Electronic Colloquium on Computational Complexity, Report No. 123 (2024)

Faster & Deterministic FPT Algorithm for Worst-Case Tensor Decomposition

Vishwas Bhargava[*] Devansh Shringi[†]

July 22, 2024

Abstract

We present a deterministic $2^{k^{\mathcal{O}(1)}}\text{poly}(n,d)$ time algorithm for decomposing d-dimensional, width-n tensors of rank at most k over $\mathbb{R}$ and $\mathbb{C}$. This improves upon the previous randomized algorithm of Peleg, Shpilka, and Volk (ITCS '24) that takes $2^{k^{k^{\mathcal{O}(k)}}}\text{poly}(n,d)$ time and the deterministic n^{k^k} time algorithms of Bhargava, Saraf, and Volkovich (STOC '21).

Our work resolves an open question asked by Peleg, Shpilka, and Volk (ITCS '24) on whether a *deterministic* FPT algorithm exists for worst-case tensor decomposition. We also make substantial progress on the fundamental problem of how the tractability of tensor decomposition varies as the tensor rank increases. Our result implies that we can achieve deterministic polynomial-time decomposition as long as the rank of the tensor is at most $(\log n)^{1/C}$, where C is some fixed constant independent of n and d. Further, we note that there cannot exist a polynomial-time algorithm for $k = \Omega(\log n)$, unless ETH fails. Our algorithm works for all fields; however, the time complexity worsens to $2^{k^{k^{\mathcal{O}(1)}}}$ and requires randomization for finite fields of large characteristics. Both conditions are provably necessary unless there are improvements in the state of the art for system solving over the corresponding fields.

Our approach achieves this by designing a proper learning (reconstruction) algorithm for set-multilinear depth-3 arithmetic circuits. On a technical note, we design a "partial" clustering algorithm for set-multilinear depth-3 arithmetic circuits that lets us isolate a cluster from any set-multilinear depth-3 circuit while preserving the structure of the circuit.

1 Introduction

Tensors, higher-dimensional analogues of matrices, are multi-dimensional arrays with entries from a field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$. The notion of tensor rank and tensor decomposition is one of the most important tools in modern science.

[*]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. Email: vishwas1384@gmail.com.

[†]Department of Computer Science, University of Toronto, Toronto, Canada. Email: devansh@cs.toronto.edu .

ISSN 1433-8092

Tensor rank and decomposition have become fundamental tools in various branches of science, with applications in machine learning, statistics, signal processing, computational complexity. Even, within machine learning theory tensor decomposition algorithms are leveraged to give efficient algorithms for phylogenetic reconstruction [MR05], topic modeling [AFH$^+$12], community detection [AGHK14], independent component analysis [MW18], and learning various mixture models [HK13, JO14]. For more details on the applications of tensor decomposition and the rich theory of tensors in general, we refer the reader to the detailed monograph by Landsberg [Lan12] and the references therein.

We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\dots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$. The *rank* of a tensor $\mathcal{T}$ is defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here, $\otimes$ denotes the Kronecker (outer) product, also known as the *tensor product*. The expression of $\mathcal{T}$ as a sum of such rank-1 tensors over the field $\mathbb{F}$ is called $\mathbb{F}$-*tensor decomposition*, or simply tensor decomposition.

Tensor decomposition is a notoriously [1] difficult problem. Håstad [Hås90] who showed that determining the tensor rank is an NP-hard over $\mathbb{Q}$ and NP-complete over finite fields. The follow up work of Schafer and Stefankovic [SS16] further improved our understanding by giving a very tight reduction from algebraic system solving. Formally, they show that for *any* field $\mathbb{F}$, given a system S of algebraic equations over $\mathbb{F}$, we can in polynomial time construct a 3-dimensional tensor τ_S and an integer k such that S has a solution in $\mathbb{F}$ *iff* τ_S has rank at most k over $\mathbb{F}$.

Despite the known hardness results, tensor decomposition remains widely studied due to its broad applicability. There is a rich history of tensor decomposition algorithms in the literature. Most of these algorithms are average-case (i.e., they work when the entries are sampled randomly from a distribution) or heuristic. See, for instance, [LRA93, DCC07, GVX14, BCMV14, AGJ15]. This area also includes interesting active research directions, such as making these algorithms noise-resilient and studying their smoothed analysis [BKS15, SS17, HSS19, KP20, DdL$^+$22].
On the other hand, tensor decomposition in the *worst-case* scenario has only recently started to receive attention. Firstly, the work of Bhargava, Saraf, and Volkovich [BSV21] provided an n^{k^k} time algorithm to decompose rank k tensors, showing that an efficient algorithm exists to decompose fixed-rank tensors.

The next natural step in furthering the understanding of this NP-hard problem is studying its *Fixed Parameter Tractability (FPT)*. In the FPT setting, the input instance comes with a parameter k with specific quantities (e.g., the optimum treewidth of a graph). In our case of tensor decomposition, this parameter is the tensor rank. This setting treats k as a parameter much smaller than the instance size n, thus relaxing the required runtime of the algorithm from $n^{\mathcal{O}(1)}$ to $f(k) \cdot n^{\mathcal{O}(1)}$, for any computable function f. The class FPT comprises parameterized problems that admit an algorithm with this running time.

Peleg, Shpilka, and Volk [PSV24] designed a randomized FPT algorithm for decomposing rank k tensors. Specifically, they provided a randomized $k^{k^{k^{\mathcal{O}(k)}}}$ poly(n, d) time algorithm for decomposing

[1]The seminal work of Lim and Hiller [HL13] aptly justifies the use of this adjective.

rank k tensors. As a direct consequence, they demonstrated that tensors can be decomposed in polynomial time even beyond constant tensor rank up to $\mathcal{O}(\log\log\log n/\log\log\log\log n)$.

Both these works, use a standard connection between tensor decomposition and decomposing special depth-3 arithmetic circuits. We start by describing this connection below.
For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$ consider the following polynomial

$$f_{\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d)\in[n_1]\times\cdots\times[n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2}\cdots x_{d,j_d}.$$

Let $C(X) = \sum_{i=1}^{k}\prod_{j=1}^{d}\ell_{i,j}(X_j)$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ respecting the partition $\sqcup_{j\in[d]}X_j$ (denoted by $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$), and computing $f_{\mathcal{T}}(X)$. Then observe that

$$\mathcal{T} = \sum_{i=1}^{k}\mathbf{v}(\ell_{i,1})\otimes\cdots\otimes\mathbf{v}(\ell_{i,d})$$

where $\mathbf{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$. Indeed, it is easy to see that a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(r)$ circuit. Therefore, the rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit. We say that a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit has *width* w, if $|X_j| \le w$ for all j.

Keeping this connection in mind, we get that if we can *learn* the optimal representation of f_τ as an $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit, that will directly give an optimal tensor decomposition of τ as well. The problem of learning a circuit representation of a polynomial from black-box (a.k.a. oracle/membership query) access to the polynomial is referred to as arithmetic circuit reconstruction. This problem is the algebraic analogue of exact learning in Boolean circuit complexity [Ang88]. In the past few years, much attention has focused on reconstruction algorithms for various interesting subclasses of arithmetic circuits, both in the worst-case setting [BBB+00, KS01, KS06, FS12a, BSV20] as well as in the average-case setting [GKL11, GKQ14, KNST17, KNS18, KS19, GKS20a, BGKS22]. Given the natural expressibility of polynomials (and algebraic circuits), arithmetic circuit reconstruction links and connects to many other learning problems. It is not just restricted to tensor decomposition but also extends to other fundamental learning problems like learning mixtures of Gaussians and subspace clustering [JLV23, GKS20b, CGK+24]. It is a very interesting research direction to find other such connections.

Returning to the aforementioned connection, we now focus on learning these special depth-three circuits. A *set-multilinear* depth-3 circuit with respect to a partition X and top fan-in k, denoted by $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$, computes a set-multilinear polynomial of the form:

$$C(X) = \sum_{i=1}^{k}\prod_{j=1}^{d}\ell_{i,j}(X_j),$$

where $\ell_{i,j}(X_j)$ is a linear form in $\mathbb{F}[X_j]$. A polynomial $P \in \mathbb{F}[X]$ is called *set-multilinear* with respect to the partition X if every monomial that appears in P is of the form $x_{i_1}x_{i_2}\cdots x_{i_d}$, where $x_{i_j} \in X_j$. The relation between tensors and depth-three set-multilinear circuits is discussed in detail in Section 4.2. We assume black-box (or oracle/membership query) access to the polynomial f rather than access to the full tensor τ. Simply reading the entries of the tensor would require n^d time. However, in the low-rank case, we can learn the decomposition with far fewer measurements. This approach, known as studying or decomposing tensors using evaluations of f or "measurements" of the tensor τ, has been extensively studied in the literature [FS12b, CM20].

Non-uniqueness of tensor decomposition: One reason for the hardness of tensor decomposition is its non-uniqueness. This is evident even in the average-case setting due to the limitations of any tensor decomposition algorithm beyond Kruskal's uniqueness bounds [LP23, Wei23]. Naturally, in the worst-case setting, things become even more convoluted.

$$\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}1\\0\\0\end{pmatrix} - 5\begin{pmatrix}1\\1\end{pmatrix}\otimes\begin{pmatrix}1\\1\end{pmatrix}\otimes\begin{pmatrix}1\\1\\1\end{pmatrix} + 10\begin{pmatrix}1\\2\end{pmatrix}\otimes\begin{pmatrix}1\\2\end{pmatrix}\otimes\begin{pmatrix}1\\2\\4\end{pmatrix}$$
$$= 10\begin{pmatrix}1\\3\end{pmatrix}\otimes\begin{pmatrix}1\\3\end{pmatrix}\otimes\begin{pmatrix}1\\3\\9\end{pmatrix} - 5\begin{pmatrix}1\\4\end{pmatrix}\otimes\begin{pmatrix}1\\4\end{pmatrix}\otimes\begin{pmatrix}1\\4\\16\end{pmatrix} + \begin{pmatrix}1\\5\end{pmatrix}\otimes\begin{pmatrix}1\\5\end{pmatrix}\otimes\begin{pmatrix}1\\5\\25\end{pmatrix}.$$

Figure 1: An example of non-uniqueness of tensor decomposiytion by Derksen [Der13].

Fortunately, thanks to strong structural results on polynomial identities of depth-3 circuits, we can obtain strong *almost*-uniqueness results for such circuit representations. These are structural results for identically zero $\Sigma\Pi\Sigma(k)$ circuits, and essentially show that under some mild conditions, any $\Sigma\Pi\Sigma(k)$ circuit which computes the identically zero polynomial must have its linear forms contained in a “low-dimensional” space. And, in the case of set-multilinear depth-3 circuits it implies that the circuit is of “low-degree” after stripping the linear factors. Observe that two different ways of writing a polynomial constitute an elaborate polynomial identity. This directly yields that two $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ decompositions of f_τ will differ on only poly(k)-linear forms. For more details into these structural results for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ decompositions, we refer the reader to check out [BSV21, Section 3.4.2].

2 Our Results

As described in the previous section, the problem of worst-case tensor decomposition, or equivalently, proper learning of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits, was studied in [BSV21], where the authors presented a deterministic algorithm with a running time of $\text{poly}(d^{k^3}, k^{k^{k^{10}}}, n)$. This algorithm runs in polynomial time when k is a constant. This was extended beyond constant rank in the recent work of [PSV24], where they proposed a randomized algorithm with a running time of $k^{k^{k^{\mathcal{O}(k)}}} \cdot \text{poly}(n, d)$.

This algorithm runs in randomized polynomial time for tensor rank $k = \mathcal{O}\left(\frac{\log\log\log n}{\log\log\log\log n}\right)$.

In [PSV24, Section 1.4], two open problems were posed: Firstly, to understand how the tractability of tensor decomposition changes as tensor rank increases; and secondly, whether a deterministic fixed-parameter tractable (FPT) algorithm exists for worst-case tensor decomposition.

We make significant progress in answering the first question and completely resolve the second.

We now state our result for learning set-multilinear $\Sigma\Pi\Sigma(k)$ arithmetic circuits.

Theorem 2.1 (Learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits)**.** *Given blackbox access to degree d, n variate polynomial f computable by a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C with top fan-in k over $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, then there exists a* deterministic *algorithm that outputs a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$ with top fan-in k computing f in time $F(k, n, d) = 2^{k^{\mathcal{O}(1)}} \cdot \mathrm{poly}(n, d)$.*

As a direct consequence of the above theorem, we get the following corollary.

Corollary 2.2 (Decomposing rank k tensors)**.** *Let $\mathcal{T} \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ be a d-dimensional tensor of rank at most k with $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. Let $n = \sum_{i=1}^{d} n_i$. Given black-box access to measurements of $\mathcal{T}$ (equivalently to evaluations of $f_{\mathcal{T}}$), there exists a* deterministic $\mathrm{poly}(2^{k^{\mathcal{O}(1)}}, d, n)$ *time algorithm for computing a decomposition of $\mathcal{T}$ as a sum of at most k rank 1 tensors.*

Note that just reading the entries of the tensor will require n^d time. However, the above corollary states that in the low-rank case, we can learn the decomposition with much fewer measurements of the tensor.

As a direct consequence of the above result, we can perform tensor decomposition for any tensor with tensor rank $k = \mathcal{O}((\log n)^{1/C})$ in $\mathrm{poly}(n, d)$ time for a fixed constant C. This substantially improves the previously known tensor rank bound of $k = \mathcal{O}\left(\frac{\log\log\log n}{\log\log\log\log n}\right)$.

Remark 2.3. *On the intractability/hardness side, using Håstad's [Hås90] tight reduction between tensor rank and SAT, we see that even testing tensor rank will require $2^{\Omega(k)}$ time assuming the Exponential Time Hypothesis (ETH, [IP01]). This demonstrates that our time complexity is somewhat tight (with respect to the parameter k). To see this, we first note that Håstad ([Hås90, Lemma 2]) converts a 3-SAT instance with n variables and m clauses to a tensor $\tau \in \mathbb{F}^{(2+n+2m)\times 3n \times (3n+m)}$ of size with rank $k = 4n + 2m$ if and only if the 3-SAT instance is satisfiable. Thus, the overall reduction only incurs a linear blow-up, which directly implies the hardness result assuming ETH.*

Comparison with previous works Two main works [BSV21, PSV24] have previously addressed this problem using similar approaches, and below we describe why exactly the approaches in these two works fail to achieve the required deterministic running time of $2^{k^{\mathcal{O}(1)}}$.

- In [BSV21], the authors develop a way of computing almost all of the linear forms in the circuit C. After this, they have to brute-force search over all subsets of the variable partition $\{\sqcup_j X_j\}$ of size $\mathcal{O}(k)$ to obtain linear forms from each of the gates. This fails the required running time for an FPT algorithm as it adds a $d^{\mathcal{O}(k)}$ factor to the running time.

- In [PSV24], the authors focus on the reconstruction of multilinear circuits and consequently provide a reconstruction algorithm for set-multilinear circuits. Their approach hinges on obtaining black-box access to a cluster of multilinear gates such that the cluster representation is unique. The benefit of this approach is that, thanks to the uniqueness, we can aim to obtain black-box access to a cluster of gates. However, even after removing the linear factors from these clusters, the degree can be as high as $k^{k^{\mathcal{O}(k)}}$. To learn this linear factor-free part of the cluster (referred to as the simple part), they have to solve a system of polynomial equations with $k^{k^{\mathcal{O}(k)}}$ variables, which takes time $k^{k^{k^{\mathcal{O}(k)}}}$ over fields $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. Although fixed-parameter tractable (FPT), this leads to a very inefficient exponential tower dependence on k. Another important distinction is that the algorithm in [PSV24] is *randomized*, placing tensor rank in randomized FPT time. As observed in [PSV24, Remark 7.3], in the process of getting black-box access to a cluster, the authors have to do polynomial identity testing of a degree $n^{k^{k^{\mathcal{O}(k)}}}$ polynomial, which is well beyond current PIT techniques to do this in $T(k) \cdot \mathrm{poly}(n)$ time deterministically. Therefore, there is no easy derandomization of their algorithm.

We summarize the comparison in the table below.

Results	Algorithm Type	Running Time	Key Bottleneck
[BSV21]	Deterministic	$\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n)$	Brute-force search in $\binom{d}{\mathcal{O}(k)}$ sized list
[PSV24]	Randomized	$k^{k^{k^{\mathcal{O}(k)}}} \cdot \mathrm{poly}(n,d)$	System solving with $k^{k^{\mathcal{O}(k)}}$ unknowns
This Work	Deterministic	$2^{k^{\mathcal{O}(1)}} \cdot \mathrm{poly}(n,d)$	System solving with $k^{\mathcal{O}(1)}$ unknowns

Table 1: Comparison of our results to previous work

Can we improve the time complexity further to $\mathbf{2^{\mathcal{O}(k)}\mathrm{poly}(n,d)}$? Our approach is based on using "uniqueness" results (or rank bounds in the algebraic complexity literature) of tensor decomposition, which requires $d = \Omega(k)$. If we use polynomial system solving to learn this low-degree set-multilinear circuit, it will create a system of equations in $\Omega(k^2)$ variables. With these parameters, the currently known best techniques for solving a system of polynomial equations will require a $2^{k^{\mathcal{O}(1)}}$ running time, making it difficult to directly improve the running time to $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(n,d)$. An improvement to $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(n,d)$ would either require a new approach to learning low-degree circuits or a detailed study of these algebraic systems of equations to achieve better time complexity in finding their solutions. However, we cannot rule out the possibility of the *decision* version of this problem being solved in $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(n)$ time. We leave this as an open problem.

Over other fields: Our algorithm works for all fields; however, the time complexity worsens to $2^{k^{k^{\mathcal{O}(1)}}}$ and requires randomization for finite fields of large characteristics. Both conditions are provably necessary, as we will discuss now. Let's start by stating our result in generality for all fields.

Theorem 2.4 (Learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ over general field $\mathbb{F}$)**.** *Given blackbox access to a degree d, n-variate polynomial f computable by a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C over $\mathbb{F}$, there exists an algorithm that outputs a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f over a $\mathrm{poly}(k^{k^{\mathcal{O}(1)}})$ degree extension of $\mathbb{F}$ in time $F(k,n,d) = 2^{k^{k^{\mathcal{O}(1)}}} \cdot \mathrm{poly}(n,d) \cdot c_{\mathbb{F}}(2^{k^{\mathcal{O}(1)}})$.*

Here, $c_{\mathbb{F}}(N)$ denotes the time complexity of factorizing a univariate polynomial of degree N over $\mathbb{F}$.

The reason for the different time complexity over different fields lies in the difficulty of system solving over these fields. Our algorithm uses as a subroutine an algorithm for solving algebraic systems with $k^{O(1)}$ unknowns. The above result follows directly from using the best algorithm for algebraic system solving over any field $\mathbb{F}$. This sensitivity to the underlying field is natural in tensor decomposition, as different fields can significantly affect the computational complexity. For example, over the rationals ($\mathbb{Q}$), determining the exact tensor rank—even for constant tensor rank—is believed to be undecidable [Shi16, SS16].

In fact, a tight reduction in [SS16] solidifies this connection, demonstrating a tight reduction between the feasibility of algebraic systems over $\mathbb{F}$ and computing tensor rank over $\mathbb{F}$:

Theorem 2.5 ([SS16])**.** *For any field $\mathbb{F}$, given a system of m algebraic equations S over $\mathbb{F}$, we can construct in polynomial time a 3-dimensional tensor $\mathcal{T}_S$ of shape $[3m] \times [3m] \times [n+1]$ and an integer $k = 2m+n$ such that S has a solution in $\mathbb{F}$ if and only if $\mathcal{T}_S$ has rank at most $2m+n$ over $\mathbb{F}$.*

Even restricting ourselves to finite fields, the time complexity of Theorem 2.4 depends on the efficiency of univariate polynomial factorization over the field. Efficient randomized algorithms exist for this task, but derandomizing them remains a notoriously difficult problem, as noted in [AM94, Problem 15].

Interestingly, the hardness of derandomizing univariate polynomial factorization over $\mathbb{F}$ directly impacts tensor decomposition, as observed by Volkovich [Vol16]. Specifically, there is no known way to derandomize Theorem 2.4 over finite fields unless we can derandomize the factorization of univariate (even quadratic) polynomials over finite fields.

Theorem 2.6 ([Vol16, Theorem 5])**.** *Let $k \in \mathbb{N}$. Suppose the class of (set)-multilinear $\Sigma\Pi\Sigma(2)$ or $\Sigma\Pi\Sigma\Pi(2)$ circuits over the field $\mathbb{F}$ is exactly learnable, with hypotheses being multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits of polynomial size. Then, in time polynomial in $\log|\mathbb{F}|$, the exact learning algorithm can be deterministically converted into a square root oracle over $\mathbb{F}$.*

3 Proof Idea

As the introduction mentions, we will use the connection between tensor rank and set multilinear depth-3 circuits. That is for our purpose we assume that we are given black-box access to f_τ.

Through some preprocessing, we can assume the following:

- We know k, the rank of the tensor or equivalently the minimum k such that f_τ is computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$. Indeed, we can assume the value of k and output the first k for which the learning algorithm works, since we can always test if our output is correct deterministically. This affects the time complexity by a multiple of k.

- We can strip off f_τ with any linear factors ($Lin(f_\tau) := \{\ell : \ell | f_\tau\}$), since any optimal decomposition(with tensor rank k) of $NonLin(f_\tau) = \frac{f_\tau}{\prod_{\ell \in Lin(f_\tau)} \ell}$ gives an optimal decomposition for f_τ as well. This comes from a result on the factoring of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits from [SV10] described in Lemma 4.13 that shows that if f_τ is computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit then any irreducible factor $NonLin(f_\tau)$ can also be computed by a product of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. It also describes how to obtain black-box access to these irreducible factors in $2^{\mathcal{O}(\log^2 k)} \cdot \text{poly}(n, d)$ time which we can

- We can assume that $|X_j| \leq k$. This follows from the width reduction step, see [BSV21, Section 5.1] and Lemma 4.15. In the low-degree reconstruction in Lemma 4.16, the system of polynomial equations has kwd variables, which go into the exponent in the running time required to find a solution. Therefore we must be able to reduce the width to k, so we can obtain a FPT algorithm.

- Note that there can be multiple optimal $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ decompositions for f_τ, but for the sake of argument, we will fix one representation C and argue the proof using this fixed representation C.

Low Degree Reconstruction using system solving : If $d < k^5$, then we can simply set a system of a $k^{\mathcal{O}(1)}$-variate algebraic system. Indeed if the degree is small, the number of monomials appearing in f is small, and the total number of variables appearing in f is small. One can invoke black-box reconstruction algorithms for sparse polynomials [KS01, BOT88] to learn f as a sum of monomials. Then, keeping the coefficients of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ representation as unknowns we set up a system of polynomial equations in $\text{poly}(k)$ variables such that every solution to the system corresponds to a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

Our Computational budget : Note that for low-degree learning (of degree $k^{\mathcal{O}(1)}$), we require at least $k^{k^{\mathcal{O}(1)}}$ time to learn the low-degree gates, as it requires solving a system of equations with at least $k^{\mathcal{O}(1)}$ variables. Our goal is to ensure that all algebraic manipulations necessary to learn the full circuit fit within this time budget. It is unclear whether the [KS09] and [PSV24]-style clustering as the learning approach that needs solving a system of polynomial equations in $k^{k^{\mathcal{O}(k)}}$

variables and therefore $k^{k^{k^{k^{\mathcal{O}(k)}}}}$ time can be performed within this time constraint. Therefore, we combine ideas from both [BSV21] and [PSV24] to adopt partial clustering—just enough to isolate a gate (or its corresponding cluster). Subsequently, we can subtract this cluster and reduce it to a top $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ circuit, which we can then learn by induction.

Learning almost all the gates of the circuit: We adopt the same approach as in [BSV21, Sections 5.3, Section 5.4] to learn almost all the gates of the circuit. The exact result we need is described in Lemma 6.1. The idea is to use *almost*-uniqueness for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits twice. Firstly, we will project the circuit to a low degree (approximately k^2) and then learn this projection using low-degree reconstruction. Note that the representation we will learn will have some linear forms that are the same as the original representation C, by *almost*-uniqueness. In fact, we can ensure to get two distinct linear forms supported on the same variable set.

In the next step, we will use these linear forms to learn most of the linear forms of C. Once we learn ℓ_1 and ℓ_2 appearing in C, we try to learn more linear forms as follows. The algorithm applies a suitable setting of the variables of ℓ_1 in the polynomial f that makes ℓ_1 evaluate to 0, resulting in a circuit with fewer than k multiplication gates. Call the restricted polynomial f_{R_1} and let $C_{f_{R_1}}$ be the restricted version of C_f. By the inductive hypothesis, we can learn an $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of f_{R_1}, which will be close to the original representation by *almost*-uniqueness. Repeating the same with ℓ_2, once we have this, by iterating over all ways of matching up the multiplication gates[2] and choices of overlap, we can generate a list of k gates $T'_1, \ldots, T'_k$ such that $\Delta(T_i, T'_i) < 2k$. Here, $\Delta(T_i, T'_i) := deg\left(\frac{T_i}{gcd(T_i, T'_i)}\right)$ is a measure of how many linear forms are different in T_i and T'_i. We refer to $\sum_{i\in[k]} T'_i$ as the almost circuit.

3.1 Using linear forms learned in Almost Circuit

Now, we have reconstructed the following almost circuit $C' \equiv \sum_{i\in[k]} T'_i$ such that $\Delta(T_i, T'_i) < 2k$. We know that at most k linear forms from each T'_i are incorrect in the representation we learned in C'. Therefore, there are at most k^2 incorrect linear forms, and so there are at least $d - k^2$ variable parts for which we know all linear forms in C' are correct. We call this set of partitions Consistent-VarPart(C, C').

$$\text{Consistent-VarPart}(C, C') := \{j \in [d] \mid \forall\, i \in [k]\ \ell_{i,j} \in T_i \cap T'_i\}$$

As discussed, $|\text{Consistent-VarPart}(C, C')| \geq d - k^2$. In [BSV21], the authors guessed this set, which introduced d^k-type dependence, but we will do something different. Note that, for any subset $P \subseteq [d]$ such that $|P| \geq k^2 + 1$, there would be at least one variable part in Consistent-VarPart(C, C').

For any variable part $j \in$ Consistent-VarPart(C, C'), we observe that if there exists some $i \in [k]$ such that $\ell_{i,j} \notin \text{sp}(\ell_{1,j}, \ldots, \ell_{i-1,j}, \ell_{i+1,j}, \ldots, \ell_{k,j})$, then we can reconstruct T_i exactly. To do this,

[2]This matching step involves a $k^{O(k)}$ brute-force matching, which might seem wasteful, but it fits within our computational budget, so we don't need to optimize this.

we substitute $X_j = \boldsymbol{\alpha}$ such that $\ell_{1,j}(\boldsymbol{\alpha}) = \ldots = \ell_{i-1,j}(\boldsymbol{\alpha}) = \ell_{i+1,j}(\boldsymbol{\alpha}) = \ldots = \ell_{k,j}(\boldsymbol{\alpha}) = 0$ and $\ell_{i,j}(\boldsymbol{\alpha}) \neq 0$. Now using black-box factoring on C after the substitution, we can learn $T_i|_{X_j=\boldsymbol{\alpha}}$ as all other terms vanish. Due to unique factorization, we can say that we learn the projection correctly and we can find T_i simply using $T_i = T_i|_{X_j=\boldsymbol{\alpha}} \cdot \frac{\ell'_{i,j}}{\ell'_{i,j}(\boldsymbol{\alpha})}$. Once we have learned T_i exactly, we can just subtract it from the rest of C and learn $C - T_i$ as an set-multilinear $\Sigma\Pi\Sigma(k-1)$ circuit.

What if there is no variable part for which there is some linear form not in the span of the other linear forms of that variable part in the circuit? That is, $\forall j \in [d]$, $i \in [k]$ $\ell_{i,j} \in \mathrm{sp}(\ell_{1,j}, \ldots, \ell_{i-1,j}, \ell_{i+1,j}, \ldots, \ell_{k,j})$. One could try to use the above technique iteratively decreasing the top fan-in using variable parts in Consistent-VarPart, i.e., you pick a variable part $j \in$ Consistent-VarPart(C, C') and fix the variables to a value $\boldsymbol{\alpha}_j$ such that one linear form in C' (and also in C as $j \in$ Consistent-VarPart(C, C')) is set to zero while keeping T_1 non-zero. This will decrease the fan-in by at least one, until we are left with our target gate T_1, which we can learn using the above-mentioned technique. This approach also fails, as there may be other gates that differ from T_1 on a few variable parts, none of which are in Consistent-VarPart(C, C'), and hence cannot be differentiated using just C'. To avoid this issue, we will focus on learning a *cluster* of gates, that is, multiplication gates (the T_i's in the circuit) that differ on only a few ($\mathrm{poly}(k)$) linear forms instead of learning just one multiplication gate. See Lemma 5.2 for the formal definition of clusters.

3.2 Set-Multilinear Clustering

Karnin and Shpilka [KS09] introduced the notion of clustering multiplication gates in any depth-3 circuit. Just like clustering points in space, where close points form a cluster and distant points form different clusters, clustering multiplication gates with $\Delta(T_i, T_j)$ as a distance metric ensures that gates in one cluster differ on a few linear forms, while gates in different clusters differ on substantially more linear forms. One significant benefit of studying clustered representation is that it is unique! Furthermore, if we can get black-box access to a single low simple rank cluster, then we can learn a circuit representation of it. Indeed, we can strip off the gcd among different multiplication gates in the cluster, and what is left is just a low-degree circuit. In fact, this exact approach has been used in [KS09], [BSV21], and [PSV24] for learning (multilinear) $\Sigma\Pi\Sigma$ circuits.

However, one major drawback of the multilinear clustering used in [PSV24] and [KS09] is that the rank of the simple part[3] of any cluster has an upper bound of $k^{\mathcal{O}(k)}$. When we try to learn this simple part as a low-degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit, it requires solving a system of equations in $k^{\mathcal{O}(k)}$ variables. This was one of the main culprits behind the exponential tower dependence in k in [PSV24].

We develop a *partial* cluster representation (Lemma 5.2) specifically designed for *isolating* a single cluster from an $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit. For instance, if we want to isolate a cluster containing the gate T_1, then our clustering algorithm will output a set $A \subseteq [k]$, the partial cluster containing 1, such that the degree of the simple part of the cluster $C_A := \sum_{i \in A} T_i$ is at most $k^4 + k^3$, while

[3]Resulting polynomial after stripping off the linear factors.

ensuring that the 'distance' between the isolated cluster and other gates is high enough. Formally, $\Delta(C_A, T_i) \geq k^2 + k$ for $i \notin A$.

Note that, if we can isolate the cluster C_A, that is, get black-box access to a cluster C_A, we can reconstruct it using low-degree reconstruction as the degree of the simple part of C_A is less than $2k^4$. Our clustering mechanism ensures that $\Delta(C_A, T_i) \geq k^2 + k$ for $i \notin A$. This further implies that $\Delta(C_A, T'_i) \geq k$.

A natural approach would be to use these k-variable parts and T'_i to kill all the gates not in A, while ensuring that T_1 doesn't vanish. However, we don't have any idea what these k variable parts are. Obviously, any brute force search for them will have a d^k-type dependence. Furthermore, our approach should also ensure that any projection doesn't kill T_1 or $\mathrm{sim}(C_A)$.

3.3 Good T_1-isolating Projections

We will now describe how to handle the above issues. Firstly, if we ensure that the variable parts we are using are not only in Consistent-VarPart(C, C') but also that the linear forms depending on these variable parts are linear factors of C_A, i.e., Consistent-VarPart$(C, C') \cap$ Support$(\mathit{Lin}(C_A))$. Fixing variables using variable parts from Consistent-VarPart$(C, C') \cap$ Support$(\mathit{Lin}(C_A))$ that give us C_A up to a few linear forms is what we call a *good* T_1-isolating projection as defined in Definition 7.2.

And secondly, for searching for those k-variable parts that differ from $\mathit{Lin}(C_A)$, in Section 7.1, we describe an algorithm to search for these in time $k^{\mathcal{O}(k)} \cdot \mathrm{poly}(n, d)$. We design a recursive approach that, using the structural guarantees of C', outputs a list of $(k^{\mathcal{O}(k)})$ candidates for these variable parts, with a guarantee that one of them will help us project to the cluster C_A. The reduction of the search time for these k-variable parts from d^k to $(k^{\mathcal{O}(k)})$ is the main technical contribution of our work.

We now elaborate on the structural guarantees of C' that let us do this. Since the rank of $\mathrm{sim}(C_A)$ is less than $2k^4$, the size of Support$(\mathit{Lin}(C_A))$ is at least $d - 2k^4$ and therefore $|$Consistent-VarPart$(C, C') \cap$ Support$(\mathit{Lin}(C_A))| \geq d - 2k^4 - k^2$. Therefore, our algorithm will pick $k^5 + 1$ variable parts, one of which is guaranteed to be in Consistent-VarPart$(C, C') \cap$ Support$(\mathit{Lin}(C_A))$, such that the linear forms in the gates not yet set to 0 depending on those parts in C' have a dimension of at least 2. Then, for each of these, we set a linear form not in the span of $\ell_{1,j}$ to 0 while keeping $\ell'_{1,j}$ non-zero, decreasing the top fan-in, and then recursively call the function. The distance condition in Lemma 5.2 ensures that if there is a gate not in A that has not yet been set to zero, we will be able to find such a variable part. Therefore, we have $k^{\mathcal{O}(k)}$ recursive calls, one of which will be such that the gates remaining are only in A and the variable parts fixed all belong to Consistent-VarPart$(C, C') \cap$ Support$(\mathit{Lin}(C_A))$, and thus we can learn C_A.

For the correct choice of these variable parts, we can ensure that the linear forms in T_1 depending on the part don't vanish. This ensures that any other gate in A also doesn't vanish, as it is part of their gcd (the initial representation is such that $\gcd(C_A) = \mathit{Lin}(C_A)$). Also, due to unique factorization and since all the linear forms that got fixed to constant non-zero values are linear factors of C_A, we can simply multiply back the linear forms we learned in T_1' (which will be the same as T_1 as $j \in \text{Consistent-VarPart}(C, C')$ and other gates in A as the linear forms are part of $Lin(C_A)$) for those variable parts.

Once we have learned C_A, we subtract it from C and learn the smaller top fan-in circuit. We use an efficient $2^{\mathcal{O}(\log^2 k)} \cdot \text{poly}(n)$ FPT polynomial-time PIT at the end, which ensures that the only circuit output is the correct one.

4 Notation and Preliminaries

In this section, we will define notations and develop the basic preliminaries in Algebraic complexity and reconstruction required to understand this work. An experienced reader can presumably skip to Section 5.

Notations. Let $\mathbb{N} := \{0, 1, 2, \ldots\}$ and $\mathbb{N}^+ := \{1, 2, \ldots\}$. Denote $\{1, 2, \ldots, n\}$ by $[n]$. The cardinality of a set S is denoted by $|S|$. $\mathbb{F}$ is usually used to denote the underlying field. $\mathbb{R}$ refers to the field of real numbers, and $\mathbb{C}$ refers to the field of complex numbers. Denote by $\log a$ the logarithm of a with base two.

Throughout the paper, we use uppercase letters X, Y to denote sets of variables, lowercase x_i to denote variables, $\mathbf{x}, \mathbf{y}$ to denote vectors/tuples of variables, and $\mathbf{v}$ to denote a vector/tuple of field constants. We sometimes abuse notation by referring to a circuit as a collection of multiplication "$\Pi\Sigma$" gates. For any circuit $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$, we say that the circuit is an optimal circuit computing a particular polynomial (say f) if no circuit (in that respective class) can compute f with a smaller fan-in.

Whenever we say linear forms divide a multiplication gate, we mean up to scalar multiples. For a polynomial f, $Lin(f)$ denotes the set of linear factors of f, and $NonLin(f)$ refers to $\frac{f}{\prod_{\ell \in Lin(f)} \ell}$. The time required to solve a system of m polynomial equations over a field $\mathbb{F}$ with n variables each with degree at most d is denoted by $\text{Sys}_{\mathbb{F}}(n, m, d)$. We use $sp(\ell_1, \ldots, \ell_r)$ to refer to the vector space that is the span of the linear forms $\sum_{i=1}^{r} \alpha_i \ell_i$ for $\alpha_i \in \mathbb{F}$. For a vector space $\mathbf{V}$, $dim(\mathbf{V})$ denotes the dimension of $\mathbf{V}$.

4.1 Depth-3 Circuits

In this section, we formally introduce the general model of depth-3 circuits and the specialization of set-multilinear depth-3 circuits, which is the focus of our paper.

Definition 4.1. *A depth-*3 $\Sigma\Pi\Sigma(k)$ *circuit C computes a polynomial of the form*

$$C(X) = \sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X),$$

where the $\ell_{i,j}$-s are linear functions; $\ell_{i,j}(X) = \sum\limits_{t=1}^{n} a_{i,j}^t x_t + a_{i,j}^0$ with $a_{i,j}^t \in \mathbb{F}$.
A multilinear $\Sigma\Pi\Sigma(k)$ *circuit is a* $\Sigma\Pi\Sigma(k)$ *circuit in which each T_i is a multilinear polynomial. In*

particular, each such T_i is a product of variable-disjoint linear functions.
Given a partition $X = \sqcup_{j\in[d]} X_j$ of X, a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ *circuit is a further specialization of a multilinear circuit to the case when each $\ell_{i,j}$ is a linear* form *in* $\mathbb{F}[X_j]$*. That is, each $\ell_{i,j}$ is defined over the variables in X_j and $a^0_{i,j} = 0$.*

We say that C is minimal *if no subset of the multiplication gates sums to zero. We define* $\gcd(C)$ *as the linear product of all the non-constant linear functions that belong to all the T_i-s. I.e.* $\gcd(C) = \gcd(T_1, \ldots, T_k)$*. We say that C is* simple *if* $\gcd(C) = 1$*. The simplification of C, denoted by* $\mathrm{sim}(C)$*, is defined as $C/$* $\gcd(C)$*. In other words, the circuit resulting upon the removal of all the linear functions that appears in* $\gcd(C)$*. Finally, we say that a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit has* width *w, if $|X_j| \le w$ for all j.*

Throughout the paper, we will be referring to this quantity as the *width* of a polynomial, width of a circuit, since our model is is $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits, it all essentially means the same.

Definition 4.2 (Rank of circuit)**.** *The* rank *of a circuit $C(X) = \sum\limits_{i=1}^{k} T_i(X) = \sum_{i=1}^{k}\prod_{j=1}^{d_i} \ell_{i,j}(X)$ is defined as the dimension of the space spanned by all the linear forms in the circuit $dim(sp(\{\ell_{i,j} : i \in [k], j \in [d_i]\}))$.*

Definition 4.3 (Rank of Simple part of circuit)**.** *The* rank *of a circuit $C(X) = \sum\limits_{i=1}^{k} T_i(X) = \sum_{i=1}^{k}\prod_{j=1}^{d_i} \ell_{i,j}(X)$ is defined as the rank of the simple part(obtain after removing the gcd of T_i's). We will denote the simple rank of C using $\Delta(C) = rank(sim(C))$. This also defines a distance measure between 2 circuits C_1, C_2 as $\Delta(C_1, C_2) = rank(sim(C_1 + C_2))$.*

4.2 Tensors and Set-Multilinear Depth-3 Circuits

Tensors, higher dimensional analogues of matrices, are multi-dimensional arrays with entries from some field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1\times n_2\times n_3}$ and 2-dimensional tensors simply corresponds to traditional matrices. We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$, here $[n_1]\times\cdots\times[n_d]$ refers to the shape of the tensor and n_i as length of tensor in i-th dimension. Just like any matrix has a natural definition of rank, there is an analogue for tensors as well.

The rank of a tensor $\mathcal{T}$ can be defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here $\otimes$ is the Kronecker (outer) product a.k.a *tensor product.* The expression of $\mathcal{T}$ as a sum of such rank-1 tensors, over the field $\mathbb{F}$ is called $\mathbb{F}$*-tensor decomposition* or just tensor decomposition, for short. The notion of Tensor rank/decomposition has become a fundamental tool in different branches of modern science with applications in statistics, signal processing, complexity of computation, psychometrics, linguistics and chemometrics. We refer the reader to a monograph by Landsberg [Lan12] and the references therein for more details on application of tensor decomposition.

For our application, it would be useful to think of tensors as a restricted form of multilinear polynomials that are called *set-multilinear* polynomials. To this end, let us fix the following notation throughout the paper.
Let $d \in \mathbb{N}$. We will refer to d as the *dimension*. For $j \in [d]$ let $X_j = \{ x_{j,1}, x_{j,2}, \ldots, x_{j,n_j} \}$, where $n_j = |X_j|$. Finally, let $X = \sqcup_{j\in[d]} X_j$. That is, $\{ X_j \}_{\{ j\in[d] \}}$ form a partition of X.

Definition 4.4 (Set-Multilinear polynomial)**.** *A polynomial $P \in \mathbb{F}[X]$ is called* set-multilinear *w.r.t (the partition) X, if every monomial that appears in P is of the form $x_{i_1} x_{i_2} \cdots x_{i_d}$ where $x_{i_j} \in X_j$.*

In other words, each monomial of a set-multilinear polynomial picks up exactly one variable from each part in the partition. These polynomial have been well studied in the literature [Raz13, LST24] in particular since many natural polynomials like the Determinant, the Permanent, Nisan-Wigderson and others are set-multilinear w.r.t appropriate partitions of variables. Furthermore, each tensor can be regraded as a set-multilinear polynomial.

Definition 4.5. *For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$ consider the following polynomial*

$$f_{\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d)\in[n_1]\times\cdots\times[n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d}.$$

Observe that $f_{\mathcal{T}}(X)$ is a set-multilinear polynomial w.r.t X. More interestingly, there is a direct correspondence between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ in the model of set-multilinear depth-3 circuits. We first define the model formally.

Definition 4.6 (Set-Multilinear Depth-3 Circuits)**.** *A set-multilinear depth-3 circuit w.r.t to (a partition) X with top fan-in k, denoted by $\Sigma\Pi\Sigma_{\{ \sqcup_j X_j \}}(k)$ computes a (set-multilinear) polynomial of the form*

$$C(X) \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}(X_j)$$

where $\ell_{i,j}(X_j)$ is a linear form in $\mathbb{F}[X_j]$.

To gain some intuition, suppose that $f_{\mathcal{T}}(X) = \ell_{i,1}(X_1) \cdot \ell_{i,2}(X_2) \cdots \ell_{i,d}(X_d)$ for some tensor $\mathcal{T}$. We can observe that in this case $\mathcal{T}$ is a rank-1 tensor. Extending this observation, the following provides a formal connection between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ by set-multilinear depth-3 circuits.

Observation 4.7. *Let $C(X) = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ computing $f_{\mathcal{T}}(X)$ for a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$. Then*

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_{i,1}) \otimes \cdots \otimes \bar{v}(\ell_{i,d})$$

where $\bar{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$.

Note that this connection is, in fact, a correspondence: any $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ gives a circuit over $\mathbb{F}$. This leads to the following important lemma:

Lemma 4.8. *A tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(r)$ circuit. Therefore, rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(k)$ circuit.*

Proof. The proof is straightforward. Note that, $\ell_{i,1}(X_1)\cdot\ell_{i,2}(X_2)\cdots\ell_{i,d}(X_d)$ exactly corresponds to a rank-1 tensors. Thus, C_f gives a rank k $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ and any $\mathbb{F}$-tensor decomposition gives a circuit over $\mathbb{F}$. □

4.3 Complexity of Solving a System of Polynomial Equations

Solving a system of polynomial equations is the following problem: For a field $\mathbb{F}$, we are given m polynomials $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1,\ldots,x_n]$, each of degree at most d. We want to test if there exist a solution (this is the decision version) to $f_1 = 0, f_2 = 0, \ldots, f_m = 0$ in $\mathbb{F}^n$, or find a solution if it exists (this is the search version). A straightforward reduction from 3-SAT shows that polynomial system solving is NP-hard in general. This is a fundamental problem in computational algebra, and it has received lot of attention over various fields. To mention a few, system solving is NP-complete for finite fields, in PSPACE over $\mathbb{R}$ [Can88] and in Polynomial Hierarchy (Σ_2), assuming GRH [Koi96].

Interestingly, for $\mathbb{F} = \mathbb{Q}$ system solving is not even known to be decidable! In fact, if we restrict the question to integral domains (like $\mathbb{Z}$) then the problem is undecidable. This was the well-known Hilbert's tenth problem, which asks if a given Diophantine equation has an integral solution, and was famously proved to be undecidable in the 70's, see [MR75].

In this work, we are mainly concerned with polynomial system solving when the number of variables involved is small (such as a $o(\log n)$). In this case, polynomial system solving turns out is efficient under various settings. We will use the following lemma to describe the current known complexity of solving a system of equations under various settings.

Theorem 4.9. *Let $f_1, f_2, \ldots f_m \in \mathbb{F}[x_1,\ldots,x_n]$ be n-variate polynomials of degree at most d. Then, the complexity of finding a single solution to the system $f_1(x) = 0, \ldots, f_m(x) = 0$ (if one exists) over various fields is as follows:*

1. *For all fields $\mathbb{F}$, the $\mathrm{Sys}_{\mathbb{F}}(n,m,d) = \mathrm{poly}((nmd)^{3^n}) \cdot \mathrm{c}_{\mathbb{F}}(d^{2^n})$. Here, $\mathrm{c}_{\mathbb{F}}(N)$ denotes the time complexity of factorizing a univariate polynomial of degree n over $\mathbb{F}$, randomized or deterministic. This follows from standard techniques in elimination theory, see [CLO15] for details. For a detailed sketch of the argument and a bound on the size of the extension, see [BSV21, Appendix A].*

2. *[GV88] For $\mathbb{F} = \mathbb{R}$, we have $\mathrm{Sys}_{\mathbb{F}}(n,m,d) = \mathrm{poly}((md)^{n^2})$ deterministic time. Here the authors assumed that the constants appearing in the system are integers (or rationals). Note*

that for all computational applications we can WLOG assume this by simply approximating/truncating a given real number at some number of bits.

3. *[Ier89] For $\mathbb{F} = \mathbb{C}$ (or any algebraically closed field)* $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = (mn)^{O(n)} \cdot d^{O(n^2)}$ deterministic *time.*

4.4 Hardness of computing Tensor rank.

The first step towards understanding the computational complexity was by Håstad [Hås90] who showed that determining the tensor rank is an NP-hard over $\mathbb{Q}$ and NP-complete over finite fields. A better way to understand hardness results for computing tensor rank is to study its connection to solving system of polynomial equations.

Theorem 4.10 ([SS16]). *For any field $\mathbb{F}$, given a system of m algebraic equations S over $\mathbb{F}$, we can in polynomial time construct a 3 dimension tensor $\mathcal{T}_S$ of shape $[3m] \times [3m] \times [n+1]$ and an integer $k = 2m + n$ such that S has a solution $\in \mathbb{F}$ iff $\mathcal{T}$ has rank atmost $2m + n$ over $\mathbb{F}$.*

This shows equivalence between system solving and computing tensor rank. This along with complexity of system solving (discussed in the previous section) shows that computing tensor rank is NP-complete over finite fields, over $\mathbb{R}$ it is in PSPACE [Can88] and is in the Polynomial Hierarchy (Σ_2), assuming the GRH [Koi96].

Similar, reductions also hold for integral domains (e.g. $\mathbb{Z}$) [Shi16], thus showing that computing Tensor rank is *undecidable* over $\mathbb{Z}$ and not known to be decidable over $\mathbb{Q}$. Due to the equivalence between tensor rank computation and learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits with optimal top fan-in, we get the corresponding hardness consequences for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$-circuit reconstruction as well.

Such results also hold for symmetric rank computation, see [Shi16]. Concretely, for 3-dimensional tensors of length n, Shitov showed that we can convert general tensors $\mathcal{T}$ to symmetric tensors $\mathcal{T}_{sym}$ s.t. $rank(\mathcal{T}) + 4.5(n^2 + n) = \text{symmetric-rank}(\mathcal{T}_{sym})$, thus transferring the results mentioned above for general tensors to symmetric tensors as well. Again, these hardness results along with equivalence between symmetric tensor rank computation and reconstructing optimal (w.r.t top fan-in) $\Sigma\wedge\Sigma$ circuits implies that proper learning (with optimal top-fan-in) for $\Sigma\wedge\Sigma$ circuits is as hard as polynomial system solving. In particular, it is NP-hard for most fields and maybe even undecidable over $\mathbb{Q}$.

4.5 Polynomial Identity Testing

A crucial ingredient of our *deterministic* FPT algorithm for tensor decomposition is the fast deterministic PIT algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits. Although most deterministic PIT results for subclasses of depth-3 circuits have $n^{f(k)}$-type time complexity, we will instead rely on a recent improved algorithm by Guo and Gurjar [GG20]. We would like to remark that [PSV24] also used the same strategy to avoid $n^{f(k)}$-type dependence in their running time.

Below we state the theorem of [GG20] specialized to the class of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits. The original statement is phrased in terms of any-order ROABP, but with standard techniques (see, for instance, [PSV24, Corollary 2.10]), we can replace it with $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits.

Theorem 4.11 ([GG20, Theorem 3])**.** *Let $\mathcal{C}$ be the family of set-multilinear polynomials $f \in \mathbb{F}[X]$ computed by $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ such that $\max |X_i| \leq r$.*

1. *If $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n^4$, then there exists an explicit hitting-set for $\mathcal{C}$ of size $\mathrm{poly}(n, k^{\log\log k})$. In particular, the hitting-set has size $\mathrm{poly}(n)$ for $r = 2^{\mathcal{O}(\log(n)/\log\log(n))}$.*
2. *In arbitrary characteristic, there exists an explicit hitting-set for $\mathcal{C}$ of size*

$$\mathrm{poly}\left(k^{\log\log k}, n^{1+\frac{\log\log k}{\max\{1, \log\log(n) - \log\log k\}}}\right).$$

For the purpose of our application and simplicity of presentation, we will work with a weaker statement which works for all fields and have $f(k)\cdot\mathrm{poly}(n,d)$ time complexity. The below corollary follows immediately from above theorem after realizing that

Corollary 4.12. *Over any field $\mathbb{F}$, there's a deterministic polynomial time $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$ for the class of set-multilinear polynomials computed by depth-3 set-multilinear circuits of degree d and top fan-in k.*

Proof. If $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n^4$, then the result follows directly.

For all other cases, note that

$$\max\{1, \log\log(n) - \log\log k\} = \log\log(n) - \log\log k \quad \text{as long as } n > k^2.$$

Case 1 When $k < 2^{\log^{0.5} n}$, we have

$$n^{\frac{\log\log k}{\max\{1, \log\log(n) - \log\log k\}}} = \mathrm{poly}(n).$$

Case 2 When $k \geq 2^{\log^{0.5} n}$, then,

$$n^{\log\log k} \leq 2^{\mathcal{O}(\log^2 k)}.$$

Thus, in all cases, the running time of the algorithm mentioned in Theorem 4.11 is bounded by $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$. ☐

4.6 Factoring : structural and algorithmic results

Another crucial ingredient of our learning algorithm is studying the factors of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits, and designing efficient deterministic algorithms for reconstructing arithmetic circuits for these factors.

In their work [SV10], showed that for any class $\mathcal{C}$ of multilinear polynomials, one can derandomize polynomial factoring using PIT algorithms for $\mathcal{C}$. Using that fact directly with the PIT algorithm in previous subsection we get that.

Lemma 4.13 ([SV10, Corollary 1.2])**.** *There is a deterministic algorithm that given a black-box access to a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C, outputs black-boxes for the irreducible factors of C, in time $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$. In addition, each such irreducible factor is computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit.*

As a corollary, we can efficiently simulate a black-box access to $\mathrm{sim}(C)$ given a black-box access to C. The main observation is that a linear function can be learnt and tested for linearity in time $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$.

Corollary 4.14. *There is a deterministic algorithm that given a black-box access to a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C outputs linear functions $L_1, \ldots, L_r$ and black-box access to a simple set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit $\hat{C}$ such that $C = \prod_{i=1}^{r} L_i \cdot \hat{C}$, in time $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$.*

4.7 Width Reduction for set-multilinear $\Sigma\Pi\Sigma(k)$

In [BSV21], the authors presented an algorithm for learning set-multilinear $\Sigma\Pi\Sigma(k)$ circuits of arbitrary width w in roughly the same amount of time it takes to learn set-multilinear $\Sigma\Pi\Sigma(k)$ circuits of width k.

The algorithm is a direct consequence of the fact that if f is computable by an set-multilinear $\Sigma\Pi\Sigma(k)$ circuit, then each variable part is just k-variate up to an invertible linear transformation. Finding such an invertible linear transformation deterministically requires additional ideas from PIT. The following lemma follows directly from the result of [BSV21, Corollary 5.3] and the fast PIT algorithm of [GG20].

Lemma 4.15 ([BSV21, Corollary 5.3])**.** *Suppose A is an algorithm that has the following behavior. On input black-box access to degree d, n-variate polynomial $f \in \mathbb{F}[X]$ such that f is computable by a width k $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, runs in deterministic time $\mathcal{A}(n,d,k)$ and outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. Then there is another algorithm A' that has the following behavior. On input black-box access to degree d, n-variate polynomial $f \in \mathbb{F}[X]$ such that f is computable by an arbitrary width $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, runs in deterministic $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n,d) \cdot \mathcal{A}(n,d,k)$ time and outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f.*

We refer the reader to Section 5.1 and Section 5.6 of [BSV21] for the details.

4.8 Low Degree Reconstruction

In [BSV21], the authors gave an algorithm that reduced the reconstruction of a low-degree set-multilinear $\Sigma\Pi\Sigma(k)$ into solving a system of polynomial equations with few variables. We will be using it mainly in the case where $d \leq 2k^4$. The result is as follows

Lemma 4.16 ([BSV21], Lemma 5.5). *Given black-box access to a degree d polynomial $f \in \mathbb{F}[X]$ such that f is computable by a width w $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, there is a deterministic $\mathrm{Sys}_{\mathbb{F}}(kwd, w^d, d) + \mathrm{poly}(w, k, d)$ time algorithm that outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. Further using Lemma 4.15, we can reduce the width to k and get the running time of $\mathrm{Sys}_{\mathbb{F}}(k^2 d, k^d, d) \cdot \mathrm{poly}(n, d)$.*

Remark 4.17. *When $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, the output circuit is over the same underlying field $\mathbb{F}$. In general the output circuit might be over a $2^{k^{\mathcal{O}(1)}}$ degree algebraic extension of $\mathbb{F}$.*

5 Partial Set-Multilinear Cluster Representation

We will be clustering all gates close to the gate T_1 to create a representation with a cluster of gates containing T_1, such that the remaining gates are far from the cluster. The gates clubbed together will be represented by a set $A \subseteq [k]$, with $C_A = \sum_{i \in A} T_i$, fulfilling the properties described in Lemma 5.2.

For any set of gates $A \subseteq [k]$ of circuit C, we can only consider the initial representation of C such that $Lin(C_A)$ and the gcd of the gates in A are the same, i.e., $sim(C_A)$ has no linear factors. Such a representation will always exist for the polynomial computed by C_A with a depth 3 set-multilinear circuit and the same top fan-in, which can be inferred from Lemma 4.13.

Next, we introduce the following definition of distance that we will be using to define the cluster representation.

Definition 5.1 (Cluster Distance). *For circuit $C = T_1 + \ldots + T_k$ with cluster $A \subseteq [k]$ with the C representation such that $sim(C_A)$ has no linear factors, we define $\Delta_A(C_A, T_i)$ for $i \notin A$,i.e. the cluster distance between cluster C_A and gate T_i, as the number of variable parts on which the linear form in $Lin(C_A)$ and T_i differ. It can be represented as follows in the mathematical notation*

$$\Delta_A(C_A, T_i) = |\{j \in [d] : \ell_{1,j} \in Lin(C_A) \text{ and } \ell_{i,j} \nmid T_i\}|$$

We will use the cluster representation with the properties described in the following lemma.

Lemma 5.2. *For a polynomial f computed by set-multilinear $\Sigma\Pi\Sigma(k)$ $C = T_1 + \ldots + T_k$, there exist a set $A \subseteq [k]$ such that for $C_A = \sum_{i \in A} T_i$ and C being a representation such that $sim(C_A)$ doesn't have any linear factors, we have*

- $1 \in A$
- $\Delta(C_A) \leq k^4 + k^3$
- *For all remaining gates T_i, $Lin(C_A)$ and T_i differ on at least $k^2 + k$ variable parts, i.e.*

$$\forall i \in [k] \setminus A \ \ |\{j \in [d] : \ell_{1,j} \in Lin(C_A) \text{ and } \ell_{1,j} \nmid T_i\}| \geq k^2 + k$$

$$i.e.\ \forall i \in [k] \setminus A \ \ \Delta_A(C_A, T_i) \geq k^2 + k$$

Proof. For any $A \subseteq [k]$, we can write the circuit as follows

$$C = T_1 + \ldots + T_k = C_A + \sum_{i \notin A} T_i = \left(\prod_{j=1}^{d'} \ell_{1,j} \right) \cdot h_A(\ell_1, \ldots, \ell_r) + \sum_{i \notin A} T_i$$

such that h_A has r essential variables with no linear factors and is computable by a $\Sigma\Pi\Sigma(|A|)$ set-multilinear circuit. This comes by using the initial representation of C following Lemma 4.13.

Now, we give a constructive way to show that such a set A exists with all properties mentioned in Lemma 5.2. Consider the following Algorithm 1.

Algorithm 1 Computing Partial Cluster

```
Input: White-box access to set-multilinear $\Sigma^k\Pi\Sigma$ Circuit $C = T_1 + \ldots + T_k$
1: Set $A := \{1\}$
2: while $\exists i \in [k] \setminus A$ s.t. $\Delta_A(C_A, T_i) < k^2 + k$ do
3:     $A \leftarrow A \cup \{i\}$
4: Output $A$
```

The algorithm outputs a set $A \subseteq [k]$ such that $1 \in A$ as we start with $A = \{1\}$. In the algorithm, we keep the representation C fixed and use $Lin(C_A)$ as the set of linear factors of C_A. Once, A is fixed, we ensure that the representation of C is such that for that A, we have $sim(C_A)$ has no linear factors. As the while loop only terminates if there is no gate $i \notin A$ such that $\Delta_A(C_A, T_i) < k^2 + k$, we have that for any A output by Algorithm 1 that for any $i \in [k] \setminus A$ $\Delta_A(C_A, T_i) \geq k^2 + k$. Therefore, conditions 1 and 3 of the lemma are satisfied by every output of Algorithm 1.

In each iteration of the while loop the set $[k] \setminus A$ decreases by at least 1, and therefore, the algorithm runs for at most $k - 1$ iterations and always terminates and outputs a set A.

We can also assume without loss of generality, that if multiple gates i satisfy the condition $i \in [k] \setminus A$ s.t. $\Delta_A(C_A, T_i) < k^2 + k$, the gate added into the cluster C_A is the one with the smallest i. This ensures that every time Algorithm 1 is run, it outputs the same maximal cluster A for fixed input representation C as it adds the first gate not satisfying the distance property to the cluster, and so on.

So, we are left with showing that the output of Algorithm 1 will also satisfy condition 2 of the lemma, i.e. $rk(sim(C_A)) \leq k^4 + k^3$. To show this, we observe that whenever a gate i is added into A, the number of variable partitions that $sim(C_A)$ depends on increases by at most $k^2 + k$. Therefore, after adding at most k gates to A, the number of variable partitions that $sim(C_A)$ depends on is at most $k^3 + k^2$. Since the linear forms that depend on a particular variable partition are at most k due to set-multilinearity, the rank of the simple part is at most $k^4 + k^3$.

Therefore, the algorithm terminates and outputs an A which satisfies all the conditions of Lemma 5.2 and so such a set A must exist. □

Remark 5.3. *Note that as described in the proof of Lemma 5.2, the output cluster A of Algorithm 1 is unique for a given representation C. We will from now on assume that we will be talking about this A cluster containing T_1 with all properties described in Lemma 5.2.*

Remark 5.4. *Comparing this to the syntactic clustering result in [KS09] and semantic clustering results in [PSV24], we see that the upper bound on rank of the simple part of the cluster in Lemma 5.2 ($k^4 + k^3$) is much better than the ones obtained in previous works ($k^{k^{O(k)}}$). We can do this as we get worse distance conditions and focus on finding 1 cluster, which is why we refer to it as partial clustering. The distance condition in both* [KS09], [PSV24] *are such that for all cluster $\Delta(C_i) \leq r$ and for $i \neq j$ $\Delta(C_i, C_j) \geq \tau r$ for $\tau = k^{k^k}$ and $r = k^{k^{\mathcal{O}(k)}}$. In contrast, even though our representation has $\Delta(C_A) \leq k^4 + k^3$, the distance condition is only $\Delta(C_A, T_i) \geq \Delta(C_A) + 2(k^2 + k)$. Our reconstruction approach for set-multilinear circuits finds these weaker distance conditions enough. Also, Lemma 5.2 and the definition of distance Definition 5.1 only makes sense for a set-multilinear circuit while their clustering works for any $\Sigma\Pi\Sigma(k)$ circuit.*

6 Almost circuit, Learning most gates

Given a $\Sigma\Pi\Sigma_{\cup_j X_j}(k)$ circuit $C = T_1 + T_2 + \ldots + T_k = \sum_{i\in[k]} \prod_{j\in[d]} \ell_{i,j}$ computing polynomial f, we can use the techniques in [BSV21] to learn a circuit $C' = T_1' + \ldots + T_k'$ such that $\forall\ i \in [k]\ \Delta(T_i, T_i') < 2k$.

We briefly explain how such a C' is obtained in time $k^{\mathcal{O}(k)} \cdot Sys_{\mathbb{F}}(k^{\mathcal{O}(1)}) \cdot \mathrm{poly}(n, d)$. We will use Lemma 4.16 to do deterministic reconstruction when the degree is $d \leq k^3$ in time $Sys_{\mathbb{F}}(k^2 d, k^d, d)$. Now, similar to Lemma 5.6 in [BSV21], we set all but k^2 variable parts to some random values and reconstruct the polynomial, which will be close to C (on the variable parts that were not fixed), to obtain 2 independent linear forms $\ell_1, \ell_2 \in C$ such that they are supported on the same variable part X_i. By setting ℓ_1, ℓ_2 to 0 one at a time, and recursively calling reconstruction for set-multilinear $\Sigma\Pi\Sigma(k-1)$ circuits, we obtain representations close to C. These close representations have each multiplication gate that is close to a multiplication gate in C due to rank bounds as the gates that contain ℓ_1 are obtained when we go $\mod \ell_2$, and vice versa. As described in Lemma 5.7 of [BSV21], we obtain $S = \{M_1, \ldots, M_{|S|}\}$ of at most $2k-2$ $\Pi\Sigma$ circuits such that $\forall\ i \in [k],\ \exists\ j \in [2k-2]$ such that $\Delta(T_i, M_j) < 2k$. Working with k^{2k-2} possibilities, we get at least one circuit $C' = T_1' + \ldots + T_k'$ such that $\forall\ i \in [k]\ \Delta(T_i, T_i') < 2k$.

Lemma 6.1. *Given black-box access to an set-multilinear $\Sigma\Pi\Sigma(k)$ circuit $C = T_1 + T_2 + \ldots + T_k$ computing f, there exists an algorithm that runs in time $2k^3 \cdot \left(2F(n, d, k-1) + k^{2k-2}\right) + Sys_{\mathbb{F}}(2k^4, k^{2k^2}, 2k^2) + poly(k, d) \cdot 2^{\log^2 k}$ and outputs a list of size $k^{\mathcal{O}(k)}$ set-multilinear $\Sigma\Pi\Sigma(k)$ circuits such that one of the circuits $C' = T_1' + T_2' + \ldots + T_k'$ has the property that $\forall\ i \in [k]\ \Delta(T_i, T_i') < 2k$.*

Proof Sketch. The proof involves the following three steps:

1. Using Lemma 5.6 from [BSV21], we find $2k^3$ pairs, one of which is $(\ell_1, \ell_2) \in C$ such that they are supported on the same variable part X_i and are independent. This step runs in time $Sys_{\mathbb{F}}(2k^4, k^{2k^2}, 2k^2) + \text{poly}(k, d) \cdot 2^{\log^2 k}$.

2. Using Lemma 5.7 from [BSV21], for each pair (ℓ_1, ℓ_2), we construct in time $2F(n, d, k-1) + \text{poly}(n, k, d)$ a set $S = \{M_1, \ldots, M_{|S|}\}$ of at most $2k-2$ $\Pi\Sigma$ circuits such that $\forall\, i \in [k]$, $\exists\, j \in [2k-2]$ such that $\Delta(T_i, M_j) < 2k$.

3. For each (ℓ_1, ℓ_2) pair, we consider all possibilities for each T_i' from the set S, generating a list of size at most k^{2k-2}, one of which is the desired circuit C'.

7 Finding a *good* Projection

The focus of this section will be on learning the cluster C_A (which contains T_1). Due to our clustering algorithm [4], we know there exists a representation of $C = T_1 + \ldots + T_k = C_A + \sum_{i \notin A} T_i$ such that $\Delta(C_A) \leq k^4 + k^3$ and $\Delta_A(C_A, T_i) \geq k^2 + k$.

We will now define our *useful* variable parts by excluding from Consistent-VarPart(C, C') the partitions for which the linear forms in C_A are not part of $Lin(C_A)$. Therefore, we define

$$S(C, C') := \text{Consistent-VarPart}(C, C') \cap \text{Support}(Lin(C_A)).$$

As $\text{rk}(sim(C_A)) \leq k^4 + k^3$, we have $|S(C, C')| \geq d - k^4 - k^3 - k^2$. Our target remains to find a set of variable parts in S, such that we can fix them to some values that vanish other gates, but C_A remains non-zero. We define any such fixing of variables X_i to $\boldsymbol{\alpha}_i$, contained as tuples in a set denoted by $\mathcal{L}$, that keeps gates G (containing T_1) alive and sets the rest to 0 as a T_1-isolating projection.

Definition 7.1 (T_1**-isolating Projection**)**.** *A T_1-isolating projection is a tuple $(G, \mathcal{L})$ such that $G \subseteq [k], 1 \in G$, and $\mathcal{L}$ is a set of tuples $(j, \boldsymbol{\alpha}_j)$ such that $j \in [d]$ is a variable part and $\boldsymbol{\alpha}_j \in \mathbb{F}^{|X_j|}$ is an assignment of variables in the j-th variable part. All gates except those in G vanish after the substitution of variables according to $\mathcal{L}$, i.e.,*

$$C\Big|_{\forall (j, \boldsymbol{\alpha}_j) \in \mathcal{L}, X_j = \boldsymbol{\alpha}_j} = \left(\sum_{i \in G} T_i\right)\Bigg|_{\forall (j, \boldsymbol{\alpha}_j) \in \mathcal{L}, X_j = \boldsymbol{\alpha}_j} \neq 0$$

The goal of our computation will be getting black-box access to C_A, thus we define a *good* T_1-isolating projection which let's us compute the black-box access to C_A.

Definition 7.2 (*Good* T_1 isolating Projection)**.** *A T_1-isolating projection $(G, \mathcal{L})$ is good for circuit C', C if $G = A$ in Lemma 5.2 and for all $(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$, $j \in S(C', C)$. We refer to the unique A for circuit C as described in Remark 5.3.*

[4] We actually never run the clustering algorithm; it is just used for existential arguments.

7.1 Computing a *good* T_1-isolating projection

This subsection will describe how we can compute a *good* T_1-isolating projection using white-box access to C'. Our idea is straightforward. We know that for each $i \in [k] \setminus A$, T_i' *differs* from $Lin(C_A)$ on $k^2 + k$ variable parts. By the 'closeness' of C and C', we know that even in T_i we will have at least k variable parts that differ from $Lin(C_A)$. If we know exactly what these variable parts are for each $i \in [k] \setminus A$, and the fact that they are not the same (up to scalar multiple), this means there is an assignment that kills at least one linear form of T_i (and thus T_i itself) for $i \in [k] \setminus A$ while keeping $Lin(C_A)$ alive. However, we don't have any idea what these k variable parts are. Obviously, any brute force search for them will have a d^k-type dependence. We design a recursive approach that, using the structural guarantees of C', outputs a small ($k^{\mathcal{O}(k)}$) list of candidates for these variable parts, with a guarantee that one of them will help us project to the cluster C_A.

We now elaborate on the structural guarantees of C' that let us do this. If we pick a set of variable parts of size at most k^5 (for simplicity, in Algorithm 2, we use $k^4 + k^3 + k^2 + 1$) such that for each variable part, the span of the set of linear forms in the gates of C' in G has a dimension of at least 2, this means for each of the variable parts j in this set, we can pick at least one linear form $\ell'_{i,j}$ from T_i' such that $\ell'_{i,j}$ is not a scalar multiple of $\ell'_{1,j}$. Since $\ell'_{i,j}$ and $\ell'_{1,j}$ are linearly independent, we can find an assignment of variables $X_j = \boldsymbol{\alpha}_j$ such that $\ell'_{i,j}(\boldsymbol{\alpha}_j) = 0$ (and therefore $T_i'|_{X_j = \boldsymbol{\alpha}_j} = 0$) while keeping $\ell'_{1,j}(\boldsymbol{\alpha}_j)$ and $T_1'|_{X_j = \boldsymbol{\alpha}_j}$ non-zero. Thus, we have found a projection for C' with the top fan-in decreased by at least 1. This projection $(j, \boldsymbol{\alpha}_j)$ is added to $\mathcal{L}$ and all gates in C' that are set to zero are removed from G for the execution that continues after fixing this variable part. We do this recursively until we reach an execution level where we cannot find any variable parts for our set, which can happen if $|G| = 1$ or the remaining gates in C' have the same linear forms (up to scalar multiples) as T_1' on all the variable parts that have not been fixed to some value. After, doing this for all variable parts in the at most k^5 sized set, the algorithm adds the gates and projection that gets to a global list *GoodProjList*.

Algorithm 2 Computing List of Candidate *good* Projections

Input: Black-box access to $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit $C = T_1 + \ldots + T_k$ and white-box access to $C' = T'_1 + \ldots + T'_k$ such that $\Delta(T_i, T'_j) < 2k$

```
1: Global $GoodProjList = \phi$          %Global list where all the $T_1$-isolating projections are added
2: function $CandidateGoodProjections(parts, G, \mathcal{L})$
3:     $count = 0$
4:     for $j \in parts$ and $|G| \neq 1$ do
5:         $L'_j := \{\ell'_{i,j} : i \in G, \ell'_{i,j} | T'_i\}$          %Linear forms in $C'$ from gates in $G$ supported on $X_j$
6:         if $dim(sp(L'_j)) = 1$ then
7:             $parts \leftarrow parts \setminus \{j\}$
8:         else
9:             Pick a linear form $\ell'_{i,j}$ in $L'_j$ such that $\ell'_{i,j} \notin sp(\ell'_{1,j})$.
10:            Find $\boldsymbol{\alpha}_j \in \mathbb{F}^{|X_j|}$ such that $\ell'_{1,j}(\boldsymbol{\alpha}_j) \neq 0$ and $\ell'_{i,j}(\boldsymbol{\alpha}_j) = 0$.
11:            $G' := G \setminus \{i' : \ell'_{i',j}(\boldsymbol{\alpha}_j) = 0\}$
12:            $count = count + 1$
13:            CandidateGoodProjections($parts \setminus \{j\}, G', \mathcal{L} \cup \{(j, \boldsymbol{\alpha}_j)\}$)
14:            Break out of the loop, if $count = k^4 + k^3 + k^2 + 1$.
15:    Add $(G, \mathcal{L})$ to $GoodProjList$.
16: $CandidateGoodProjections(parts = [d], G = [k], \mathcal{L} = \phi)$
17: Output: $GoodProjList$
```

Now, to see that this set has a good T_1-isolating projection, we observe that since we picked k^5 variable parts or Algorithm 2 couldn't find k^5 variable parts with the required property. In the former case, using $|S(C, C')| \geq d - k^4 - k^3 - k^2$, there would be at least one variable part in our set that came from $S(C, C')$. We use the distance property of the clustering to show in the proof of Lemma 7.3 that if there is a gate $i \in G \setminus A$, then there is a variable part j in $S(C, C')$ on which the gate T_i (and T'_i as $j \in S(C, C')$) has an independent linear form from $\text{Lin}(C_A)$ and therefore T_1 (and T'_1). Therefore, in the latter case as well, either we pick a variable part in S or $G \subseteq A$.

We focus our attention on the path of execution where each variable part picked was from $S(C, C')$. Since all variable parts in $S(C, C')$ are also in Consistent-VarPart(C, C'), setting $T'_i|_{X_j = \boldsymbol{\alpha}_j} = 0$ using $j \in$ Consistent-VarPart(C, C') also sets $T_i|_{X_j = \boldsymbol{\alpha}_j} = 0$ as the linear forms depending on the variable part are same for C and C' as described in the definition of Consistent-VarPart(C, C'), while keeping $T'_1|_{X_j = \boldsymbol{\alpha}_j}$ nonzero keeps $T_1|_{X_j = \boldsymbol{\alpha}_j}$ nonzero. Also, as all variable parts in $S(C, C')$ are also in Support$(\text{Lin}(C_A))$ and the circuit is set-multilinear, keeping $T_1|_{X_j = \boldsymbol{\alpha}_j}$ non-zero also keeps $T_i|_{X_j = \boldsymbol{\alpha}_j}$ for all $i \in A$. Therefore, along this execution path, at each recursive level, at least one gate not in A gets set to zero, while keeping all the gates in A nonzero by fixing variable parts X_j with $j \in S(C, C')$, until $G = A$. In the execution call on the path with $G = A$, the $(G, \mathcal{L})$ that is added to GoodProjList is a good T_1-isolating Projection and occurs at a depth of at most $k - |A| \leq k - 1$.

This computation can be seen as a k^5-arity tree with depth at most $k - 1$, with each node a recursive call of the function. We start with $G = [k]$ and make at most k^5 choices of variable parts such that at least one of them is in $S(C, C')$. In the execution of the algorithm along this path, at each step, at least one gate not in A is set to zero by fixing the chosen variable part in $S(C, C')$ while keeping all gates in A non-zero, until we reach $G = A$, which adds a good T_1-isolating Projection to the list.

Lemma 7.3. *Algorithm 2 when given black-box access to a set-multilinear $\Sigma\Pi\Sigma(k)$ circuit C and white-box access to an almost circuit C' from Lemma 6.1 computes a list of T_1'-isolating projections for C' of size at most $k^{\mathcal{O}(k)}$ such that it has at least 1 good T_1-isolating projection for C, C' in deterministic $k^{\mathcal{O}(k)} \cdot \mathrm{poly}(n, d)$ time.*

Proof. We break our proof into the following 3 claims:

1. There is a path of execution of Algorithm 2 such that there is a recursive call of the function CandidateGoodProjections with $G = A$ and all variable parts included in $\mathcal{L}$ are in $S(C, C')$, i.e. $G = A$ and $\forall (j, \boldsymbol{\alpha}_j) \in \mathcal{L} \;\; j \in S$.

2. There is at least 1 good T_1-isolating projection for C, C' in the output of Algorithm 2.

3. Size of the list is $k^{\mathcal{O}(k)}$. Running time is $k^{\mathcal{O}(k)} \cdot \mathrm{poly}(n, d)$.

Proof of Claim 1: Algorithm 2 in each execution of the function makes at most $k^4 + k^3 + k^2 + 1$ recursive calls on Line 14 for different variable parts from variable partition $\{\sqcup_j X_j\}$ which are set to $\boldsymbol{\alpha}_j$ in their respective branches of execution. Recall, $|S(C, C')| \geq d - k^4 - k^3 - k^2$. Therefore, if the algorithm stopped searching for variable parts that have at least 2 independent linear forms, i.e. breaks out of the loop on Line 14, then there must be at least 1 recursive call from each execution of CandidateGoodProjections that uses a variable part from S. The other case is that the local variable *parts* did not have $k^4 + k^3 + k^2 + 1$ variable parts such that the linear forms from the gates in G spanned a dimension 2 subspace. We will argue later that in this case, if none of the variable parts chosen for the next calls are from $S(C, C')$ then $G \subseteq A$. All gates in A have the same (up to scalar factor) linear forms on all variable parts in S. This can be seen as all variable parts in S are also in the support of $Lin(C_A)$ and therefore in gcd of the gates in A. As a result of set-multilinearity, there will be no linear form supported on the variable parts in S. This ensures that when we keep $\ell_{1,j}$ non-zero, we also keep all gates in A non-zero. Let us say there is a gate T_i with $i \in G \setminus A$. Therefore, setting a linear form different from $\ell_{1,j}$ will ensure you set a gate T_i' not in A to zero which also sets a gate in T_i as the variables parts in S are also in Consistent-VarPart(C, C'). Since, we know $\Delta_A(C_A, T_i) \geq k^2 + k$, we have $Lin(C_A)$ and T_i differ on at least $k^2 + k$ variable partitions. As $|\text{Consistent-VarPart}(C, C')| \geq d - k^2$, there are at least k variable partitions in S on which $Lin(C_A)$ and T_i differ. In each recursive call, the size of G decreases by at least 1, so the depth of recursion could at most be k and $|\mathcal{L}| < k$, which means there will be at least 1 variable partition in S on which $Lin(C_A)$ and T_i differ. As the depth of the execution can at most be $k - 1$, the number of variable parts fixed will be at most $k - 1$, and there will be at least 1 variable part in S on which $Lin(C_A)$ and T_i differ. This also means that if none of the recursive calls from the execution call are in $S(C, C')$ then there is not $i \in G \setminus A$ implying $G \subseteq A$. Also, All the variables parts removed in Line 7 of Algorithm 2 are such that all gates in G agree on them with $Lin(C_A)$, so all k variable partitions will be in *parts* except those who have been added to $\mathcal{L}$.

Proof of Claim 2: We will first observe that every element in *GoodProjList* is a T_1' Isolating Projection for C' as we keep T_1' non-zero and remove all gates that get set to 0. We can guarantee to find such an $\boldsymbol{\alpha}_j$ in Step 10 because the dimension of the span of $\ell'_{1,j}, \ell'_{i,j}$ is at least 2. From Claim 1, we know there will be a path when $G = A$ and all the variable partitions in $\mathcal{L}$ are in $S(C, C')$. For this to be a good T_1 Isolating Projection, we only need to argue that this is a T_1 Isolating Projection for C. This is true as all variable partitions in S are also in *Consistent-VarPart* and therefore when we set a gate T_i' to 0 by setting $\ell'_{(i,j)}(\boldsymbol{\alpha}_j) = 0$, we also set T_i to 0 keeping C_A non-zero as $\ell'_{1,j}$ will also be in $Lin(C_A)$. Therefore, there is at least 1 good T_1-isolating Projection in the list *GoodProjList* output by Algorithm 2.

Proof of Claim 3: As the top fan-in decreases by at least 1 at each level, the recursion depth will be at most $k-1$. At each level, at most $k^4 + k^3 + k^2 + 1$ recursive calls are made, creating the recursive $(k^4 + k^3 + k^2 + 1)$-ary tree with depth $k-1$. As the number of leaves of a $(k^4 + k^3 + k^2 + 1)$-ary tree and k depth is bounded by $(k^4 + k^3 + k^2 + 1)^k$, the number of projections in *GoodProjList* is also less than $(k^4 + k^3 + k^2 + 1)^k$. Finding $\boldsymbol{\alpha}_i$ in Step 10 of Algorithm 2 can be done in poly(n) time as it is just solving a system of linear equations. Removing the variable partitions in gcd also takes poly(d) time across any path as each partition is only considered once in each computational path of the recursion tree. Therefore, each recursion call runs in poly(n, d) time and there are $(k^4 + k^3 + k^2 + 1)^k$ such calls, and therefore the entire running time is $(k^4 + k^3 + k^2 + 1)^k \cdot \text{poly}(n, d) = k^{\mathcal{O}(k)} \cdot \text{poly}(n, d)$. $\square$

8 Reconstruction using a good Projection

Now, we will describe how we can obtain a $k^{\mathcal{O}(k)}$ sized list of candidate circuits for C_A using the list computed in Lemma 7.3, one of which will be computing C_A.

Algorithm 3 Candidate Circuits using Good Partitions

Input: Black-box access to set-multilinear $\Sigma^k\Pi\Sigma$ Circuit $C = T_1 + \ldots + T_k$ and white-box access to $C' = T'_1 + \ldots + T'_k$ such that $\Delta(T_i, M_j) < 2k$ and a list containing a *good* Projection

```
1: function $Candidate\text{-}Cluster\text{-}Circuit(C, C', GoodProjList)$
2:     $Candidate\text{-}Circuits = \phi$
3:     for $(G, \mathcal{L}) \in GoodProjList$ do
4:         Let $\Phi(C) = C|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j}$ denote the circuit $C$ with values in $X_j$ set to $\boldsymbol{\alpha}_j$ for all $(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$.
5:         Use Corollary 4.14 to obtain the linear factors of $\Phi(C)$(denoted by $Lin(\Phi(C))$) and black-box access to $NonLin(\Phi(C))$
6:         Reconstruct the set-multilinear circuit $NonLin(\Phi(C)')$ using low degree (degree $\leq k^4 + k^3$) from Lemma 4.16 black-box access to $NonLin(\Phi(C))$ with top-fan-in set to $|G|$.
7:         If Reconstruction fails in case $deg(NonLin(\Phi(C))) \geq k^4 + k^3$, move to next $(G, \mathcal{L})$. Set $\Phi(C') = \left(\prod_{\ell \in Lin(\Phi(C)} \ell\right) \cdot NonLin(\Phi(C)')$
8:         Obtain $\Phi(C') = \Phi(C') \cdot \left(\prod_{(j,\boldsymbol{\alpha}_j)\in\mathcal{L}} \frac{\ell_{1,j}}{\ell_{1,j}(\boldsymbol{\alpha}_j)}\right)$
9:         Add $(\Phi(C'), |G|)$ to $Candidate\text{-}Circuits$
10:    Output $Candidate\text{-}Circuits$
```

Lemma 8.1. *Given a list of T_1-isolating Projections from Lemma 7.3, black-box access to a set-multilinear circuit C computing f with top fan-in k, and white-box access to circuit C' as in Lemma 6.1, then Algorithm 3 computes a $k^{\mathcal{O}(k)}$ sized list of depth 3 set-multilinear circuits $\Phi(C)'$ such that at least one of them computes C_A in time $k^{\mathcal{O}(k)} \cdot Sys_{\mathbb{F}}(2k^6, k^{2k^4}, 2k^4) \cdot \mathrm{poly}(n, d)$.*

Proof. From Lemma 7.3, we have that there will be at least one *good* T_1-isolating projection in the list *GoodProjList*. We consider only this T_1-isolating projection and show that when $(G, \mathcal{L})$ is a *good* T_1-isolating projection, then $\Phi(C')$ at end of the algorithm is computing C_A. When $(G, \mathcal{L})$ is a *good* T_1-isolating projection, we know from Definition 7.2 that $G = A$ and $\forall(j, \boldsymbol{\alpha}_j) \in \mathcal{L}, j \in S(C', C)$. Also, Definition 7.1 of T_1-isolating projection we know gates other than G vanish when we substitute values to variable parts according to $\mathcal{L}$, i.e. $C|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j} = \left(\sum_{i\in G} T_j\right)|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j} \neq 0$. As this is a *good* T_1- Isolating projection, we have $G = A$, which means $C|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j} = \left(\sum_{i\in A} T_j\right)|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j}$ and gives us $(C_A)|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j} = \Phi(C)$. From Lemma 5.2, we can see that C_A can be written as $C_A = \left(\prod_{j=1}^{d'} \ell_{1,j}\right) \cdot h_A(\ell_1, \ldots, \ell_r)$ with $h_A = sim(C_A)$ and therefore $r \leq rk(sim(C_A)) \leq k^4 + k^3$. From Corollary 4.14, we can get $Lin(\Phi(C))$ and black-box access to $NonLin(\Phi(C))$. As we fixed the representation such that $sim(C_A)$ has no linear factors, we have $NonLin(\Phi(C)) = sim(C_A)$. From Lemma 4.13, we know that a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit can also compute h_A. The rank of the polynomial acts like an upper bound on the degree due to set-multilinearity and therefore, we can learn it using low-degree reconstruction for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits with top fan-in $|A|$. As $\forall(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$ we have $j \in S_{C,C'}$ and so $j \in Support(Lin(C_A))$. So, when we consider $(C_A)|_{\forall(j,\boldsymbol{\alpha}_j)\in\mathcal{L}, X_j=\boldsymbol{\alpha}_j}$, it only fixes linear forms in $Lin(C_A)$ to their evaluations at $\boldsymbol{\alpha}_j$ which are not zero. Therefore, the circuit learned for $NonLin(\Phi(C))$ in the case when we are working with a good T_1-isolating projection will be equal to $sim(C_A) = h_A$. So, we have learned C_A correctly except for the linear forms that were set to constant nonzero values by Φ. These will be $(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$, $\ell_{1,j}$ which got set to $\ell_{1,j}(\boldsymbol{\alpha}_j)$. As the variable parts in $\mathcal{L}$ are in $S(C, C')$ and so also in Consistent-VarPart(C, C'), we have $\ell_{1,j} = \ell'_{1,j}$ which we already know. Therefore, we can obtain a circuit for C_A by multiplying with $\frac{\ell'_{1,j}}{\ell'_{1,j}(\boldsymbol{\alpha}_j)}$ for all $(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$.

The algorithm adds at most 1 circuit for each element of *GoodProjList* which we know from Lemma 7.3 has size at most $k^{\mathcal{O}(k)}$, and therefore, $|Candidate\text{-}Circuits| \leq k^{\mathcal{O}(k)}$. The size bound of *GoodProjList* also means that the loop on Line 3 of Algorithm 3 will also run $k^{\mathcal{O}(k)}$ times. Factoring in Line 5 takes $2^{\mathcal{O}(\log^2 k)} \cdot \text{poly}(n, d)$ time from Corollary 4.14. Line 6 which has the low degree reconstruction with fan-in at most k and degree at most $k^4 + k^3 \leq 2k^4$, which by Lemma 4.16 can be done in $\text{Sys}_{\mathbb{F}}(k^2 d, k^d, d) \cdot \text{poly}(n, d) = \text{Sys}_{\mathbb{F}}(2k^6, k^{2k^4}, 2k^4) \cdot \text{poly}(n, d)$. Line 7 takes time $\mathcal{O}(|\mathcal{L}|)$ for which we showed in Lemma 7.3 that $|\mathcal{L}| \leq k - 1$. Therefore, the entire algorithm runs in time $k^{\mathcal{O}(k)} \cdot \text{Sys}_{\mathbb{F}}(2k^6, k^{2k^4}, 2k^4) \cdot \text{poly}(n, d)$. $\square$

9 Proof of Theorem 2.1

Algorithm 4 Reconstruction of set-multilinear $\Sigma\Pi\Sigma(k)$ circuits

Input: Black-box access to set-multilinear $\Sigma^k\Pi\Sigma$ Circuit $C = T_1 + \ldots + T_k$

```
 1: function $TensorReconstruction(C, k)$
 2:     if $k = 0$ then
 3:         Output 0
 4:     else
 5:         if $k = 1$ then
 6:             Factor using Corollary 4.14 to obtain all the linear factors.
 7:             Output the circuit that is product of all factors.
 8:     Use Lemma 6.1 to obtain a list of $2k^{2k+1}$ almost circuits $L$
 9:     for $C' \in L$ do
10:         Run Algorithm 2 with $C$ and $C'$ to obtain $GoodProjList$
11:         Run Algorithm 3 with input $C, C', GoodProjList$ to obtain $Candidate\text{-}Circuits$
12:         for $(C_G, |G|) \in Candidate\text{-}Circuits$ do
13:             $\tilde{C} = TensorReconstruction(C - C_G, k - |G|)$
14:             Run blackbox PIT algorithm on $\tilde{C} + C_G - C$, if identity output $\tilde{C} + C_G$
15:     Output $Nil$
```

Proof of Theorem 2.1. The correctness of Algorithm 4 follows from Lemma 3 that outputs at least 1 circuit computing C_A which decreases the fan-in by at least 1. The blackbox PIT check ensures that either the circuit outputs the correct circuit computing C or Nil. For the correct value of k, it will output the correct circuit when $G = A$ and $C_G = C_A$. We can run Algorithm 4 for $k := \{1, 2, \ldots, d\}$ which will give us the minimum rank Tensor Decomposition.

Let $F(n, d, k)$ be the running time of Algorithm 4 for an input n-variate d degree set-multilinear polynomial computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit. From Lemma 6.1, we know we can obtain the list of $2k^{2k+1}$ almost circuits in time $2k^3 \cdot \left(2F(n, d, k-1) + k^{2k-2}\right) + \text{Sys}_{\mathbb{F}}(2k^4, k^{2k^2}, 2k^2) +$

$\mathit{poly}(k,d)\cdot 2^{\log^2 k}$. From Lemma 7.3 and Lemma 8.1, we know Algorithm 2 and Algorithm 3 run in $k^{\mathcal{O}(k)}\cdot \mathrm{poly}(n,d)$ time and $k^{\mathcal{O}(k)}\cdot \mathrm{Sys}_{\mathbb{F}}(2k^6, k^{2k^4}, 2k^4)\cdot \mathrm{poly}(n,d)$ time respectively. The loop on Line 6 runs at most $k^{\mathcal{O}(k)}$ times taking time $F(n,d,k-|A|)\le F(n,d,k-1)$ on Line 7 and $2^{\mathcal{O}(\log^2 k)}\cdot \mathrm{poly}(n,d)$ time for PIT on Line 8 due to Corollary 4.12. The base case of $F(n,d,\le 1)$ takes time $\mathrm{poly}(n,d)$. Therefore, the total running time can be described by the following recurrence relation

$$\begin{aligned}
F(n,d,k) &= 2k^3\cdot\left(2F(n,d,k-1)+k^{2k-2}\right)+\mathrm{Sys}_{\mathbb{F}}(2k^4,k^{2k^2},2k^2)+\mathit{poly}(k,d)\cdot 2^{\log^2 k}\\
&\quad + k^{\mathcal{O}(k)}\cdot(k^{\mathcal{O}(k)}\cdot\mathrm{poly}(n,d)+k^{\mathcal{O}(k)}\cdot \mathrm{Sys}_{\mathbb{F}}(2k^6,k^{2k^4},2k^4)\cdot\mathrm{poly}(n,d)\\
&\quad + k^{\mathcal{O}(k)}\cdot(F(n,d,k-1)+2^{\mathcal{O}(\log^2 k)}\cdot\mathrm{poly}(n,d)))\\
&\le k^{\mathcal{O}(k)}F(n,d,k-1)+k^{\mathcal{O}(k)}\cdot\mathrm{Sys}_{\mathbb{F}}(2k^6,k^{2k^4},2k^4)\cdot\mathrm{poly}(n,d)\\
&\le k^{\mathcal{O}(k^2)}F(n,d,1)+k^{\mathcal{O}(k)}\cdot\mathrm{Sys}_{\mathbb{F}}(2k^6,k^{2k^4},2k^4)\cdot\mathrm{poly}(n,d)\\
&= \left(k^{\mathcal{O}(k^2)}+k^{\mathcal{O}(k)}\cdot\mathrm{Sys}_{\mathbb{F}}(2k^6,k^{2k^4},2k^4)\right)\cdot\mathrm{poly}(n,d)
\end{aligned}$$

Now, as described in Theorem 4.9, we can observe that we can substitute $\mathrm{Sys}_{\mathbb{F}}(n,m,d)$ to $\mathrm{poly}((md)^{n^2})$ when $\mathbb{F}=\mathbb{R}$ or $\mathbb{C}$. This gives the running time of

$$\begin{aligned}
F(n,d,k) &= \left(k^{\mathcal{O}(k^2)}+k^{\mathcal{O}(k)}\cdot\mathrm{poly}((k^{2k^4}\cdot 2k^4)^{(2k^6)^2})\right)\cdot\mathrm{poly}(n,d) && \mathbb{F}=\mathbb{R}\text{ or }\mathbb{C}\\
&= 2^{\mathcal{O}(k^{16}\log k)}\cdot\mathrm{poly}(n,d)\\
&= 2^{k^{\mathcal{O}(1)}}\cdot\mathit{poly}\,(n,d) && \text{Deterministic, }\mathbb{F}=\mathbb{R}\text{ or }\mathbb{C}
\end{aligned}$$

□

Proof of Theorem 2.4. For all other fields $\mathbb{F}$, Theorem 4.9 gives $\mathrm{Sys}_{\mathbb{F}(n,m,d)}\mathrm{poly}((nmd)^{3^n})\cdot c_{\mathbb{F}}(d^{2^n})$ where, $c_{\mathbb{F}}(N)$ denotes the time complexity of factorizing a univariate polynomial of degree n over $\mathbb{F}$, randomized or deterministic respectively. Upon substitution into our expression, we get

$$\begin{aligned}
F(n,d,k) &= \left(k^{\mathcal{O}(k^2)}+k^{\mathcal{O}(k)}\cdot\mathrm{poly}((2k^2\cdot k^{2k^4}\cdot 2k^4)^{(2k^6)^{2k^6}})\right)\cdot\mathrm{poly}(n,d)\cdot c_{\mathbb{F}}(2^{k^{\mathcal{O}(1)}})\\
&= 2^{\mathcal{O}(k^{k^6}\log k)}\cdot\mathrm{poly}(n,d)\cdot c_{\mathbb{F}}(2^{k^{\mathcal{O}(1)}})\\
&= 2^{k^{k^{\mathcal{O}(1)}}}\cdot\mathrm{poly}\,(n,d)\cdot c_{\mathbb{F}}(2^{k^{\mathcal{O}(1)}})
\end{aligned}$$

□

References

[AFH$^+$12] Anima Anandkumar, Dean P Foster, Daniel J Hsu, Sham M Kakade, and Yi-Kai Liu. A spectral algorithm for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 917–925, 2012.

[AGHK14] Animashree Anandkumar, Rong Ge, Daniel Hsu, and Sham M Kakade. A tensor approach to learning mixed membership community models. *The Journal of Machine Learning Research*, 15(1):2239–2312, 2014.

[AGJ15] Animashree Anandkumar, Rong Ge, and Majid Janzamin. Learning overcomplete latent variable models through tensor methods. In *Conference on Learning Theory*, pages 36–112. PMLR, 2015.

[AM94] L. M. Adleman and K. S. McCurley. Open problems in number theoretic complexity, II. In *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, pages 291–322, 1994.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[BBB$^+$00] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

[BCMV14] Aditya Bhaskara, Moses Charikar, Ankur Moitra, and Aravindan Vijayaraghavan. Smoothed analysis of tensor decompositions. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 594–603, 2014.

[BGKS22] Vishwas Bhargava, Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning generalized depth three arithmetic circuits in the non-degenerate case. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

[BKS15] Boaz Barak, Jonathan A Kelner, and David Steurer. Dictionary learning and tensor decomposition via the sum-of-squares method. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 143–151, 2015.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 301–309, 1988.

[BSV20] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *Journal of the ACM (JACM)*, 67(2):1–28, 2020.

[BSV21] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Reconstruction algorithms for low-rank tensors and depth-3 multilinear circuits. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 809–822, 2021.

[Can88] J. Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467, 1988.

[CGK$^+$24] Pritam Chandra, Ankit Garg, Neeraj Kayal, Kunal Mittal, and Tanmay Sinha. Learning Arithmetic Formulas in the Presence of Noise: A General Framework and Applications to Unsupervised Learning. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[CLO15] D. A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (4. ed.)*. Undergraduate texts in mathematics. Springer, 2015.

[CM20] S. Chen and R. Meka. Learning polynomials in few relevant dimensions. In Jacob D. Abernethy and Shivani Agarwal, editors, *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 1161–1227. PMLR, 2020.

[DCC07] Lieven DeLathauwer, Josphine Castaing, and Jean-Franois Cardoso. Fourth-order cumulant-based blind identification of underdetermined mixtures. *IEEE Transactions on Signal Processing*, 55(6):2965–2973, 2007.

[DdL$^+$22] Jingqiu Ding, Tommaso d'Orsi, Chih-Hung Liu, David Steurer, and Stefan Tiegel. Fast algorithm for overcomplete order-3 tensor decomposition. In *Conference on Learning Theory*, pages 3741–3799. PMLR, 2022.

[Der13] Harm Derksen. Kruskal's uniqueness inequality is sharp. *Linear Algebra and its Applications*, 438(2):708–712, 2013.

[FS12a] M. A. Forbes and A. Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:115, 2012.

[FS12b] Michael A Forbes and Amir Shpilka. On identity testing of tensors, low-rank recovery and compressed sensing. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 163–172, 2012.

[GG20] Zeyu Guo and Rohit Gurjar. Improved explicit hitting-sets for roabps. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020.

[GKL11] A. Gupta, N. Kayal, and S. V. Lokam. Efficient reconstruction of random multilinear formulas. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 778–787, 2011.

[GKQ14] A. Gupta, N. Kayal, and Y. Qiao. Random arithmetic formulas can be reconstructed efficiently. *Computational Complexity*, 23(2):207–303, 2014.

[GKS20a] A. Garg, N. Kayal, and C. Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. *arXiv preprint arXiv:2004.06898*, 2020.

[GKS20b] Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 889–899. IEEE, 2020.

[GV88] D. Yu. Grigor'ev and N.N. Vorobjov. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation*, 5(1):37 – 64, 1988.

[GVX14] Navin Goyal, Santosh Vempala, and Ying Xiao. Fourier pca and robust tensor decomposition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 584–593, 2014.

[Hås90] Johan Håstad. Tensor rank is np-complete. *J. Algorithms*, 11(4):644–654, 1990.

[HK13] Daniel Hsu and Sham M Kakade. Learning mixtures of spherical gaussians: moment methods and spectral decompositions. arXiv preprint arXiv:1306.0021, 2013. Presented at the 4th Conference on Innovations in Theoretical Computer Science (ITCS).

[HL13] Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *J. ACM*, 60(6), nov 2013.

[HSS19] Samuel B Hopkins, Tselil Schramm, and Jonathan Shi. A robust spectral algorithm for overcomplete tensor decomposition. In *Conference on Learning Theory*, pages 1683–1722. PMLR, 2019.

[Ier89] D. Ierardi. Quantifier elimination in the theory of an algebraically-closed field. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 138–147, New York, NY, USA, 1989. Association for Computing Machinery.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[JLV23] Nathaniel Johnston, Benjamin Lovitz, and Aravindan Vijayaraghavan. Computing linear sections of varieties: quantum entanglement, tensor decompositions and beyond. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1316–1336. IEEE, 2023.

[JO14] Prateek Jain and Sewoong Oh. Learning mixtures of discrete product distributions using spectral decompositions. arXiv preprint arXiv:1404.4604, 2014. Presented at the Conference on Learning Theory (COLT).

[KNS18] N. Kayal, V. Nair, and C. Saha. Average-case linear matrix factorization and reconstruction of low width algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:29, 2018.

[KNST17] N. Kayal, V. Nair, C. Saha, and S. Tavenas. Reconstruction of full rank algebraic branching programs. In *32nd Computational Complexity Conference, CCC 2017.*, pages 21:1–21:61, 2017.

[Koi96] P. Koiran. Hilbert's nullstellensatz is in the polynomial hierarchy. *Journal of complexity*, 12(4):273–286, 1996.

[KP20] Bohdan Kivva and Aaron Potechin. Exact nuclear norm, completion and decomposition for random overcomplete tensors via degree-4 sos. *arXiv preprint arXiv:2011.09416*, 2020.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–223, 2001.

[KS06] A. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of computing*, 2(10):185–206, 2006.

[KS09] Z. S. Karnin and A. Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 274–285, 2009. Full version at http://www.cs.technion.ac.il/ shpilka/publications/KarninShpilka09.pdf.

[KS19] N. Kayal and C. Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019.*, pages 413–424, 2019.

[Lan12] J. Landsberg. Tensors: geometry and applications. *Representation theory*, 381(402):3, 2012.

[LP23] Benjamin Lovitz and Fedor Petrov. A generalization of kruskal's theorem on tensor decomposition. In *Forum of Mathematics, Sigma*, volume 11, page e27. Cambridge University Press, 2023.

[LRA93] Sue E Leurgans, Robert T Ross, and Rebecca B Abel. A decomposition for three-way arrays. *SIAM Journal on Matrix Analysis and Applications*, 14(4):1064–1083, 1993.

[LST24] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. *Communications of the ACM*, 67(2):101–108, 2024.

[MR75] Y. Matijasevič and J. Robinson. Reduction of an arbitrary diophantine equation to one in 13 unknowns. *Acta Arithmetica*, 27(1):521–553, 1975.

[MR05] Elchanan Mossel and Sébastien Roch. Learning nonsingular phylogenies and hidden markov models. arXiv preprint arXiv:1506.08512, 2005. Presented at the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC).

[MW18] Ankur Moitra and Alexander S Wein. Spectral methods from tensor networks. *arXiv preprint arXiv:1811.00944*, 2018.

[PSV24] Shir Peleg, Amir Shpilka, and Ben Lee Volk. Tensor Reconstruction Beyond Constant Rank. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 87:1–87:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Raz13] R. Raz. Tensor-rank and lower bounds for arithmetic formulas. *J. ACM*, 60(6):40:1–40:15, 2013.

[Shi16] Y. Shitov. How hard is the tensor rank? *arXiv preprint arXiv:1611.01559*, 2016.

[SS16] M. Schaefer and D. Stefankovic. The complexity of tensor rank. *CoRR*, abs/1612.04338, 2016.

[SS17] Tselil Schramm and David Steurer. Fast and robust tensor decomposition with applications to dictionary learning. In *Conference on Learning Theory*, pages 1760–1793. PMLR, 2017.

[SV10] A. Shpilka and I. Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *Automata, Languages and Programming, 37th International Colloquium (ICALP)*, pages 408–419, 2010. Full version at https://eccc.weizmann.ac.il/report/2010/036.

[Vol16] Ilya Volkovich. A guide to learning arithmetic circuits. In *Conference on Learning Theory*, pages 1540–1561. PMLR, 2016.

[Wei23] Alexander S Wein. Average-case complexity of tensor decomposition for low-degree polynomials. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1685–1698, 2023.