Sparse Nonnegative Convolution Is Equivalent to Dense Nonnegative Convolution[*]

Karl Bringmann
Saarland University and Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

Nick Fischer
Saarland University and Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

Vasileios Nakos
Saarland University and Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

ABSTRACT

Computing the convolution $A \star B$ of two length-n vectors A, B is an ubiquitous computational primitive, with applications in a variety of disciplines. Within theoretical computer science, applications range from string problems to Knapsack-type problems, and from 3SUM to All-Pairs Shortest Paths. These applications often come in the form of *nonnegative* convolution, where the entries of A, B are nonnegative integers. The classical algorithm to compute $A \star B$ uses the Fast Fourier Transform (FFT) and runs in time $O(n \log n)$.

However, in many cases A and B might satisfy sparsity conditions, and hence one could hope for significant gains compared to the standard FFT algorithm. The ideal goal would be an $O(k \log k)$-time algorithm, where k is the number of non-zero elements in the output, i.e., the size of the support of $A \star B$. This problem is referred to as *sparse* nonnegative convolution, and has received a considerable amount of attention in the literature; the fastest algorithms to date run in time $O(k \log^2 n)$.

The main result of this paper is the first $O(k \log k)$-time algorithm for sparse nonnegative convolution. Our algorithm is randomized and assumes that the length n and the largest entry of A and B are subexponential in k. Surprisingly, we can phrase our algorithm as a reduction from the sparse case to the dense case of nonnegative convolution, showing that, under some mild assumptions, sparse nonnegative convolution is equivalent to dense nonnegative convolution for constant-error randomized algorithms. Specifically, if $D(n)$ is the time to convolve two nonnegative length-n vectors with success probability $2/3$, and $S(k)$ is the time to convolve two nonnegative vectors with output size k with success probability $2/3$, then $S(k) = O(D(k) + k(\log \log k)^2)$.

Our approach uses a variety of new techniques in combination with some old machinery from linear sketching and structured linear algebra, as well as new insights on linear hashing, the most classical hash function.

[*]This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

STOC '21, June 21–25, 2021, Virtual, Italy
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8053-9/21/06.
https://doi.org/10.1145/3406325.3451090

CCS CONCEPTS

• **Theory of computation → Streaming, sublinear and near linear time algorithms**.

KEYWORDS

Convolution, Sparsity, FFT, Linear Hashing

ACM Reference Format:
Karl Bringmann, Nick Fischer, and Vasileios Nakos. 2021. Sparse Nonnegative Convolution Is Equivalent to Dense Nonnegative Convolution. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451090

1 INTRODUCTION

Computing convolutions is an ubiquitous task across all science and engineering. Some of its special cases are as fundamental as the general case; we first introduce the most important problem variants.

- **Boolean Convolution** is the problem of computing for given vectors $A, B \in \{0,1\}^n$ the vector $C = A \circledast B \in \{0,1\}^{2n-1}$ defined by $C_k = \bigvee_i A_i \wedge B_{k-i}$. This formalizes a situation in which we split a computational problem into two subproblems, so that in total there is a solution of size k if and only if for some i there is a solution of the left subproblem of size i and there is a solution of the right subproblem of size $k - i$. Therefore, it is a natural task that frequently arises in algorithm design. Boolean convolution is also equivalent to *sumset computation*, where for given sets $A, B \subseteq \{0, 1, \dots, n-1\}$ the task is to compute their sumset $A + B$ consisting of all sums $a+b$ with $a \in A,\ b \in B$. It therefore frequently comes up in algorithms for Subset Sum, 3SUM, and similar problems, see, e.g., [8, 11, 15, 32, 39, 45].
- **Nonnegative Convolution** is the problem of computing for given vectors $A, B \in \mathbf{Z}^n$ with nonnegative entries the vector $C = A \star B \in \mathbf{Z}^{2n-1}$ defined by $C_k = \sum_i A_i \cdot B_{k-i}$. For instance, if A_i and B_i count the number of size-i solutions of the left and right subproblem, then C_k counts the number of size-k solutions of the whole problem. It also comes up in string algorithms when computing the Hamming distance of a pattern and each sliding window of a text; this connection was found by Fischer and Paterson [18] and has been exploited in many string algorithms, see, e.g., [2, 6, 37, 40]. As an operation, nonnegative convolution is frequent also in computer vision, image processing and computer graphics; a prototypical such an example is the process of blurring an

image by a Gaussian kernel in order to remove noise and detail [1]. Note that nonnegative convolution generalizes Boolean convolution, as $A \circledast B$ is simply the support of the nonnegative convolution $A \star B$. In this paper our focus is on the nonnegative convolution problem.

- **General Convolution**, or simply "convolution", denotes the general case obtained by dropping the nonnegativity assumption from the previous problem variant. This problem is central in signal processing and is also equivalent to polynomial multiplication, one of the most fundamental problems of computer algebra, and thus has a wealth of applications. We remark that general convolution can be reduced to nonnegative convolution (and thus they are equivalent), by replacing $A'_i := A_i + M$ and $B'_j := B_j + M$, which are nonnegative for sufficiently large M, and noting that $(A \star B)_k = (A' \star B')_k \bmod M$. However, this reduction destroys the *sparsity* of the input and output, and thus is not applicable in the context of this paper.

Algorithms in the Dense Case. The standard algorithm for these problems uses the Fast Fourier Transform (FFT) and runs in time $O(n \log n)$ on the RAM model. This running time is conjectured to be optimal (at least for general convolution), but proving this is a big open problem. There is some evidence in favor of this conjecture, for instance nonnegative convolution can be used to multiply integers and the latter is connected to the network coding conjecture [3]. For Boolean convolution, the evidence is less clear, since there exists a Boolean convolution algorithm by Indyk [28] running in time $O(n)$ with the guarantee that any fixed output entry is correct with constant probability (see Theorem 6.2). However, in this paper we focus on algorithms where the whole output vector is correct with constant probability, and boosting Indyk's algorithm to such a guarantee would again result in running time $O(n \log n)$. Therefore, for all three problem variants it is plausible that time $O(n \log n)$ is optimal, even for constant-error randomized algorithms.

Algorithms in the Sparse Case. A long line of work has considered convolution in a sparse setting, see, e.g., [7, 15, 16, 23, 42, 46, 47, 53, 55, 57]. Here the running time is expressed not only in terms of n, but also in terms of the output size k, defined as the number of nonzero entries of $A \star B$. All variants of convolution listed above admit randomized algorithms running in near-linear time $k \operatorname{polylog} n$. This was first achieved by Cole and Hariharan [16] for nonnegative convolution with a Las Vegas algorithm running in time $O(k \log^2 n + \operatorname{polylog}(n))$, in [47] for general convolution with running time[1] $\widetilde{O}(k \log^2 n + \operatorname{polylog}(n))$. The latter was improved by Giorgi, Grenet and Perret du Cray [23] to a *bit complexity* of $\widetilde{O}(k \log n)$; it seems that on the RAM model their algorithm would run in time $O(k \log^5 k \operatorname{polyloglog} n)$.[2] Implementations of sparse convolution algorithms exist in Maple [43, 44] and Magma [56].

[1]Here we use the notation $\widetilde{O}(T) = \bigcup_{c>0} O(T \log^c T)$.

[2]To determine their running time on the RAM model, from the last paragraph of the proof of their Lemma 4.7 one can infer that the bottleneck of their running time stems from $\Theta(\log^2 k)$ many dense convolutions on vectors of length $\Theta(k \log(k \log n) \log(k \log \log n)) = \Theta(k \log^2 k \operatorname{polyloglog} n)$. Since one dense convolution of length d can be performed in time $O(d \log d)$ on the RAM model, they require time $O(k \log^5 k \operatorname{polyloglog} n)$.

This research is closely related to the extensively studied sparse Fourier transform problem, e.g. [19, 21, 25, 29, 30]. Indeed, the same running time of $O(k \log^2 n)$, albeit with a more complicated algorithm and under the assumption that complex exponentials can be evaluated at constant time, can be obtained by combining the state-of-the-art sparse Fourier transform with the semi-equispaced Fourier transform, see Section 3.1.

In summary, for nonnegative convolution on the RAM model, the state of the art requires time $\Omega(k \log^2 n)$ or $\Omega(k \log^5 k)$. In view of the conjecture that $O(n \log n)$ is optimal for the dense case, the best running time we could expect for the sparse case would be $O(k \log k)$. The driving question of this work is thus:

Can sparse nonnegative convolution be solved in time $O(k \log k)$?

We note that the need for sparse convolution arises in many different areas of algorithm design, for example algorithms for the sparse cases of Boolean and nonnegative convolution have been used for clustered 3SUM and similar problems [15], output-sensitive Subset Sum algorithms [12], pattern matching on point sets [13], sparse wildcard matching [16], and other string problems [4, 5].

1.1 Results

We present a novel connection between the sparse and dense case of nonnegative convolution, which can be viewed as work at the intersection of sparse recovery and fine-grained complexity.

We work on the Word RAM model where each cell stores a word consisting of w bits, and standard operations on w-bit integers can be performed in constant time; this includes addition, multiplication, and division with remainder. We always assume that the length n of the input vectors as well as each input entry fit into a word, or more precisely into a constant number of words. For nonnegative convolution, this means that the input consists of vectors $A, B \in \{0, 1, \dots, \Delta\}^n$ with $n, \Delta \le 2^{O(w)}$. In this machine model, the standard algorithm for dense convolution uses FFT and runs in time $D(n) = O(n \log n)$. In the following, we denote by $D_\delta(n)$ the running time of a randomized algorithm for dense nonnegative convolution with failure probability δ (for any $0 \le \delta \le 1/3$). Note that this notation hides the dependence on Δ.

In the sparse setting, we denote the output size by k, i.e., k is the number of nonzero entries of the convolution $A \star B$. Also in this setting we will always assume that the length n of the input vectors as well as the largest input entry Δ fit into a constant number of words. We will denote by $S_\delta(k)$ the running time of a randomized algorithm for sparse nonnegative convolution with failure probability δ; this hides the dependence on n and Δ.

The main result of this paper is a novel Monte Carlo algorithm for nonnegative sparse convolution.

Theorem 1.1. *The sparse nonnegative convolution problem has a randomized algorithm with running time $O(k \log k + \operatorname{polylog}(n\Delta))$ and failure probability $2^{-\sqrt{\log k}}$.*

Here and throughout the paper we implicitly mean Monte Carlo randomization. Naturally, the same algorithm also can be used for Boolean convolution, where $\Delta = 1$. For Boolean convolution, this is the first algorithm that improves upon the dense case's running time of $O(n \log n)$ for all $k = o(n)$; previous algorithms required

$k = o(n/\log n)$. For nonnegative convolution, the same statement is true assuming that $\Delta \le 2^{n^{o(1)}}$. Moreover, this answers our driving question for $k \gg \operatorname{polylog}(n\Delta)$, by a randomized algorithm.

In fact, our algorithm can be phrased as a reduction from the sparse case to the dense case of nonnegative convolution:

Theorem 1.2. *Any randomized algorithm for dense nonnegative convolution running in time $D_{1/3}(n)$ can be turned into a randomized algorithm for sparse nonnegative convolution running in time*[3]

$$S_\delta(k) = O\big(D_{1/3}(k) + k\log^2(\log(k)/\delta) + \operatorname{polylog}(n\Delta)\big).$$

Here we assume for technical reasons that the function $D_\delta(n)/n$ is nondecreasing, as is to be expected from any natural running time.

Since $D_{1/3}(k) = O(k\log k)$, setting $\delta = 2^{-\sqrt{\log k}}$ yields time $O(k\log k + \operatorname{polylog}(n\Delta))$, which proves Theorem 1.1. Furthermore, any future algorithmic improvement for the dense case automatically yields an improved algorithm for the sparse case by our reduction. In fact, under the mild conditions that $k \gg \operatorname{polylog}(n\Delta)$ and that the optimal running time $D_{1/3}(k)$ is $\Omega(k(\log\log k)^2)$, we obtain an *asymptotic equivalence* with respect to constant-error randomized algorithms:

- $S_{1/3}(k) = O(D_{1/3}(k))$ holds by Theorem 1.2 and the mild conditions,
- $D_{1/3}(n) = O(S_{1/3}(n))$ holds since the sparse case trivially is a special case of the dense one.

1.2 Discussion and Open Problems

Our work raises a plethora of open problems that we discuss in the following.

Improving our Reduction. We can ask for improvements of our reduction, specifically of the parameters of Theorem 1.2:

(1) Can the error probability of the reduction be reduced? Specifically, can Theorem 1.1 be improved from $\delta = 2^{-\sqrt{\log k}}$ to $1/\operatorname{poly}(k)$ or even $1/\operatorname{poly}(n)$?
(2) Can the $\operatorname{polylog}(n\Delta)$ term in Theorem 1.2 be removed, to make it work also for very small k? This would require a quite different approach than the one we take here, since already for finding a prime field large enough to store n and Δ, or for computing a single multiplicative inverse in such a prime field, the fastest algorithms that we are aware of run in time $O(\operatorname{polylog}(n\Delta))$, even for the Word RAM model.
(3) Can we obtain further improvements by bit packing, say for Boolean convolution?

General Convolution. Here we focused on nonnegative convolution, what about the general case?

(4) Does sparse general (not necessarily nonnegative) convolution have a randomized algorithm running in time $O(k\log k + \operatorname{polylog}(n\Delta))$?
(5) Are sparse general convolution and dense general convolution asymptotically equivalent?

Deterministic Algorithms. Chan and Lewenstein [15] presented a deterministic $k \cdot n^{o(1)}$-time algorithm for sparse nonnegative convolution, assuming that they are additionally given a small superset of the output.

(6) Is there a deterministic algorithm for sparse nonnegative convolution with running time $k\operatorname{polylog}(n)$?
(7) Are sparse and dense nonnegative convolution asymptotically equivalent with respect to deterministic algorithms?

Sparse Fourier Transform. Computing convolutions is intimately connected to the Fast Fourier Transform (FFT). In fact, in the dense case these two problems are known to be equivalent: if one of these problems can be solved in time $T(n)$ then the other can be solved in time $O(T(n))$. One direction of this equivalence follows from the standard algorithm for convolution that uses FFT, the other direction follows from an old trick invented by Bluestein [10], see also [24, pp. 213–215], showing how to express the discrete Fourier transform as a convolution.[4]

The sparse case of Fourier transform, where one has oracle access to x and wants to compute $\widehat{x}$ under a k-sparsity assumption, is also extensively studied [19, 21, 25, 26, 29, 30, 34–36, 48, 51]. We can ask whether the results presented in this paper also work for computing Fourier transforms:

(8) Are sparse Fourier transform and dense Fourier transform asymptotically equivalent? The algorithm in [25] runs time $O(k\log(n\Delta))$, but the running time is not dominated by the calls to FFT.

Note that positive answers to Questions 5 and 8 would, together with the known equivalence of dense convolution and dense Fourier transform, show an asymptotic equivalence of sparse general convolution and sparse Fourier transform. Currently, reductions among these problems that lose one log-factor are known, see also Section 3.1.

Note that since dense nonnegative convolution is equivalent to general dense convolution (as mentioned in the introduction), and since the latter is equivalent also to (dense) DFT computation, our work places sparse nonnegative convolution to the aforementioned equivalent class, under the assumptions made.

We hope that our work ignites further work on revealing connections between all these fundamental problems.

1.3 Organization

This paper is organized as follows. Section 2 starts with some preliminary definitions. In Section 3 we sketch our algorithm and describe some technical difficulties and highlights. The detailed reduction is split across several sections starting with Section 4 which gathers some algorithmic tools, followed by the two most interesting steps of the reduction in Sections 5 and 6. Due to space constraints we omitted some reduction steps in this version of the paper.

2 PRELIMINARIES

Let $\mathbf{Z}$, $\mathbf{N}$, $\mathbf{Q}$ and $\mathbf{C}$ to denote the integers, nonnegative integers, rationals, and complex numbers, respectively. For any nonnegative integer n, let $\mathbf{Z}_n$ denote the ring of integers modulo n. We set

[3]To be precise, we should take the dependence on Δ (and n) into account. Expressing the running time for the dense case as $D_\delta(n,\Delta)$ and for the sparse case as $S_\delta(k,n,\Delta)$, our reduction actually shows that $S_\delta(k,n,\Delta) = O(D_{1/3}(k,\operatorname{poly}(n\Delta)) + k\log^2(\log(k)/\delta) + \operatorname{polylog}(n\Delta))$.

[4]As a technical detail, this reduction assumes that terms of the form $\exp(2\pi i x)$ can be evaluated in constant time.

$[n] = \{0, 1, \ldots, n-1\}$. The Iverson bracket $[P] \in \{0, 1\}$ denotes the truth value of a proposition P. We write log for the base-2 logarithm, $\mathrm{poly}(n) = n^{O(1)}$ and $\mathrm{polylog}(n) = \log^{O(1)} n$.

We mostly denote vectors by letters A, B, C with A_i referring to the i-th coordinate in A. The *convolution* of two length-n vectors A and B is the vector $A \star B$ of length $2n-1$ with

$$(A \star B)_i = \sum_{0 \le j \le i} A_j B_{i-j}.$$

The *cyclic convolution* of two length-n vectors A, B is the length-n vector $A \star_n B$ with

$$(A \star_n B)_i = \sum_{0 \le j \le n-1} A_j B_{(i-j) \bmod n}.$$

We let $\mathrm{supp}(A) = \{i \in [n] : A_i \neq 0\}$, $\|A\|_0 = |\mathrm{supp}(A)|$ and $\|A\|_\infty = \max_i |A_i|$. Furthermore, we often write $A \bmod m$ for the vector with

$$(A \bmod m)_{i'} = \sum_{i = i' \pmod m} A_i,$$

and more generally, for a function $\mathbf{Z} \to [m]$, we write $f(A)$ for the length-m vector with

$$f(A)_{i'} = \sum_{i : f(i) = i'} A_i.$$

For sets X, Y, we define the *sumset* $X + Y = \{x + y : (x, y) \in X \times Y\}$ and some other shorthand notation: $a + X = \{a + x : x \in X\}$, $aX = \{ax : x \in X\}$, $X \operatorname{div} a = \{\lfloor \frac{x}{a} \rfloor : x \in X\}$ and $X \bmod a = \{x \bmod a : x \in X\}$. More generally, for a function defined on X we set $f(X) = \{f(x) : x \in X\}$.

3 TECHNICAL OVERVIEW

3.1 Previous Techniques

Possibly the earliest work on sparse convolution is a quite complicated $O(k \log^2 n + \mathrm{polylog}(n))$-time[5] algorithm due to Cole in Hariharan [16] for the nonnegative case. Their approach builds on linear hashing and string algorithms in order to identify $\mathrm{supp}(A \star B)$, and involves many ideas such as encoding characters with complex entries before applying convolution. The more recent approaches [7, 23, 47, 53, 54, 57, 58] (the last two of which can also solve the general convolution problem) heavily build on hashing modulo a random prime number. This approach suffers from the loss of one log factor due to the density of the primes given by the Prime Number Theorem. Therefore, these approaches seem hopeless of getting time $O(k \log k)$, or even time $o(k \log^2 k)$.

On the other hand, a quite different $O(k \log^2(n\Delta))$ algorithm, not explicitly written down as far as we know, is attainable using techniques from the sparse Fourier transform (assuming that complex exponentials can be evaluated in constant time). It has been established in the celebrated work of Hassanieh, Indyk, Katabi and Price [25] that one can recover a k-sparse vector $x \in \mathbf{C}^n$ in time $O(k \log(n\Delta))$ by only accessing a small subset of its Fourier transform $\widehat{x}$. This alone might not seem sufficient, but spelling out the details of [25] reveals that the pattern of accesses to $\widehat{x}$ is a random arithmetic progression of length $O(k \log(n\Delta))$. In light of this, one can additionally leverage known techniques from semi-equispaced Fourier transforms [17], [30, Section 12] to obtain a $O(k \log^2(n\Delta))$-time algorithm. The semi-equispaced Fourier transform is a well-studied subfield of computational Fourier transforms, and results from that area show that s equally spaced Fourier coefficients of a length-n and s-sparse vector can be computed in time $O(s \log(n\Delta))$ [30, Section 12]. Combining this with the algorithm of [25] yields an $O(k \log^2(n\Delta))$-time algorithm for sparse convolution. The inherent reason for this logarithmic blow-up is that going back and forth in Fourier and time domain is more costly in the sparse case than in the dense case. Furthermore, the above algorithm cannot yield a reduction between the sparse and dense convolution (more generally, the approach of [25] cannot yield such an equivalence, as their running time is not dominated by calls to FFT). It is a very interesting open question in that area to show any equivalence between some variant of sparse and dense Fourier transform, as well as to achieve $O(k \log k)$ running time.

There are other techniques for sparse convolution using polynomial interpolation, see [55], but they do not seem sufficient in going beyond a $O(k \,\mathrm{polylog}\, n)$-time algorithm in any variation of the problem, owing to the usage of a variety of tools from structured linear algebra which come with additional polylog factors.

3.2 Our Approach

The goal is to solve the following problem in output-sensitive time:

Problem (SparseConv).
Input: *Nonnegative vectors A, B and a parameter $\delta > 0$.*
Task: *Compute $A \star B$ with success probability $1 - \delta$.*

In what follows, we assume that we are given a number k such that $\|A \star B\|_0 \le k$, and we want to recover $A \star B$ in time $O(k \log k)$. This assumption will can removed using standard techniques. For the sake of simplicity, we will focus on how to obtain a constant-error randomized algorithm for sparse convolution from a deterministic algorithm for dense convolution.

The Obstructions Created by Known Recovery Techniques. So far, hashing-based approaches on computing sparse convolutions build on either of two well-known hash functions mapping $[n] \to [m]$:

- $g(x) = x \bmod p$, where p is a random prime of appropriate size.
- *Linear hashing*: $h(x) = ((\sigma x + \tau) \bmod p) \bmod m$, where p is a sufficiently large fixed prime number and σ, τ are random.[6]

The first hash function satisfies $g(x + y) = (g(x) + g(y)) \bmod m$, in particular it is *affine*, in the sense that $g(x + y) + g(0) \equiv g(x) + g(y) \pmod m$. In comparison, the second hash function is only *almost-affine*, in the sense that $h(x + y) + h(0) - h(x) - h(y)$ can only take a constant number of different values. Although almost-affinity is an amenable issue in many situations, e.g. [14, 49], in our case it appears to be a more serious obstruction for reasons outlined later.

In turn, the first hash function is only $O(\log n)$-universal. Thus, if we want to hash a size-k set X using g, such that a fixed $x \in X$ is isolated from every other $x' \in X$, we must pick $p = \Omega(k \log n)$. This results in a multiplicative $O(\log n)$ overhead on top of the

[5] The claimed running time in their paper is $O(k \log^2 n)$, however they need to pick a prime $p \in [n, 2n]$, which requires time $\mathrm{polylog}(n)$ (this additive overhead disappears if the algorithm is allowed to hardcode p).

[6] One can also use $h(x) = \lfloor ((\sigma x + \tau) \bmod p) m / p \rfloor$ for a sufficiently large prime p, which enjoys similar properties [38].

number of buckets. In comparison, linear hashing is $O(1)$-universal, so setting $m = O(k)$ suffices for proper isolation.

Before delving deeper, let us sketch how to design an $O(k \log k)$-time algorithm, assuming that we had an "ideal" hash function $\iota : [n] \to [m]$ that is $O(1)$-universal and affine, i.e., combines the best of $g(x)$ and $h(x)$. Then the hashed convolution could be easily computed as $\iota(A \star B) = \iota(A) \star_m \iota(B)$. The next ingredient is the derivative operator from [27]. Defining the vector ∂A with $(\partial A)_i = i \cdot A_i$, and similarly ∂B with $(\partial B)_i = i \cdot B_i$, we have that $\partial(A \star B) = (\partial A) \star B + A \star (\partial B)$, which when combined with the ideal hash function ι gives $\iota(\partial(A \star B)) = \iota(\partial A) \star_m \iota(B) + \iota(A) \star_m \iota(\partial B)$. The b-th coordinate of this vector is

$$\iota(\partial(A \star B))_b = \sum_{i:\iota(i)=b} i \cdot (A \star B)_i,$$

which can be accessed by computing the length-m convolutions $\iota(\partial A) \star_m \iota(B)$ and $\iota(A) \star_m \iota(\partial B)$ and adding them together. By setting $m = O(k)$, we can now infer a constant fraction of elements $i \in \operatorname{supp}(A \star B)$ by performing the division

$$\frac{\iota(\partial(A \star B))_b}{\iota((A \star B))_b} = \frac{\sum_{i:\iota(i)=b} i \cdot (A \star B)_i}{\sum_{i:\iota(i)=b} (A \star B)_i}$$

for all $b \in [m]$. This yields the locations of all isolated elements in $\operatorname{supp}(A \star B)$ under ι. In particular, we obtain a vector $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le k/2$, say.

Now, a classical linear sketching technique [20] kicks in. The idea is that we can recover the residual vector $A \star B - \widetilde{C}$ by iteratively hashing to a geometrically decreasing number of buckets and performing the same recovery step as before. The number of buckets in the ℓ-th iteration is $m_\ell = O(k/2^\ell)$, and the goal is to obtain a sequence of vectors $\widetilde{C}^\ell$ such that $\|A \star B - \widetilde{C}^\ell\|_0 \le k/2^\ell$. The crucial observation is that since ι is affine, we can *cancel out* the contribution of the found elements $\widetilde{C}^\ell$ by the fact that $\iota(A \star B) - \iota(\widetilde{C}^\ell) = \iota(A \star B - \widetilde{C}^\ell)$. Thus, after $R = O(\log k)$ iterations [20] we obtain a vector $\widetilde{C}^R$ such that $\|A \star B - \widetilde{C}^R\|_0 = 0$, recovering $A \star B$. The running time is dominated by the first iteration, where an FFT over vectors of length $O(k)$ is performed.

Unfortunately, we do not have access to such an ideal function ι. Replacing ι by h or g runs into issues: If we use h as a substitute, we *cannot cancel out* the contribution of the found elements, since h is only almost affine but not affine. Specifically, the sparsity of $h(A \star B) - h(\widetilde{C}^\ell)$ does not necessarily decrease in the next iteration, which renders the geometric decreasing number of buckets impossible and thus precludes iterative recovery. If we use g as a substitute, we need to pay additional log factors to ensure isolation of most coordinates, even in the very first iteration.

Given this discussion, it seems that the known hash functions reach a barrier on the way to designing $O(k \log k)$-time algorithms. We show how to remedy this state of affairs.

In the following we describe our approach in five steps.

Step 0: Universe Reduction from Large to Small. The first step is to reduce our problem to a universe of size $U = \operatorname{poly}(k)$. We will refer to this regime of U as a *small* universe, and say that U is *large* if there is no bound on U. Formally, we introduce the following problem.

Problem (SmallUniv-SparseConv).
Input: *Nonnegative vectors* A, B *of length* $U = \operatorname{poly}(k)$, *an integer* k *such that* $\|A \star B\|_0 \le k$.
Task: *Compute* $A \star B$ *with success probability* $1 - \delta$.

We sketch how to reduce the general problem of computing $A \star B$ in a large universe n to three instances in a small universe U. The main observation is that in this parameter regime the linear hash function h is *perfect* with probability $1 - 1/\operatorname{poly}(k)$. In combination with the derivative operator ∂, it suffices to compute the three convolutions $h(A) \star_U h(B)$, $h(\partial A) \star_U h(B)$, $h(A) \star_U h(\partial B)$. Note that the cyclic convolution $\star_U$ can be reduced in the nonnegative case to the non-cyclic convolution at the cost of doubling the sparsity of the underlying vector, i.e., $\|h(A) \star h(B)\|_0 \le 2\|h(A) \star_U h(B)\|_0 \le 2\|A \star B\|_0 \le 2k$. This yields the claimed reduction.

This universe reduction ensures that from now on the function $g(x) = x \bmod p$ is $O(\log k)$-universal, i.e., we have removed its undesired dependence on n, which will be important for the next step. We stress as a subtle detail that this step crucially relies on the fact that we are dealing with nonnegative convolution, for more details see Section 3.3.

Step 1: Error Correction. In the next step, we show that it suffices to compute the convolution $A \star B$ up to $k/\operatorname{polylog} k$ errors, since we can correct these errors by iterative recovery with the affine hash function g. More precisely assume that we can somehow recover a vector $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le k/\operatorname{polylog} k$. In other words, suppose that we could efficiently solve the following problem for an appropriate parameter γ (think of $\gamma = 1/\log k$).

Problem (SmallUniv-Approx-SparseConv).
Input: *Nonnegative vectors* A, B *of length* $U = \operatorname{poly}(k)$, *an integer* k *such that* $\|A \star B\|_0 \le k$.
Task: *Compute* $\widetilde{C}$ *such that* $\|A \star B - \widetilde{C}\|_0 \le \gamma k$ *with success probability* $1 - \delta$.

If we are able to do so, then the remaining goal is to correct the error between $A \star B$ and $\widetilde{C}$. We can access the residual vector via $g(A) \star_m g(B) - g(\widetilde{C}) = g(A \star B - \widetilde{C})$, for $g : [U] \to [O(k)]$. Thus, since the new universe size is a log factor larger than the sparsity of the residual vector, it is possible to continue in an iterative fashion using g and still be within the $O(k \log k)$ time bound. Note that (i) it is crucial that we have recovered a $(1 - 1/\log k)$-fraction of the coordinates of $\widetilde{C}$ rather than only a constant fraction, and (ii) it can (and will) be the case that $\operatorname{supp}(\widetilde{C}) \setminus \operatorname{supp}(A \star B) \ne \emptyset$, i.e., there are spurious elements, but those spurious elements will be removed upon iterating.

There is one catch: Iterative recovery creates a sequence of successive approximations $\widetilde{C}^1, \widetilde{C}^2, \dots$ to $A \star B$, and the time to hash each such vector, i.e., to perform the subtraction $g(A) \star_m g(B) - g(\widetilde{C}^\ell)$, is $O(k)$. Since there are $O(\log k)$ such subtractions, the total cost spent on subtractions is $O(k \log k)$, which suffices for Theorem 1.1 but not for Theorem 1.2. The natural solution is to reduce the number of successive approximations (iterations), which is closely related to the column sparsity of linear sketches that allow iterative recovery. More sophisticated iterative loop invariants exist [22, 31, 52], but these all get $\Omega(\log k)$ column sparsity. What we observe is that, surprisingly, a small modification of the iterative loop in [20] finishes in $O(\log \log k)$ iterations, rather than $O(\log k)$. In the ℓ-th iteration we hash to $O(k/\ell^2)$ buckets, and let $k_\ell = \|A \star B - \widetilde{C}^\ell\|_0$. An easy argument yields that with probability $1 - 1/\ell^2$ we have $k_{\ell+1} \leq 1/10 \cdot k_\ell^2/k \cdot \ell^4$, which yields $k_L < 1$ for $L = O(\log\log k)$. This means that the subtraction is performed $O(\log\log k)$ times, so the additive running time overhead is only $O(k \log\log k)$. A more involved implementation of this idea (due to the fact that we are interested in $o(1)$ failure probability) appears in the full version of this paper.

An Attempt using Prony's Method. So far we have reduced to small universe and established that we can afford $k/\operatorname{polylog} k$ errors. In the following we want to recover a $(1 - 1/\log k)$-fraction of the coordinates "in one shot". Consider the following line of attack. Fix a parameter $T \ll k$ and a linear hash function $h : [U] \to [k/T]$. We aim to recover, for each bucket $b \in [k/T]$, all entries of the convolution $A \star B$ that are hashed to bucket b.[7] This corresponds to hashing $A \star B$ to k/T buckets; we expect to have T elements per bucket and thus most buckets contain at most $2T$ elements, say. Note that we no longer expect isolated buckets, so we cannot use the derivative operator. However, we can instead get access to the first $4T$ Fourier coefficients of each vector $(A \star B)_{h^{-1}(b)}$ in the following way. Let ω be a U-th root of unity. For each $t \in [2T]$, set $(\omega^t \bullet A)_i = \omega^{ti} A_i$ and $(\omega^t \bullet B)_i = \omega^{ti} B_i$ and perform the convolution $h(\omega^t \bullet A) \star_{k/T} h(\omega^t \bullet B)$. This yields

$$(h(\omega^t \bullet A) \star_{k/T} h(\omega^t \bullet B))_b = \sum_{(h(i)+h(j)) \bmod k/T = b} \omega^{t(i+j)} \cdot A_i B_j,$$

which is essentially the t-th Fourier coefficient of $(A \star B)_{h^{-1}(b)}$.

The time to perform these $4T$ convolutions is $O((k/T)\log(k/T)) \cdot 4T = O(k \log k)$. Now, a classical algorithm due to Gaspard de Prony in 1796 (rediscovered several times since then, for decoding BCH codes [59] and in the context of polynomial interpolation [9]) postulates that any $2T$-sparse vector can be efficiently reconstructed from its first $4T$ Fourier coefficients. However, Prony's method is known to be unstable with finite precision arithmetic as it solves a polynomial equation (even if it was, we do not know how to compute the evaluations $h(\omega^t \bullet A)$ in the desired time bound when ω is a root of unity), and alternatives working with finite precision or over a finite field are not available, to the best of our knowledge.

Nevertheless, there is another serious problem with this approach. Since we want to recover a $(1 - 1/\log k)$-fraction of elements in $A \star B$, for a $(1 - 1/\log k)$-fraction of support elements $i \in \operatorname{supp}(A \star B)$ it must be the case that $|h^{-1}(h(i))| \leq 2T$. This is a necessary condition in order to recover $(A \star B)_{h^{-1}(h(i))}$ using $4T$ Fourier coefficients. If h was three-wise independent, a standard argument using Chebyshev's inequality would show the desired concentration bound. However, since the linear hash function h is only pairwise independent, we need to take a closer look at concentration of linear hashing.

Intermezzo on Linear Hashing. A beautiful paper of Knudsen [38] shows that the linear hash function h, despite being only pairwise independent, satisfies refined concentration bounds.

Theorem 3.1 (Informal Version of [38, Theorem 5]). *Let $X \subseteq [U]$ be a set of k keys. Randomly pick a linear hash function h that hashes to m buckets, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $y, z \in X$ be chosen independently and uniformly at random. Then:*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \leq \frac{1}{m^2} + \frac{2^{O(\sqrt{\log k \log\log k})}}{mk}. \quad (1)$$

Using the above theorem and Chebyshev's inequality, Knudsen arrives at a concentration bound on the number of elements falling in a fixed bucket, see [38, Theorem 2].[8] Up to the factor $2^{O(\sqrt{\log k \log\log k})} = k^{o(1)}$, this would indeed be the concentration bound satisfied by three-wise independent hash functions. However, this additional $k^{o(1)}$ factor is crucial for our application. Moreover, as we show in the full version, the analysis in [38] is nearly tight. In particular, we show the existence of a set X such that the $k^{o(1)}$ factor is necessary.

Theorem 3.2 ([38, Theorem 5] is Almost Optimal). *Let k and U be arbitrary parameters with $U \geq k^{1+\epsilon}$ for some constant $\epsilon > 0$, and let h be a random linear hash function which hashes to m buckets. Then there exists a set $X \subseteq [U]$ of k keys, a fixed key $x \notin X$ and buckets $a, b \in [m]$ such that for uniformly random $y, z \in X$ we have*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \geq \frac{1}{mk} \exp\left(\Omega\left(\sqrt{\min\left(\frac{\log k}{\log\log k}, \frac{\log U}{\log^2 \log U}\right)}\right)\right).$$

This brings us to an unclear situation. The concentration bounds for linear hashing seem to leave small room for improvement, and additionally the structured linear algebra machinery of Prony's method does not seem be sufficiently strong. However, we show again how to remedy this state of affairs.

Our first trick (Step 2) is to reduce to a *tiny* universe of size $k \operatorname{polylog} k$. Note that then Theorem 3.2 is no longer applicable, and indeed we show improved concentration bounds for linear hashing as we shall see later. Another technical step is to approximate the support of $A \star B$ (Step 3), which can be done efficiently when the universe is tiny. This replaces the computationally expensive part of Prony's method. After that, we are ready to make the attempt work (Step 4). These steps are described in the following.

Step 2: Universe Reduction from Small to Tiny. We further reduce the universe size to $U = k \operatorname{polylog} k$; let us call this regime of U *tiny*. This is the smallest universe we can hash to while ensuring that with constant probability a $(1 - 1/\log k)$-fraction of coordinates is isolated under the hashing. Apart from this difference the reduction is very similar to Step 0. It remains to solve the following computational problem (again, you may think of $\gamma = 1/\log k$).

Problem (TinyUniv-Approx-SparseConv).
Input: *Nonnegative vectors A, B of length $U \leq k/\gamma^2$, an integer k such that $\|A \star B\|_0 \leq k$*
Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \leq \gamma k$ with success probability $1 - \delta$.*

[7] Here and in the following for ease of exposition we ignore the issue that entries of $A \star B$ can be split up, due to h being only almost-affine.

[8] We are referring to the FOCS proceedings version, which differs in an important way from the arXiv version.

Step 3: Approximating the Support. Next we want to approximate the support $\operatorname{supp}(A \star B)$. Specifically, we want to recover a set X of size $|X| = O(k)$ such that $|\operatorname{supp}(A \star B) \setminus X| \leq k/\operatorname{polylog} k$. Since $\operatorname{supp}(A \star B) = \operatorname{supp}(A) + \operatorname{supp}(B)$, for $Y = \operatorname{supp}(A)$, $Z = \operatorname{supp}(B)$ we formally want to solve the following problem.

Problem (TinyUniv-ApproxSupp).
Input: *Sets $Y, Z \subseteq [U]$ and $k \in \mathbf{N}$, such that $U \leq k/\gamma$ and $|Y+Z| \leq k$.*
Task: *Compute a set X of size $O(k)$ such that $|(Y + Z) \setminus X| \leq \gamma k$.*

To this end, we create a sequence of successive approximations to $Y + Z$. Consider the sets

$$Y_\ell = \left\{ \left\lfloor \frac{y}{2^\ell} \right\rfloor : y \in Y \right\}, \quad Z_\ell = \left\{ \left\lfloor \frac{z}{2^\ell} \right\rfloor : z \in Z \right\},$$

for $0 \leq \ell \leq \log(U/k)$. For $\ell \geq \log(U/k)$, we have $Y_\ell, Z_\ell \subseteq [k]$, and thus we can compute $X_\ell := Y_\ell + Z_\ell$ by one Boolean convolution in time $O(k \log k)$. Since U is tiny, the number of levels is just $\log(U/k) = O(\log\log k)$. It remains to argue how to go from level $\ell + 1$ to ℓ, to finally approximate $Y_0 + Z_0 = Y + Z$. We say that a set X_ℓ *closely approximates* $Y_\ell + Z_\ell$ if $|X_\ell| = O(k)$, and $|(Y_\ell + Z_\ell) \setminus X_\ell| \leq k/\operatorname{polylog} k$. Given a set $X_{\ell+1}$ which closely approximates $Y_{\ell+1} + Z_{\ell+1}$, we want to find a set X_ℓ which closely approximates $Y_\ell + Z_\ell$. It is not hard to see that a candidate for X_ℓ is $2X_{\ell+1} + \{0, 1, 2\}$. Hence the main problem is keeping the size of X_ℓ small by filtering out false positives. One way to do so would be to compute $h(Y_\ell) + h(Z_\ell)$, for a random linear hash function $h : [U] \to [O(k)]$. We then throw away all coordinates $i \in 2X_{\ell+1} + \{0, 1, 2\}$ for which the bucket $h(i)$ is empty. Naively computing the convolution would lead to time $\Omega(k \log k \log\log k)$. To improve this, we apply an algorithm due to Indyk:

Theorem 3.3 (Randomized Boolean Convolution [28]). *There exists an algorithm which takes as input two sets $Y', Z' \subseteq [U]$, and in time $O(U)$ outputs a set $O \subseteq Y' + Z'$, such that for all $x \in Y' + Z'$ we have $\mathbf{P}(x \in O) \geq \frac{99}{100}$.*

Since Indyk's algorithm has a small probability of not reporting an element in the sumset, this leads to losing some of the elements in $\operatorname{supp}(A) + \operatorname{supp}(B)$, but we are fine with $k/\operatorname{polylog} k$ errors. On the positive side, compared to standard Boolean convolution this reduces the running time by a factor $\log k$. Putting everything together carefully, we show that $\operatorname{supp}(A \star B)$ can be approximated in time $O(k(\log\log k)^2)$. For the complete proof we refer to Section 6.

Step 4: Approximate Set Query. With all reductions and preparations discussed so far, it remains to solve the following problem to finish our algorithm, for details see Section 5.

Problem (TinyUniv-Approx-SetQuery).
Input: *Nonnegative vectors A, B of length $U \leq k/\gamma^2$, an integer k such that $\|A \star B\|_0 \leq k$ and a set X with $|X| = O(k)$ and $|\operatorname{supp}(A \star B) \setminus X| \leq o(\gamma^2 k)$.*
Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \leq \gamma k$ with success probability $1 - \delta$.*

This is the last step of the algorithm. As in the approach using Prony's method that we discussed above, we pick a parameter T, hash to k/T buckets, and get access to $h(\omega^t \bullet A) \star_{k/T} h(\omega^t \bullet B)$. Here, ω is an appropriate element in $\mathbf{Z}_q^\times$ for q a sufficiently large prime. The surprising observation is that in a tiny universe U the lower bound on the concentration bound of linear hashing does not apply, and in fact a much stronger concentration bound is attainable. In particular, we obtain the analogue of (1) where the term $2^{O(\sqrt{\log k \log\log k})}$ is replaced by $\operatorname{polylog} k$. This can be proved using the machinery established in [38] as well as some elementary number theory, and is actually simpler than the complete analysis of [38].

Furthermore, we can now circumvent the computationally expensive part of Prony's method, since we have knowledge of most of the support $\operatorname{supp}(A \star B)$. It turns out that we only need to solve $O(k/T)$ transposed Vandermonde systems of size $O(T) \times O(T)$ over $\mathbf{Z}_q$. The part of the support we do not know might mess up some the estimates due to collisions, but it is such a small fraction that cannot make us misestimate more than a $1/\operatorname{polylog} k$-fraction of the coordinates in X (and the errors that will be introduced due to misestimation will be cleaned up by the iterative recovery loop in Step 2). Using the improved concentration bound for linear hashing, a fast transposed Vandermonde solver [41], and some additional tricks to compute all $h(\omega^t \bullet A)$ simultaneously, we can pick $T = \operatorname{polylog} k$ and arrive at a $O(k \log k)$-time algorithm, that is also a reduction from sparse to dense convolution.

One last detail is that Vandermonde system solvers compute multiplicative inverses, which cost time $\Omega(\log q) = \Omega(\log(n\Delta))$ each, and thus account for time $\Omega(k \log(n\Delta))$ in total. We observe that, since we are solving several (in particular, k/T) Vandermonde systems, we can run all of them in parallel and batch the inversions across calls. We can then simulate k/T inversions using $O(k/T)$ multiplications and just one division, see Lemma 4.4. This yields $O(k \log k)$ running time and, as claimed in Theorem 1.2, an additive $\operatorname{polylog}(n\Delta)$ term (which is already present, only for choosing the prime q).

3.3 What Makes General Convolution Harder?

The reader may ask whether general convolution can be attacked using our techniques. We want to stress that a linear hash function, which is one of our building blocks and at the core of almost all the steps of the algorithm, seems not to be suited for general convolution, due to the fact that it is almost-affine, but not affine. For an element $x \in [n]$ consider the quantities

$$c_1 = \sum_{\substack{y+z=x,\\ h(y)+h(z)\equiv h(0)+h(x)}} A_y B_z,$$

$$c_2 = \sum_{\substack{y+z=x,\\ h(y)+h(z)\equiv h(0)+h(x)+p}} A_y B_z,$$

$$c_3 = \sum_{\substack{y+z=x,\\ h(y)+h(z)\equiv h(0)+h(x)-p}} A_y B_z,$$

where, for convenience we write $\equiv$ for equality modulo m, and p, m are parameters of the linear hash function. By the almost-affinity of linear hashing we have $(A \star B)_x = c_1 + c_2 + c_3$ (see Lemma 4.1). In general, it can happen that $(A \star B)_x = 0$, not contributing at all to the output size, whereas $c_1, c_2, c_3 \neq 0$. This means that what is hashed to m buckets is a vector with sparsity much larger than k. Handling the presence of cancellations in $A \star B$ is a significant obstruction to an $O(k \log k)$ general convolution algorithm. Note that even Step 0 is non-trivial to implement for general convolution.

Unless one can somehow handle this issue, we can only work with $g(x) = x \bmod p$, which comes with additional log factor losses. We believe that a very different approach is needed to obtain time $O(k \log k)$ in the general case, which is a very interesting open question.

4 TOOLS

4.1 Linear Hashing

In many of our algorithms, the goal is to reduce the dimension of some vectors in a convolution-preserving way. To that end, we often use the classic textbook hash function

$$h(x) = ((\sigma x + \tau) \bmod p) \bmod m.$$

In our case p is always some (fixed) prime, $m \leq p$ is the (fixed) number of buckets and $\sigma, \tau \in [p]$ are chosen uniformly and independently at random. We say that h is a *linear hash function* with parameters p and m. We start with some well-known fundamental properties of linear hashing:

Lemma 4.1 (Linear Hashing Basics). *Let h be a linear hash function with parameters p and m drawn uniformly at random. Then the following properties hold:*

- Universality: *For distinct keys x, y and $a \in [m]$:* $\mathbf{P}(h(x) = h(y) + a \pmod m) \leq \frac{1}{m} + \frac{3}{p} \leq \frac{4}{m}$.
- Pairwise Independence: *For distinct keys x, y and arbitrary buckets $a, b \in [m]$:* $\mathbf{P}(h(x) = a \wedge h(y) = b) \leq \frac{1}{m^2} + \frac{3}{mp} \leq \frac{4}{m^2}$.
- Almost-Affinity: *For arbitrary keys x, y there exists one out of three possible offsets $o \in \{-p, 0, p\}$ such that $h(x) + h(y) = h(0) + h(x + y) + o \pmod m$.*

Proof. Universality follows directly from pairwise independence, so we start proving pairwise independence. Let $h(x) = \pi(x) \bmod m$, where $\pi(x) = (\sigma x + \tau) \bmod p$ for uniformly random $\sigma, \tau \in [p]$. The first step is to prove that $\mathbf{P}(\pi(x) = a' \wedge \pi(y) = b') = 1/p^2$ for distinct keys x, y and arbitrary $a', b' \in [p]$. Note that the event $\pi(x) = a'$ and $\pi(y) = b'$ can be rewritten as $\pi(x) = a'$ and $\pi(y) - \pi(x) = b' - a' \pmod p$ and the claim follows immediately by observing that the random variables $\pi(x)$ and $\pi(y) - \pi(x) = \sigma(y - x) \pmod p$ are independent.

We get back to h. It clearly holds that $\mathbf{P}(h(x) = a \wedge h(y) = b) = \sum_{a',b'} \mathbf{P}(\pi(x) = a' \wedge \pi(y) = b')$, where the sum is over all $a', b' \in [p]$ with $a = a' \bmod m$ and $b = b' \bmod m$. As there are at most $p/m + 1$ such values a' and b', respectively, we conclude that the desired probability is at most $(p/m + 1)^2/p^2 \leq \frac{1}{m^2} + \frac{3}{mp}$.

Finally, we prove that h is almost-affine. It is clear that $\pi(x) + \pi(y) = \pi(0) + \pi(x + y) \pmod p$. As each side of the equation is a nonnegative integer less than $2p$, it follows that $\pi(x) + \pi(y) = \pi(0) + \pi(x + y) + o$, where $o \in \{-p, 0, p\}$. By taking residues modulo m, the claim follows. □

In our reduction we also crucially rely on the following improved concentration bound for linear hashing, which we prove in the full version of the paper.

Theorem 4.2 (Overfull Buckets). *Let $X \subseteq [U]$ be a set of k keys. Randomly pick a linear hash function h with parameters $p > 4U^2$ and $m \leq U$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $F = \sum_{y \in X} [h(y) = b]$. Then:*

$$\mathbf{E}(F \mid h(x) = a) = \mathbf{E}(F) = \frac{k}{m} + O(1),$$

and, for any $\lambda > 0$,

$$\mathbf{P}(\,|F - \mathbf{E}(F)| \geq \lambda\sqrt{\mathbf{E}(F)} \mid h(x) = a) \leq O\left(\frac{U \log U}{\lambda^2 k}\right).$$

4.2 Algebraic Computations

On more than one occasion we need to efficiently perform algebraic computations such as computing powers or inverses. The next two lemmas describe how to easily obtain improved algorithms for bulk-evaluation.

Lemma 4.3 (Bulk Exponentiation). *Let R be a ring. Given an element $x \in R$, and a set of nonnegative exponents $e_1, \ldots, e_n \leq e$, we can compute $x^{e_1}, \ldots, x^{e_n}$ in time $O(n \log_n e)$ using $O(n \log_n e)$ ring operations.*

The naive way to implement exponentiations is via repeated squaring in time $O(n \log e)$. There are methods [50, 60] improving the dependence on e, but for our purposes this simple algorithm suffices.

Proof. First, compute the base-n representations of all exponents $e_i = \sum_j e_{i,j} n^j$; then $e_{i,j} \in [n]$ where $j = 0, \ldots, \lceil \log_n e \rceil$. We precompute all powers x^{in^j} for $i = 1, \ldots, n$ and $j = 0, \ldots, \lceil \log_n e \rceil$ using the rules $x^{n^{j+1}} = (x^{n^j})^n$ and $x^{(i+1)n^j} = x^{in^j} x^{n^j}$. Finally, every output x^{e_i} can be computed as a product of $\lceil \log_n e \rceil$ numbers $\prod_j x^{e_{i,j} n^j}$. The correctness is immediate and it is easy to check that every step takes time $O(n \log_n e)$. □

Lemma 4.4 (Bulk Division). *Let F be a field. Given n field elements $a_1, \ldots, a_n \in F$, we can compute their inverses $a_1^{-1}, \ldots, a_n^{-1} \in F$ in time $O(n)$ using $O(n)$ multiplications and a single inversion.*

Proof. First, we compute the n prefix products $b_j = a_1 \cdots a_i$. It takes a single inversion to compute b_n^{-1}. Then, for $i = n, n - 1, \ldots, 2$, we compute $a_i^{-1} = b_i^{-1} b_{i-1}$ and $b_{i-1}^{-1} = b_i^{-1} a_i$. Finally, $a_1^{-1} = b_1^{-1}$. As claimed, this algorithm takes time $O(n)$ and it uses $O(n)$ multiplications and a single inversion. □

Finally, a crucial ingredient to our core algorithm is the following theorem about solving transposed Vandermonde systems by Li [41]. By carefully inspecting the algorithm one can reduce the number of necessary divisions to polylog(n); we give more details in the full version of this paper.

Theorem 4.5 (Transposed Vandermonde Systems [41]). *Let F be a field. Given $a, y \in F^n$ with pairwise distinct entries a_i, the solution $x \in F^n$ to the system of equations*

$$y = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ a_1^2 & a_2^2 & \cdots & a_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{bmatrix} x$$

can be computed in time $O(n \log^2 n)$ using at most $O(n \log^2 n)$ ring operations (additions, subtractions, multiplications) and using at most polylog(n) *divisions. Furthermore, given a and x, the matrix-vector product y can be computed in the same running time.*

5 SET QUERIES IN A TINY UNIVERSE

As the first step in our chain of reductions, the goal of this section is to give an efficient algorithm for the TinyUniv-Approx-SetQuery problem:

Problem (TinyUniv-Approx-SetQuery).
Input: *Nonnegative vectors A, B of length $U \le k/\gamma^2$, an integer k such that $\|A \star B\|_0 \le k$ and a set X with $|X| = O(k)$ and $|\operatorname{supp}(A \star B) \setminus X| \le o(\gamma^2 k)$.*
Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \gamma k$ with success probability $1 - \delta$.*

Lemma 5.1. *Let $\log k \le 1/\gamma \le \operatorname{poly}(k)$ and let $1/\delta \le \operatorname{poly}(k)$. There is an algorithm for the TinyUniv-Approx-SetQuery problem running in time*

$$O(D(k) + k \log^2(1/\gamma) + k \log(1/\delta) + \operatorname{polylog}(k, \|A\|_\infty, \|B\|_\infty)).$$

We proceed in three steps. In Section 5.1 we give two important preliminary lemmas. In Section 5.2 we present and analyze the algorithm which proves Lemma 5.1. Finally, in Section 5.3, we will strengthen Lemma 5.1 and show that we can in fact achieve the same running time with $D_{1/3}(k)$ in place of $D(k)$, i.e., it suffices to assume that we only have black-box access to an efficient dense convolution algorithm with constant error probability.

5.1 Folding & Unfolding

For a vector A and a scalar ω, let $\omega \bullet A$ denote the vector defined by $(\omega \bullet A)_i = \omega^i A_i$. A straightforward calculation reveals that the $\bullet$-product commutes nicely with taking (non-cyclic) convolutions:

Proposition 5.2. *Let A, B be vectors and let ω be a scalar. Then $(\omega \bullet A) \star (\omega \bullet B) = \omega \bullet (A \star B)$.*

Proof. For any coordinate x:

$$\begin{aligned}((\omega \bullet A) \star (\omega \bullet B))_x &= \sum_{y+z=x} (\omega \bullet A)_y (\omega \bullet B)_z \\ &= \sum_{y+z=x} \omega^{y+z} A_y B_z = (\omega \bullet (A \star B))_x. \quad \square\end{aligned}$$

The goal of this section is to show that the we can efficiently evaluate, and under certain restrictions also invert, the following map:

$$A \longrightarrow (\omega^0 \bullet A) \bmod m, \dots, (\omega^{T-1} \bullet A) \bmod m.$$

We will vaguely refer to these two directions as *folding* and *unfolding*, respectively. For the remainder of this subsection we assume as before that A is an arbitrary length-U vector with sparsity k. We further assume that A is over some finite field $\mathbf{Z}_q$ in order avoid precision issues in the underlying algebraic machinery. We also need the technical assumption that $\omega \in \mathbf{Z}_q^\times$ has multiplicative order at least U.

Lemma 5.3 (Folding). *Let m be a parameter and let $T = \lceil 2k/m \rceil$. There is a deterministic algorithm* Fold *computing $(\omega^0 \bullet A) \bmod m, \dots, (\omega^{T-1} \bullet A) \bmod m$ in time $O(k \log^2(k/m) + k \log_k U + \operatorname{polylog} q)$.*

Let us postpone the proof of Lemma 5.3 and instead outline how to (approximately) invert the folding. A crucial assumption is that we are given a close approximation X of $\operatorname{supp}(A)$. The quality of the recovery is controlled by the following measure: The *flatness of X modulo m* is defined as

$$F_m(X) = \sum_{x \in X} \left[\sum_{x' \in X} \left[x = x' \pmod{m} \right] > \frac{2|X|}{m} \right],$$

and we say that X is *α-flat modulo m* if $F_m(X) \le \alpha$. Some intuition about this definition: Recall that when hashing a set X into m buckets, the average bucket receives $|X|/m$ elements. Therefore, the flatness is the number of elements falling into overfull buckets under the very simple hash function $x \bmod m$.

Lemma 5.4 (Unfolding). *Let m be a parameter and let $T = \lceil 2k/m \rceil$. There is a deterministic algorithm* Unfold *which, given $(\omega^0 \bullet A) \bmod m, \dots, (\omega^{T-1} \bullet A) \bmod m$ and a size-k set $X \subseteq [U]$, computes a vector $\widetilde{A}$ such that*

$$\|A - \widetilde{A}\|_0 \le T \cdot |\operatorname{supp}(A) \setminus X| + F_m(X).$$

The algorithm runs in time $O(k \log^2(k/m) + k \log_k U + \operatorname{polylog} q)$.

We will next prove Lemmas 5.3 and 5.4.

Proof of Lemma 5.3. Given a k-sparse vector A the goal is to simultaneously compute $A^0 = (A \bullet \omega^0) \bmod m, \dots, A^{T-1} = (A \bullet \omega^{T-1}) \bmod m$. We first precompute all powers ω^x for $x \in X = \operatorname{supp}(A)$ using the bulk exponentiation algorithm (Lemma 4.3).

Next, we partition X into several *chunks* $X_{i,j}$. We start with $X_i = \{x \in X : x \bmod m = i\}$ and then greedily subdivide every part X_i into chunks $X_{i,1}, X_{i,2}, \dots$ such that all chunks have size $|X_{i,j}| \le T$. In fact, all chunks except for the last one have size exactly T. We note that in this way we have constructed at most $O(m)$ chunks: On the one hand, there can be at most m chunks of size exactly T since A has sparsity $k = \Theta(mT)$. On the other hand, there can be at most m chunks of size less than T by the way the greedy algorithm works.

Now focus on an arbitrary chunk $X_{i,j}$; for simplicity assume that $|X_{i,j}| = T$ and let $x_1, \dots, x_T$ denote the elements of $X_{i,j}$ in an arbitrary order. We set up the following transposed Vandermonde system with indeterminate $y^{i,j} \in \mathbf{Z}_q^T$:

$$y^{i,j} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{x_1} & \omega^{x_2} & \cdots & \omega^{x_T} \\ \omega^{2x_1} & \omega^{2x_2} & \cdots & \omega^{2x_T} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(T-1)x_1} & \omega^{(T-1)x_2} & \cdots & \omega^{(T-1)x_T} \end{bmatrix} \begin{bmatrix} A_{x_1} \\ A_{x_2} \\ \vdots \\ A_{x_T} \end{bmatrix}.$$

Since ω has multiplicative order at least U, the numbers $\omega^{x_1}, \dots, \omega^{x_T}$ are distinct and we can apply Theorem 4.5 to compute y. It remains to return the vectors $(\omega^t \bullet A) \bmod m$ for all $t \in [T]$, computed as $((\omega^t \bullet A) \bmod m)_i = \sum_j y_t^{i,j}$. It is easy to check that $y_t^{i,j}$ equals $((\omega^t \bullet A') \bmod m)_i$ for A' the vector obtained from A by restricting the support to $X_{i,j}$. The correctness of the whole algorithm follows immediately.

Finally, we analyze the running time. Precomputing the powers of ω using Lemma 4.3 accounts for time $O(k \log_k U)$. The construction of the chunks takes time $O(mT) = O(k)$, and also writing down all vectors $(\omega^t \bullet A) \bmod m$ takes time $O(k)$ given the $y^{i,j}$'s. The dominant step is to solve a Vandermonde system for every chunk. Since there are $O(m)$ chunks in total and the running time for solving a single system is bounded by $O(T \log^2 T)$ (by Theorem 4.5), the total running time is $O(mT \log^2 T)$ plus $O(mT \log^2 T)$ ring operations and $O(m \operatorname{polylog}(T))$ divisions in $\mathbf{Z}_q$.

Additions, subtractions and multiplications take constant time each on a random-access machine and can therefore by counted into the time bound. However, divisions in a prime field are computationally more expensive. The common way is to implement inversions by Euclid's algorithm in time $O(\log q)$ and so the naive time bound becomes $O(m \operatorname{polylog}(T) \cdot \log q)$. This can be optimized by exploiting Lemma 4.4: Recall that we are executing Theorem 4.5 m times in parallel, and each call requires up to $\operatorname{polylog}(T)$ inversions. Therefore, we can apply Lemma 4.4 to replace m inversions by $O(m)$ multiplications and a single inversion in time $O(m + \log q)$. In that way, it takes time $O(\operatorname{polylog}(T) \cdot (m + \log q)) = O(m \operatorname{polylog}(T) + \operatorname{polylog}(q))$ to deal with all divisions and the total running time is $O(mT \log^2 T + \operatorname{polylog}(q)) = O(k \log^2(m/k) + \operatorname{polylog}(q))$. □

Proof of Lemma 5.4. Given $A^0 = (A \bullet \omega^0) \bmod m, \dots, A^{T-1} = (A \bullet \omega^{T-1}) \bmod m$, the goal is to recover a good approximation $\widetilde{A}$ of A, provided that an approximation X of $\operatorname{supp}(A)$ is given. As before, we partition X into buckets $X_i = \{x \in X : x \bmod m = i\}$. We say that the bucket X_i is *overfull* if $|X_i| > T$. In contrast to Fold, we can afford to ignore all overfull buckets here, so focus on an arbitrary bucket X_i with $|X_i| \le T$. Letting $x_1, \dots, x_T$ denote the elements in X_i in an arbitrary order (and assuming for the sake of simplicity that there are exactly T of these), it suffices to solve the following Vandermonde system with indeterminates $\widetilde{A}_{x_1}, \dots, \widetilde{A}_{x_T}$:

$$\begin{bmatrix} A_i^0 \\ A_i^1 \\ A_i^2 \\ \vdots \\ A_i^{T-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{x_1} & \omega^{x_2} & \cdots & \omega^{x_T} \\ \omega^{2x_1} & \omega^{2x_2} & \cdots & \omega^{2x_T} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(T-1)x_1} & \omega^{(T-1)x_2} & \cdots & \omega^{(T-1)x_T} \end{bmatrix} \begin{bmatrix} \widetilde{A}_{x_1} \\ \widetilde{A}_{x_2} \\ \vdots \\ \widetilde{A}_{x_T} \end{bmatrix}.$$

The running time can be analyzed in the same way as before, so let us focus on proving that $\|A - \widetilde{A}\|_0$ is small. We say that a bucket X_i is *successful* if (i) it is not overfull, and if (ii) there exists no support element $x \in \operatorname{supp}(A) \setminus X$ with $x \bmod m = i$. The claim is that whenever X_i is successful, then $\widetilde{A}_x = A_x$ for all $x \in X_i$. Indeed, for any successful bucket one can verify by the definition of the $\bullet$-product that the equation system is valid for A_x in place of $\widetilde{A}_x$, and as the Vandermonde matrix has full rank this is the unique solution.

Therefore, it suffices to bound the total size of all non-successful buckets: On the one hand, the number of elements in buckets for which condition (i) holds but (ii) fails is at most $T \cdot |\operatorname{supp}(A) \setminus X|$. On the other hand, the contribution of elements in buckets for which condition (i) fails is exactly the flatness of X modulo m, by definition. Together, these yield the claimed bound on $\|A - \widetilde{A}\|_0$. □

Algorithm 1 TinyUniv-Approx-SetQuery(A, B, U, k, X)

```
Input: • Nonnegative vectors $A, B$ of length $U \le k/\gamma^2$
       • An integer $k$ such that $\|A \star B\|_0 \le k$
       • A set $X \subseteq [U]$ of size $O(k)$ with $|\operatorname{supp}(A \star B) \setminus X| \le o(\gamma^2 k)$
Output: A vector $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \gamma k$ with probability $1 - \delta$
    (Part 1: Find a suitable linear hash function)
 1: Let $m = \Theta(\gamma k)$, let $T = \lfloor 2|X|/m \rfloor$ and let $p \ge 4U^2$ be a prime
 2: repeat
 3:     Pick $\sigma, \tau \in [p]$ uniformly at random
 4:     Let $\pi(x) = (\sigma x + \tau) \bmod p$
 5:     $\widetilde{X} \leftarrow \pi(X) + \{0, p\}$
 6: until $\widetilde{X}$ is $\gamma k/2$-flat modulo $m$
    (Part 2: Set up a sufficiently large finite field)
 7: Let $q > U^3 \|A\|_\infty \|B\|_\infty$ be a prime; the following calculations are over $\mathbf{Z}_q$
 8: Pick $\omega \in \mathbf{Z}_q^\times$ uniformly at random
    (Part 3: Fold – Convolve – Unfold)
 9: $A^0, \dots, A^{T-1} \leftarrow$ Fold$(\pi(A), \omega)$
10: $B^0, \dots, B^{T-1} \leftarrow$ Fold$(\pi(B), \omega)$
11: for $t \leftarrow 0, \dots, T-1$ do
12:     $C^t \leftarrow A^t \star_m B^t$ (using the dense convolution algorithm)
13: $\widetilde{R} \leftarrow$ Unfold$(C^0, \dots, C^{T-1}, \omega, \widetilde{X})$
14: $\widetilde{C} \leftarrow \pi^{-1}(\widetilde{R})$
15: return $\widetilde{C}$ (cast back to an integer vector)
```

5.2 The Algorithm

We are ready to prove Lemma 5.1 by analyzing the pseudo-code given in Algorithm 1. The analysis is split into three parts corresponding to the three parts in Algorithm 1. The first step is to prove that the loop in Part 1 quickly terminates.

Lemma 5.5 (Analysis of Part 1). *With probability $1 - \delta/2$, the loop in Lines 2–6 terminates in time $O(k \log(1/\delta))$.*

Proof. We prove that a single iteration of the loop succeeds with constant probability. Having established that fact, it is clear that the loop is left after at most $O(\log(1/\delta))$ independent iterations with probability at least $1 - \delta/2$. Recall that the loop ends if $\widetilde{X}$ is $\gamma k/2$-flat modulo m, that is,

$$\sum_{x \in \widetilde{X}} \left[\sum_{x' \in \widetilde{X}} \left[x = x' \pmod m \right] > \frac{2|\widetilde{X}|}{m} \right] \le \frac{\gamma k}{2}. \qquad (2)$$

Since by definition $\widetilde{X} = \pi(X) + \{0, p\}$, we may fix offsets $o, o' \in \{0, p\}$ and instead bound

$$\sum_{x \in X} \left[\sum_{x' \in X} \left[h(x) + o = h(x') + o' \pmod m \right] > \frac{2|X|}{m} \right] \le \frac{\gamma k}{4}, \quad (3)$$

where $h(x) = \pi(x) \bmod m$ is a linear hash function with parameters p and m. Indeed, if the latter event happens (simultaneously for all offsets o, o'), then also the former event happens. Fix o, o'

and fix any $x \in X$. Then:

$$\mathbf{P}\left(\sum_{x' \in X} \left[h(x) + o = h(x') + o' \pmod{m}\right] > \frac{2|X|}{m}\right)$$
$$= \sum_{a \in [m]} \mathbf{P}(h(x) = a)$$
$$\cdot \mathbf{P}\left(\sum_{x' \in X} \left[h(x') = (a + o - o') \bmod m\right] > \frac{2|X|}{m} \;\middle|\; h(x) = a\right)$$

This is where our concentration bounds come into play: Observe that the conditional probability can be bounded by Theorem 4.2 with buckets a and $b = (a + o - o') \bmod m$. Let $F = \sum_{x' \in X} [h(x') = b]$, then $\mathbf{E}(F) = |X|/m + O(1)$. It follows that:

$$= \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot \mathbf{P}\left(F > \frac{2|X|}{m} \;\middle|\; h(x) = a\right),$$
$$= \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot \mathbf{P}\left(F - \mathbf{E}(F) > \frac{|X|}{m} - O(1) \;\middle|\; h(x) = a\right),$$
$$\leq \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot O\left(\frac{mU \log U}{|X|^2}\right)$$
$$= O\left(\frac{mU \log U}{|X|^2}\right)$$

where for the inequality we applied Theorem 4.2 with $\lambda = \sqrt{|X|/m} - O(1)$. Choosing $m = \Theta(\gamma k)$ for some small enough constant, this becomes

$$\leq \frac{\gamma k U \log U}{12k^2} \leq \frac{\gamma k (k/\gamma^2) \log(k/\gamma^2)}{12k^2} \leq \frac{\log k}{12\gamma} \leq \frac{1}{12}.$$

Here we used the assumption $\log k \leq 1/\gamma \leq \mathrm{poly}(k)$. By a union bound over the three possible values of $o - o'$ and by Markov's inequality, we conclude that the event in (3) (and thereby the event in (2)) happens with probability at least $1/2$.

As we just proved, with probability $1 - \delta/2$ the loop in Lines 2–6 runs for at most $O(\log(1/\delta))$ iterations. Moreover, each execution of the loop body takes time $O(k)$, and thus the loop terminates in time $O(k \log(1/\delta))$. □

Lemma 5.6 (Analysis of Part 2). *With probability* $1 - 1/\mathrm{poly}(k)$ *and in* $\mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty)$ *time we correctly compute* q *and* ω *such that* ω *has multiplicative order at least* U *in* $\mathbf{Z}_q^\times$.

Proof. Computing q takes time $\mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty)$ and succeeds with high probability. The interesting part is to show that ω is as claimed. It is well-known that $\mathbf{Z}_q^\times$ is isomorphic to the cyclic group $\mathbf{Z}_{q-1}$ and thus our problem is equivalent to finding an element in $\mathbf{Z}_{q-1}$ with (additive) order at least U. In a cyclic group there can be at most i elements with order i (the only possible candidates are multiples of $(q-1)/i$) and thus there are at most $\sum_{i \leq U} i \leq U^2$ elements with order at most U. Hence, the probability of sampling ω as claimed is at least $1 - U^2/q \geq 1 - 1/U \geq 1 - 1/\mathrm{poly}(k)$. □

Lemma 5.7 (Analysis of Part 3). *With probability* $1 - 1/\mathrm{poly}(k)$, *Part 3 correctly outputs a vector* $\widetilde{C}$ *with* $\|A \star B - \widetilde{C}\|_0 \leq \gamma k$ *and runs in time* $O(D(k) + k \log^2(1/\gamma) + \mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty))$.

Proof. In the event that the previous parts succeeded the technical condition of Lemmas 5.3 and 5.4 is satisfied (namely that ω has large multiplicative order) and we may apply Fold and Unfold. In Lines 9 and 10 we thus correctly compute $A^t = (\omega^t \bullet \pi(A)) \bmod m$, for all $t \in [T]$, and similarly for B. As we are assuming (for now) that the dense convolution algorithm succeeds with probability 1, in Line 12 we correctly compute the cyclic convolutions $C^t = A^t \star_m B^t$.

The interesting step is to analyze the unfolding in Line 13. By Proposition 5.2 and some elementary identities about cyclic convolutions we have

$$\begin{aligned} C^t &= A^t \star_m B^t \\ &= (\omega^t \bullet \pi(A)) \star_m (\omega^t \bullet \pi(B)) \\ &= ((\omega^t \bullet \pi(A)) \star (\omega^t \bullet \pi(B))) \bmod m \\ &= (\omega^t \bullet (\pi(A) \star \pi(B))) \bmod m, \end{aligned}$$

i.e. it holds that $C^t = (\omega^t \bullet R) \bmod m$ for $R = \pi(A) \star \pi(B)$. For that reason, Lemma 5.4 guarantees that the call to Unfold will approximately recover R and the approximation quality is bounded by

$$\|R - \widetilde{R}\|_0 \leq T \cdot |\operatorname{supp}(R) \setminus \widetilde{X}| + F_m(\widetilde{X}).$$

By the loop guard in Line 6 we can assume that $F_m(\widetilde{X}) \leq \gamma k/2$. We can put the same bound on the term $T \cdot |\operatorname{supp}(R) \setminus \widetilde{X}|$. Indeed, note that since $\operatorname{supp}(R) \subseteq \pi(\operatorname{supp}(A \star B)) + \{0, p\}$ and $\widetilde{X} = X + \{0, p\}$, we must have that $|\operatorname{supp}(R) \setminus \widetilde{X}| \leq 2|\operatorname{supp}(A \star B) \setminus X|$. It follows that

$$T \cdot |\operatorname{supp}(R) \setminus \widetilde{X}| \leq 2T \cdot |\operatorname{supp}(A \star B) \setminus X| \leq \frac{o(\gamma^2 k^2)}{m},$$

which becomes $\gamma k/2$ for sufficiently large k since we picked $m = \Theta(\gamma k)$. All in all, this shows that $\|R - \widetilde{R}\|_0 \leq \gamma k$ as claimed.

The remaining steps are easy to analyze: Since p is a prime, the function $\pi(x) = (\sigma x + \tau) \bmod p$ is invertible on $[p]$ (assuming that $\sigma \neq 0$, which happens with high probability). As $\pi(A \star B) = R$ and as $A \star B$ is a vector of length $U < p$ it follows that $A \star B = \pi^{-1}(R)$. In the same way, we obtain for $\widetilde{C} = \pi^{-1}(\widetilde{R})$ that $\|A \star B - \widetilde{C}\|_0 \leq \gamma k$. In the final step we use that q is large enough (larger than any entry in the convolution $A \star B$), so we can safely cast $\widetilde{C}$ back to an integer vector.

Let us finally bound the running time of Part 3. The calls to Fold and Unfold take time $O(k \log^2(k/m) + k \log_k U + \mathrm{polylog}(q)) = O(k \log^2(1/\gamma) + \mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty))$. Computing $T = O(1/\gamma)$ convolutions of vectors of length $m = O(\gamma k)$ takes time at most

$$O\left(\frac{1}{\gamma} \cdot D(\gamma k)\right) = O\left(\frac{\gamma k}{\gamma} \cdot \frac{D(\gamma k)}{\gamma k}\right) = O\left(k \cdot \frac{D(k)}{k}\right) = O(D(k)),$$

assuming that $D(n)/n$ is nondecreasing. Summing the two contributions yields the claimed running time. □

In combination, Lemmas 5.5, 5.6 and 5.7 show that Algorithm 1 is correct and runs in the correct running time with probability at least $1 - \delta/2 - 1/\mathrm{poly}(k) \geq 1 - \delta$. This finishes the proof of Lemma 5.1.

5.3 Corrections for Randomized Dense Convolution

In the previous subsection, we assumed that we have black-box access to a deterministic algorithm computing the dense convolution of two length-n vectors in time $D(n)$. We will now prove that it suffices to assume that the black-box algorithm errs with constant probability, say 1/3.

Lemma 5.8. *Let $\log k \leq 1/\gamma \leq \mathrm{poly}(k)$ and let $1/\delta \leq \mathrm{poly}(k)$. There is an algorithm for the TinyUniv-Approx-SetQuery problem running in time*

$$O(D_{1/3}(k) + k\log^2(1/\gamma) + k\log(1/\delta) + \mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty)).$$

The idea is simple: Every call to the randomized dense convolution algorithm is followed by a call to the verifier presented in Lemma 5.9. If a faulty output is detected, then we repeat the convolution (with fresh randomness) and test again.

Lemma 5.9 (Dense Verification). *Given three vectors A, B, C of length U, there is a randomized algorithm running in time $O(U + \mathrm{polylog}(U, \|A\|_\infty, \|B\|_\infty))$, which checks whether $A \star B = C$. The algorithm fails with probability at most $1/\mathrm{poly}(U)$.*

The proof of Lemma 5.9 is by a standard application of the classical Schwartz-Zippel lemma; in the full version we prove a more general statement about a sparse verifier. We also need the following tail bound on the sum of geometric random variables [33]:

Theorem 5.10 ([33, Theorem 2.1]). *Let $X_1, \dots, X_n$ be independent, identically distributed geometric random variables, and let $X = \sum_i X_i$. Then, for any $\lambda \geq 1$:*

$$\mathbf{P}(X > \lambda\,\mathbf{E}(X)) \leq \exp(-n(\lambda - 1 - \ln\lambda)).$$

Proof of Lemma 5.8. The overall proof follows Lemma 5.1 exactly, we merely substitute the black-box calls to the dense convolution algorithm. The only place where this algorithm is directly called is in the proof of Lemma 5.7, where we compute $T = O(1/\gamma)$ convolutions of length $m = O(\gamma k)$. Each such call is replaced by a test-and-repeat loop using the verifier in Lemma 5.9. As the failure probability of the verifier is at most $1/\mathrm{poly}(m) = 1/\mathrm{poly}(k)$, we can afford a union bound and assume that the verifier never fails, i.e., we uphold the assumption that dense convolution succeeds.

It remains to bound the running time overhead. A single iteration of the test-and-repeat loop takes time $O(D_{1/3}(m) + m) = O(D_{1/3}(m))$. To bound the number of iterations $X = \sum_i X_i$, let X_i model the number of iterations caused by the i-th dense convolution call. Observe that X_i is geometrically distributed with parameter $p = 2/3$ and thus $\mathbf{E}(X) = 3T/2$. By Theorem 5.10 with, say, $\lambda = 4$, it follows that $\mathbf{P}(X > 4\,\mathbf{E}(X)) \leq \exp(-T) \leq \exp(-\Omega(1/\gamma))$. Using that $\log k \leq 1/\gamma$, the number of iterations is bounded by $6T$ with high probability $1 - 1/\mathrm{poly}(k)$ and therefore the total running time to answer all dense convolution queries is bounded by $O(TD_{1/3}(m)) = O(D_{1/3}(k))$. □

6 APPROXIMATING THE SUPPORT SET

This section is devoted to finding a set X which closely approximates $\mathrm{supp}(A \star B)$. To that end, our goal is to solve the following problem, which is later applied with $Y = \mathrm{supp}(A)$ and $Z = \mathrm{supp}(B)$.

Algorithm 2 TinyUniv-ApproxSupp(Y, Z, U, k)

Input: Sets $Y, Z \subseteq [U]$ over a universe $U \leq k/\gamma$ with $|Y + Z| \leq k$
Output: A set $X \subseteq [U]$ of size $O(k)$ such that $|(Y + Z) \setminus X| \leq \gamma k$

```
1:  Let $m = 40k$ and pick a prime $p \geq U$
2:  Let $L = \lceil \log(1/\gamma) \rceil$
3:  $X_L \leftarrow \{0, 1, \dots, \lceil U/2^\ell \rceil\}$
4:  for $\ell \leftarrow L-1, \dots, 1, 0$ do
5:      $Y_\ell \leftarrow Y$ div $2^\ell$
6:      $Z_\ell \leftarrow Z$ div $2^\ell$
7:      $M \leftarrow 2X_{\ell+1} + \{0, 1, 2\}$
8:      repeat $R = \Theta(\log(1/\gamma) + \log(1/\delta))$ times
9:          Sample a linear hash function $h$ with param. $p$ and $m$
10:         $O \leftarrow$ output of Indyk's algorithm (Theorem 6.2)
                with input $h(Y_\ell), h(Z_\ell)$
11:         for $x \in M$ do
12:             if $(h(0) + h(x) + o) \bmod m \in (O \bmod m)$
                    for some $o \in \{-p, 0, p\}$ then
13:                 Give a vote to $x$
14:     $X_\ell \leftarrow$ all elements in $M$ with at least $3R/4$ votes
15: return $X = X_0$
```

Problem (TinyUniv-ApproxSupp).
Input: *Sets $Y, Z \subseteq [U]$ and $k \in \mathbf{N}$, such that $U \leq k/\gamma$ and $|Y+Z| \leq k$.*
Task: *Compute a set X of size $O(k)$ such that $|(Y + Z) \setminus X| \leq \gamma k$.*

Lemma 6.1. *There is an $O(k\log(1/\gamma)\log(\frac{1}{\gamma\delta}))$-time algorithm for the TinyUniv-ApproxSupp problem.*

A key ingredient to the algorithm is the following routine to approximately compute sumsets, which we shall refer to as Indyk's algorithm.

Theorem 6.2 (Randomized Boolean Convolution [28]). *There exists an algorithm which takes as input two sets $Y, Z \subseteq [U]$, and in time $O(U)$ outputs a set $O \subseteq Y + Z$, such that for all $x \in Y + Z$ we have $\mathbf{P}(x \in O) \geq \frac{99}{100}$.*

The algorithm claimed in Lemma 6.1 is given in Algorithm 2. For the remainder of this section, we will analyze this algorithm in several steps. We shall call the iterations of the outer loop *levels* and call an element x a *witness at level ℓ* if $x \in Y_\ell + Z_\ell$. Otherwise, we say that x is a non-witness. Fix a level ℓ and consider a single iteration of the inner loop (Lines 8–13). The *voting probability of x at level ℓ* is the probability that x is given a vote in Line 13. Recall that in every such iteration, we pick a random linear hash function $h : [U] \to [m]$ using fresh randomness. The following lemmas prove that witnesses have large voting probability and non-witnesses have small voting probability.

Lemma 6.3 (Witnesses have Large Voting Probability). *At any level ℓ, the voting probability of a witness x is at least $\frac{99}{100}$.*

Proof. Recall that if x is a witness at level ℓ, then $x = y + z$ for some $y \in Y_\ell$ and $z \in Z_\ell$. By the almost-affinity of linear hashing (Lemma 4.1), it holds that $h(y) + h(z) = h(x) + h(0) + o \pmod m$ for some offset $o \in \{-p, 0, p\}$. It follows that $(h(x) + h(0) + o) \bmod m$ is an element of the sumset $(h(Y_\ell) + h(Z_\ell)) \bmod m$. However, in order for x to gain a vote, this condition must be true for the set O returned by Indyk's algorithm. By the guarantee of Theorem 6.2, O contains every element of $h(Y_\ell) + h(Z_\ell)$ with probability at least $\frac{99}{100}$, which yields the claim. □

Lemma 6.4 (Non-Witnesses have Small Voting Probability). *At any level ℓ, the voting probability of a non-witness x is at most $1/2$.*

Proof. Given the fact that Indyk's algorithm never returns a false positive, it suffices to prove that none of the three values $(h(0)+h(x)+\{-p,0,p\}) \bmod m$ is contained in the sumset $(h(Y_\ell)+h(Z_\ell)) \bmod m$, with sufficiently large probability. By the almost-affinity of h, we have

$$h(Y_\ell) + h(Z_\ell) \bmod m \subseteq (h(0) + h(Y_\ell + Z_\ell) + \{-p, 0, p\}) \bmod m.$$

So fix some offsets $o, o' \in \{-p, 0, p\}$ and some witness $x' \in Y_\ell + Z_\ell$. As x is not a witness, we must have $x \neq x'$. It suffices to bound the following bound the probability:

$$\mathbf{P}(h(0) + h(x) + o = h(0) + h(x') + o' \bmod m)$$
$$= \mathbf{P}(h(x) = (h(x') + o' - o) \bmod m) \leq \frac{4}{m},$$

where in the last step we applied the universality of h (Lemma 4.1). By a union bound over the five possible values of $o' - o$ and over all witnesses x', we conclude that the voting probability of x is at most $20|Y_\ell + Z_\ell|/m \leq 20k/m \leq 1/2$. □

We are now ready to prove Lemma 6.1. We shall do it in two steps: First we bound the running time and the number of false positives, i.e. $|X \setminus (Y+Z)|$, and second the number of false negatives, i.e. $|(Y + Z) \setminus X|$.

Lemma 6.5 (Running Time of Algorithm 2). *With probability $1 - \delta/2$, Algorithm 2 outputs a set X of size $O(k)$, and its running time is $O(k \log(1/\gamma) \log(\frac{1}{\gamma\delta}))$.*

Proof. Fix any level ℓ. By Lemma 6.4 we know that the voting probability of any non-witness x is at most $1/2$. Thus, by an application of Chernoff's bound, the probability that x receives more than $3R/4$ votes over all $R = \Omega(\log L + \log(1/\delta))$ rounds is at most $2^{-\Omega(R)} \leq \delta/(12L)$ by appropriately choosing the constant in the definition of R (in the upcoming Lemma 6.6 we will see why R is even slightly larger). By Markov's inequality, we obtain that with probability $1 - \delta/(2L)$ the number of non-witness elements in M which will be inserted in X_ℓ is at most $|M|/6 \leq 3|X_{\ell+1}|/6$. By a union bound over all levels, with probability $1 - \delta/2$ we get that

$$|X_\ell| \leq k + \frac{1}{2}|X_{\ell+1}|,$$

for all $\ell \in [L]$. As initially $|X_L| \leq k$ it follows by induction that $|X_\ell| \leq (\sum_{i=0}^{\infty} 1/2^i)k = 2k$. In particular we have that $|X| = |X_0| = O(k)$, as claimed.

The total running time of the algorithm can be split into two parts: (i) the time spent on running Indyk's algorithm in Line 10, and (ii) the time needed to iterate over all elements $x \in M$ across all levels and assign them votes (Line 13). The former is $O(mLR) = O(kLR)$ (recall that Indyk's algorithm runs for sets over the universe $[m]$) and also the latter is

$$\sum_{\ell \in [L]} O(|X_\ell|R) = \sum_{\ell \in [L]} O(kR) = O(kLR).$$

Together, we obtain the desired bound on the running time $O(kLR) = O(k \log(1/\gamma) \log(\frac{1}{\gamma\delta}))$. □

Lemma 6.6 (Correctness of Algorithm 2). *With probability $1-\delta/2$, Algorithm 2 correctly outputs a set X with $|(Y+Z) \setminus X| \leq \gamma k$.*

Proof. Fix any $y \in Y, z \in Z$ and define $y_\ell = \lfloor \frac{y}{2^\ell} \rfloor$, $z_\ell = \lfloor \frac{z}{2^\ell} \rfloor$ and $x_\ell = y_\ell + z_\ell$. The first step is to prove that $x_\ell \in 2\{x_{\ell+1}\} + \{0, 1, 2\}$. Indeed, from the basic inequalities $2\lfloor a \rfloor \leq \lfloor 2a \rfloor \leq 2\lfloor a \rfloor + 1$, for all rationals a, it follows directly that

$$x_\ell - 2x_{\ell+1} = \left\lfloor \frac{y}{2^\ell} \right\rfloor + \left\lfloor \frac{z}{2^\ell} \right\rfloor - 2\left\lfloor \frac{y}{2^{\ell+1}} \right\rfloor - 2\left\lfloor \frac{z}{2^{\ell+1}} \right\rfloor \leq 2,$$

and in the same way $x_\ell - 2x_{\ell+1} \geq 0$.

Coming back to the algorithm, we claim that with probability $1 - \delta\gamma/2$, $x = y + z$ will participate in X. It suffices to show that with the claimed probability, for all levels ℓ the element x_ℓ belongs to X_ℓ. Note that trivially $x_L \in X_L$. Fix a specific level ℓ. Conditioning on $x_{\ell+1} \in X_{\ell+1}$, it will be the case that x_ℓ is inserted into $M = 2X_\ell + \{0, 1, 2\}$ in Line 7, by the fact that $x_\ell \in 2\{x_{\ell+1}\} + \{0, 1, 2\}$. Moreover, recall that x_ℓ is a witness at level ℓ and thus, by Lemma 6.3, its voting probability is at least $\frac{99}{100}$. Therefore it receives more than $3R/4$ votes and is inserted into X_ℓ with probability at least $1 - 2^{-\Omega(R)} \geq 1 - \delta\gamma/(2L)$. Taking a union bound over all levels we obtain that x is contained in X with probability $1 - \delta\gamma/2$, and hence we can apply Markov's inequality to conclude that with probability $1 - \delta/2$ it is the case that $|(Y + Z) \setminus X| \leq \gamma k$. □

This finishes the proof of Lemma 6.1. Putting together the results from the previous section (Lemma 5.8) and this section (Lemma 6.1 with $\gamma' = o(\gamma^2)$), we have established an efficient algorithm to approximate convolutions in a tiny universe:

Lemma 6.7. *Let $\log k \leq 1/\gamma \leq \mathrm{poly}(k)$ and let $1/\delta \leq \mathrm{poly}(k)$. There is an algorithm for the TinyUniv-Approx-SparseConv problem running in time*

$$O(D_{1/3}(k) + k \log(1/\gamma) \log(\tfrac{1}{\gamma\delta}) + \mathrm{polylog}(k, \|A\|_\infty, \|B\|_\infty)).$$

REFERENCES

[1] Gaussian smoothing. https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.
[2] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.
[3] Peyman Afshani, Casper B. Freksen, Lior Kamma, and Kasper G. Larsen. Lower bounds for multiplication via network coding. In *Proceedings of the 46th International Colloquium Automata, Languages, and Programming*, ICALP '19, pages 10:1–10:12. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.
[4] Amihood Amir, Ayelet Butman, and Ely Porat. On the relationship between histogram indexing and block-mass indexing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372, 2014.
[5] Amihood Amir, Oren Kapah, and Ely Porat. Deterministic length reduction: Fast convolution in sparse data and applications. In *Proceedings of the 18th Symposium on Combinatorial Pattern Matching*, CPM '07, pages 183–194. Springer, 2007.
[6] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.
[7] Andrew Arnold and Daniel S. Roche. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *Proceedings of the 40th International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, pages 29–36. ACM, 2015.
[8] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th ACM Symposium on Theory of Computing*, STOC '18, pages 1269–1282. ACM, 2018.
[9] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, STOC '88, pages 301–309. ACM, 1988.

[10] Leo I. Bluestein. A linear filtering approach to the computation of discrete Fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.

[11] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1073–1084. SIAM, 2017.

[12] Karl Bringmann and Vasileios Nakos. Top-k-convolution and the quest for near-linear output-sensitive subset sum. In *Proceedings of the 52nd ACM Symposium on Theory of Computing*, STOC '20, pages 982–995. ACM, 2020.

[13] David E. Cardoze and Leonard J. Schulman. Pattern matching for spatial point sets. In *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 156–165. IEEE Computer Society, 1998.

[14] Timothy M. Chan and Qizheng He. Reducing 3SUM to convolution-3SUM. In *Proceedings of the 3rd Symposium on Simplicity in Algorithms*, SOSA '20, pages 1–7. SIAM, 2020.

[15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the 47th ACM Symposium on Theory of Computing*, STOC '15, pages 31–40. ACM, 2015.

[16] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, STOC '02, pages 592–601. ACM, 2002.

[17] A. Dutt and Vladimir Rokhlin. Fast Fourier transforms for nonequispaced data. *SIAM J. Comput.*, 14(6):1368–1393, 1993.

[18] Michael J. Fischer and Michael S. Paterson. String matching and other products. *Complexity of Computation*, 7:113–125, 1974.

[19] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, S. Muthukrishnan, and Martin J. Strauss. Near-optimal sparse Fourier representations via sampling. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, STOC '02, pages 152–161. ACM, 2002.

[20] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: Optimizing time and measurements. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, STOC '10, pages 475–484. ACM, 2010.

[21] Anna C. Gilbert, S. Muthukrishnan, and Martin J. Strauss. Improved time bounds for near-optimal space Fourier representations. *Proceedings of SPIE – The International Society for Optical Engineering*, 2005.

[22] Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. L_2/L_2-foreach sparse recovery with low risk. In *Proceedings of the 40th International Colloquium Automata, Languages, and Programming*, ICALP '13, pages 461–472. Springer, 2013.

[23] Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. Essentially optimal sparse polynomial multiplication. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*, ISSAC '20, pages 202–209. ACM, 2020.

[24] Bernard Gold and Charles M. Rader. *Digital processing of signals*. Krieger, 1969.

[25] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse Fourier transform. In *Proceedings of the 44th ACM Symposium on Theory of Computing*, STOC '12, pages 563–578. ACM, 2012.

[26] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled Fourier matrices. In *Geometric aspects of functional analysis*, pages 163–179. Springer, 2017.

[27] Qiao-Long Huang. Sparse polynomial interpolation over fields with large or zero characteristic. In *Proceedings of the 44th International Symposium on Symbolic and Algebraic Computation*, ISSAC '19, pages 219–226. ACM, 2019.

[28] Piotr Indyk. Faster algorithms for string matching problems: matching the convolution bound. In *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 166–173. IEEE Computer Society, 1998.

[29] Piotr Indyk and Michael Kapralov. Sample-optimal Fourier sampling in any constant dimension. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 514–523. IEEE Computer Society, 2014.

[30] Piotr Indyk, Michael Kapralov, and Eric Price. (Nearly) sample-optimal sparse Fourier transform. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 480–499. SIAM, 2014.

[31] Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 285–294. IEEE Computer Society, 2011.

[32] Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference*, ITCS '19, pages 43:1–43:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[33] Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018.

[34] Michael Kapralov. Sparse Fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, STOC '16, pages 264–277. ACM, 2016.

[35] Michael Kapralov. Sample efficient estimation and recovery in sparse FFT via isolation on average. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 651–662. IEEE Computer Society, 2017.

[36] Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse Fourier transform. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2709–2728. SIAM, 2019.

[37] Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993.

[38] Mathias B. T. Knudsen. Linear hashing is awesome. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '16, pages 345–352. IEEE Computer Society, 2016.

[39] Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019.

[40] Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern hamming distance. In *Proceedings of the 1st Symposium on Simplicity in Algorithms*, SOSA '18, pages 10:1–10:5. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[41] Lei Li. On the arithmetic operational complexity for solving Vandermonde linear equations. *Japan Journal of Industrial and Applied Mathematics*, 17, 2000.

[42] Michael Monagan and Roman Pearce. Parallel sparse polynomial multiplication using heaps. In *Proceedings of the 34th International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 263–270. ACM, 2009.

[43] Michael Monagan and Roman Pearce. POLY: A new polynomial data structure for Maple 17. In *Computer Mathematics*, pages 325–348. Springer, 2014.

[44] Michael Monagan and Roman Pearce. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *ACM Communications in Computer Algebra*, 48(3/4):166–186, 2015.

[45] Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 70–88. SIAM, 2019.

[46] Shanmugavelayutham Muthukrishnan. New results and open problems related to non-standard stringology. In *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, CPM '95, pages 298–317. Springer, 1995.

[47] Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020.

[48] Vasileios Nakos, Zhao Song, and Zhengyu Wang. (Nearly) sample-optimal sparse Fourier transform in any dimension; RIPless and filterless. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '19, pages 1568–1577. IEEE Computer Society, 2019.

[49] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, STOC '10, pages 603–610. ACM, 2010.

[50] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM J. Comput.*, 9(2):230–250, 1980.

[51] Eric Price and Zhao Song. A robust sparse Fourier transform in the continuous setting. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '15, pages 583–600. IEEE Computer Society, 2015.

[52] Eric Price and David P. Woodruff. Applications of the Shannon-Hartley theorem to data streams and recovery. In *Proceedings of the 45th IEEE International Symposium on Information Theory*, ISIT '12, pages 2446–2450. IEEE, 2012.

[53] Daniel S. Roche. Adaptive polynomial multiplication. *Proceedings of Milestones in Computer Algebra*, pages 65–72, 2008.

[54] Daniel S. Roche. Chunky and equal-spaced polynomial multiplication. *Journal of Symbolic Computation*, 46(7):791–806, 2011.

[55] Daniel S. Roche. What can (and can't) we do with sparse polynomials? In *Proceedings of the 43rd International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, pages 25–30. ACM, 2018.

[56] Allan Steel. Multivariate polynomial rings. http://magma.maths.usyd.edu.au/magma/handbook/text/223#1924, 2018.

[57] Joris Van Der Hoeven and Grégoire Lecerf. On the complexity of multivariate blockwise polynomial multiplication. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 211–218. ACM, 2012.

[58] Joris Van Der Hoeven and Grégoire Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *Journal of Symbolic Computation*, 50:227–254, 2013.

[59] Jack K. Wolf. Decoding of Bose-Chaudhuri-Hocquenghem codes and Prony's method for curve fitting. *IEEE Transactions on Information Theory*, 13(4):608–608, 1967.

[60] Andrew Chi-Chih Yao. On the evaluation of powers. *SIAM J. Comput.*, 5(1):100–103, 1976.