Faster & Deterministic FPT Algorithm for Worst-Case Tensor Decomposition

Vishwas Bhargava
California Institute of Technology, Pasadena, CA, USA

Devansh Shringi
University of Toronto, Canada

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* Fixed Parameter Tractable (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.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Algebraic circuits, Deterministic algorithms, FPT algorithm, Learning circuits, Reconstruction, Tensor Decomposition, Tensor Rank

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.28

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/123/`

1 Introduction

Tensors, higher-dimensional analogs 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 are one of the most important tools in modern science. Tensor rank and decomposition have become fundamental tools in various branches of science, with applications in machine learning, statistics, signal processing, and computational complexity. Even within machine learning theory, tensor decomposition algorithms are leveraged to give efficient algorithms for phylogenetic reconstruction [40], topic modeling [2], community detection [3], independent component analysis [39], and learning various mixture models [27, 30]. 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 [37] and the references therein.

52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 28; pp. 28:1–28:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. 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 [25] showed that determining the tensor rank is an NP-hard problem over $\mathbb{Q}$ and NP-complete over finite fields. The follow-up work of Schafer and Stefankovic [42] 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 they work when the input is non-degenerate (i.e., when the tensor components are chosen from a Zariski open set), or they are heuristic. See, for instance, [14, 4, 36]. This area also includes interesting active research directions, such as making these algorithms noise-resilient and studying their smoothed analysis [6, 43, 16].
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 [11] 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 [41] designed a randomized FPT algorithm for decomposing rank-k tensors. Specifically, they provided a *randomized* $k^{k^{\mathcal{O}(k)}}\operatorname{poly}(n,d)$ time algorithm for decomposing rank-k tensors. As a direct consequence, they demonstrated that tensors can be decomposed in polynomial time even beyond the 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 reconstruction of a special case of depth-3 circuits. We start by describing this connection below.

Tensors as depth-3 arithmetic circuits

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

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

[1] The seminal work of Lim and Hiller [26] aptly justifies the use of this adjective.

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 $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$), 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 $|X_j| \leq w$ for all j.

Now that we understand the direct correspondence between tensors and set-multilinear depth-3 arithmetic circuits, we will, for simplicity of analysis, stick to this polynomial representation of tensors. We emphasize that all operations on the polynomial associated with a tensor can be interpreted as standard efficient operations on tensors themselves. We will describe these basic operations used in our algorithm in terms of tensor notation, which follows directly from the above equivalence.

For instance, any partial evaluation of f_τ on a subset of the variable partition (say, $X_1 = \boldsymbol{a_1}, X_3 = \boldsymbol{a_3}, X_8 = \boldsymbol{a_8}$) is simply the weighted contraction of the tensor along those directions (1,3,8) with weights $\boldsymbol{a_1}, \boldsymbol{a_3}$, and $\boldsymbol{a_8}$, respectively. A complete evaluation is simply the dot product of τ and the outer-product of the evaluation points, that is $f_\tau(\boldsymbol{a_1}, \ldots, \boldsymbol{a_d}) = \langle \tau, \otimes_{i\in[d]} \boldsymbol{a_i} \rangle$. Note that the linear forms in the circuit representation of f_τ correspond to vectors in the tensor decomposition.

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 analog of exact learning in Boolean circuit complexity [5]. In recent years, much attention has been focused on reconstruction algorithms for various interesting subclasses of arithmetic circuits, both in the worst case setting [7, 35, 34, 17, 10] and in the average case setting [23, 24, 33, 19, 9]. 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 Gaussian and subspace clustering [31, 20, 12]. Finding other such connections is a very interesting research direction.

Returning to the aforementioned connection, we now focus on learning these special depth-3 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 $X = \sqcup_j X_j$ 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-3 set-multilinear circuits is discussed in detail in the full version of paper. 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 [18, 13].

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 [38, 47]. Naturally, in the worst-case setting, things become even more convoluted.

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

Figure 1 An example of non-uniqueness of tensor decomposition by Derksen [15].

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 that 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 $\text{poly}(k)$-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 [11, 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 [11], 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 [41], 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 [41, 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 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 $F(k, n, d) = 2^{k^{\mathcal{O}(1)}} \cdot \text{poly}(n, d)$.*

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

▶ **Corollary 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 the 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 3. The only point at which the field plays a crucial role in our work is in the polynomial system-solving process (which is provably necessary; see Discussion 2). Aside from this, our techniques are field-independent and *deterministic*. The computational model in Theorem 1 assumes input entries as integers (or rationals), and without loss of generality, we can maintain this assumption by approximating or truncating any given real/ complex numbers to a specific number of bits.

The coefficients in our output are derived from solutions to polynomial systems of equations. These solutions lie in an algebraic extension over the base field, and their δ-rational approximations can be computed with an additional factor of $\mathrm{poly}(\log(1/\delta))$ in the time complexity; see, for instance, [21, Remark on page 2].

On the intractability/hardness side, using Håstad's [25] tight reduction between tensor rank and SAT, we see that even *testing* tensor rank will require $2^{\Omega(k)} \cdot \mathrm{poly}(n, d)$ time assuming the Exponential Time Hypothesis (ETH, [29]). This demonstrates that our time complexity is somewhat tight (with respect to the parameter k). We formalize this in the observation below.

▶ **Observation 4.** *Given a d-dimensional tensor $\tau \in \mathbb{F}^{n \times n \times \cdots \times n}$ (with d dimensions) and $k \in \mathbb{N}$, if there exists a $2^{o(k)} \cdot \mathrm{poly}(n, d)$ time algorithm for testing if rank$(\tau) \leq k$, then ETH is false.*

Proof. Note that Håstad ([25, 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 rank $k = 4n + 2m$ if and only if the 3-SAT instance is satisfiable[2]. Thus, if there exists a $2^{o(k)} \cdot \mathrm{poly}(n, d)$ time algorithm for testing tensor rank, this would yield a $2^{o(n)}$-time algorithm for 3-SAT, since the reduction incurs only a linear blow-up. Thus refuting ETH. ◀

Comparison with previous works

Two main works [11, 41] 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 [11], the authors develop a way of computing almost all of the linear forms (i.e. the last layer) 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.

[2] We can assume $n = m$ by an easy reduction from general 3-SAT

- In [41], 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 "multiplication" gates) of multilinear gates such that the cluster representation is unique. The benefit of this approach is that thanks to its 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 [41] is *randomized*, placing tensor rank in randomized FPT time. As observed in [41, 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 $T(k) \cdot \text{poly}(n)$ 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
[11]	Deterministic	$\text{poly}(d^{k^3}, k^{k^{k^{10}}}, n)$	Brute-force search in $\binom{d}{\mathcal{O}(k)}$ sized list
[41]	Randomized	$k^{k^{k^{\mathcal{O}(k)}}} \cdot \text{poly}(n, d)$	System solving with $k^{k^{\mathcal{O}(k)}}$ unknowns
This Work	Deterministic	$2^{k^{\mathcal{O}(1)}} \cdot \text{poly}(n, d)$	System solving with $k^{\mathcal{O}(1)}$ unknowns

Can we improve the time complexity further to $2^{\mathcal{O}(k)}\text{poly}(n, d)$?

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 $2^{\mathcal{O}(k)} \cdot \text{poly}(n, d)$. An improvement to $2^{\mathcal{O}(k)} \cdot \text{poly}(n, d)$ 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 $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ time. We leave this as an open problem.

Over other fields

Our algorithm works for all fields; however, the time complexity worsens to $2^{k^{\mathcal{O}(1)}}$ 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 5** (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 $\text{poly}(k^{k^{\mathcal{O}(1)}})$ degree extension of $\mathbb{F}$ in time $F(k,n,d) = 2^{k^{k^{\mathcal{O}(1)}}} \cdot \text{poly}(n,d) \cdot c_{\mathbb{F}}(2^{k^{\mathcal{O}(1)}})$.*

Here, $c_{\mathbb{F}}(D)$ denotes the time complexity of factorizing a univariate polynomial of degree D 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 [44, 42].

In fact, a tight reduction in [42] solidifies this connection, demonstrating a tight reduction between the feasibility of algebraic systems over $\mathbb{F}$ and computing tensor rank over $\mathbb{F}$:

▶ **Theorem 6** ([42]). *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 5 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 [1, Problem 15].

Interestingly, the hardness of derandomizing univariate polynomial factorization over $\mathbb{F}$ directly impacts tensor decomposition, as observed by Volkovich [46]. Specifically, there is no known way to derandomize Theorem 5 over finite fields unless we can derandomize the factorization of univariate (even quadratic) polynomials over finite fields.

▶ **Theorem 7** ([46, Theorem 5]). *If the class of (set)-multilinear $\Sigma\Pi\Sigma(2)$ circuits over the field $\mathbb{F}$ is learnable as multilinear $\Sigma\Pi\Sigma(k)$ circuits of polynomial size with $k = O(1)$, then, in deterministic $\text{poly}(\log|\mathbb{F}|)$ time, the learning algorithm can be converted to an algorithm that computes the square root of any element in $\mathbb{F}$, if it exists.*

3 Proof Idea

As the introduction mentions, we will use the connection between tensor rank and set multilinear depth-3 circuits. For readers who are not very familiar with the algebraic circuit representation of tensors, we emphasize that we are simply performing standard efficient operations on tensors and use this notation solely for ease of analysis. We refer the reader to subsection 1 to familiarize themselves with this notation. Overall, we are simply performing weighted contractions (partial evaluations) on our tensor after a change of basis (variable reduction) to first learn most of the vectors in the decomposition. We then use the uniqueness of low-rank tensor decomposition in high-dimensional settings to learn the remaining vectors in the decomposition. We now give a detailed sketch of our algorithm using set-multilinear depth-3 circuits terminology.

Recall that we only need black-box access to f_τ, i.e., we can query evaluations of f_τ at fixed points in $\mathbb{F}^n$ in constant time, rather than requiring a dense representation of the tensor/polynomial.

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 using an existing black-box PIT algorithm. This affects the time complexity by a multiple of k.
- We can strip off f_τ with any linear factors ($\text{Lin}(f_\tau) := \{\ell : \ell | f_\tau\}$), since any optimal decomposition(with tensor rank k) of $\text{NonLin}(f_\tau) = \frac{f_\tau}{\prod_{\ell \in \text{Lin}(f_\tau)} \ell}$ gives an optimal decomposition for f_τ as well. This comes from a result of the factoring of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits from [45] described in Lemma 10 that shows that if f_τ is computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit then any irreducible factor $\text{NonLin}(f_\tau)$ 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.
- We can assume that $|X_j| \leq k$. This follows from the width reduction step, see [11, Section 5.1] and Lemma 11. In the low-degree reconstruction in Lemma 12, 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 an 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 reduce it to finding a solution of 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 [35, 8] 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 [32] and [41]-style clustering as the learning approach, which requires solving a system of polynomial equations in $k^{k^{\mathcal{O}(k)}}$ variables and therefore $k^{k^{k^{\mathcal{O}(k)}}}$ time, can be performed within this time constraint. Therefore, we combine ideas from both [11] and [41] 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 [11, Sections 5.3, Section 5.4] to learn almost all the gates of the circuit. The exact result we need is described in Lemma 17. 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 evaluates 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[3] 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 (distance measure between two gates). 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 [11], 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 $\ell_{i,j} \notin \text{span}(\ell_{1,j}, \ldots, \ell_{i-1,j}, \ell_{i+1,j}, \ldots, \ell_{k,j})$, then we can reconstruct T_i exactly. To do this, 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 the above condition holds? That is, $\forall j \in [d], i \in [k]\ \ell_{i,j} \in \text{span}(\ell_{1,j}, \ldots, \ell_{i-1,j}, \ell_{i+1,j}, \ldots, \ell_{k,j})$. 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 $\boldsymbol{\alpha}_j \in \mathbb{F}^{|X_j|}$ such that one linear form depending on X_j (except $\ell_{1,j}$) in C' (and also in C as $j \in$ Consistent-VarPart(C, C')) is set to zero while keeping T_1 non-zero (by ensuring $\ell_{1,j}(\boldsymbol{\alpha}_j) \neq 0$). This will decrease the fan-in by at least one until we are left with our target

[3] 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.

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 (poly(k)) linear forms instead of learning just one multiplication gate. See Lemma 15 for the formal definition of clusters.

3.2 Set-Multilinear Clustering

Karnin and Shpilka [32] 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 [32], [11], and [41] for learning (multilinear) $\Sigma\Pi\Sigma$ circuits.

However, one major drawback of the multilinear clustering used in [41] and [32] is that the rank of the simple part[4] 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 [41].

We develop a *partial* cluster representation (Lemma 15) 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 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 set all the gates not in A to zero 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 sim(C_A).

3.3 Good T_1-isolating Projections

We will now describe how to handle the above issues. Firstly, we will 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., we choose variable parts from Consistent-VarPart$(C, C') \cap$ Support$(Lin(C_A))$. Fixing variables that are in the variable

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

parts from the set Consistent-VarPart$(C, C') \cap$ Support$(Lin(C_A))$ will give us C_A up to a few linear forms. This fixing of variables is what we call a *good* T_1-isolating projection as defined in Definition 19.

And secondly, in Section 7.1, we describe an algorithm to search for those k-variable parts such that the linear forms depending on them in other gates differ from $Lin(C_A)$, in time $k^{\mathcal{O}(k)} \cdot \text{poly}(n, d)$. 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 sim(C_A) is less than $2k^4$, the size of Support$(Lin(C_A))$ is at least $d - 2k^4$ and therefore $|\text{Consistent-VarPart}(C, C') \cap \text{Support}(Lin(C_A))| \geq d - 2k^4 - k^2$. Therefore, our algorithm will pick k^5+1 variable parts, one of which is guaranteed to be in Consistent-VarPart$(C, C') \cap$ Support$(Lin(C_A))$, 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 15 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$(Lin(C_A))$, 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 $\gcd(C_A) = Lin(C_A)$). 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 $j \in$ 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. To ensure that the output circuit computes the correct polynomial, we use an efficient $2^{\mathcal{O}(\log^2 k)} \cdot \text{poly}(n)$ FPT polynomial-time PIT for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(2k)$ circuits in Theorem 9 at the end, which ensures that the only circuit output is the correct one.

4 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. A more detailed set of preliminaries with Notations can be found in the full version.

Complexity of Solving a System of Polynomial Equations

▶ **Theorem 8.** *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. *[21] For $\mathbb{F} = \mathbb{R}$, we have $\text{Sys}_{\mathbb{F}}(n, m, d) = \text{poly}((md)^{n^2})$ 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.*

2. *[28] 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.*
3. *For all fields $\mathbb{F}$, the* $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = \mathrm{poly}((nmd)^{3^n}) \cdot c_{\mathbb{F}}(d^{2^n})$. *Here, $c_{\mathbb{F}}(N)$ 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. For a detailed sketch of the argument and a bound on the size of the extension, see [11, Appendix A].*

Polynomial Identity Testing

▶ **Corollary 9** ([22, Theorem 3])**.** *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.*

The result in [22] is quantitatively much stronger, but we restate only what is most suited for our application.

Factoring: structural and algorithmic results

In their work [45], 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 10** ([45, 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. There is a deterministic algorithm that 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 $2^{\mathcal{O}(\log^2 k)} \cdot \mathrm{poly}(n)$.*

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

In [11], 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 following lemma follows directly from the result of [11, Corollary 5.3] and the fast PIT algorithm of [22].

▶ **Lemma 11** ([11, 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 [11] for the details.

Low Degree Reconstruction

In [11], 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 12** ([11], 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 11, 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 13. 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 15.

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 10.

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

▶ **Definition 14** (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 15.** *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*

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

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

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

Due to space constraints, we defer the formal proof of the above lemma to the full version of the paper.

▶ Remark 16. Note that as described in the proof of Lemma 15, the output cluster A of our clustering algorithm 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 15.

Check out the full version to compare it to previously known clusterings.

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 [11] 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 $k^{\mathcal{O}(k)} \cdot Sys_{\mathbb{F}}(k^{\mathcal{O}(1)}) \cdot \text{poly}(n, d)$. We will use Lemma 12 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 [11], 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 [11], 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 17.** *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 $2k^3 \cdot \left(2F(n, d, k-1) + k^{2k-2}\right) + Sys_{\mathbb{F}}(2k^4, k^{2k^2}, 2k^2) + \text{poly}(k, d) \cdot 2^{\log^2 k}$ 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 $\forall\, i \in [k]\ \Delta(T_i, T'_i) < 2k$.*

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 [5], 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 18 (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.,*

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

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

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 19** (*Good* T_1 isolating Projection). *A T_1-isolating projection $(G, \mathcal{L})$ is good for circuit C', C if $G = A$ in Lemma 15 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 16.*

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 1, 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 $GoodProjList$.

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 1 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 20 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,

Algorithm 1 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 $\Delta(T_i, T'_j) < 2k$

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

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 20.** *Algorithm 1 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 17 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 $k^{\mathcal{O}(k)} \cdot \text{poly}(n, d)$ time.*

Due to space constraints, we defer the formal proof of the above lemma to the full version of the paper.

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 20, one of which will be computing C_A.

Algorithm 2 Candidate Circuits using Good Partitions.

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

function *Candidate-Cluster-Circuit*$(C, C', GoodProjList)$
  $Candidate\text{-}Circuits = \phi$
  for $(G, \mathcal{L}) \in GoodProjList$ **do**
    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 $(j, \boldsymbol{\alpha}_j) \in \mathcal{L}$.
    Use Lemma 10 to obtain the linear factors of $\Phi(C)$(denoted by $Lin(\Phi(C))$) and black-box access to $NonLin(\Phi(C))$
    Reconstruct the set-multilinear circuit $NonLin(\Phi(C)')$ using low degree (degree $\leq k^4 + k^3$) from Lemma 12 black-box access to $NonLin(\Phi(C))$ with top-fan-in set to $|G|$.
    If Reconstruction fails in case $deg(NonLin(\Phi(C))) \geq k^4 + k^3$, move to next $(G, \mathcal{L})$.
    Set $\Phi(C') = \left(\prod_{\ell\in Lin(\Phi(C)} \ell\right) \cdot NonLin(\Phi(C)')$
    Obtain $\Phi(C') = \Phi(C') \cdot \left(\prod_{(j,\boldsymbol{\alpha}_j)\in\mathcal{L}} \frac{\ell_{1,j}}{\ell_{1,j}(\boldsymbol{\alpha}_j)}\right)$
    Add $(\Phi(C'), |G|)$ to $Candidate\text{-}Circuits$
  Output $Candidate\text{-}Circuits$

▶ **Lemma 21.** *Given a list of T_1-isolating Projections from Lemma 20, 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 17, then Algorithm 2 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 $k^{\mathcal{O}(k)} \cdot Sys_{\mathbb{F}}(2k^6, k^{2k^4}, 2k^4) \cdot \text{poly}(n, d)$.*

Due to space constraints, we defer the formal proof of the above lemma to the full version of the paper. We conclude by presenting the algorithm mentioned in our main theorem (Theorem 1). The proof of its correctness can be found in the full version of the paper.

Algorithm 3 Reconstruction of set-multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Input: Black-box access to set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ Circuit $C = T_1 + \ldots + T_k$

function *TensorReconstruction*(C, k)
  if $k = 0$ **then**
    Output 0
  else
    if $k = 1$ **then**
      Factor using Lemma 10 to obtain all the linear factors.
      Output the circuit that is product of all factors.
  Use Lemma 17 to obtain a list of $2k^{2k+1}$ almost circuits L
  for $C' \in L$ **do**
    Run Algorithm 1 with C and C' to obtain $GoodProjList$
    Run Algorithm 2 with input $C, C', GoodProjList$ to obtain $Candidate\text{-}Circuits$
    for $(C_G, |G|) \in Candidate\text{-}Circuits$ **do**
      $\tilde{C} = TensorReconstruction(C - C_G, k - |G|)$
      Run blackbox PIT algorithm on $\tilde{C} + C_G - C$, if identity output $\tilde{C} + C_G$
  Output Nil

References

1 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. doi:10.1007/3-540-58691-1.

2 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.

3 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. doi:10.5555/2627435.2670323.

4 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. URL: http://proceedings.mlr.press/v40/Anandkumar15.html.

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

6 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. doi:10.1145/2746539.2746605.

7 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. doi:10.1145/337244.337257.

8 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.

9 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.

10 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. doi:10.1145/3365667.

11 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. doi:10.1145/3406325.3451096.

12 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 *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287, pages 25:1–25:19, 2024. doi:10.4230/LIPIcs.ITCS.2024.25.

13 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. URL: http://proceedings.mlr.press/v125/chen20a.html.

14 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. doi:10.1109/TSP.2007.893943.

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

16 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. URL: https://proceedings.mlr.press/v178/ding22a.html.

17 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.

18 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. doi:10.1145/2213977.2213995.

19 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.

20 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. doi:10.1109/FOCS46700.2020.00087.

21 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. doi:10.1016/S0747-7171(88)80005-1.

22 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.

23 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. doi:10.1109/FOCS.2011.70.

24 A. Gupta, N. Kayal, and Y. Qiao. Random arithmetic formulas can be reconstructed efficiently. *Computational Complexity*, 23(2):207–303, 2014. doi:10.1007/s00037-014-0085-0.

25 Johan Håstad. Tensor rank is np-complete. *J. Algorithms*, 11(4):644–654, 1990. doi:10.1016/0196-6774(90)90014-6.

26 Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *J. ACM*, 60(6), November 2013. doi:10.1145/2512329.

27 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).

28 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, pages 138–147, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/73007.73020.

29 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

30 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).

31 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. doi:10.1109/FOCS57990.2023.00079.

32 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.

33 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. doi:10.1145/3313276.3316360.

34 A. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of computing*, 2(10):185–206, 2006. doi:10.4086/TOC.2006.V002A010.

35 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.

36 Pravesh K. Kothari, Ankur Moitra, and Alexander S. Wein. Overcomplete tensor decomposition via koszul-young flattenings, 2024. doi:10.48550/arXiv.2411.14344.

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

38 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.

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

40 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).

41 Shir Peleg, Amir Shpilka, and Ben Lee Volk. Tensor Reconstruction Beyond Constant Rank. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287, pages 87:1–87:20, 2024. doi:10.4230/LIPIcs.ITCS.2024.87.

42 M. Schaefer and D. Stefankovic. The complexity of tensor rank. *CoRR*, abs/1612.04338, 2016. arXiv:1612.04338.

43 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. URL: http://proceedings.mlr.press/v65/schramm17a.html.

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

45 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.

46 Ilya Volkovich. A guide to learning arithmetic circuits. In *Conference on Learning Theory*, pages 1540–1561. PMLR, 2016. URL: http://proceedings.mlr.press/v49/volkovich16.html.

47 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. doi:10.1145/3564246.3585232.