Streaming with Catalytic Memory

Tamara Kaplan
Blavatnik School of Computer Science and AI, Tel Aviv University, Israel

Nimrod Kaplan
Blavatnik School of Computer Science and AI, Tel Aviv University, Israel

Haim Kaplan
Blavatnik School of Computer Science and AI, Tel Aviv University, Israel

Abstract

We introduce a streaming model that uses both catalytic and regular memory. In this model, we show how to exactly compute the frequency moments using a logarithmic number of bits of regular memory and a polynomial number of bits of catalytic memory. More generally, we show how to compute arbitrary polynomials of the item frequencies exactly within the same space bounds. As an application, we obtain catalytic streaming algorithms that exactly compute the number of distinct elements in a stream, count the number of triangles (or any other small subgraph) in a graph whose edges arrive in a stream, and identify heavy hitters.

Our algorithms for frequency moments perform a constant number of passes over the stream, and for polynomial evaluation, we require one more pass than the degree of the polynomial. In particular, for the second moment, we perform three passes over the stream. By relating our catalytic streaming model to the catalytic communication model introduced in [24], we show that catalytic memory is not useful for any one pass streaming algorithms. For lower bounds on multi pass streaming algorithms, the impossibility results of [24] are not strong enough. However, using a different technique, we show that computing the second frequency moment cannot be achieved by a two pass catalytic streaming algorithm that satisfies certain natural assumptions.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases Catalytic memory, streaming algorithms, frequency moments, space complexity, polynomial evaluation

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.70

Related Version *Full Version*: `https://arxiv.org/abs/2607.09475`

Funding *Tamara Kaplan*: This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme No 949499.
Nimrod Kaplan: Work funded by the European Union (ERC, EACTP, 101142020). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
Haim Kaplan: Work supported by Israel Science Foundation (ISF) grant no. 1156/23 and the Blavatnik Research Foundation.

1 Introduction

Catalytic computation was introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [7] as a theoretical model for understanding the computational power of a "full hard drive" (i.e., memory which can be used but has to be restored at the end to its original content). Since then, catalytic memory has attracted significant attention and has led to several interesting results in complexity theory. In particular, researchers have explored the power of non-determinism [9] and randomness [12] in this model, as well as non-uniform [13, 15, 23]

and quantum [8] versions of it. Furthermore, interesting space-efficient algorithms using catalytic memory have been developed. Notable examples include the work of Henzinger, Pyne, and Ragavan [17], who improved upon Cook and Mertz's [14] results by solving the Tree Evaluation Problem using subpolynomial catalytic memory and logarithmic regular memory. Additionally, Chmel et al. [11] provided deterministic solutions for directed connectivity and sequence alignment.

A *Catalytic* Turing machine has a working tape as of standard Turing machine and in addition it has a catalytic tape initialized arbitrarily that has to be restored to its original content by the end of the computation.[1] The class *Catalytic Logspace*, denoted by CL, contains all problems computable by a catalytic Turing machine with regular tape logarithmic in the input length and polynomial size catalytic tape. Surprisingly, Buhrman et al. [7] proved that $\mathsf{TC}^1 \subseteq \mathsf{CL}$,[2] showing that catalytic memory can be exploited to solve problems not known to lie in L.[3]

Recently, Pyne, Sheffield, and Wang [24] introduced a *catalytic communication* model, in which Alice and Bob communicate by exchanging messages over a small clean memory and large catalytic memory. They show that this model is substantially stronger than the standard one as it allows to solve certain problems such as set equality or inner product over GF(2), exactly, using only one-bit of clean memory.

Standard streaming algorithms are fundamentally constrained by strict memory limitations. Motivated by these space barriers, as well as the rich interplay between streaming and communication complexity, we propose a novel catalytic streaming model and initiate the first formal study of its capabilities.

Streaming algorithms typically have memory which is only logarithmic in the input size (so they cannot store their inputs). They traverse their inputs one by one in a single or multiple passes, maintain some information in their small memory and compute the result. Streaming algorithms have been intensively studied for many problems, starting from the seminal work of Alon, Matias, and Szegedy on the frequency moment [2], for a survey see [10, 20, 22].

1.1 The Catalytic Streaming model

In the standard multi-pass streaming model [21] (see also the surveys cited above) a sequence of m elements $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ is given sequentially as input. The algorithm is then allowed to make p passes over σ while maintaining s (sublinear in m) bits of memory.

In the *catalytic streaming model* in addition to the s bits of regular memory, the algorithm has access to a large catalytic memory of c bits. Initially, the catalytic memory contains an arbitrary string $\boldsymbol{\mathcal{T}}$ of c bits. The algorithm can use this catalytic memory however it wants, with the constraint that it must be restored to contain exactly $\boldsymbol{\mathcal{T}}$ when the algorithm ends. It's formal definition is as follows.

▶ **Definition 1** (Catalytic Streaming). *A* catalytic streaming algorithm *A has a read-only input tape, a read-only stream tape (storing the current stream element), a read-write* regular memory*, and a read-write* catalytic memory*. A gets $n, m \in \mathbb{N}$ as input, possibly additional inputs on its input tape, and a stream $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ of elements from a domain D of size n.*

[1] The terminology comes from chemistry: a catalyst enables a reaction to proceed without being consumed or permanently altered.

[2] TC^1 denotes the class of decision problems solvable by a family of polynomial-size, logarithmic-depth Boolean circuits with unbounded fan-in AND, OR, and MAJORITY (aka threshold) gates.

[3] L is the class of problems computable with memory of size logarithmic in the input length.

Let $p \in \mathbb{N}$. We say that A makes p passes *if it scans its input stream σ sequentially from x_1 to x_m, p times. Thus, during each pass, at time step j, the symbol x_j appears on the stream tape and processed by A.*

Suppose that initially the catalytic memory contains an arbitrary string $\boldsymbol{\mathcal{T}}$. We say that A computes the function F from D^m to some output range $\mathcal{O}$ in p passes if, for every stream $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$, after the p-th pass of A on input $\boldsymbol{\sigma}$

1. *The value $F(x_1, x_2, \ldots, x_m)$ appears in memory, and*
2. *The catalytic memory is restored to contain exactly the string $\boldsymbol{\mathcal{T}}$.*

We say that A uses s bits of regular memory *and c* bits of catalytic memory *if, at every point during its execution, it uses at most s bits on the regular memory and at most c bits on the catalytic memory.* ◊

▶ Remark 2. Our stream elements in this paper would be from $[n] \coloneqq \{1, 2, \ldots, n\}$ unless stated otherwise. Although our complexity measure is in bits, we describe our algorithms in terms of *registers* storing integers. We perform our computations over a ring of integers modulo some $q \in \mathbb{N}$, which we denote by Z_q. Whenever additional assumptions on the ring are needed, we will state them explicitly.

Basic catalytic algorithms rely on a symmetric execution – computing forward, copying the output, and uncomputing backward to restore the memory. This often necessitates reading the input in reverse order during the uncomputation phase. In the streaming model, however, data arrives sequentially and can only be accessed in the forward direction. This restriction breaks the standard uncomputation strategy, making the design of catalytic streaming algorithms a significant challenge even with multiple passes.

1.2 Our contributions

We introduce the catalytic streaming model (Definition 1), describe algorithms for classical streaming problems in this model, and prove lower bounds as follows.

Frequency Moments

We show that a straightforward generalization of the catalytic exponentiation algorithm given in [7] can be used to compute $F_k(\boldsymbol{\sigma}) = \sum_{i \in [n]} f_i^k$ exactly, using four passes over σ for any $k \geq 2$. Here $f_i \coloneqq \left|\{j \in [m] : x_j = i\}\right|$ is the number of occurrences of item i in σ. Computing frequency moments is maybe the most classical and well studied streaming problem. This result sets a clear separation between the model of catalytic streaming and the standard streaming model, since it's well known that any moment other than F_1 cannot be computed exactly in sub-linear space using constant number of passes. Classical results give randomized sublinear algorithms for approximating the frequency moments [2].

Similarly, we show that it is possible to extend existing catalytic algorithms for polynomial evaluation to catalytic streaming algorithms.

Polynomial Evaluation

We give a new, simple, catalytic streaming algorithm that evaluates $P(f_1, \ldots, f_n)$ in $k+1$ passes where P is a multivariate polynomial of degree k. We can evaluate this polynomial over any ring Z_q (the integers modulo q) as long as this ring contains multiplicative inverses

to $2, 3, \ldots, k$. We can do so by using $O(1)$ regular registers and n catalytic registers of $O(\log q)$ bits each. Since the evaluation of $P(f_1, f_2, \ldots, f_n)$ over the integers is $O(m^k)$,[4] then by picking q of this magnitude, we in fact compute $P(f_1, \ldots, f_n)$ over the integers.

In particular we get a three pass algorithm for computing F_2 (improving above the four pass algorithm using simple exponentiation we mentioned before).

Lower Bounds

We were intrigued by the question of whether we can compute F_2 in the catalytic streaming model in less than three passes. Using results in the recently introduced catalytic communication complexity model [24], and well known reduction from the Set-Disjointness problem in communication complexity to streaming algorithms for frequency moments, we are able to easily show that one cannot compute F_k for any $k \geq 2$ using one pass catalytic streaming algorithm and only sublinear regular memory. However, in the catalytic arena this approach fails for two passes since, as we show, there is a catalytic communication protocol with small regular memory for Set-Disjointness with three rounds.

Introducing a different technique, we are able to show that such a two pass catalytic streaming algorithm with sublinear regular memory for F_2 does not exist at least among a large family of catalytic two pass streaming algorithms. A natural extension of this family to algorithms with more than two passes includes all algorithms presented here and many others.

Applications: F_0, Counting Subgraphs, and Heavy Hitters

Finally, we apply our algorithms for moments and polynomial evaluation to obtain space efficient catalytic streaming algorithms for several other well studies streaming problems. In particular, we get a four pass algorithm to compute the exact number of distinct items in a stream. We show how to count triangles (or any other small subgraph) in a multigraph whose edges arrive as a stream. We also show that we can compute the exact set of F_k heavy hitters in $O(\log n)$ passes.[5]

1.3 Roadmap

We follow the notation of Definition 1. Section 2 shows how to compute moments in four passes. Section 3 gives our polynomial evaluation algorithm. We prove lower bounds in Section 4. Section 5 shows how to apply our algorithm to compute the number of distinct elements in a stream (so called F_0), count small subgraphs in a graph stream, and compute heavy hitters.

2 Computing Exact Moments Using 4 Passes and Logarithmic Space

In this section, we study the classic problem of computing the k-th frequency moment of a stream and prove the following theorem.

▶ **Theorem 3.** *Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream over $[n]$, and let $f_1, f_2, \ldots, f_n$ denote the corresponding frequencies. Then there exists a catalytic streaming algorithm that evaluates $F_k(\boldsymbol{\sigma}) \coloneqq \sum_{i=1}^{n} f_i^k$ in 4 passes, using $O(1)$ regular registers and $n \cdot (k+1)$ catalytic registers. This algorithm works over any ring Z_q, to compute F_k over the integers it is enough to take $q = O(m^k)$.*

[4] The hidden constant depends on the largest coefficient of a monomial.
[5] All items i such that $f_i^k \geq \epsilon F_k$.

A key lemma in the proof of the above theorem is a result proved in [7] called *the powering lemma*. We formulated it here in a suitable way for our task.

▶ **Lemma 4** ([7, Powering, Lemma 10 (modified)]). *Let $k \in \mathbb{N}$. Let r and $r_1, r_2, \ldots, r_k$ be catalytic registers, initiated with arbitrary values τ, $\tau_1, \tau_2, \ldots, \tau_k$ respectively, and r_o be a regular register. There are programs I_1, I_2, and I_3 that only use the registers $\{r_i\}_{i\in[k]}$ and r_o such that for every input $f \in [n]$ the program*

$$I_1,\ r \leftarrow r + f,\ I_2,\ r \leftarrow r - f,\ I_3$$

computes $r_o \leftarrow f^k$. Moreover, the values of the other registers are as follows

$$r = \tau, \quad \text{and for every } i \in [k], \quad r_i = \tau_i - (\tau + f)^{k-i}.$$

They can be restored by executing the program in reverse.

For convenience we give the algorithm used to prove Lemma 4 by [7] in Appendix A.

Proof of Theorem 3. Let $r_1, r_2, \ldots, r_n$ and $\{r_{i,j}\}_{i\in[n],\, j\in[k]}$ be catalytic registers, and let r_o be a regular output register. Given Lemma 4, computing $F_k(\boldsymbol{\sigma})$ is straightforward. We simply run the algorithm of Lemma 4 in parallel for each item $i \in [n]$, using the frequency f_i in place of the value f appearing in that lemma.

All of these parallel executions share the same output register r_o, so their contributions accumulate to $\sum_{i=1}^{n} f_i^k = F_k(\boldsymbol{\sigma})$. On the other hand, each execution uses its own catalytic registers, i.e. for the computation corresponding to f_i, we use r_i in the role of the register r, and $r_{i,1}, r_{i,2}, \ldots, r_{i,k}$ in the roles of the registers $r_1, r_2, \ldots, r_k$. Denote by τ_i the initial value of r_i and $\tau_{i,j}$ the initial value of $r_{i,j}$.

The resulting algorithm is given in Algorithm 1. There, for $t \in \{1, 2, 3\}$, I_t^i denotes the program I_t applied to the registers dedicated for f_i. Observe that before the two additional passes mentioned in line 7 the values of the registers $r_1, r_2, \ldots, r_n$ are restored, but each register $r_{i,j}$ still contains $\tau_{i,j} - (\tau_i + f_i)^{k-j}$, thus two more passes are needed. ◀

Algorithm 1 4-Pass F_k Computation.

1: Initial State: $r_o = 0\ \forall i \in \{1, 2, \ldots, n\},\ r_i = \tau_i,\ c_i = (-1)^i \binom{k}{i}, \forall j \in \{1, \ldots, k\}, r_j^i = \tau_j^i$
2: $\forall i \in [n]$, run I_1^i
3: **for** every $x_\ell \in \boldsymbol{\sigma}$ **do** $r_{x_\ell} \leftarrow r_{x_\ell} + 1$ ▷ 1st pass
4: $\forall i \in [n]$, run I_2^i
5: **for** every $x_\ell \in \boldsymbol{\sigma}$ **do** $r_{x_\ell} \leftarrow r_{x_\ell} - 1$ ▷ 2nd pass
6: $\forall i \in [n]$, run I_3^i
7: Execute the inverse of every operation in reverse order without changing r_o using two more passes to restore $r_1, \ldots, r_n$.
8: **return** r_o

Polynomial Evaluation by Powering

Alexseev et al. [1] show how to evaluate a multivariate polynomial of degree k over n variables by representing it as a linear combination of powers ($\leq k$) of linear functions of the variables (so called Waring representation [19]). However (due to the large number of coefficients required for this representation) their program is for a specific polynomial, it does not get a representation of the polynomial as an input. Their implementation requires catalytic

memory of size exponential in k. Similarly to our catalytic streaming implementation of the computation of F_k, we can implement this polynomial evaluation scheme in 4 passes in our model. We will get a streaming algorithm for a **specific** multivariate polynomial P, that given a stream evaluates $P(f_1, \ldots, f_n)$ (it cannot get a polynomial as input). Furthermore, it requires exponential in d catalytic memory.

3 Polynomial of Degree k Using $k + 1$ Passes

In this section we show how to evaluate polynomials over the frequencies of items in a stream. We assume we get the polynomial's coefficients in our input tape, like we get m and n. The following is this section's main result.

▶ **Theorem 5.** *Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream over $[n]$, and let $f_1, f_2, \ldots, f_n$ denote the corresponding frequencies. Then there exists a catalytic streaming algorithm that for every multivariate polynomial $P \in \mathbb{F}[y_1, y_2, \ldots, y_n]$ of total degree k,[6] evaluates $P(f_1, f_2, \ldots, f_n)$ in $k + 1$ passes. This algorithm uses $O(1)$ regular register and n catalytic registers. It works over any ring Z_q that contains inverses to $2, 3, \ldots, k$, to evaluate $P(f_1, f_2, \ldots, f_n)$ over the integers it is enough to take $q = O(\alpha \cdot m^k)$ where $\alpha \in \mathbb{Z}$ is the largest coefficient of any monomial*

3.1 Warm Up: Degree 2 Polynomials

To warm up, we first show how to evaluate degree 2 polynomials.

Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream of elements from the set $[n]$ with frequencies $f_1, f_2, \ldots f_n$ and let $P \in \mathbb{F}[y_1, y_2, \ldots, y_n]$ be a multivariate polynomial of degree 2. Let

$$P(f_1, f_2, \ldots, f_n) = \sum_{1 \le i \le j \le n} a_{i,j} f_i f_j \,,$$

for some $\{a_{i,j}\} \subseteq \mathbb{F}$, i.e. we assume P only contains degree 2 monomials. Solving the problem with respect to such polynomials is easily generalized, since adding degree-1 monomials and constants is trivial using $O(\log m)$ bits of regular memory.

We allocate registers $r_1, r_2, \ldots, r_n$ in the catalytic memory, each of size $2 \log m$ bits. We denote the initial values inside these registers by $\tau_1, \tau_2 \ldots, \tau_n$. In addition, we allocate $2 \log m$ bits of regular memory as our output. We give this register a name, r_o, and think of it as the output register. The algorithm goes as follows.

1. Set $r_o \leftarrow P(r_1, r_2, \ldots, r_n)$.
2. Over the first pass of the stream, for every item x_j received, increment the register r_{x_j} by one. After this pass, r_i holds $\tau_i + f_i$.
3. Set $r_o \leftarrow r_o - 2P(r_1, \ldots, r_n)$.
4. Over the second pass, we repeat Step (2). At the end of this pass, r_i holds $\tau_i + 2f_i$.
5. Set $r_o \leftarrow r_o + P(r_1, \ldots, r_n)$.
6. Over the third pass, we reverse our previous operations, i.e. for every x_j received, we subtract 2 from its corresponding register.
7. Return $r_o/2$.

[6] $\mathbb{F}[y_1, y_2, \ldots, y_n]$ is the ring of polynomials over the field$\mathbb{F}$ with the variables $y_1, y_2, \ldots, y_n$ n. P's coefficients are given to the algorithm in the input tape.

To establish correctness, we follow the contents of r_o along the execution of these steps. We use the following definitions.

$$J = \sum_{1\le i\le j\le n} a_{ij}\tau_i\tau_j \qquad \text{(the junk term)},$$
$$C = \sum_{1\le i\le j\le n} a_{ij}(\tau_i f_j + f_i\tau_j) \qquad \text{(the cross term), and}$$
$$V = \sum_{1\le i\le j\le n} a_{ij} f_i f_j \qquad \text{(the target term).}$$

We assume that the regular memory is initialized to $\mathbf{0} = 0^s$, hence initially $r_o = 0$. Following Step (1)

$$r_o = P(\tau_1, \tau_2, \ldots, \tau_n) = \sum_{1\le i\le j\le n} a_{i,j}\tau_i\tau_j = J\,.$$

Following Step (3),

$$\begin{aligned} r_o &= J - 2P(r_1, r_2, \ldots, r_n) = J - 2P(\tau_1 + f_1, \tau_2 + f_2, \ldots, \tau_n + f_n) \\ &= J - 2\sum_{1\le i\le j\le n} a_{i,j}(\tau_i + f_i)(\tau_j + f_j) = J - 2(J + C + V)\,. \end{aligned}$$

Following Step (5),

$$\begin{aligned} r_o &= J - 2(J + C + V) + \sum_{1\le i\le j\le n} a_{i,j}(\tau_i + 2f_i)(\tau_j + 2f_j) \\ &= J - 2(J + C + V) + (J + 2C + 4V) = 2V \end{aligned}$$

It follows that when Step (5) ends, $r_o = 2V = 2P(f_1, f_2, \ldots, f_n)$. Hence $r_o/2$ is exactly $P(f_1, f_2, \ldots, f_n)$, which is the desired output.

3.2 Proof of Theorem 5

In order to prove the theorem, we first handle the case of evaluating a single monomial.

▶ **Lemma 6.** *Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream over $[n]$, and let $f_1, f_2, \ldots, f_n$ denote the corresponding frequencies. There exists a catalytic streaming algorithm such that for any given monomial $M \in \mathbb{F}[y_1, y_2, \ldots, y_n]$ of total degree k, evaluates $M(f_1, f_2, \ldots, f_n)$ in $k+1$ passes. The algorithm uses $O(1)$ regular registers and n catalytic registers. This algorithm works over any ring Z_q that contains inverses to $2, 3, \ldots, k$. To evaluate $M(f_1, \ldots, f_n)$ over the integers, it is enough to take $q = O(m^k)$.*

Proof. Let $r_1, r_2, \ldots, r_n$ be the catalytic registers, where each r_i initially contains the value τ_i, and let r_o be the regular register. Evaluating M on $(f_1, f_2, \ldots, f_n)$ goes as follows (Algorithm 2):

Algorithm 2 Compute Degree-k Monomial $M(f_1, f_2, \ldots, f_n)$.

Input: A monomial M and a sequential stream $\boldsymbol{\sigma}$. ▷ **Initially**: $r_o = 0$, $\forall i,\ r_i = \tau_i$
Output: The evaluation $M(f_1, f_2, \ldots, f_n)$.

```
r_o ← r_o + (−1)^k M(r_1, r_2, ..., r_n)
for i = 1, ..., k do
    for every x_j ∈ σ do r_{x_j} ← r_{x_j} + 1 end for        ▷ The i-th pass
    r_o ← r_o + (−1)^{k−i} C(k,i) M(r_1, r_2, ..., r_n)
end for
for every x_j ∈ σ do r_{x_j} ← r_{x_j} − k end for          ▷ The k + 1-th pass
return r_o / k!
```

Observe that for every $1 \le i \le k$ and $j \in [n]$ after the i-th pass, the value in register r_j is $\tau_j + i \cdot f_j$. Thus, by summing the values added to r_o in every pass, we get

$$\begin{aligned} r_o &= (-1)^k M(\tau_1, \ldots, \tau_n) + \sum_{i=1}^{k} (-1)^{k-i} \binom{k}{i} M(\tau_1 + if_1, \ldots, \tau_n + if_n) \\ &= \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} M(\tau_1 + if_1, \ldots, \tau_n + if_n)\,. \end{aligned} \tag{1}$$

Since the exact degree of M is k, we can choose $\{i_\alpha\}_{\alpha\in[k]} \subseteq [n]$ (these indices are not necessary distinct) such that $M(y_1, y_2, \ldots, y_n) = \prod_{\alpha=1}^{k} y_{i_\alpha}$. Therefore, by expanding M in Equation (1) in that way we get

$$r_o = \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} \prod_{\alpha=1}^{k} (\tau_{i_\alpha} + if_{i_\alpha}) = \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} \sum_{\ell=0}^{k} i^{k-\ell} \mathrm{Cross}_\ell(\boldsymbol{\mathcal{T}}, \boldsymbol{f})$$

where $\boldsymbol{\mathcal{T}} = \tau_1, \tau_2, \ldots, \tau_n$, $\boldsymbol{f} = f_1, f_2, \ldots, f_n$, and

$$\mathrm{Cross}_\ell(\boldsymbol{\mathcal{T}}, \mathbf{f}) = \sum_{\substack{\beta=\{\beta_1,\ldots\beta_\ell\}\subseteq[k] \\ \gamma_1,\ldots,\gamma_{k-\ell}=[k]\setminus\beta}} \tau_{\beta_1} \cdots \tau_{\beta_\ell} \cdot f_{\gamma_1} \cdots f_{\gamma_{k-\ell}}$$

where the sum is over all partitions of $\{i_\alpha\}_{\alpha\in[k]}$ into two sets, $\{\beta_j\}_{j\in[\ell]}$ and $\{\gamma_t\}_{t\in[k-\ell]}$.

By changing the order of summation we get that

$$r_o = \sum_{\ell=0}^{k} \mathrm{Cross}_\ell(\boldsymbol{\mathcal{T}}, \boldsymbol{f}) \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} i^{k-\ell}\,. \tag{2}$$

Recall the definition of Stirling numbers of the second kind:

$$S(n,k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} i^n\,.$$

A key property is that $S(n,k) = 0$ whenever $n < k$, and $S(k,k) = 1$. Using these identities to rewrite Equation (2) we get

$$r_o = \sum_{\ell=0}^{k} \mathrm{Cross}_\ell(\boldsymbol{\mathcal{T}}, \boldsymbol{f}) \cdot k! \cdot S(k-\ell, k) = k! \cdot \mathrm{Cross}_0(\boldsymbol{\mathcal{T}}, \boldsymbol{f}) = k! \cdot \prod_{\alpha=1}^{k} f_{i_\alpha} = k! \cdot M(f_1, f_2, \ldots, f_n)\,.$$

Therefore $\frac{r_o}{k!} = M(f_1, f_2, \ldots, f_n)$ and the returned value is correct. ◀

We are now ready to prove Theorem 5.

Proof of Theorem 5. Denote the set of monomials with nonzero coefficient in $P(\boldsymbol{y})$ by $\mathrm{Mon}(P)$ and let $a_M \in Z_q$ be the coefficient of the monomial $M(\boldsymbol{y})$ in $P(\boldsymbol{y})$. Moreover, let $r_1, r_2, \ldots, r_n$ be catalytic registers and r_o a regular register. Algorithm 3 shows how to evaluate P on $f_1, f_2, \ldots, f_n$.

Algorithm 3 Compute Degree-k Polynomials.

Input: A polynomial P and a sequential stream $\boldsymbol{\sigma}$. ▷ **Initially**: $r_o = 0$, $\forall i$, $r_i = \tau_i$
Output: The evaluation $P(f_1, f_2, \ldots, f_n)$.

for $i = 1, \ldots, k$ **do** ▷ The i-th pass
  for every $x_j \in \boldsymbol{\sigma}$ **do** $r_{x_j} \leftarrow r_{x_j} + 1$ **end for**
  for $M \in \mathrm{Mon}(P)$ **do**
    $r_o \leftarrow r_o + a_M(-1)^{\deg(M)-i}\binom{\deg(M)}{i} M(r_1, r_2, \ldots, r_n) \;/\; \deg(M)!$
  end for
end for
for every $x_j \in \boldsymbol{\sigma}$ **do** $r_{x_j} \leftarrow r_{x_j} - k$ **end for** ▷ The $k+1$-th pass
for $M \in \mathrm{Mon}(P)$ **do**
  $r_o \leftarrow r_o + a_M(-1)^{\deg(M)} M(r_1, \ldots, r_n) \;/\; \deg(M)!$
end for
return r_o

The idea is to evaluate in parallel all monomials of P using the same set of registers. Since summation is commutative different monomials do not interfere with one another. The correctness of Algorithm 3 essentially follows as in the proof of Lemma 6. ◀

In particular we get the following corollary for evaluating F_2.

▶ **Corollary 7.** *Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream over $[n]$, and let $f_1, f_2, \ldots, f_n$ denote the corresponding frequency vector. Then there exists a catalytic streaming algorithm that evaluates $F_2(\boldsymbol{\sigma}) = \sum_{i=1}^{m} f_i^2$ in 3 passes, using $O(1)$ regular registers and n catalytic registers. This algorithm works over any ring Z_q that contains inverses to $2, 3, \ldots, k$. To evaluate F_2 over the integers, it is enough to take $q = O(m^2)$.*

3.3 Polynomial Evaluation over a Field Using Primitives Roots of Unity

In Cook and Mertz [14], the authors show how to evaluate any multivariate polynomial $P(y_1, \ldots, y_n)$ of degree k over a finite field $\mathbb{F}_q$ under the assumption that $k + 1 < q$. Their algorithm requires a primitive root of unity of order $e \geq k + 1$. Note that the smallest e for which such a root exists is the smallest divisor of $q - 1$ which is larger than k. Their algorithm reads each input variable e times, and uses $\log q$ bits of regular memory together with $n \log q$ bits of catalytic memory. As we did for F_k we can modify this algorithm to a catalytic streaming algorithm that evaluates $P(f_1, \ldots, f_n)$ in $e \geq k + 1$ passes over the stream. To evaluate $P(f_1, \ldots, f_n)$ over the integers we have to use a field of characteristic $\Omega(m^k)$. Note that it is not clear how to find a field of characteristic $O(m^k)$ that contains a primitive root of unity of order $k + 1$. Hence this algorithm may require more than $k + 1$ passes. The algorithm presented in this section, which is arguably simpler, always evaluates P in exactly $k + 1$ passes and requires only milder assumptions on the underlying ring.

Goldreich [16] simplified the Cook-Mertz procedure. He observed that Cook and Mertz in fact interpolate the univariate polynomial $f(x) = P(x \cdot \tau_1 + f_1, \ldots, x \cdot \tau_n + f_n)$ at powers of a primitive root of unity, but one can also interpolate at other points. (Since we only need to extract $f(0)$ at the end.) This removes the need for a primitive root of unity, however, when we attempt to implement it in the catalytic streaming model it would require twice as many passes. This happens since to evaluate at arbitrary values of x requires two passes per value (we need to subtract away the frequencies f_i, multiple the noise by a different value of x and then add back the frequencies). But powers of a primitive root of unity allow to get away with one pass per value of x.

▶ Remark 8. In this section we've evaluated polynomials over the frequencies of the elements in the stream. Note that this can be easily generalized to evaluating ℓ-variate polynomials over every set of functions of the steam elements $\{g_1(\boldsymbol{\sigma}), \ldots, g_\ell(\boldsymbol{\sigma})\}$, such that we can compute $r_i \leftarrow g_i(\boldsymbol{\sigma})$ for every i in one pass and within our space bounds.

4 Lower Bounds

Standard streaming lower bounds are typically proved via reductions from communication complexity. In this section, we apply this approach to the catalytic setting. Specifically, in Subsection 4.1, we leverage the recently introduced framework of catalytic communication complexity [24] to establish lower bounds for the catalytic streaming model.

However, catalytic communication results cannot establish lower bounds for multi-pass catalytic streaming. As we show below, this is because a modified inner product protocol from [24] actually solves Set-Disjointness. Using alternative techniques, we instead prove that our 3-pass algorithm for F_2 is pass-optimal among a family of "natural" algorithms.

4.1 Implication of Catalytic Communication

In this section, we use results from [24] to establish a lower bound for one-pass catalytic streaming algorithms. We then adapt their method for computing inner product over GF_2 to obtain catalytic communication protocol for Set-Disjointness with three rounds of communication that uses only $O(\log n)$ bits of regular memory. This latter result shows that the typical approach to prove streaming lower bounds via reductions from Set-Disjointness cannot yield lower bounds in our new model for algorithms with more than one pass.

Briefly, the communication model of [24] is as follows (for exact definition, see [24, Definition 1]). Alice, holding an input $\boldsymbol{x} \in \{0,1\}^n$, and Bob, holding an input $\boldsymbol{y} \in \{0,1\}^n$ communicate to compute a function $f(\boldsymbol{x}, \boldsymbol{y})$. They exchange messages on c bits of catalytic memory (arbitrarily initialized) and s bits of regular memory (initialized to $\mathbf{0}$), note that the message size in every round is fixed. In their protocol the last player sending a message should send/restore the original contents of the catalytic memory. The player that gets this message should output $f(\boldsymbol{x}, \boldsymbol{y})$.

A lower bound on the amount of regular memory required by catalytic streaming algorithm for computing the frequency moment F_k follows from the following three facts: (1) We observe that the standard reduction from the Set-Disjointness problem in communication complexty to the streaming problem of computing the frequency moment works also in the catalytic setting. The Set-Disjointness problem in communication complexity asks how many bits of communication Alice, holding $x \in \{0,1\}^n$, and Bob, holding $y \in \{0,1\}^n$, have to exchange in order to compute the function

$$\mathrm{Disj}_n(x,y) = \begin{cases} 1 & \forall i \in [n] : x_i = 1 \Rightarrow y_i = 0 \\ 0 & \text{otherwise} \end{cases}.$$

Specifically, our lower bound relies on three facts: (1) A p-pass catalytic streaming algorithm for a frequency moment F_k yields a $2p$-round catalytic communication protocol for Set-Disjointness (Disj_n) with the same catalytic memory and $O(\log n)$ regular memory (a standard reduction provided in Appendix B). (2) Any catalytic communication protocol with fewer than three rounds can be simulated without catalytic memory and similar regular communication cost [24, Proposition 3]. (3) Computing Disj_n strictly requires $\Omega(n)$ standard communication [2, 18, 25]. Combining these, we immediately obtain the following:

▶ **Proposition 9.** *There is no one-pass catalytic streaming algorithm for frequency moments with sublinear regular memory.*

Proof. Assume there exists a 1-pass catalytic streaming algorithm with sublinear regular memory. By Fact (1), this yields a 2-round catalytic protocol for Disj_n with sublinear regular communication. By Fact (2), this implies a standard 2-round protocol for Disj_n with sublinear communication, which contradicts Fact (3). ◀

While similar reductions can rule out other one-pass catalytic streaming algorithms, this approach fails for multiple passes (which induce protocols with more than two rounds). This barrier is inherent: in Appendix C, we show that a simple modification of the GF_2 inner-product protocol from [24, Proposition 5] yields a 3-round catalytic protocol for Disj_n. Thus, multi-pass lower bounds cannot be obtained via this route.

4.2 A Restricted Lower Bound for Moments in Two Passes

In this section, we prove that under certain assumptions on the way it must operate no two pass catalytic streaming algorithm can compute F_k exactly for $k \geq 2$.[7]

▶ **Assumption 10** (Restriced two-pass Catalytic Streaming Algorithm)**.** *We denote by C the catalytic memory, r_i is a catalytic register associated with item i, and M is the regular memory. A restricted two-pass catalytic streaming algorithm obeys the following structure:*

1. ***Initial Update (Before we see the stream)**: $M \leftarrow M + W_s(\boldsymbol{C})$, where W_s is some function of the entire catalytic memory $\boldsymbol{C}$. It returns a vector that we use in order to update the clean memory M additively.*
2. ***Pass 1 (Forward):** For each item x_j in the stream, if $x_j = i$ then we set $r_i \leftarrow U_i(r_i)$, and then $M \leftarrow M + P_i(r_i)$. Here U_i is an item-specific bijection $U_i : \Sigma \to \Sigma$. We require that $r_i \neq r_j$ for $i \neq j$. P_i is also an item specific function that updates M additively after the change to r_i.*
3. ***Intermediate Update (End of Pass 1):** $M \leftarrow M + W_m(\boldsymbol{C})$. W_m is a function of the entire catalytic memory $\boldsymbol{C}$ that we use to update the clean memory between the two passes.*
4. ***Pass 2 (Backward):** To restore the catalytic memory, if $x_j = i$, we set $r_i \leftarrow U_i^{-1}(r_i)$, then $M \leftarrow M + Q_i(r_i)$. Q_i is an item specific function that updates M additively after the update to r_i.*
5. ***Final Update (End of Pass 2):** $M \leftarrow M + W_e(\boldsymbol{C})$. W_e is a function of the entire catalytic memory $\boldsymbol{C}$ that we use to update the clean memory between following the two passes.*

[7] Our restrictions force the algorithm to use the catalytic memory. Consequently we do not need an explicit restriction on the size of the clean memory.

▶ Remark 11. Assumption 10 can be generalized to apply to p-pass streaming algorithms. If $U_{i,j}$ is the function applied to r_i in pass j when we see item i, then we should have $U^{\ell}_{i,p}(\cdots(U^{\ell}_{i,2}(U^{\ell}_{i,1}(\tau_i))))=\tau_i$ for every $\ell \le m$. Further generalizations of this assumption allocate a set of registers to every element instead of only one register and allow to apply some function $g_{i,j}$ to each r_i after pass j such that $g_{i,p}(U^{\ell}_{i,p}(\cdots(g_{i,2}(U^{\ell}_{i,2}(g_{i,1}(U^{\ell}_{i,1}(g_{i,0}(\tau_i))))))))=\tau_i$ for every $\ell \le m$. All our algorithms obey this generalized assumption.

Now we show that no program that satisfies Assumption 10 can compute F_2 exactly using only two passes. The proof for F_k, $k>2$ is analogous. The intuition is that the catalytic tape behaves as a one time pad with respect to the items frequencies, so, in one pass it is impossible to extract information about them from the catalytic memory. Formally, we prove this by analyzing the discrete derivative of the output as a function of f_i.

The Discrete Derivative Argument

Let $\boldsymbol{C}=\boldsymbol{\mathcal{T}}$ and in particular let $r_i=\tau_i$ initially for all $i\in[n]$. For every $i\in[n]$ and $j\in[f_i]$ we denote by U_i^j the bijective function U_i applied j times. We analyze the state of M at the end of the algorithm, denoted by $M_e(\boldsymbol{\mathcal{T}},\boldsymbol{f})$.

$$M_e(\boldsymbol{\mathcal{T}},\boldsymbol{f})=W_s(\boldsymbol{\mathcal{T}})+\sum_{i=1}^{n}\sum_{j=1}^{f_i}P_i\big(U_i^j(\tau_i)\big)+W_m\big(\boldsymbol{U^f}(\boldsymbol{\mathcal{T}})\big)+\sum_{i=1}^{n}\sum_{j=0}^{f_i-1}Q_i\big(U_i^j(\tau_i)\big)+W_e(\boldsymbol{\mathcal{T}})$$

where $\boldsymbol{U^f}(\boldsymbol{\mathcal{T}})=\Big(U_1^{f_1}(\tau_1),\ldots,U_n^{f_n}(\tau_n)\Big)$. The second term from the right comes from the fact that in the second pass (see Item 4) after seeing item i for the j-th time, the value of the register r_i is $U_i^{f_i-j}(\tau_i)$.

For simplicity, for the rest of the proof we denote

$$E_i(\tau_i,f_i)=\sum_{j=1}^{f_i}P_i\big(U_i^j(\tau_i)\big)+\sum_{j=0}^{f_i-1}Q_i\big(U_i^j(\tau_i)\big)$$

▶ **Theorem 12.** *There is no algorithm satisfying Assumption 10 for which* $M_e(\boldsymbol{\mathcal{T}},\boldsymbol{f})=\sum_{i=1}^{n}f_i^2$.[8]

Proof. Assume, by way of contradiction, that for every initial state $\boldsymbol{\mathcal{T}}$ of the catalytic tape and for every set of f_i's we have an algorithm such that

$$M_e(\boldsymbol{\mathcal{T}},\boldsymbol{f})=\sum_{i=1}^{n}E_i(\tau_i,f_i)+W_s(\boldsymbol{\mathcal{T}})+W_m\big(\boldsymbol{U^f}(\boldsymbol{\mathcal{T}})\big)+W_e(\boldsymbol{\mathcal{T}})=\sum_{i=1}^{n}f_i^2 \tag{3}$$

We take the discrete partial derivative of M_e with respect to f_i, which in our case is defined as $\Delta_{f_i}(g)\coloneqq g(f_1,\ldots,f_i+1,\ldots,f_n)-g(\boldsymbol{f})$, of both sides of Equation (3). On the right hand side we get:

$$\Delta_{f_i}\left(\sum_{j=1}^{n}f_j^2\right)=(f_i+1)^2-f_i^2=2f_i+1\,.$$

[8] We assume w.l.o.g. that in the end of the algorithm we only have F_2 on our regular memory.

For the left hand side first observe that

$$\Delta_{f_i}\left(\sum_{i=1}^{n} E_i(\tau_i, f_i)\right) = P_i\big(U_i^{f_i+1}(\tau_i)\big) + Q_i\big(U_i^{f_i}(\tau_i)\big) = P_i\big(U_i(y)\big) + Q_i\big(y\big) \ ,$$

where $y = U_i^{f_i}(\tau_i)$ is the state of register r_i at the end of the forward pass. Note that $\Delta_{f_i}\big(W_s(\boldsymbol{\mathcal{T}})\big) = \Delta_{f_i}\big(W_e(\boldsymbol{\mathcal{T}})\big) = 0$. Overall, we get

$$\Delta_{f_i}\big(M_e(\boldsymbol{\mathcal{T}}, \boldsymbol{f})\big) = P_i\big(U_i(y)\big) + Q_i\big(y\big) + \\ W_m\left(U_1^{f_1}(\tau_1), \ldots, \boldsymbol{U_i(y)}, \ldots, U_n^{f_n}(\tau_n)\right) - W_m\left(U_1^{f_1}(\tau_1), \ldots, \boldsymbol{y}, \ldots, U_n^{f_n}(\tau_n)\right) \ .$$

We argue that $\Delta_{f_i}\big(M_e(\boldsymbol{\mathcal{T}}, \boldsymbol{f})\big) = 2f_i + 1$ is not possib which would give a contradiction.

We fix f_j and τ_j for $j \neq i$, and think of $\Delta_{f_i}\big(M_e(\boldsymbol{\mathcal{T}}, \boldsymbol{f})\big)$ as a function of $y = U_i^{f_i}(\tau_i)$, which we denote by $H_i(y)$. Note that the dependency of $\Delta_{f_i}\big(M_e(\boldsymbol{\mathcal{T}}, \boldsymbol{f})\big)$ on f_i is only through $y = U_i^{f_i}(\tau_i)$. Let τ_i and f_i be random variables, each distributed uniformly over its set of possible values. The intuition is as follows. Since τ_i is a uniform random variable and U_i is a bijection it follows that y is also distributed uniformly at random and independent of f_i. Therefore it cannot hold any information about f_i. This is formalized in the following claim.

▷ Claim 13. Let f_i and τ_i be independent random variables over the uniform distribution and let $I(\cdot, \cdot)$ be the mutual information function. Then $I(f_i, y) = 0$.

Proof. By Assumption 10, U_i is a bijection, thus it holds that $\forall f_i,\ U_i^{f_i} : \Sigma \to \Sigma$ is also a bijection. Therefore we have that y is a random variable that is distributed uniformly over Σ. In particularly he have

$$\Pr_{\tau_i \sim U_\Sigma}\left[y = \sigma \mid f_i = k\right] = \Pr_{\tau_i \sim U_\Sigma}\left[U^k(\tau_i) = \sigma\right] = \frac{1}{|\Sigma|}$$

Where the last equality follows from the fact that $U_i^k(\cdot)$ is a bijection. Since the conditional probability of y is completely independent of the choice of f_i we have that $I(f_i, y) = 0$. ◁

Since $H_i(y)$ is a deterministic function of y we have that $I(f_i, H_i(y)) \leq I(f_i, y) = 0$ by the data processing inequality and Claim 13. We've assumed the algorithm always computes F_2 exactly, therefore $H_i(y) = 2f_i + 1$, which implies that $I\big(f_i, H(y)\big) = 1$ in contradiction. ◀

The last proof works with small changes for every F_k with $k \geq 2$.

In Appendix D we show where the proof fails for 3 passes.

5 Applications

In this section, we apply our polynomial evaluation algorithm to design multi-pass streaming algorithms for the exact computation of quantities that are notoriously difficult to evaluate exactly in the standard streaming model. Specifically, we present algorithms for counting distinct elements, counting occurrences of small subgraphs in an edge stream, and identifying the exact set of frequent elements.

5.1 F_0 Using Powering

The zero moment of a stream $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ over $[n]$, is $F_0(\boldsymbol{\sigma}) := \sum_{i=1}^{n} \mathbf{1}_{f_i > 0}$; that is the number of distinct items that appear in σ. Computing F_0 in the standard streaming model using a constant number of passes is known to be hard. In particular, [2] show that any p-pass algorithm requires $\Omega(min\{m, n\}/p)$ bits of memory. In contrast, in the catalytic streaming model we have the following Lemma.

▶ **Lemma 14** (F_0 Computation). *Let $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$ be a stream over $[n]$. There exists a catalytic streaming algorithm that evaluates $F_0(\boldsymbol{\sigma})$ in 4 passes, using one regular register and n catalytic registers. This algorithm works over the field $\mathbb{F}_p$ for every prime $p > m$.*

Proof. By Fermat's little theorem we have $f_i^{p-1} \equiv 1 \pmod{p}$ for every $f_i \not\equiv 0 \pmod{p}$, and 0 whenever $f_i \equiv 0 \pmod{p}$. Therefore, evaluating the $p-1$-th moment of $\boldsymbol{\sigma}$ over the field $\mathbb{F}_p$ gives us $F_0(\boldsymbol{\sigma})$. ◀

▶ Remark 15. Note that by Bertrand's postulate there exists a prime $m < p \leq 2m$, so the registers used require at most $\log(m) + 1$ bits.

5.2 Finding Subgraphs in a Stream Using Polynomials

Consider a multigraph $G = (V, E)$ with $|V| = n$ and $|E| = m$. In the graph streaming model, G is revealed sequentially as a stream of edges $e_1, e_2, \ldots, e_m$. A fundamental problem in this setting is computing the number of occurrences of a small target subgraph H (e.g., a triangle or four cycle) within G, or simply detecting its presence. This task is difficult because local structural information is fragmented across the stream. In fact, exact counting, or even detecting a single instance of a simple subgraph like a triangle, requires $\Omega(m/p)$ bits of memory in the worst case for any p-pass streaming algorithm [3, 4, 6].

▶ **Lemma 16** (Subgraph Counting). *Let $H = (V_H, E_H)$ be a fixed target multigraph. There exists an algorithm that, given an edge stream of length m over a vertex set V of size n, computes the exact number of occurrences of H in the underlying graph G. The algorithm requires $|E_H|+1$ passes over the stream, uses $O(|E_H|+|V_H|)$ regular registers and n^2 catalytic registers, and works over Z_q with $q = O\big(\max\{m^{|E_H|}, n\}\big)$.*

Proof. For simplicity, consider first the case where H is a triangle.

▶ **Observation 17.** *Denote by $x_{u,v}$ the number of occurrences of the edge (u, v) in G. The exact number of triangles in G is therefore given by the polynomial:*

$$P(\{x_{u,v} \mid (u,v) \in E\}) = \sum_{u<v<w \in V} x_{u,v} \cdot x_{v,w} \cdot x_{u,w} \,.$$

Given this observation, we can compute the number of triangles by evaluating P over the stream using Algorithm 3 for cubic polynomials.

We generate these coefficients of the polynomial $P(\{x_{u,v} \mid (u,v) \in E\})$ on the fly. To do so, we allocate 3 registers in our regular memory, to index the vertices u, v, and w. By incrementally updating these registers, we iterate through all triplets (u, v, w) such that $u < v < w$. For each such triplet, the corresponding monomial $x_{u,v} \cdot x_{v,w} \cdot x_{u,w}$ appears in P with coefficient 1.

Generalization to Arbitrary Subgraphs

To generalize this approach to an arbitrary subgraph $H = (V_H, E_H)$, we define a polynomial of degree $k = |E_H|$. For every set of $|V_H|$ vertices and assignment of labels to the vertices we generate a monomial. This generates each monomial as many times as the number of automorphisms of H. So we divide the result by the cardinality of the automorphism group of H. This number should either be given as input or if H is small it can be computed in (regular) memory polynomial in the size of H, by checking all permutations of the vertices in H lexicographically. We evaluate this polynomial over the frequencies of edges using

Algorithm 3. As for triangles we do not store the polynomial but traverse the monomials using $|V_H|$ registers of size $\log n$ each. Applying Theorem 5, this takes $|E_H|+1$ passes, and calculations are over some ring which requires registers of size at most $2\left(\frac{m}{|E_H|}\right)^{|E_H|}$ bits since $\left(\frac{m}{|E_H|}\right)^{|E_H|}$ is an upper bound on the number of occurrences of H in a multigraph with m edges. ◀

5.3 F_2 Heavy Hitters

We use our three pass algorithm for F_2 to compute ϵF_2-heavy hitters. We say that an element i is ϵF_2-heavy hitter if $f_i^2 \geq \epsilon F_2$. We do it in logarithmic number of passes and with $O(1/\epsilon)$ registers of clean memory. This is about the same amount of memory required by the classical state of the art (one pass) streaming algorithm for F_2-heavy hitters [5], just that we identify them exactly rather than approximately using a simple divide an conquer scheme.[9] Specifically we show the following.

▶ **Lemma 18.** *There is an algorithm that given a stream $\boldsymbol{\sigma} = x_1, x_2, \ldots, x_m$, returns the exact set of ϵF_2-heavy hitters of $\boldsymbol{\sigma}$ in $3 + 3\log \epsilon n$ passes using $O(1/\epsilon)$ registers of regular memory and $O(n)$ registers of catalytic memory, over a ring of size Z_q where $q = O(m^2)$.*

Proof. The formal description of the algorithm is given in Algorithm 4. In short we maintain a set $\mathcal{B}$ of at most $1/\epsilon$ buckets, each is an interval of consecutive elements from $[n]$, containing the heavy hitters. In each iteration we partition each bucket into two buckets. Then we estimate the contribution to F_2 of the elements in each bucket simultaneously by running a copy of the Algorithm of Corollary 7 separately on the elements of each bucket. There could be at most $1/\epsilon$ buckets whose contribution is more than ϵF_2, we continue with them to the next iteration and discard the rest.

We need three passes to compute F_2 initially and then three passes in each iteration to compute the F_2 contributions of the buckets. The number of iterations is at most $\log \epsilon n$, as the initial size of the buckets is ϵn. We keep in regular memory the boundaries of the buckets in $\mathcal{B}$ and the F_2 values of the elements in each of at most $2/\epsilon$ buckets. Thus we need $O(1/\epsilon)$ registers over Z_q for $q = O(m^2)$. The catalytic memory we need is the same as required by the algorithm in Corollary 7 since the number of elements occuring in all buckets together is at most n. ◀

Extension to F_k-heavy hitters. Notice we can use the same idea in order to compute ϵF_k-Heavy Hitters using Algorithm 1. In this case we would need $4 + 4\log \epsilon n$ passes and $O(kn)$ catalytic registers over the ring Z_q with $q = O(nm^k)$.

6 Concluding Remarks

In this paper, we introduced the *catalytic streaming model*, which equips algorithms with a large auxiliary memory that must be restored to its initial state by the end of the computation. The central question of this model is identifying which streaming problems inherently benefit from this restricted memory.

[9] Maybe a more common notion is ϵ ℓ_2-heavy hitter, defined to be any item i such that $f_i \geq \epsilon\sqrt{F_2}$. These notions are the same up to squaring ϵ.

Algorithm 4 F_2-Heavy Hitters.

Input: Stream $\boldsymbol{\sigma} \in [n]^m$, threshold $\epsilon \in (0, 1)$

$g \leftarrow F_2(\boldsymbol{\sigma})$; $k \leftarrow \lceil 2/\epsilon \rceil$ ▷ Compute global F_2

Partition $[n]$ into k disjoint buckets $\mathcal{B} = \{B_1, \ldots, B_k\}$. Bucket B_i contains the elements from $l_i = (i-1) \cdot (n/k) + 1$ to $r_i = i \cdot (n/k)$ and is represented by the pair of indices $\{l_i, r_i\}$.

$H \leftarrow \emptyset$

while $\mathcal{B} \neq \emptyset$ **do**

  $\mathcal{B}_{next} \leftarrow \emptyset$

  $b_1, \ldots, b_{|\mathcal{B}|} \leftarrow F_2^{\mathcal{B}}(\boldsymbol{\sigma})$ ▷ Compute the F_2 contribution of the elements in each bucket

  for all $i \in \{1, \ldots, |\mathcal{B}|\}$ **do**

    if $b_i \geq \epsilon \cdot g$ **then**

      if $l_i = r_i$ **then** $H \leftarrow H \cup \{l_i\}$

      else $mid \leftarrow \lfloor (l_i + r_i)/2 \rfloor$; $\mathcal{B}_{next} \leftarrow \mathcal{B}_{next} \cup \{[l_i, mid], [mid + 1, r_i]\}$

      end if

    end if

  end for

  $\mathcal{B} \leftarrow \mathcal{B}_{next}$

end while

return H

While recent results in catalytic communication [24] imply that catalytic memory offers no advantage for one-pass algorithms, we demonstrated that it is remarkably powerful given multiple passes. Specifically, we showed how to exactly evaluate multivariate polynomials of stream frequencies using $O(1)$ regular memory of logarithmically many bits. We also show that with two passes this task is impossible for a broad family of algorithms restricted to satisfy certain mild assumptions.[10] We believe the catalytic streaming model opens a rich landscape for future work. Several compelling open problems remain:

1. **Higher-degree pass optimality:** Our current lower bound technique is limited to two passes and puts restrictions on the algorithms that it applies to. Can one prove more general lower bounds that relate the number of required passes to the degree of the evaluated polynomial ?
2. **Exact Heavy Hitters:** Can heavy hitters be computed exactly using a constant number of passes in the catalytic model?
3. **Further catalytic separations:** Which other fundamental streaming problems admit catalytic algorithms that require substantially less regular memory than their standard streaming counterparts?

References

1 Yaroslav Alekseev, Yuval Filmus, Ian Mertz, Alexander Smal, and Antoine Vinciguerra. Catalytic Computing and Register Programs Beyond Log-Depth. In *50th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 6:1–6:18, 2025. doi:10.4230/LIPIcs.MFCS.2025.6.

[10] Following the submission of this paper we've discovered a two pass algorithm for F_2 that breaks the mild restrictions required by the lower bound of Section 4.2. We will include this algorithm in the full version of this paper (https://arxiv.org/abs/2607.09475).

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. doi:10.1006/JCSS.1997.1545.

3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545464.

4 Suman K. Bera and Amit Chakrabarti. Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams. In *34th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 11:1–11:14, 2017. doi:10.4230/LIPIcs.STACS.2017.11.

5 Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P. Woodruff. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 361–376, 2017. doi:10.1145/3034786.3056117.

6 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *40th International Conference on Automata, Languages, and Programming (ICALP)*, pages 244–254, 2013. doi:10.1007/978-3-642-39206-1_21.

7 Harry Buhrman, Richard Cleve, Michal Kouckỳ, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *46th annual ACM symposium on Theory of computing (STOC)*, pages 857–866, 2014.

8 Harry Buhrman, Marten Folkertsma, Ian Mertz, Florian Speelman, Sergii Strelchuk, Sathyawageeswar Subramanian, and Quinten Tupker. Quantum catalytic space. In *20th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, pages 11:1–11:24, 2025. doi:10.4230/LIPIcs.TQC.2025.11.

9 Harry Buhrman, Michal Kouckỳ, Bruno Loff, and Florian Speelman. Catalytic space: Nondeterminism and hierarchy. *Theory of Computing Systems*, 62(1):116–135, 2018. doi:10.1007/S00224-017-9784-7.

10 Amit Chakrabarti. Data stream algorithms: Lecture notes. Dartmouth College Computer Science, 2020. URL: https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf.

11 Petr Chmel, Aditi Dudeja, Michal Koucký, Ian Mertz, and Ninad Rajgopal. Frontier space-time algorithms using only full memory. *arXiv preprint*, 2026. arXiv:2602.21089, doi:10.48550/arXiv.2602.21089.

12 James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. In *57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 554–564, 2025. doi:10.1145/3717823.3718112.

13 James Cook and Ian Mertz. Trading time and space in catalytic branching programs. In *37th Computational Complexity Conference (CCC)*, pages 8:1–8:21, 2022. doi:10.4230/LIPIcs.CCC.2022.8.

14 James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \log \log n)$. In *56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1268–1278, 2024. doi:10.1145/3618260.3649664.

15 Vincent Girard, Michal Kouckỳ, and Pierre McKenzie. Nonuniform catalytic space and the direct sum for space. Technical Report TR15-138, Electronic Colloquium on Computational Complexity (ECCC), 2015.

16 Oded Goldreich. Solving tree evaluation in o(log n · log log n) space. Technical Report TR24-124, Electronic Colloquium on Computational Complexity (ECCC), 2024.

17 Alexandra Henzinger, Edward Pyne, and Seyoon Ragavan. Catalytic tree evaluation from matching vectors. *arXiv preprint*, 2026. arXiv:2602.14320.

18 Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. doi:10.1137/0405044.

19 Joseph M. Landsberg. *Tensors: Geometry and Applications*, volume 128 of *Graduate Studies in Mathematics*. American Mathematical Society, 2012.

20 Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

21 J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

22 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005. doi:10.1561/0400000002.

23 Aaron Potechin. A note on amortized branching program complexity. In *32nd Computational Complexity Conference (CCC)*, pages 4:1–4:12, 2017. doi:10.4230/LIPIcs.CCC.2017.4.

24 Edward Pyne, Nathan S. Sheffield, and William Wang. Catalytic communication. In *16th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 79:1–79:24, 2025. doi:10.4230/LIPIcs.ITCS.2025.79.

25 A.A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992. doi:10.1016/0304-3975(92)90260-M.

A Powering Lemma

We give here the powering algorithm of Lemma 4. We denote $c_i \coloneqq (-1)^i\binom{k}{i}$.

Algorithm 5 Powering Algorithm (Lemma 10) [Modified].

Initial State: $r = \tau,\ r_o = 0,\ \forall i \in [k] : r_i = \tau_i$

1: I_1: **for** $i = 1$ **to** k **do** $r_o \leftarrow r_o + c_i \cdot r_i \cdot r^i$ ▷ $r_o = \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i \cdot \tau^i$

2: Execute $r \leftarrow r + f$ ▷ $r = \tau + f$

3: I_2: **for** $i = 1$ **to** k **do** $r_i \leftarrow r_i - r^{k-i}$ ▷ $\forall i \in [k],\ r_i = \tau_i - (\tau + f)^{k-i}$

4: I_2: $r_o \leftarrow r_o + r^k$ ▷ $r_o = (\tau + f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i\tau^i$

5: Execute $r \leftarrow r - f$ ▷ $r = \tau$

6: I_3: **for** $i = 1$ **to** k **do** $r_o \leftarrow r_o - c_i \cdot r_i \cdot r^i$

▷ $r_o = (\tau + f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i\tau^i - \sum_{i=1}^{k}(-1)^i\binom{k}{i}(\tau_i - (\tau + f)^{k-i})\tau^i$

▷ $= (\tau + f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau^i(\tau + f)^{k-i}$

After the execution of line 1, the output register satisfies

$$r_o = \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i\tau^i.$$

After the execution of line 3, we have

$$\forall i \in [k], \qquad r_i = \tau_i - (\tau + f)^{k-i}.$$

After the execution of line 4, the output register satisfies

$$r_o = (\tau + f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i\tau^i.$$

Hence, after the execution of line 6, we obtain

$$\begin{aligned} r_o &= (\tau+f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau_i\tau^i - \sum_{i=1}^{k}(-1)^i\binom{k}{i}\left(\tau_i-(\tau+f)^{k-i}\right)\tau^i \\ &= (\tau+f)^k + \sum_{i=1}^{k}(-1)^i\binom{k}{i}\tau^i(\tau+f)^{k-i} \\ &= f^k, \end{aligned}$$

where the last equality follows from the binomial theorem.

Therefore, at the end of the computation, the registers contain

$$r=\tau, \qquad r_o = f^k, \quad \text{and for every } i\in[k], \quad r_i = \tau_i - (\tau+f)^{k-i}.$$

B Set-Disjointness

▶ **Proposition 19.** *Suppose there exists a catalytic streaming algorithm A that computes F_2 in p passes using s bits of regular memory. Then there exists a catalytic communication protocol for $Disj_n$ with $2p$ rounds that uses $s+\log(n)+c_0$ bits of regular memory, where c_0 is some absolute constant.*

Proof. Let $\boldsymbol{x}$ be Alice's input and $\boldsymbol{y}$ be Bob's input. To compute $\mathrm{Disj}_n(\boldsymbol{x},\boldsymbol{y})$, Alice and Bob simulate the streaming algorithm A on an appropriate stream, while using the shared memory exactly as A would.

First, Alice inserts into the stream every item $i\in[n]$ such that $x_i=1$. She then passes the inner state of A plus it's memory configuration to Bob.

Bob then inserts in a similar way $\boldsymbol{y}$ into the stream. Moreover, he calculates $\|\boldsymbol{y}\|_0$[11] and passes both A and $\|\boldsymbol{y}\|_0$ to Alice.

They continue in this way, alternately simulating the execution of A and passing it's inner state together with it's current configuration, until the execution of A is complete.

Let a denote the output of A on the simulated stream. Alice accepts if and only if

$$a = \|\boldsymbol{x}\|_0 + \|\boldsymbol{y}\|_0.$$

Note that $\|\boldsymbol{x}\|_0+\|\boldsymbol{y}\|_0$ is exactly the length of the simulated stream.

▷ Claim 20. Alice accepts if and only if $\mathrm{Disj}_n(\boldsymbol{x},\boldsymbol{y})=1$.

Proof. Suppose first that $\mathrm{Disj}_n(\boldsymbol{x},\boldsymbol{y})=1$. Then the supports of $\boldsymbol{x}$ and $\boldsymbol{y}$ are disjoint, so each item $i\in[n]$ appears in the stream at most once. Hence, for every $i\in[n]$, we have $f_i\in\{0,1\}$, and therefore $f_i^2=f_i$. It follows that

$$F_2 = \sum_{i=1}^{n} f_i^2 = \sum_{i=1}^{n} f_i = \|\boldsymbol{x}\|_0+\|\boldsymbol{y}\|_0.$$

Since A computes F_2, we have $a=\|\boldsymbol{x}\|_0+\|\boldsymbol{y}\|_0$, and so Alice accepts.

[11] The L_0 norm of $\boldsymbol{y}$, i.e. the number of 1 bits in $\boldsymbol{y}$

Conversely, suppose that $\mathrm{Disj}_n(\boldsymbol{x}, \boldsymbol{y}) = 0$. Then there exists some $i \in [n]$ such that $x_i = y_i = 1$, and hence $f_i = 2$. More generally, there is at least one index i for which $f_i > 1$. Since for every $t \in \mathbb{Z}_{\geq 0}$, $t^2 \geq t$,with strict inequality whenever $t > 1$, we obtain

$$F_2 = \sum_{i=1}^{n} f_i^2 > \sum_{i=1}^{n} f_i = \|\boldsymbol{x}\|_0 + \|\boldsymbol{y}\|_0.$$

Thus $a > \|\boldsymbol{x}\|_0 + \|\boldsymbol{y}\|_0$, and Alice rejects. ◁

The claim about the number of rounds is trivial. Each time the simulation is passed from one player to the other, the players need to communicate the current configuration of A, which contributes $s + c_0$ bits of regular memory, s bits for the memory size of A and c_0 for description of it's inner state. In addition, Bob communicates the value $\|\boldsymbol{y}\|_0$ to Alice, which requires at most $\log n$ bits of regular memory. ◀

▶ Remark 21. Observe that the last reduction works for algorithm A which computes any moment other than F_1, not just F_2.

C A Three-Round Catalytic Communication Protocol for Set-Disjointness

In [24, Proposition 5], it is shown that the inner product of two vectors over GF_2 can be computed by a three-round catalytic communication protocol using only 1 bit of regular memory and n bits of catalytic memory. We observe that essentially the same construction, with a minor modification, yields a protocol for Disj_n. For completeness, we include the proof here, following their argument with the necessary changes.

▶ **Proposition 22.** *There exists a catalytic communication protocol for Disj_n that uses $\log n + 1$ bits of regular memory, $n(\log n + 1)$ bits of catalytic memory, and requires 3 rounds.*

Proof. Let $\boldsymbol{x} \in \{0,1\}^n$ be Alice's input and let $\boldsymbol{y} \in \{0,1\}^n$ be Bob's input.

Partition the catalytic tape into n registers

$$r_1, r_2, \ldots, r_n,$$

each of size $\log(n) + 1$ bits. For each $i \in [n]$, let τ_i denote the initial contents of register r_i. The regular memory consists of $\log(n) + 1$ bits, so throughout the protocol all arithmetic operations are performed modulo $2n$. Alice and Bob exchange the clean abd catalytic memory in three rounds as follows.

1. **Alice** increments r_i for every $i \in [n]$ such that $x_i = 1$.
2. **Bob** adds r_i to regular memory for every $i \in [n]$ such that $y_i = 1$.
   Following this round, the regular memory contains

$$\sum_{i:\, y_i = 1} r_i \pmod{2n} = \sum_{i:\, y_i = 1} \big(\tau_i + \mathbf{1}[x_i = 1]\big) \pmod{2n}, \tag{4}$$

   where $\mathbf{1}[x_i = 1]$ is the indicator of the event $x_i = 1$.
3. **Alice** decrements r_i for every $i \in [n]$ such that $x_i = 1$.
   Following this step, every catalytic register is restored to its initial value. .

Finally, once Bob gets the second message from Alice, he Bob subtracts r_i from the value in the regular memory for every i such that $y_i = 1$. After doing so, using Equation (4), the value Bob obtains is

$$\sum_{i\,:\,y_i=1} \mathbf{1}[x_i = 1] \pmod{2n}. \tag{5}$$

Bob outputs 1 if this value is 0, and outputs 0 otherwise.

The space bounds are immediate from the construction, the catalytic memory contains n registers of $\log n + 1$ bits each, for a total of $n(\log n + 1)$ bits, while the regular memory uses exactly $\log n + 1$ bits.

To prove correctness, observe that

$$\sum_{i\,:\,y_i=1} \mathbf{1}[x_i = 1] = |\{i \in [n] : x_i = y_i = 1\}|,$$

namely, the number of coordinates in which both $\boldsymbol{x}$ and $\boldsymbol{y}$ contain a 1. Therefore,

$$\mathrm{Disj}_n(\boldsymbol{x}, \boldsymbol{y}) = 1 \iff \sum_{i\,:\,y_i=1} \mathbf{1}[x_i = 1] = 0.$$

Moreover, this sum is always at most n, and hence is strictly smaller than $2n$. Thus its value modulo $2n$ is zero if and only if the sum itself is zero. ◀

D Why the Discrete Derivative Argument Breaks in Three Passes

The impossibility result strictly relies on the two-pass constraint (one to hold some information about the input and one to clean the catalytic tape). We claim that if the algorithm is permitted a third pass, the information-theoretic contradiction vanishes. In a three-pass model, we have three bijection matrices $U_{i,1}$, $U_{i,2}$, $U_{i,3}$, one for each pass such that for every f_i, and every τ_i, applying $(U_{i,3})^{f_i}\Big((U_{i,2})^{f_i}\big((U_{i,1})^{f_i}(\tau_i)\big)\Big) = \tau_i$. If we apply the same proof technique then the derivative we get in this case will depend on $y_1 = (U_{i,1})^{f_i}(\tau_1)$ and $y_2 = (U_{i,2})^{f_i}\big((U_{i,1})^{f_i}(\tau_i)\big)$. So, we will have some deterministic function $H_i(\cdot,\cdot)$ that satisfies $H_i(y_1, y_2) = 2f_i + 1$. Crucially, while y_1 and y_2 are individually uniformly distributed and independent of f_i, they jointly can determine f_i. In particular they indeed determine f_i if there is a unique k for which $y_2 = (U_{i,2})^k(y_1)$.