Electronic Colloquium on Computational Complexity, Report No. 295 (2026)

Fast Algorithms for Low Rank Tensor Decomposition

Shubhangi Saraf* Devansh Shringi†

November 3, 2026

Abstract

We give fast algorithms for low-rank tensor decomposition. Over $\mathbb{R}$ and $\mathbb{C}$ we give a deterministic algorithm that takes as input an order-d, width-n, rank-k tensor, gets measurement access to the tensor, runs in time $k^{O(k^2)} \cdot \text{poly}(n, d)$, and finds a minimal decomposition as a sum of rank-one tensors. This algorithm also immediately gives an algorithm for tensor rank computation with similar time complexity. This improves upon the previous best runtime of $k^{O(k^{16})} \cdot \text{poly}(n, d)$ by the work of Bhargava and Shringi.

In particular, we get the first subexponential time algorithms for tensor decomposition of order-d tensors of rank as large as $\sqrt{n}/\log n$. One interesting consequence is that the run time of our algorithm for decomposing order-d tensors approaches the fastest algorithms we know for decomposing order-3 tensors ($2^{O(k^2)} \cdot \text{poly}(n)$). Moreover, known NP-hardness results for tensor rank computation suggest that an exponential dependence on k is necessary.

Our approach is via the equivalent problem of proper learning for the class of set multilinear depth-3 circuits of top fan-in k. The main technical tools involve iteratively applying appropriate pseudorandom projections combined with solving multivariate polynomial systems.

1 Introduction

A *tensor* is a multi-way array $T \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_d}$ with entries denoted by $T_{i_1,i_2,\ldots,i_d}$ for $i_j \in [n_j]$, where d is the order of the tensor, and width $n = \sum_{j=1}^{d} n_j$. Tensors are higher-dimensional analogs of matrices. A rank-1 tensor is an array in $\mathbb{F}^{n_1 \times n_2 \times \ldots n_d}$ expressible as $a_1 \otimes a_2 \otimes \ldots \otimes a_d$, where $\otimes$ denotes the Kronecker (outer/tensor) product, for $a_1 \in \mathbb{F}^{n_1}, \ldots a_d \in \mathbb{F}^{n_d}$. The *rank* of a tensor T is the minimum k such that T can be written as a sum of k rank-1 tensors.

Tensor Decomposition is the algorithmic task of given a rank k tensor T, finding the constituent k rank-1 summands of T. For a tensor T, *measurement* of T with respect to vectors $v_1 \in \mathbb{F}^{n_1}, \ldots, v_d \in \mathbb{F}^{n_d}$ is the quantity $\sum_{(i_1,\ldots,i_d) \in [n_1] \times \ldots \times [n_d]} T_{i_1,\ldots,i_d} v_{1,i_1} v_{2,i_2} \ldots v_{d,i_d}$.

Tensors are fundamental objects in mathematics and computer science and the tasks of finding the rank and minimal rank-1 decomposition of a given tensor are important and central problems in computer science, with applications throughout statistics, signal processing, data science, complexity theory, and machine learning. Just in machine learning, it appears as an important primitive for obtaining efficient algorithms for phylogenetic reconstruction [MR05], topic modeling [AFH+12], community detection [AGHK14], independent component analysis [MW19], and learning various

*Department of Mathematics & Department of Computer Science, University of Toronto, Toronto, Canada. Research partially supported by an NSERC Discovery Grant and a McLean Award. Email: shubhangi.saraf@utoronto.ca

†Department of Computer Science, University of Toronto, Toronto, Canada. Research partially supported by an NSERC Discovery Grant and a McLean Award. Email: devansh@cs.toronto.edu

ISSN 1433-8092

mixture models [HK13, JO14]. We refer the reader to detailed monographs [Lan12],[Moi18] and references within for more background and applications of tensors.

Unlike matrices (which are tensors of order 2), the tasks of computing the tensor rank and tensor decompositions of higher order tensors are significantly more challenging. Håstad [Hås90] showed that determining the tensor rank and finding the tensor decomposition is an NP-hard problem for $d \geq 3$. Thus, we do not expect to find a polynomial-time worst-case algorithm for the problem. In fact, a follow-up work of Schaefer and Stefankovic [SS16] gives a reduction from solving general multivariate polynomial systems to computation of tensor rank. Formally, this work shows that for *any* field $\mathbb{F}$, given a system S of polynomial equations in $\mathbb{F}[x_1, \ldots, x_n]$, we can in polynomial time construct a 3-dimensional tensor T_S and an integer k such that S has a solution in $\mathbb{F}$ *iff* T_S has rank at most k over $\mathbb{F}$. Thus, the problem becomes undecidable over $\mathbb{Z}$[1], while it is NP-complete over finite fields.

Despite the hardness, the tensor rank problem is still extremely interesting under various relaxations or restrictions of parameters. An important setting and one which has received a lot of attention is when the input tensor comes from a random distribution, or we add a perturbation with noise (smoothed analysis model) [GVX14, BCMV14, AGJ15, MSS16, Wei23]. Another well-studied case is when the vectors defining the rank-1 components of the input tensors are generically chosen, i.e., do not follow a set of polynomial constraints [LRA93, JLV23, Koi25, KMW25].

In the study of tensors and tensor decompositions, a very natural question to ask is the following - is the minimal decomposition of the tensor as a sum of rank 1 tensors unique? There have been several works trying to understand criteria under which such uniqueness results can be proved [Kru77, LP23], and perhaps the most well known of these is the theorem of Kruskal [Kru77] which gives a simple and clean criterion under which a tensor is guaranteed a unique minimum-rank decomposition. Recent exciting works have explored algorithms for tensor decomposition when the input tensor satisfies these uniqueness conditions [BSS26] and also in some situations trying to obtain decompositions in more general settings [KO25].

The goal in this paper will be to obtain the fastest possible algorithms in the *worst case setting*, when the input tensor is arbitrary but has bounded rank. This study was initiated recently in [BSV21], where the authors gave a $\text{poly}(n, d^{k^k})$ time algorithm, which was a polynomial-time algorithm for constant rank tensors. A natural next goal was to improve the run time to get polynomial run time even with super constant values of k, for instance an algorithm with running time that has FPT style dependence on the rank of tensor, i.e. $f(k) \cdot \text{poly}(n, d)$ for some function $f(k)$. In [PSV24], this was achieved, where the authors presented a randomized $k^{k^{k^{O(k)}}} \text{poly}(n, d)$ time algorithm. The work of [BS25] significantly improved this running time - this work gave a deterministic $poly(k^{k^{16}}, n, d)$ time algorithm, thus derandomizing the previous work and also giving a much better dependence on k. Due to the NP-hardness of tensor rank computation, it is widely believed that an exponential dependence in k on the running time is expected for the tensor decomposition problem[2]. This still leaves a large gap in our understanding of the polynomial in the exponent of the best tensor decomposition algorithms. In this work we try to bridge this gap and give significantly faster algorithms for the low-rank tensor decomposition problem.

Our result (informal): Let the underlying field be $\mathbb{R}$ or $\mathbb{C}$. *Let* $T \in \mathbb{F}^{n_1 \times \ldots \times n_d}$ *be an order* d *tensor with* $n = \sum_{i \in [d]}(n_i)$. *Let* k *be an upper bound on the rank of* T.

[1] Also, by a recent result [Ope26b], it is undecidable over $\mathbb{Q}$

[2] Note that tensor rank computation is simpler than tensor decomposition with smallest k, since the tensor decomposition algorithm can iterate over all values of k and find the least one where there is a decomposition - and this will compute the tensor rank

Given k and given measurement access to T, there exists a deterministic algorithm that runs in time $\mathrm{poly}(k^{O(k^2)}, n, d)$ and computes the tensor rank of T as well as the minimal decomposition of T as a sum of rank 1 tensors.

Our runtime also depends polynomially on the bit complexities of a representation of $\mathcal{T}$ as a sum of k rank-1 tensors (as do the prior works mentioned above). For a more formal theorem statement, see Theorem 1.1.

Several of the prior works that study worst-case tensor decomposition, as well as our current paper, study it through the lens of reconstructing set-multilinear depth-3 arithmetic circuits. Indeed, through this lens, it follows that tensor evaluations exactly correspond to evaluations of a certain set-multilinear depth-3 arithmetic circuit, and the tensor decomposition problem corresponds to reconstruction algorithms for this class of circuits. We now describe this equivalence and then restate our results in the language of polynomial reconstruction.

1.1 Reconstruction of Arithmetic Circuits

Arithmetic circuits are directed acyclic graphs (DAGs) that succinctly compute multivariate polynomials. They have variables as the base, and repeatedly use addition ($+$) gates and multiplication ($\times$) gates to compute polynomials.

The problem of *Reconstruction* of arithmetic circuits is as follows: Design an efficient algorithm (randomized or deterministic) that, given black-box access to evaluations of a polynomial computed by an arithmetic circuit C from a class $\mathcal{C}$, outputs a circuit C' computing the same polynomial as the input circuit. This is the algebraic analog of exact learning for boolean circuits [Ang87]. If there is an added constraint that the output circuit C' also needs to be from the same circuit class $\mathcal{C}$, then we call the task *proper learning*.

Tensor Decomposition and Reconstruction of Depth-3 Set-Multilinear Arithmetic Circuits For a tensor $T \in \mathbb{F}^{n_1 \times \ldots \times n_d}$, consider the set-multilinear polynomial f_T over the variable sets $X_1, \ldots, X_d$ with $X_i = \{x_{i,1}, \ldots, x_{i,n_i}\}$ as follows:

$$f_T := \sum_{(i_1,\ldots,i_d) \in [n_1] \times \ldots \times [n_d]} T_{i_1,\ldots,i_d} x_{1,i_1} x_{2,i_2} \ldots x_{d,i_d}.$$

If f_T can be computed by a depth-3 set-multilinear circuit with top fan-in k (denoted by $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$), then f_T has the form

$$f_T = \sum_{i=1}^{k} \prod_{j=1}^{d} l_{ij}$$

where l_{ij} is a linear form in the X_j variables. For each linear form $l_{ij} \in \mathbb{F}[X_j]_1$, we can look at its dual form, which corresponds to the vector $\bar{v}(l_{ij}) \in \mathbb{F}^{n_j}$. The above form therefore gives us a corresponding rank-k decomposition of T as

$$T = \sum_{i=1}^{k} \bar{v}(l_{i1}) \otimes \bar{v}(l_i 2) \otimes \ldots \otimes \bar{v}(l_{id}).$$

Similarly, one can go from a rank-k decomposition of T to a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of f_T. The blackbox evaluations of f_T correspond to *measurements* of T, and have been

studied in several models [FS12, CM20] along with [BSV21, PSV24, BS25]. We recommend the reader to Section 3.2 for more details of the equivalence.

The problem of reconstructing arithmetic circuits is central in algebraic complexity, but remains notoriously difficult. Recently, several algorithms for various restricted but natural subclasses of arithmetic circuits have been developed [BBB+00, KS01, KS06, GKL12, FS13, BSV20, ST26].

Special attention was given to depth-3 arithmetic circuits for their connections to various notions of rank, and structural results in polynomial identity testing (PIT) [Shp07, KS09, Sin16, Sin22, SS25, SSV26], where we now have quasi-polynomial-time randomized algorithms for constant top fan-in circuits. Set-multilinear and multilinear depth-3 circuits have also received a fair bit of attention for their connections to tensor decompositions [BSV21, PSV24, BS25]. These questions have also been explored when the input circuit is generic [GKL11, GKQ14, KNST19, KNS19, KS19, GKS20, BGKS22], and these found connections in capturing learning problems for various interesting models.

In this work, we will focus on the model of set-multilinear depth-3 circuits of top fan-in k, and try to obtain fast algorithms for this model when k is bounded.

1.2 Our Results

In this paper, we will give a worst-case tensor decomposition algorithm for tensors of rank-k (or equivalently, reconstruction of set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits) with running time $k^{O(k^2)} \cdot \mathrm{poly}(n, d)$. In the language of circuit reconstruction, the algorithm will assume black-box access to evaluations of the circuit (equivalently, measurements of the tensors) over the field. When $\mathbb{F} = \mathbb{F}_q$, if $|\mathbb{F}| \leq O(d)$, then we allow evaluations from an extension field of $\mathbb{F}$.

Theorem 1.1 (Learning $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits)**.** *Let $\mathbb{F} = \mathbb{R}$, $\mathbb{C}$, or $\mathbb{F}_q$. Let $f \in \mathbb{F}[X_1, \ldots, X_d]$ be a degree-d, n-variate set-multilinear polynomial ($n = \sum_{j\in[d]}(|X_j|)$) computable by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C. Then*

- *Suppose that $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, and C has all constants in $\mathbb{Q}$, with bit complexity at most τ. Then there exists a deterministic algorithm that knows n, k, d, τ, makes blackbox queries to f, runs in time $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \tau)$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\mathbb{F}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\mathbb{F}$.*
- *Suppose that $\mathbb{F} = \mathbb{F}_q$. Then there exists a randomized algorithm that knows n, k, d, makes blackbox queries to f, runs in time $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \log q)$ and outputs, with high probability, a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\overline{\mathbb{F}_q}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\overline{\mathbb{F}_q}$.*

Note that even if the circuit C has rational numbers appearing as coefficients, the output circuit might have algebraic numbers appearing in its representation. However, they will be efficiently representable, with complexity at most $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \tau)$. Essentially, the complexity of the output coefficients is what is guaranteed by a multivariate polynomial system solver which takes as input $\mathrm{poly}(k^k)$ equations over $O(k^2)$ variables and of degree $O(k)$, with coefficients that have complexity τ (See Theorem 13.11, [BPR06]). For the case of $\mathbb{F} = \mathbb{F}_q$, our output circuit will be in a $k^{O(k^2)}$-degree extension of $\mathbb{F}_q$, but the algorithm will be computing the $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\overline{\mathbb{F}_q}$ with minimum k'. This also comes from solving a similar system of polynomial equations (See Theorem 2.1, [Roj99]).

Our run-time directly improves on the previous best algorithm by [BS25], which had a run time of $\mathrm{poly}(k^{k^{16}}, n, d)$. In [BS25], the algorithm combines structural analysis of the circuits combined with off-the-shelf algorithms for polynomial system solving. The algorithm of [BS25] needs to solve instances of systems of $k^{O(k^4)}$ polynomial equations in $O(k^6)$ variables and of degree $O(k^4)$, but they did not use the state-of-the-art system solvers. By using the improved system solvers as used in this work, this immediately improves their runtime to $\mathrm{poly}(k^{k^6}, n, d)$. The current work improves it significantly, giving a runtime of $\mathrm{poly}(k^{k^2}, n, d)$ by using stronger analyses of projections of circuits. Our run time approaches the best run time known for constant-order tensors (which is $\mathrm{poly}(2^{k^2}, n, d)$) up to a logarithmic factor of k in the exponent.

Due to the equivalence between the above model and the tensor decomposition problem, we immediately get the following corollaries.

Corollary 1.2 (Decomposing rank k tensors from measurements)**.** *Let $\mathcal{T} \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ be an order-d tensor over $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, with rank at most k and coefficients in the rank k representation from $\mathbb{Q}$ with bit complexity τ. Let $n = \sum_{i=1}^{d} n_i$. Given the values of k and n, and given black-box access to measurements of $\mathcal{T}$ (equivalent to evaluations of $f_{\mathcal{T}}$), there exists a deterministic $\mathrm{poly}(k^{O(k^2)}, d, n, \tau)$ time algorithm for computing the tensor rank of $\mathcal{T}$ and a minimal decomposition of $\mathcal{T}$ as a sum of rank 1 tensors.*

Note that in the above algorithm there is an additional polynomial dependence on τ where τ is an upper bound on the bit complexity of some representation of $\mathcal{T}$ as a sum of k rank 1 tensors.

The above algorithm gives the first polynomial-time algorithm for tensor decomposition and tensor rank computation for the rank as large as any function in $o\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ as well as the first subexponential algorithm for rank as large as any function in $o\left(\sqrt{n/\log n}\right)$.

Note that if we do not have access to measurements but only query access to the entries of the tensor, then we can recover access to the measurements by a polynomial overhead of n^d.

Corollary 1.3 (Decomposing rank k tensors)**.** *Let $\mathcal{T} \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ be an order-d tensor over $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, with rank at most k and coefficients in the representation from $\mathbb{Q}$ with bit complexity τ. Let $n = \sum_{i=1}^{d} n_i$. Given $\mathcal{T}$ as a grid, there exists a deterministic $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \tau) \cdot \left(\prod_{j\in[d]} n_j\right)^{O(1)}$ time algorithm for computing the tensor rank of $\mathcal{T}$ and a minimal decomposition of $\mathcal{T}$ as a sum of rank 1 tensors.*

Even in this setting, we give the first subexponential tensor decomposition for $k = o\left(\sqrt{n/\log n}\right)$ and $d = o(n)$.

AI disclosure

LLMs were used only for minor copy-editing and for improved clarity of exposition. The generation of proofs and essentially all the writing was by the human authors.

2 Proof Overview

Our algorithm will involve several instances of solving systems of multivariate polynomial equations along with several variable reduction procedures that ensure that the number of variables these systems depend on is small, i.e bounded by $O(k^2)$. It is known that over $\mathbb{R}$ and $\mathbb{C}$, any system of polynomial equations in t variables with m equations and degree r can be solved deterministically

in time $\text{poly}(r^t \cdot m)$. In order to keep the run time efficient, we will make sure each system we are solving has at most $O(k^2)$ variables and of degree at most $O(k)$.

The most basic version of how we can reduce rank-k tensor decomposition to polynomial system solving in $O(k^2)$ variables comes from considering low or constant dimensional tensors, and we first describe how to do this. Such a procedure was already observed in several works.

We note that naively, doing the same for d-dimensional tensors would reduce to solving systems in $O(d \cdot k^2)$ variables, which is too much.

It will be helpful to describe our algorithms using the perspective of reconstruction algorithms for set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits, and henceforth in the proof overview we will stick with the language of reconstruction algorithms for set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits. Recall that the setting is that we have blackbox access to a polynomial f that is computable by a circuit C that is a set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit, and we are trying to output a set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit C' for f with the smallest possible value of k'. Note that if the representation of C has bit complexity bounded by τ, then it is possible that another representation of f with smaller top fanin might have bit complexity significantly larger than τ, and our algorithm needs to address these issues. For the purpose of the proof overview, instead of learning a representation with the minimal k', we will show how to learn a representation as a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit, a setting where we are already promised that there exists a circuit computing f with constants of bounded bit complexity. This already will contain most of the ideas in the final proof. Additional care is needed to output representations with minimal top fanin, but we skip that for the purpose of the proof overview.

As a first step, we will show how to obtain reconstruction algorithms for set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits of constant degree (which corresponds to constant-order tensors).

Reconstruction of constant degree set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits: This corresponds to reconstructing set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits of constant degree, given blackbox query access to the underlying polynomial f. Suppose the degree of f is r where r is a constant.

Let us fix a representation C of a circuit computing f as follows:

$$C = \sum_{i \in [k]} T_i = \sum_{i \in [k]} \prod_{j \in [r]} \ell_{i,j},$$

where $\ell_{i,j} \in \mathbb{F}[X_j]_1$.

The total number of linear forms appearing in the underlying circuit C is at most $k \cdot r$, with k linear forms in each set of variables. It follows that the number of "essential variables" (See Definition 8) in each set of variables is at most k. By applying an appropriately constructed invertible linear map to each set of variables (Lemma 3.17), we first reduce our reconstruction problem to one where each set of variables has only size k.

We now set up a system of polynomial equations whose solution will give us the coefficients of the linear forms appearing in some set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit for f.

Let the coefficients of the linear forms $\ell_{i,j}$ be the variables in the system, and thus we have $k^2 \cdot r$ unknowns. By polynomial interpolation, we can compute the monomial representation of the polynomial f that is computed by C. By multiplying out the expression with the unknown variables in the expression $\sum_{i \in [k]} \prod_{j \in [r]} \ell_{i,j}$, we observe that each coefficient of the monomials in $X_1, X_2, \ldots, X_r$ is a polynomial in the unknown $k^2 \cdot r$ variables. Equating it with the coefficients of monomials after interpolating f gives a system of polynomial equations in the $k^2 \cdot r$ variables. Clearly, any solution would give a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit computing f.

This above described algorithm is not new, and indeed variations of this appeared in [BSV21, PSV24, BS25]. The challenge is to do this for large values of d and still do this using polynomial

systems in only $O(k^2)$ variables.

Low rank $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits This is the case where our circuit C is as follows:

$$C = \sum_{i\in[k]} T_i = \sum_{i\in[k]} \prod_{j\in[d]} \ell_{i,j},$$

where $\ell_{i,j} \in \mathbb{F}[X_j]_1$.

The degree d might be superconstant, but we have a promise that the subspace spanned by all the $d \cdot k$ linear forms in the circuit is only $O(k)$-dimensional. Notice that in this model this automatically implies that the degree d is at most $O(k)$ and indeed it could be even much smaller. This quantity, $\dim(\text{span}(\{\ell_{i,j}\}))$ is called $\text{rank}(C)$[3].

It is not too hard to show that this case is not so different from the previous case.

The main idea is to reduce the number of variables in each variable part (by applying invertible linear transformations), such that the number of variables across all variable parts at the end equals the rank (which we assume is $O(k)$) , and at the same time we have preserved the set-multilinear structure. We then construct a system of polynomial equations whose unknown variables are the coefficients of the variables in the linear forms in the circuit, and such that the solutions correspond to set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit representations for f. Solving this system gives us what we want. If we do the accounting carefully, this system still only has $O(k^2)$ variables.

Low degree $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits This is the first challenging case. We consider set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits whose degree is at most $5k$ (it could have been anything that is $O(k)$, but $5k$ will be a convenient choice in the proof). Observe that such circuits could have rank as large as $\Theta(k^2)$ and hence naively recovering the circuit by solving a system of polynomial equations would have k^3 variables, even after mapping the circuit only to its "essential variables".

The main idea is to consider a projection of the circuit, where only a few variable parts are kept alive and the rest are set to random (or pseudorandom) constants. This projection will make the rank *small enough* to use the reconstruction algorithm in the *low rank* case to reconstruct some representation (need not be unique) of the projected circuit. Yet, the projection will also keep the rank *large enough* that *rank bounds* for $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits (see Theorem 3.9) will be helpful to reveal some information about the original hidden circuit C. Now suppose only the variable parts indexed by $S \subset [d]$ are kept alive, and the projected circuit is C_S computing some polynomial f_S. The reconstruction algorithm might return another $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of C_S, say C'_S. Now we know that there exists a "lift" of C_S to C, but we do not know that C'_S can be lifted to get a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit representation of the original polynomial f. This is where rank bounds for $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits (see Theorem 3.9) come to our rescue. We see that C'_S still reveals a lot of information about C_S. Since $C_S - C'_S \equiv 0$, the rank bounds imply that a significant number of linear forms appearing in C_S also appear in C'_S. Indeed, we can show that all the linear forms in C_S which do not have this property lie in a subspace W of dimension at most $4k - 2$.

By an iterative and exhaustive search, we will show that we can assume the algorithm "knows" the set of all linear forms in C_S except those that lie in W, and it also knows which gates they appear in. Thus we would have learned a big chunk of C_S (all linear forms but for those that lie in the low-dimensional space W). Now we undo some part of the projection and make more variable parts "alive". Thus we have grown the set S to a larger set S'. We set up a new system of polynomial equations (still in $O(k^2)$ variables) to learn a set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation

[3]When we formally define the rank, we will strip out the common gcd of the product gates and then look at the span of the leftover linear forms.

of $f_{S'}$ (this is a variation of the low rank reconstruction algorithm) that is consistent with the linear forms learned previously, and it allows us to learn even more linear forms, now from $C_{S'}$, since we now have more variable parts to learn from. We keep doing this until all d variable parts have been made alive (or unprojected), and the final polynomial system that we solve then yields a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of the full unprojected polynomial.

General (high degree) $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits As in the previous case, carefully constructed projections will again play a crucial role. We will consider a projection of the circuit C to only the first $5k$ variable parts, by substituting appropriately chosen field constants into the other variable parts, such that certain nonzeroness properties of the resulting circuit are preserved. Let the resulting circuit be C_S. After the projection, we are in the bounded degree case, and we can use the reconstruction procedure described above for degree $5k$ circuits. The reconstructed circuit output by the algorithm might not equal C_S, though it will compute the same polynomial as C_S. Let us call it C'_S. Note that C'_S might not be liftable or extendable to a set multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of f, though by construction, we know that C_S is liftable.

We will use rank bounds to show that the learned circuit C'_S will be "close" to C_S. We will use this closeness to show that every product gate in C_S agrees with some other product gate in C'_S in all but very few linear forms. By a careful accounting we will conclude that there are $k+2$ variable parts such that restricted to those parts, the gates in C'_S can allow us to recover *all* the linear forms in C_S. Once we have all the linear forms in several of the variable parts, this turns out to be specially useful, as we will now describe.

Let the input (within the blackbox) circuit C be of the form $C = \sum_{i\in[k]} T_i = \sum_{i\in[k]} \prod_{j\in[d]} \ell_{i,j}$.

Since there are $k+2$ variable parts wherein we have learnt all the linear forms, WLOG let us assume we know the parts corresponding to $[k+2]$. Thus for each $i \in [k]$, the algorithm has learned the linear forms in gate T_i corresponding to the variable parts in $[k+2]$. Let $A_i := \prod_{j\in[k+2]} \ell_{ij}$. Clearly, for all $i \in [k]$, A_i divides T_i, and we are in the case where we have learned $A_1, \ldots, A_k$. By some initial preprocessing we will assume $\gcd(T_1, \ldots, T_k) = 1$, and hence it follows that $\gcd(A_1, \ldots, A_k) = 1$ as well. Note that thus C is of the form $\sum_{i=1}^{k} A_i \cdot B_i$ where each B_i is a product of linear forms. Moreover each A_i is supported on the first $k+2$ variable parts and each B_i is supported on the remaining variable parts. We first illustrate a simple case. Suppose $A_1, \ldots, A_k$ are linearly independent. Observe that by plugging in suitable values to the variables of X_i for each $i \in [k+2]$, we get access to linear combinations (over the base field) of $B_1, \ldots, B_k$ where we know the coefficients of the linear combination. By doing this for enough values, and using the assumption that $A_1, \ldots, A_k$ are linearly independent, we show that we can in fact get enough linearly independent linear combinations, so that we can actually recover blackbox access to the individual B_i's themselves. Once we can do this, we can apply a basic factoring algorithm to learn the linear forms of each B_i and we are done.

However in general there is no reason for the A_i's to be linearly independent. Here is another situation that can arise. $A_1 = A_2$, but $A_2, A_3, \ldots, A_k$ are linearly independent. Thus the circuit C looks like

$$A_2(B_1 + B_2) + A_3 \cdot B_3 + \cdots + A_k \cdot B_k$$

Notice that this is again ok, since $A_2, A_3, \ldots, A_k$ are linearly independent and we can individually obtain blackbox access to the things they are multiplying, i.e. to $B_1 + B_2, B_3, \ldots, B_k$. However $B_1 + B_2$ is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(2)$ circuit, and we can assume recursively or inductively that we have reconstruction algorithms for $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuits efficiently for all $k' < k$. Thus we can learn a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(2)$ representation for $B_1 + B_2$ given blackbox access to it.

Here is yet another situation. Suppose that $A_1 = A_2 + A_3$, and $A_2, A_3, \ldots, A_k$ are linearly independent.

Thus the circuit looks like

$$(A_2 + A_3) \cdot B_1 + A_2 \cdot B_2 + A_3 \cdot B_3 + \cdots + A_k \cdot B_k$$

which equals

$$A_2 \cdot (B_1 + B_2) + A_3 \cdot (B_1 + B_3) + A_4 \cdot B_4 + \cdots + A_k \cdot B_k$$

Now we can recover blackbox access to $B_1 + B_2, B_1 + B_3, B_4, \ldots, B_k$ and hope to proceed like before, but we run into an issue. $B_1 + B_2$ and $B_1 + B_3$ are not independent. They share a gate in common. Thus we cannot independently learn $B_1 + B_2$ and $B_1 + B_3$ as $\Sigma\Pi\Sigma_{\sqcup_j X_j}(2)$ circuits and use the output. If we do learn them independently, we would get a circuit of larger top fan-in.

Instead, we do something else. We observe that $A_1 = A_2 + A_3$ actually gives us a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(3)$ identity! In other words since $A_1 - A_2 - A_3 \equiv 0$, we can use rank bounds (see Theorem 3.7) to observe that A_1, A_2 and A_3 are very similar. They agree in all except 1 of the variable partitions! WLOG, let us assume they agree on the first $k+1$ variable parts. Thus, we let A'_i be the part of A_i which only corresponds to the first $k+1$ parts and we add the $(k+2)$th part into B_i and call it B'_i, then the circuit C can be represented as

$$C = A'_3 \cdot (B'_1 + B'_2 + B'_3) + A'_4 \cdot B'_4 + \cdots + A'_k \cdot B'_k.$$

Now, if the $A'_3, A'_4, \ldots, A'_k$ were linearly independent, then we would be done since we could recover blackbox access to $B'_1 + B'_2 + B'_3, B'_4, \ldots, B'_k$. However, they may not be independent, but we still made some progress. We started off with k different A_i's, and now we have fewer of them. Each time we find a linear dependency, we can repeat the process we just described to shrink the number of A_i's. We show that this process will terminate and eventually we will find a representation of C of the form

$$C = A'_1 \cdot C'_1 + A'_2 \cdot C'_2 + \cdots + A'_r \cdot C'_r$$

where $r \geq 2$, $A'_1, \ldots A'_r$ are linearly independent and each C'_i is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k_i)$ circuit where $k_i < k$ for each i and $\sum_{i=1}^{r} k_i = k$. We will be able to recover blackbox access to each C'_i and hence recursively learn the C'_i. Along with the A'_i, this will enable us to recover a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of the original circuit.

3 Preliminaries

Notation: 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. We will denote by $\mathbb{F}[x_1, \ldots, x_n]$ the ring of polynomials over n variables with coefficients from the field $\mathbb{F}$. $\mathbb{F}[x_1, \ldots, x_n]_d$ will denote the set of homogeneous degree d polynomials in $\mathbb{F}[x_1, \ldots, x_n]$.

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. 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 $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ (see Section 3.3 for details). We use $\mathrm{span}(l_1, \ldots, l_r)$ to refer to the vector space that is the span of the linear forms $\sum_{i=1}^{r} \alpha_i l_i$ for $\alpha_i \in \mathbb{F}$. For a vector space $\mathbf{V}$, $\dim(\mathbf{V})$ denotes the dimension of $\mathbf{V}$.

3.1 Depth-3 Circuits

We begin by defining depth-3 circuits and their set-multilinear specialization, which will be the focus of this paper.

Definition 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 forms; $\ell_{i,j}(X) = \sum_{t=1}^{n} a_{i,j}^t x_t + a_{i,j}^0$ 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 forms.*
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 $a_{i,j}^0 = 0$.*

We say that C is minimal *if no proper nonempty 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 appear in* $\gcd(C)$.

Definition 2 (Rank of simple part of circuit)**.** *The* rank *of a circuit C such that* $\mathrm{sim}(C) = \sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X)$ *is defined as the dimension of the space spanned by all the linear functions in the simple part of the circuit (after removing* $\gcd(C)$*)* $\dim(\mathrm{span}(\{\ell_{i,j} : i \in [k], j \in [d_i]\}))$.

3.2 Tensors and Set-Multilinear Depth-3 Circuits

For our purposes, it will be useful to view tensors as a special class of multilinear polynomials, called *set-multilinear* polynomials. We first fix the notation that we will use throughout the paper. Let $d \in \mathbb{N}$ denote the *order*. For each $j \in [d]$, let $X_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,n_j}\}$, where $n_j = |X_j|$, and let $X = \sqcup_{j\in[d]} X_j$. Thus, the sets X_j form a partition of X.

Definition 3 (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, every monomial contains exactly one variable from each part of the partition. Set-multilinear polynomials have been studied extensively [Raz13, LST24], partly because many natural polynomials, including the determinant, the permanent, and the Nisan-Wigderson polynomials, are set-multilinear with respect to suitable partitions of their variables. We can also associate a set-multilinear polynomial with every tensor, as defined below.

Definition 4. *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*

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

Observe that $f_{\mathcal{T}}(X)$ is set-multilinear with respect to X. Moreover, decompositions of $\mathcal{T}$ correspond directly to set-multilinear depth-3 circuits computing $f_{\mathcal{T}}(X)$. We state the circuit model explicitly before describing this correspondence.

Definition 5 (Set-Multilinear Depth-3 Circuits)**.** *A set-multilinear depth-3 circuit w.r.t (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 $\mathbb{F}[X_j]_1$.

To see the intuition behind this correspondence, 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 $\mathcal{T}$. In this case, $\mathcal{T}$ is a rank-1 tensor. Applying this observation to each multiplication gate gives the following connection between tensor decomposition and set-multilinear depth-3 circuits.

Remark 3.1. *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 $\mathbb{F}$.

The correspondence also holds in the other direction: any $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ gives a set-multilinear depth-3 circuit over $\mathbb{F}$ computing $f_{\mathcal{T}}(X)$. Together, these two directions give the following lemma.

Lemma 3.2. *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, 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.*

Proof. Each product $\ell_{i,1}(X_1) \cdot \ell_{i,2}(X_2) \cdots \ell_{i,d}(X_d)$ corresponds to a rank-1 tensor over $\mathbb{F}$. Thus, a circuit with k multiplication gates computing $f_{\mathcal{T}}(X)$ gives a decomposition of $\mathcal{T}$ into k rank-1 tensors over $\mathbb{F}$. Conversely, a decomposition into k rank-1 tensors gives a set-multilinear depth-3 circuit with k multiplication gates over the same field. Therefore, the minimum number of rank-1 tensors in a decomposition equals the minimum top fan-in of a circuit computing $f_{\mathcal{T}}(X)$. □

3.3 Complexity of Solving a System of Polynomial Equations

We will be using subroutines to find a solution of a system of m polynomials $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ of degree at most d. We are interested in the cases where $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$. The complexity of finding a single solution will be denoted by $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ (over $\mathbb{F}_q$, we will allow solutions in $\overline{\mathbb{F}_q}$). We will also address whether the algorithm is randomized or deterministic next to the statement. These

algorithms will output $\perp$ if there is no solution to the system in $\mathbb{F}^n$ (no solution in $\overline{\mathbb{F}}^n$ for $\mathbb{F} = \mathbb{F}_q$), and if the solution set is non-empty, then they output any one of the possible solutions.

When $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, we will assume that the coefficients of $f_1, \ldots, f_m$ are in $\mathbb{Z}$ (for $\mathbb{C}$ the coefficients would be of the form $a+ib$ for $a, b \in \mathbb{Z}$) and have bit size τ, and the output of the solution will be a real algebraic solution, encoded by a real univariate representation over $\mathbb{Z}$. To obtain the required running time, we want the subroutine to find a solution in deterministic $\text{poly}(m, \tau) \cdot d^{O(n)}$ time. This is known in the literature. One can observe that a system of m polynomial equations over $\mathbb{R}$, can be reduced to finding a solution of a single equation $\sum_{i=1}^{m} f_i^2 = 0$ (degree is now $2d$) in polynomial time. Similarly, finding a solution with polynomials in $\mathbb{C}[x_1, \ldots, x_n]$ reduces to finding solutions of $2m$ real equations over $2n$ variables. Thus, an algorithm that finds a real solution of a single polynomial equation of degree d over n variables in time $d^{O(n)}$ suffices for our purpose. Such an algorithm is known and can be found in [BPR06] (Algorithm 13.2).

When $\mathbb{F} = \mathbb{F}_q$, the algorithm will output an explicit extension $\mathbb{E}$ of $\mathbb{F}$ of degree at most $d^{O(n)}$ and a point $a \in \mathbb{E}^n$. As observed in [AGS26, Lemma 4.10], a system with m equations can be reduced to a system of $n+1$ equations by taking random combinations of equations. This can be converted into an $(n+1)$- equation, $(n+1)$- variable square system using a dummy variable. In [Roj99], the authors presented an algorithm that gave us the required running time for a square system (n equations, n variables). If a solution exists, the algorithm outputs the solution with probability $1 - \varepsilon$, and with probability ε it outputs $\perp$. All the steps can be combined to give a total running time of $\text{poly}(m, \log q, \log \frac{1}{\varepsilon}) \cdot d^{O(n)}$.

We combine the above discussion into the following theorem:

Theorem 3.3. *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,*

- *[BPR06, Theorem 13.11] Let $\mathbb{F}$ be $\mathbb{R}$ or $\mathbb{C}$. Let the coefficients of the polynomials be in $\mathbb{Q}$ with bit complexity upper bounded by τ. Then,* $\text{Sys}_{\mathbb{F}}(n, m, d) = \text{poly}(m, \tau) \cdot d^{O(n)}$ *for* deterministic *algorithms.*
- *[Roj99, Theorem 2.1] Let $\mathbb{F}$ be $\mathbb{F}_q$ for some prime power q. Then,* $\text{Sys}_{\overline{\mathbb{F}}}(n, m, d) = \text{poly}(m, \log q) \cdot d^{O(n)}$ *for* randomized *algorithms with solutions in a degree $d^{O(n)}$ explicit extension.*

We are optimistic that the second result might be derandomizable using the ideas of the recent breakthrough on the deterministic univariate factorization result from OpenAI [Ope26a].

3.4 Polynomial Identity Testing (PIT)

Polynomial identity testing (PIT) is the problem of determining whether a given arithmetic circuit computes the zero polynomial. In the black-box setting, we only have access to evaluations of the polynomial. We recall the notions of hitting sets and hitting set generators [SY10, Section 4.1].

Definition 6 (Hitting set)**.** *Let $\mathcal{C} \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be a class of polynomials. A finite set $H \subseteq \mathbb{F}^n$ is a* hitting set *for $\mathcal{C}$ if for every nonzero $f \in \mathcal{C}$, there exists $a \in H$ such that $f(a) \neq 0$.*

Given a hitting set for $\mathcal{C}$, we can deterministically test whether $f \in \mathcal{C}$ is zero by evaluating it at every point of H.

Definition 7 (Hitting set generator)**.** *Let $\mathcal{C} \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be a class of polynomials. A polynomial map $\mathcal{G} = (G_1, \ldots, G_n) : \mathbb{F}^t \to \mathbb{F}^n$, where $G_i \in \mathbb{F}[y_1, \ldots, y_t]$, is a* hitting set generator *for $\mathcal{C}$ if for every $f \in \mathcal{C}$,*

$$f \equiv 0 \quad \iff \quad f(G_1, \ldots, G_n) \equiv 0.$$

The number t is called the seed length *of the generator.*

A polynomial-time blackbox algorithm gives a hitting set, which also gives a hitting set generator by interpolation [SY10, Lemma 4.1].

Our focus of the paper is going to be on set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits.

Theorem 3.4 ([GG20, Theorem 3]). *Let $\mathcal{C}$ be the family of set-multilinear polynomials $f \in \mathbb{F}[X]$ computed by $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ such that $\sum_{i\in[d]} |X_i| \leq n$.*

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 $k = 2^{O(\log(n)/\log\log(n))}$.*
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 has $f(k)\cdot\mathrm{poly}(n,d)$ time complexity. The below corollary follows immediately from the above theorem.

Corollary 3.5 ([BS25]). *Over any large enough field $\mathbb{F}$, there's a deterministic $2^{O(\log^2 k)}\cdot\mathrm{poly}(n)$ PIT algorithm for the class of set-multilinear polynomials computed by depth-3 set-multilinear circuits of degree d and top fan-in k.*

Corollary 3.6. *Over any large enough field $\mathbb{F}$, there's a deterministic $2^{O(k+\log^2 k)}\cdot\mathrm{poly}(n,d)$ algorithm that outputs a set of points $\mathcal{H} \subset \mathbb{F}^n$ such that for any set-multilinear polynomial $f \neq 0$ computed by depth-3 set-multilinear circuit $C = \sum_{i=1}^{k} T_i$ of degree d and top fan-in k, there exists a point $\alpha \in \mathcal{H}$, such that $f(\alpha) \neq 0$ and for any subset $S \subseteq [k]$, if $\sum_{i\in S} T_i \neq 0$, then $\sum_{i\in S} T_i(\alpha) \neq 0$.*

Proof. Using the hitting-set H in Corollary 3.5, we can obtain a degree $2^{O(\log^2 k)}\cdot\mathrm{poly}(n,d)$ hitting set generator $G(t) := (G_1(t),\ldots,G_n(t))$ corresponding to H by interpolation.

Consider any $S \subseteq [k]$, and $f_S = \sum_{i\in S} T_i \neq 0$. As H is a hitting set for $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits, we see $f_S(G) \neq 0$. Consider $\prod_{S\subseteq[k],\sum_{i\in S} T_i\neq 0} f_S(G)$. This is a degree at most $(2^k-1)\cdot d\cdot|H|$ univariate polynomial, and hence if we evaluate it on $(2^k-1)\cdot d\cdot|H|+1 \leq 2^{k+O(\log^2 k)}\cdot\mathrm{poly}(n,d)$ points, we will have a point over which it is non-zero and hence all $f_S(G)$ are non-zero. These univariate values, using G, give us the set $\mathcal{H}$. □

3.5 Rank and degree bounds for set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ identities

In developing PIT algorithms for set-multilinear depth-3 circuits, we developed a deep understanding of the structure of polynomial identities computed by set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits. These can be translated into the following degree and rank bounds:

Theorem 3.7 ([BSV21]). *Let $C \equiv \sum_{i=1}^{k} T_i$ be a simple and minimal set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ circuit computing the zero polynomial. Then for all $i \in [k] : \deg(T_i) \leq k-2$.*

For the rank bound, we are going to use the following Splitting Theorem developed in [LP23] to obtain a generalization of Kruskal's theorem. They define (in [LP23, Definition 3]) a set of non-zero vectors $V = \{v_1,\ldots,v_n\}$ splits if there exists a subset $S \subset V$ such that $1 \leq |S| \leq n-1$ and $\mathrm{span}(V) = \mathrm{span}(S)\oplus\mathrm{span}(V\setminus S)$ (i.e. $\mathrm{span}(V) = \mathrm{span}(S)+\mathrm{span}(V\setminus S)$ and $\mathrm{span}(S)\cap\mathrm{span}(V\setminus S) = \{0\}$).

Theorem 3.8 ([LP23, Theorem 4]). *Let $n \geq 2$, $d \geq 2$ be integers. Let $V = V_1 \otimes V_2 \otimes \ldots \otimes V_d$ be a vector space over a field $\mathbb{F}$, let*

$$E = \{x_{a,1} \otimes \ldots \otimes x_{a,d} : a \in [n]\} \subseteq V \setminus \{0\}$$

be a multiset of product tensors, and for each $j \in [d]$, let

$$r_j = \dim(\text{span}(\{x_{a,j} : a \in [n]\})).$$

If $\dim(\text{span}(E)) \leq \sum_{j=1}^{d}(r_j - 1)$, then E splits.

This can be translated into the language of set-multilinear depth 3 circuits to obtain the following rank bound.

Theorem 3.9. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a simple and minimal set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ circuit computing the zero polynomial. Then,* $\text{rank}(C) \leq 2k - 2$.

Proof. For every linear form $l_{ij} \in \mathbb{F}[X_j]$, such that $l_{ij} | T_i$, we can obtain a vector $x_{ij} \in \mathbb{F}^{|X_j|}$ by considering its dual. Therefore each T_i, can be captured by the vector $V_i = x_{i1} \otimes x_{i2} \otimes \ldots \otimes x_{id}$. Therefore, our set of nonzero vectors is $E = \{V_i : i \in [k]\}$. Note that $r_j = \dim(\text{span}(\{x_{a,j} : a \in [k]\})) = \dim(\text{span}(\{l_{a,j} : a \in [k]\}))$. As the circuit is set-multilinear, we have $\text{rank}(C) = \sum_{j \in [d]} r_j$.

We first observe that the minimality condition ensures that E doesn't split. That is because for any $S \subset [k]$ such that $1 \leq |S| \leq k - 1$, minimality implies

$$\sum_{i \in S} T_i = -\sum_{i \in ([k] \setminus S)} T_i \neq 0$$

otherwise we have smaller fan-in identities. This implies $\text{span}(\{V_i : i \in S\})$ and $\text{span}(\{V_i : i \in ([k] \setminus S)\})$ have an element in their intersection other than 0, corresponding to $\sum_{i \in S} T_i$. Therefore, E doesn't split.

Thus, from Theorem 3.8, we know $\dim(\text{span}(E)) > \sum_{j=1}^{d}(r_j - 1)$. As $|E| = k$, we have $\dim(\text{span}(E)) \leq k$. Therefore, we have $k \geq \text{rank}(C) - d$, which gives $\text{rank}(C) \leq k + d$. From Theorem 3.7, we know $d \leq k - 2$, giving us the required $\text{rank}(C) \leq 2k - 2$. □

3.6 Factoring: structural and algorithmic results

Another tool that we will need is to be able to gain access to the factors of $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ circuits using efficient deterministic algorithms and reconstruct arithmetic circuits for these factors.

In [SV10], the authors 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 of Corollary 3.5, we have the following:

Lemma 3.10 ([SV10, Corollary 1.2]). *There is a deterministic algorithm that, given 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 $2^{O(\log^2 k)} \cdot \text{poly}(n)$. In addition, each such irreducible factor is computable by a $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ circuit.*

An important observation of the above is that for any $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C, one can find a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C' such that C and C' compute the same polynomial, and $\text{sim}(C')$ doesn't have any linear factors. To observe this, consider any $l \in \mathbb{F}[X_j]$ such that $l | \text{sim}(C)$. Therefore, there exists a polynomial $g \in \mathbb{F}[X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_d]$ such that $l \cdot g = C$. Now, let us substitute X_j with a random value α_j such that $l(\alpha_j) \neq 0$. This gives us $l(\alpha_j) \cdot g = C|_{X_j = \alpha_j}$. As $l(\alpha_j)$ is a

non-zero constant, g is computable by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit $C' = \frac{1}{l(\alpha_j)} \cdot C|_{X_j=\alpha_j}$. Using C', we get a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ representation of C as $l \cdot C'$ where the simple part is no longer divisible by l. Doing this with every l that divides $\text{sim}(C)$ gives the required representation.

One can look at the factors from Lemma 3.10, and separate out all the linear factors by testing linearity and reconstructing linear forms, and hence compute $\gcd(C)$ (the product of all linear factors) and $\text{sim}(C)$ (the product of all remaining factors), where additionally $\text{sim}(C)$ doesn't have linear factors. As a corollary, we can efficiently simulate a black-box access to $\text{sim}(C)$ given a black-box access to C.

Corollary 3.11 ([BS25]). *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 $2^{O(\log^2 k)} \cdot \text{poly}(n)$.*

3.7 Essential Variables of a Polynomial

This notion will be useful in reconstruction when the simple part of the input set-multilinear circuit has low rank. We start by defining essential variables in a polynomial.

Definition 8 (Essential variables, [Kay11]). *The* number of essential variables *in $f(x_1, \ldots, x_n)$ is the smallest t for which there exists an invertible linear transformation $A \in \mathbb{F}^{(n\times n)}$ such that $f(A \cdot \overline{x})$ depends on only t variables.*

The number of *redundant variables* is equal to the number of essential variables subtracted from n. We will use the following result from [Car06] that allows us to compute the number of essential variables t and the linear transformation A from the above definition.

Theorem 3.12 ([Car06],[Kay11]). *Let n, d be positive integers and $\mathbb{F}$ be a field with $\text{char}(\mathbb{F}) > d$ or $= 0$. There is a randomized algorithm that takes as input black-box access to an n-variate degree-d polynomial $f(x) \in \mathbb{F}[\bar{x}]$ with t essential variables, runs in time $(nd)^{O(1)}$, and outputs an invertible matrix $A \in \mathbb{F}^{(n\times n)}$ such that $f(A \cdot \bar{x})$ depends only on the first t variables.*

The partial derivative $\partial_i f$ is used to represent $\frac{\partial f}{\partial x_i}$. We use ∂f to denote $(\partial_1 f, \ldots, \partial_n f)$. We define the partial derivative matrix of a polynomial f, $M(f)$, as the matrix with columns indexed by monomials over n variables and of degree at most $d-1$, while the rows are indexed by $[n]$, and $M_{i,j} = \text{coeff}_j(\partial_i f)$ where $\text{coeff}_j(g)$ is the coefficient of monomial j (represented as vector) in g.

We denote $\partial f^{\perp}$ as the set of vectors $\mathbf{a} \in \mathbb{F}^n$ such that $\mathbf{a} \cdot \partial f = 0$. The proof of the above theorem relies on the following lemma, which describes the relation between the partial derivative matrix and the number of essential variables.

Lemma 3.13 ([Car06], Lemma B.1[Kay11]). *The number of redundant variables in a polynomial $f(x_1, \ldots, x_n)$ equals the dimension of $\partial f^{\perp}$. In particular, the number of essential variables of f is the rank of the partial derivative matrix $M(f)$.*

Thus, to compute A from Theorem 3.12, we just need to compute a basis for $\partial f^{\perp}$; extending it to a basis of $\mathbb{F}^n$ gives the columns of A. In the proof of Theorem 3.12, the authors evaluate $\partial f/\partial x_j$ at random values, and the kernel of the resulting matrix gives a basis for $\partial f^{\perp}$.

We will derandomize the above algorithm for some circuit classes using appropriate hitting sets and computation of the relevant basis of $\partial f^{\perp}$.

Lemma 3.14 ([PSV24, Lemma 2.7])**.** *Let C be a class of polynomials and let $f_1, \ldots, f_t \in C$. Let $\mathcal{H}$ be a hitting set for $\Sigma^t C$. Denote by $f_i|_{\mathcal{H}}$ the vector (of length $|\mathcal{H}|$) $(f_i(\beta))\beta \in \mathcal{H}$. Then for any $\alpha_1, \ldots, \alpha_t \in \mathbb{F}$,*

$$\sum_{i=1}^{t} \alpha_i f_i = 0 \iff \sum_{i=1}^{t} \alpha_i f_i|_{\mathcal{H}} = 0.$$

In particular, the polynomials $f_1, \ldots, f_t$ are linearly independent if and only if the vectors $f_1|_{\mathcal{H}}, \ldots, f_t|_{\mathcal{H}}$ are linearly independent.

Lemma 3.15. *Let $f_1, \ldots, f_t$ be polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ of degree d that can be computed by circuits from class C. Let $\dim(\text{span}(f_1, \ldots, f_t)) = r$. Consider the space*

$$V := \{(a_1, \ldots, a_t) : \sum_{i=1}^{t} a_i f_i = 0\}.$$

Let $\mathcal{H}$ be a hitting set for $\Sigma^r C$ circuits. Then, given black-box access to $f_1, \ldots, f_t$, there exists a deterministic $\text{poly}(n, d, t, |\mathcal{H}|)$ time algorithm that can compute a basis of V.

Proof. Let $h := |\mathcal{H}|$. Start with $S := \{f_1\}$, and greedily add $f_2, \ldots, f_t$ to S such that f_i is added to S iff f_i is linearly independent of polynomials in S. We stop when $|S| = r$. To check if $S \cup \{f_i\}$ is linearly independent, using Lemma 3.14, we check if $f_j|_{\mathcal{H}} \in \mathbb{F}^h$ are independent.

For every $f_i \notin S$, we can find constants $\alpha_{ij} \in \mathbb{F}$ such that $f_i|_{\mathcal{H}} := \sum_{j \in S} \alpha_{ij} \cdot f_j|_{\mathcal{H}}$. From Lemma 3.14, we know $f_i = \sum_{j \in S} \alpha_{ij} \cdot f_j$. Thus, if we order elements of S as $f_{S1}, \ldots, f_{Sr}$, we obtain a matrix $M \in \mathbb{F}^{t \times r}$ using $M_{ij} := \alpha_{ij}$ such that

$$M \begin{pmatrix} f_{S1} \\ f_{S2} \\ f_{S3} \\ \vdots \\ f_{Sr} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_t \end{pmatrix}$$

and the left kernel of M is V, whose basis we can compute easily having access to M. To see this, consider any $v := (v_1, \ldots, v_t)$ such that $vM = 0$. Then, we know for every $j \in [r]$, $\sum_{i=1}^{t} \alpha_{ij} v_i = 0$. Multiplying the j-th equation with f_{Sj}, and adding gives $\sum_{j=1}^{r} \left(\sum_{i=1}^{t} \alpha_{ij} v_i\right) \cdot f_{Sj} = 0$, which is $\sum_{i=1}^{t} v_i \left(\sum_{j=1}^{r} \alpha_{ij} f_{Sj}\right) = \sum_{i=1}^{t} v_i f_i = 0$, and hence $v \in V$. □

We will now consider the set-multilinear version of essential variables.

Definition 9 (Set-multilinear Essential variables)**.** *The* number of essential variables *in X_i of a set-multilinear polynomial $f(X_1, \ldots, X_d)$ is the smallest t_i for which there exists an invertible linear transformation $A_i \in \mathbb{F}^{(|X_i| \times |X_i|)}$ such that $f(X_1, \ldots, A_i \cdot X_i, \ldots, X_d)$ depends on only t_i variables of X_i.*

We will need the set-multilinear version of the above lemma. It was observed in [PSV24], and simply follows from the above by considering the spaces corresponding to each variable part.

Remark 3.16. *Observe that Theorem 3.12 required the $\text{char}(\mathbb{F}) > d$ or $\text{char}(\mathbb{F}) = 0$, but the same is not true for the lemma below. This is a result of the fact that each transformation A_j we are applying only acts individually on each variable part X_j, and as the input polynomial f is set-multilinear, f is linear in each variable part separately. Therefore, the characteristic of the underlying field is no longer an issue.*

Lemma 3.17. *Let d, k be positive integers. Consider a set-multilinear degree-d polynomial $f(x) \in \mathbb{F}[\bar{x_1}, \ldots, \bar{x_d}]$ over variables $X_1, \ldots, X_d$, $|X_i| = n_i$ and $N = \sum_{i=1}^{d} n_i$, with a_i essential variables in variable part X_i computable by a depth-3 set-multilinear circuit $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$. Note that $a_i \leq k$. Then, there is a deterministic algorithm that takes as input black-box access to f, runs in time $\mathrm{poly}(N, d) \cdot k^{O(\log k)}$, and outputs a set of invertible matrices $A_1, \ldots, A_d$ where $A_i \in \mathbb{F}^{(n_i \times n_i)}$ such that $f(A_1 \cdot X_1, \ldots, A_d \cdot X_d)$ depends only on the first a_i variables of X_i for all $i \in [d]$.*

Proof. For every $i \in [d]$, consider the spaces V_i defined as follows:

$$V_i := \left\{ v \in \mathbb{F}^{n_i} : \sum_{j=1}^{n_i} v_j \frac{\partial f}{\partial x_{i,j}} = 0 \right\}$$

As f can be computed by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit, $\frac{\partial f}{\partial x_{i,j}}$ can also be computed by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit for any $x_{i,j}$ as a result of set-multilinearity. Let $\dim(V_i) = t_i$. Consider $\mathcal{H}$ as hitting set for set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k^2)$ circuits from Corollary 3.5. Observe that f is multilinear, $\frac{\partial f}{\partial x_{i,j}} = f|_{x_{i,j}=1} - f|_{x_{i,j}=0}$. Using Lemma 3.15, with $\left\{ \frac{\partial f}{\partial x_{i,j}} : j \in [n_i] \right\}$, $\mathcal{H}$, we can obtain a basis for V_i in deterministic $\mathrm{poly}(N, n_i, d, |\mathcal{H}|)$ time.

Extend the basis of V_i to a basis of $\mathbb{F}^{n_i}$, and let A_i be an invertible matrix formed with vectors of this basis as columns such that the last t_i columns form the chosen basis of V_i. Set

$$g = f(A_1 \cdot \bar{x_1}, \ldots, A_d \cdot \bar{x_d})$$

By the chain rule, we have

$$\frac{\partial g}{\partial x_{i,j}} = \left(\sum_{k=1}^{n_i} (A_i)_{k,j} \frac{\partial f}{\partial x_{i,k}} \right) (A_1 \cdot \bar{x_1}, \ldots, A_d \cdot \bar{x_d})$$

As the j-th column of A_i ($(A_i)_{*,j}$) lies in V_i (subspace of $\partial f^{\perp}$) for $j > n_i - t_i$, we have $\left(\sum_{k=1}^{n_i} (A_i)_{k,j} \frac{\partial f}{\partial x_{i,k}} \right) = 0$ and hence $\partial_{x_{i,j}} g = 0$. Since g remains set-multilinear, it depends only on the first $n_i - t_i$ variables of each X_i.

Conversely, any invertible set-multilinear substitution eliminating s variables from X_i gives s linearly independent vectors in V_i, by the same chain-rule argument. Thus $s \leq t_i$, and hence $a_i = n_i - t_i$. As $a_i \leq k$, we have $|\mathcal{H}| = 2^{O(\log^2 k^2)} \cdot \mathrm{poly}(N) = 2^{O(\log^2 k)} \cdot \mathrm{poly}(N) = k^{O(\log k)} \cdot \mathrm{poly}(N)$. Thus, total running time is $\mathrm{poly}(N, d) \cdot k^{O(\log k)}$. □

4 Reconstructing Low Rank Circuits

We first handle the case when the rank of the input set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C is such that $\mathrm{rank}(\mathrm{sim}(C)) < 9k$. As in Theorem 3.3, let $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ be the complexity of finding a solution of m polynomial equations over n variables of degree at most d. Recall that for $\mathbb{F} = \mathbb{R}$ and $\mathbb{C}$, such that the coefficients of the polynomials are in $\mathbb{Q}$, with bit complexity upper bounded by τ, $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = \mathrm{poly}(m, \tau) \cdot d^{O(n)}$ deterministic time. For $\mathbb{F} = \mathbb{F}_q$, $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = \mathrm{poly}(m, \log q) \cdot d^{O(n)}$ randomized time with solution in $\overline{\mathbb{F}}$.

Lemma 4.1. *Let $\mathbb{F} = \mathbb{R}$, $\mathbb{C}$, or $\mathbb{F}_q$. Let $f \in \mathbb{F}[X_1, \ldots, X_d]$ be a degree-d, n-variate set-multilinear polynomial ($n = \sum_{j \in [d]} (|X_j|)$) computable by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C such that $\mathrm{rank}(\mathrm{sim}(C)) < 9k$.*

- *If $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, and C has constants in $\mathbb{Q}$, with bit complexity upper bounded by τ, then there exists a deterministic algorithm that knows n, k, d, τ, makes blackbox queries to f, runs in time $\mathrm{poly}(n, d, \tau) \cdot k^{O(\log k)} + \mathrm{Sys}_{\mathbb{F}}(9k^2, k^{9k}, 9k)$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\mathbb{F}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\mathbb{F}$.*
- *If $\mathbb{F} = \mathbb{F}_q$ then there exists a randomized algorithm that knows n, k, d, makes blackbox queries to f, runs in time $\mathrm{poly}(n, d, \log q) \cdot k^{O(\log k)} + \mathrm{Sys}_{\mathbb{F}}(9k^2, k^{9k}, 9k)$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\overline{\mathbb{F}}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\overline{\mathbb{F}}$.*

Proof. The main idea is to reduce the number of variables in each variable part (by applying invertible linear transformations), such that the number of variables at the end equals the rank, and at the same time we have preserved the set-multilinear structure. We then construct a system of polynomial equations whose variables are the coefficients of the variables in the linear forms in the circuit, and such that the solutions correspond to set-multilinear depth-3 representations of the input polynomial. Solving this system gives us what we want.

We now describe how to implement the above strategy. We first show that WLOG we can assume the algorithm knows k'. This is because the algorithm can iterate over $\mathbb{N}$, starting from $k' = 1$ and increasing k' until the algorithm is successful, which can be detected with a PIT from Corollary 3.5. So now onwards we will assume k' is known.

Let $C^{\min}$ be a representation of f as a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit. Note that blackbox access to C is the same as blackbox access to $C^{\min}$. Thus using Corollary 3.11, we can get blackbox access to $\mathrm{sim}(C^{\min})$ and the $\gcd(C^{\min})$ in time $k^{O(\log k)} \cdot \mathrm{poly}(n)$. So, we will only focus on reconstructing $\mathrm{sim}(C^{\min})$ and multiply the gcd at the end. Therefore, moving forward we will assume $C^{\min} = \mathrm{sim}(C^{\min})$. Let us assume $C^{\min}$ has a representation as follows:

$$C^{\min} = \sum_{i \in [k']} T_i = \sum_{i \in [k']} \prod_{j \in [d]} \ell_{i,j},$$

where $\ell_{i,j} \in \mathbb{F}[X_j]_1$ and $\gcd(T_1, \ldots, T_{k'}) = 1$. For every $j \in [d]$, let

$$b_j := \dim\left(\mathrm{span}(\ell_{1,j}, \ldots, \ell_{k',j})\right)$$

As $C^{\min}$ is set-multilinear, the upper bound on the rank translates to

$$\mathrm{rank}(C^{\min}) = \sum_{j \in [d]} b_j < 9k$$

Note that we assumed τ was an upper bound on bit complexity of C, and this might not be true of $C^{\min}$. However since C has bounded bit complexity, thus evaluations of C, evaluations of $C^{\min}$, evaluations of f as well as coefficients of f all have bit complexity bounded by $\mathrm{poly}(n, d, \tau)$.

Using the set-multilinear essential variable computation of Lemma 3.17 in time $k^{O(\log k)} \cdot \mathrm{poly}(n, d, \tau)$, we can compute $a_1, \ldots, a_d$ such that a_i is the number of essential variables of variable part X_i. Observe $a_i \leq b_i$, and hence $\sum_{j \in [d]} a_j \leq \sum_{j \in [d]} b_j < 9k$. For each variable part $j \in [d]$, it also gives us an invertible linear transformation A_j on X_j such that, after the transformation, $C^{\min}$ depends only on the first a_j variables in the j-th variable part. Let

$$g(\overline{x}_1, \ldots, \overline{x}_d) := C^{\min}(A_1\overline{x}_1, \ldots, A_d\overline{x}_d)$$

Note g only depends on the first a_j variables of X_j. Therefore, g has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation of the form

$$g = \sum_{i\in[k']} \prod_{j\in[d]} \left(\sum_{m\in[a_j]} \alpha_{i,j,m} x_{j,m} \right)$$

Our goal will be to recover any one valid choice of the coefficients $\alpha_{i,j,m}$. To do this, we will interpolate g to write it as a sum of monomials, and then set up a system of polynomial equations in variables $\alpha_{i,j,m}$ by equating the coefficients of each monomial on the two sides of the equation.

As g is set-multilinear, and with the total number of variables in each variable part at most $k' \leq k$, we have the number of monomials at most k^d. As the circuit is simple and we have $\text{rank}(\text{sim}(C)) < 9k$, we also have $d < 9k$. Therefore, the number of monomials is at most k^{9k}. Thus, we can interpolate g in time $\text{poly}(d, k^{9k})$ by setting up an appropriate linear system with an appropriate number of evaluations.

Once we have monomial access to g, we can set up the system of equations by equating the coefficient of any monomial in terms of $\alpha_{i,j,m}$ to the obtained coefficient of g. These would give us at most k^{9k} polynomial equations in $k \cdot \sum_{j\in[d]} a_j < 9k^2$ variables and degree $d < 9k$. Using Theorem 3.3, we can find a solution to it in time $\text{Sys}_{\mathbb{F}}(9k^2, k^{9k}, 9k)$. We then use $A_1^{-1}, \ldots, A_d^{-1}$ as

$$C^{\min} = g(A_1^{-1}\overline{x}_1, \ldots, A_d^{-1}\overline{x}_d)$$

and multiply $\gcd(C)$ into the obtained circuit, and thus get a circuit for f in time $\text{poly}(n, d) \cdot k^{O(\log k)} + \text{Sys}_{\mathbb{F}}(9k^2, k^{9k}, 9k)$.

□

5 Reconstructing Low Degree Circuits

Now, we will show how to reconstruct circuits where the set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit is such that $\deg(C) \leq 5k$. As in Theorem 3.3, let $\text{Sys}_{\mathbb{F}}(n, m, d)$ be the complexity of finding a solution of m polynomial equations over n variables of degree at most d. For $\mathbb{F} = \mathbb{R}$ and $\mathbb{C}$, $\text{Sys}_{\mathbb{F}}(n, m, d) = \text{poly}(m, \tau) \cdot d^{O(n)}$ deterministic time. For $\mathbb{F} = \mathbb{F}_q$, $\text{Sys}_{\mathbb{F}}(n, m, d) = \text{poly}(m, \log q) \cdot d^{O(n)}$ randomized time with solution in $\overline{\mathbb{F}}$.

Lemma 5.1. *Let $\mathbb{F} = \mathbb{R}$, $\mathbb{C}$, or $\mathbb{F}_q$. Let $f \in \mathbb{F}[X_1, \ldots, X_d]$ be a degree-d, n-variate set-multilinear polynomial ($n = \sum_{j\in[d]}(|X_j|)$) computable by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C such that $d \leq 5k$.*

- *If $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, and C has constants in $\mathbb{Q}$, with bit complexity upper bounded by τ, then there exists a deterministic algorithm that knows n, k, d, τ, makes blackbox queries to f, runs in time $\text{poly}(n, d, \tau) \cdot k^{O(k^2)} \cdot \text{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\mathbb{F}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\mathbb{F}$.*
- *If $\mathbb{F} = \mathbb{F}_q$ then there exists a randomized algorithm that knows n, k, d, makes blackbox queries to f, runs in time $\text{poly}(n, d, \log q) \cdot k^{O(k^2)} \cdot \text{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\overline{\mathbb{F}}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\overline{\mathbb{F}}$.*

Proof. Again, observe that WLOG we can assume the algorithm knows k'. This is because the algorithm can iterate over $\mathbb{N}$, starting from $k' = 1$ and increasing k' until the algorithm is successful, which can be detected with a PIT from Corollary 3.5. So, now onwards, we assume that k' is known.

Let $C^{\min}$ be a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ computing f over $\mathbb{F}$ for $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, while $C^{\min}$ will be a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ computing f over $\overline{\mathbb{F}_q}$. WLOG, we will assume that C doesn't have subsets of gates that sum to zero either. If it did, we will just remove the subset of gates from C, and swap k with the smaller top fan-in.

The main idea is to consider a projection of the circuit $C^{\min}$ (or equivalently f), where only a few variable parts are kept alive and the rest are set to random (or pseudorandom) constants. This projection (which we call $C^{\min}_S$) will make the rank small enough to use Lemma 4.1 to reconstruct some representation (need not be unique) of the projected circuit (which we call C'_S) and then use this representation to learn a significantly large subset of linear forms in the original circuit $C^{\min}_S$. We do this by observing that there is an overlap in linear forms between $C^{\min}_S$ and C'_S using the rank bound (Theorem 3.9). We fix these linear forms which are common in C'_S and in $C^{\min}_S$ (by a "brute force" guess of the correct ones). We show that the remaining ones lie in a low-dimensional space. To be completely precise, we will choose to give up on the linear forms that have bit complexity greater than $\text{poly}(\tau)$[4] (We will give up on linear forms with coefficients in extensions of $\mathbb{F}_q$ after normalization). The existence of the circuit representation C with bounded bit complexity will imply (again using rank bounds Theorem 3.9) that the projection of C (i.e. C_S) and $C^{\min}_S$ have a significant overlap in linear forms. Thus, we will only assume we learn those linear forms in $C^{\min}_S$ that also appear in C_S, and that will suffice. Thus, at this point we would have learnt several linear forms from $C^{\min}_S$, all with $\text{poly}(\tau)$ bit complexity (be in $\mathbb{F}_q[x_1, \ldots, x_n]_1$ for $\mathbb{F} = \mathbb{F}_q$), and all the ones we don't learn lie in a low-dimensional space.

Now we undo some part of the projection and make more variable parts alive. We use a variation of Lemma 4.1 to learn even more linear forms. We keep doing this until all d variable parts have been made alive (or unprojected), and the final polynomial system that we solve then yields a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation of the full polynomial.

We now show how to implement the above proof strategy.

Getting access to the simple circuit and variable reduction: Using Corollary 3.11, we can get blackbox access to $\text{sim}(C^{\min})$ and the $\gcd(C^{\min})$ in time $k^{O(\log k)} \cdot \text{poly}(n)$. So, we will only focus on reconstructing $\text{sim}(C^{\min})$ and multiply the gcd at the end. Therefore, moving forward, we will assume $C^{\min} = \text{sim}(C^{\min})$, just as in Lemma 4.1. Let us fix a representation $C^{\min}$ as follows:

$$C^{\min} = \sum_{i \in [k']} T_i = \sum_{i \in [k']} \prod_{j \in [d]} \ell_{i,j},$$

where $\ell_{i,j} \in \mathbb{F}[X_j]_1$.

Using the set-multilinear essential variable computation of Lemma 3.17, we can compute $a_1, \ldots, a_d$, where a_i is the number of essential variables of f in X_i. For each variable part $j \in [d]$, it also gives us an invertible linear transformation M_j on X_j such that, after the transformation, f depends only on the first a_j variables in the j-th variable part. We apply this invertible transformation as in Lemma 4.1. Then the resulting polynomial will only depend on the a_j of the X_j variables (even though the individual linear forms may not), but we just set all the other variables in X_j to 0 and observe that it does not change the polynomial being computed (after the invertible transformation). At the end, after learning the circuit, we will undo the invertible transformation to recover the circuit computing the original polynomial. Thus, moving forward we will assume we have computed the invertible transformations for each variable part, set certain variables to 0, and

[4] what poly we use depends on how we are representing the coefficients of C. If we normalized all linear forms in C to have the leading variable have coefficient 1, then we could just use τ

hence now each variable part depends only on a_j variables. In particular, for every $j \in [d]$,

$$a_j = \dim\left(\text{span}(\ell_{1,j}, \ldots, \ell_{k',j})\right).$$

Also, note that each $a_j \leq k' \leq k$.

Projecting to a low rank circuit: We first observe that if $\sum_{j\in[d]} a_j < 9k$, then we have $\text{rank}(\text{sim}(C))$ after the transformation of $A_1, \ldots, A_d$ is less than $9k$. Thus, we can use Lemma 4.1 to reconstruct a circuit. Therefore, moving forward, we will assume $\sum_{j\in[d]} a_j \geq 9k$. For a set $S \subseteq [d]$, define $\text{rank}(S) := \sum_{j\in S} a_j$. We first greedily choose a minimal set $S \subseteq [d]$ whose rank is at least $8k$. Thus, if any element i of S is excluded from S, then the rank of the smaller set is less than $8k$. Since the variable part corresponding to i has rank at most k (this is actually k' but it will not matter), therefore it follows that the rank of S can be at most $9k$. In other words,

$$8k \leq \sum_{j\in S} a_j < 9k.$$

Since $C^{\min}$ is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ representation of f with a minimum top fan-in, no non-empty subset of the product gates sums to 0. We will find a substitution of field constants into all variables in the parts in $[d] \setminus S$ which preserves this property. Note that such a substitution can be obtained in $2^{O(k+\log^2 k)} \cdot \text{poly}(n, d)$ time via a hitting set generator for set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits as in Corollary 3.6[5]. We will call this projected polynomial after the substitution f_S. Given the representation $C^{\min}$ of f, we get a representation $C^{\min}_S$ computing f_S:

$$C^{\min}_S = \sum_{i\in[k']} \gamma_i \prod_{j\in S} \ell_{i,j},$$

where every γ_i is nonzero (and additionally no subset of the product gates sums to 0). Since $C^{\min}$ was simple, so is $C^{\min}_S$. By design, the rank of $C^{\min}_S$ is less than $9k$ and we have blackbox access to it. Hence we can use Lemma 4.1, to reconstruct a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(\bar{k})$ circuit C'_S computing f_S such that $\bar{k} \leq k' \leq k$. We output fail if $\bar{k} > k'$.

Learning some linear forms in $C^{\min}_S$ Now, we have learnt C'_S, but we want to learn $C^{\min}_S$ because C'_S may not be extendable to a representation of $C^{\min}$. However, we will see that C'_S still has a lot of information that we care about. Notice that $C^{\min}_S - C'_S \equiv 0$ and hence is a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(\bar{k} + k')$ identity. It is not necessarily a minimal identity. There may be subsets of the product gates which individually sum to 0. These give lots of minimal sub-identities. We consider a partition of all the product gates into minimal identities $D_1, \ldots, D_q$, and let r_h be the number of product gates in D_h. For each $h \in [q]$, Theorem 3.9 gives

$$\text{rank}(\text{sim}(D_h)) \leq 2r_h - 2.$$

Let $\bar{W}_h$ be the span of the linear forms in $\text{sim}(D_h)$ and let $W = \text{span}(\bar{W}_1 \cup \bar{W}_2 \cup \ldots \cup \bar{W}_q)$. Since $\sum_{h\in[q]} r_h \leq k + k' \leq 2k$, we have

$$\dim(W) \leq \sum_{h\in[q]} (2r_h - 2) \leq 4k - 2.$$

[5]To be more precise, we will iterate over all elements of a hitting set that has the required properties, and for the right choice of element our algorithm will succeed. The incorrect choices will lead to an eventual error that will get pruned out.

Note that every product gate of $C_S^{\min}$ belongs to some minimal identity D_h. Also, D_h must necessarily contain some gate of C'_S. These come from the fact that no subset of gates vanish in $C_S^{\min}$ and C'_S has minimum top fan-in. Fix any such gate of $C_S^{\min}$ that belongs to D_h. Every linear form in this gate which does not lie in $\bar{W_h}$ must divide $\gcd(D_h)$. It therefore also appears in every gate of C'_S that belongs to D_h. It follows that every linear form that appears in $C_S^{\min}$ and which does not lie in W appears as a linear form in C'_S.

Similarly, observe $C^{\min} - C$ is also a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k'+k)$ identity, and therefore by the same arguments the linear forms that are in $C^{\min}$ but not in C (up to multiplication by a scalar) must lie in a space W_0 of dimension at most $2(k'+k)-2 \leq 4k-2$.

It follows that if we look at the set of all linear forms appearing in $C_S^{\min}$ and remove all those that lie in $W_0 \cup W$, then the rest of them must appear as linear forms in C'_S and also in C. Moreover, since they are included in C, they must have bit complexity bounded by τ (be in $\mathbb{F}_q[x_1,\ldots,x_n]_1$ for $\mathbb{F}=\mathbb{F}_q$) after normalization so that the leading coefficient is 1. Let us call this set of linear forms, the *good* linear forms, and notice that they all appear in C'_S which was the circuit that was learned.

Now, we do not know which of the linear forms that we have learned in C'_S are the *good* ones. We also don't know which gates of $C_S^{\min}$ the good ones will eventually correspond to. So, we "guess" all of this. In other words, we brute-force search over all possibilities of good linear forms and for each good one, which gates of $C_S^{\min}$ they are contained in. One of the guesses will lead to the correct reconstruction, which we will be able to identify at the end of the algorithm. The brute-force search works as follows. First, for each gate in $C_S^{\min}$, we guess a corresponding gate in C'_S which shows up in the same minimal identity D_h. There are $\bar{k}^{k'} \leq k^k$ such possibilities for guesses. There are at most $k' \cdot |S|$ linear forms in $C_S^{\min}$ that we are dealing with, which can be correctly learned from C'_S.

Therefore, we can have $2^{k' \cdot |S|}$ possibilities of which ones got learnt from a gate of C'_S. Therefore, the total number of guesses is at most $k^k \cdot 2^{k' \cdot |S|}$. As $|S| \leq d \leq 5k$, we have $k^{O(k^2)}$ total guesses. One of these is correct. We will run the algorithm for each guess in parallel; one of them will lead to the correct reconstruction, and we will reject the rest. Thus, for now we assume that we made the correct guesses and learnt a large collection of linear forms in $C_S^{\min}$. We will normalize these linear forms by dividing by a constant such that the first non-zero coefficient becomes 1. This prevents the bit-complexity of the coefficients from blowing up beyond $\mathrm{poly}(\tau)$ (and ensures coefficients are in $\mathbb{F}_q$ when $\mathbb{F}=\mathbb{F}_q$). The ones that we do not know lie in the low-dimensional space $\mathrm{span}(W \cup W_0)$ whose dimension is at most $8k-2$. We call this the end of iteration 1. We relabel S as S_1, and thus $C_S^{\min}$ is now $C_{S_1}^{\min}$ and the learned circuit is C'_{S_1}. Let the set of linear forms learned (i.e. the good ones) be L_1.

The plan is to recursively learn more linear forms by undoing the projection on some of the variable parts. In other words, we will grow S_1 to a superset S_2, learn more linear forms, and keep iterating till eventually we cover all variable parts.

Recursively learning more linear forms after fixing the learned linear forms: We assume the following invariants are true before each recursive call (and indeed in the correct guess branch this will be the case). Assume we are at the end of the i'th iteration. We will assume:

- We have correctly identified or learned a subset of linear forms L_i appearing in $C_{S_i}^{\min}$ (and hence in $C^{\min}$).
- All unknown linear forms in $C_{S_i}^{\min}$ lie in a space $\mathrm{span}(W_0 \cup W_i)$ where each of W_0 and W_i has dimension at most $4k-2$.

The plan will be to grow S_i to $S_{i+1} \subseteq [d]$ while maintaining the above properties and keeping the runtime efficient (by only solving polynomial systems in few variables).

We will show how to grow S_i by at least one (unless $S_i = [d]$), and then recurse this process at most $d = O(k)$ times.

For $j \in S_i$, let $W_{i,j}$ correspond to the subspace of $\mathrm{span}(W_0 \cup W_i)$ defined by the span of the unknown linear forms in $C^{\min}_{S_i}$ depending only on the variables in X_j. Define $b_{i,j} := \dim(W_{i,j}) \leq k$. We guess each of the $b_{i,j}$'s, i.e. try all $(k+1)^{|S_i|} = k^{O(k)}$ possible values for them and assume we are in the case where we guessed them correctly (the wrong guesses will eventually get rejected by the algorithm). A bound on the dimension of $\mathrm{span}(W_0 \cup W_i)$, means we have

$$\sum_{j \in S_i} b_{i,j} \leq 8k - 4.$$

Thus all unknown linear forms span a space of dimension exactly $\sum_{j \in S_i} b_{ij}$.

We will now greedily add more variable parts to S_i and grow it to S_{i+1} such that either

$$8k \leq \sum_{j \in S_i} b_{ij} + \sum_{j \in (S_{i+1} \setminus S_i)} a_j < 9k.$$

or $S_{i+1} = [d]$. As $\sum_{j \in S_i} b_{ij} \leq \dim(\mathrm{span}(W_0 \cup W_i)) \leq 8k - 4$, we add at least one variable part in each iteration unless S_i already equals $[d]$.

If $S_i = [d]$, then we do not need to grow any more and we simply output C'_{S_i} as the final learned circuit.

Let $f_{S_{i+1}}$ be the polynomial computed by $C^{\min}_{S_{i+1}}$. We would like to learn linear forms in $C^{\min}_{S_{i+1}}$ that are a superset of L_i, which were the linear forms of $C^{\min}_{S_i}$ learnt in the previous iteration. Thus we do not just want to learn any set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation of $f_{S_{i+1}}$, but a representation which respects the linear forms L_i learnt so far, and the positions of these linear forms in the different gates. For this, we set up a system of polynomial equations which takes this information into account. Moreover, since all the linear forms learnt so far have bit complexity at most $\mathrm{poly}(\tau)$ (or are in $\mathbb{F}_q$ when $\mathbb{F} = \mathbb{F}_q$), this ensures that the polynomial system we will set up will continue to have all coefficients having bit complexity at most $\mathrm{poly}(\tau, n, d, k)$.

For every $j \in S_i$, we know that there exist linear forms (which we will try to learn) $u_{j,1}, \ldots, u_{j,b_{i,j}} \in \mathbb{F}[X_j]$ such that every unknown linear form of $C^{\min}_{S_{i+1}}$ that depends on the variables X_j will be a linear combination of $u_{j,1}, \ldots, u_{j,b_{i,j}}$. For $j \in S_{i+1} \setminus S_i$, observe that every linear form in $C^{\min}_{S_{i+1}}$ that depends on the variables of X_j is a linear form in only the a_j variables of X_j (see Lemma 4.1.)

We now set up an appropriate system of polynomial equations to learn a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation of $f_{S_{i+1}}$ which takes into account the linear forms in L_i and their positions in the product gates. Observe that the linear forms in L_i are all by assumption in C and hence have bounded coefficients (or are in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$) after normalization.

Each $u_{j,i}$ can be expressed as $u_{j,i} := \sum_{t=1}^{a_j} \alpha_{i,j,t} x_{j,t}$ where $\alpha_{i,j,t}$ are unknown variables in our system. For each $j \in S_i$, the number of variables $\alpha_{i,j,t}$ of $u_{j,1}, \ldots, u_{j,b_{i,j}}$ will be at most $a_j \cdot b_{i,j} \leq k \cdot b_{i,j}$. Summing it up for all $j \in S_i$, gives us an upper bound of $k \cdot (8k - 4) = O(k^2)$. For $j \in S_i$ and $(v \in [k])$, each linear form l_{vj} that is not in L_i, can be written as a linear combination of $u_{j,1}, \ldots, u_{j,b_{i,j}}$, i.e. $l_{v,j} = \sum_{t=1}^{b_{i,j}} \beta_{v,j,t} u_{j,t}$, and these $\beta_{v,j,t}$ are also unknown variables in our system. Therefore, for each fixed j, the number of β variables is at most $k \cdot b_{i,j}$. Summing it up over all $j \in S_i$, the number of β variables is at most $k \cdot (\sum_{j \in S_i} b_{i,j}) \leq k(8k - 4) = O(k^2)$.

For $j \in S_{i+1} \setminus S_i$, $v \in [k]$, define the linear form depending on variables X_j in gate T_v as $l_{vj} := \sum_{t=1}^{a_j} \gamma_{v,t,j} x_{j,t}$, where $\gamma_{v,t,j}$ will be additional unknown variables in our system. The number of γ variables will be $k \cdot (\sum_{j \in (S_{i+1} \setminus S_i)} a_j) \leq 9k^2$. Therefore, the total number of variables in the

system (which includes the α, β and γ variables) is $O(k^2)$. For every already "learned" linear form (in L_i), we just assume it is already present in the correct spot in the circuit representation. We interpolate to get a monomial representation of $f_{S_{i+1}}$ and compare the monomial coefficients with the corresponding coefficients coming from the unknown variables we have introduced and thus set up a system of polynomial equations. Note that every solution exactly corresponds to a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation for $f_{S_{i+1}}$ which respects the learned linear forms in the i'th iteration.

The number of equations we get will be the number of monomials of $f_{S_{i+1}}$, which is at most $k'^d = k^{O(k)}$, and the degree of the system is at most $O(d) = O(k)$. Thus, we can solve this system using Theorem 3.3 in time $\mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$. Thus, we learn a circuit $C'_{S_{i+1}}$ computing $f_{S_{i+1}}$ with the linear forms from L_i in their correct positions.

If $S_{i+1} = [d]$, then we can stop and output $C'_{S_{i+1}}$.

Otherwise, we observe that $C^{\min}_{S_{i+1}} - C'_{S_{i+1}}$ will be an identity computed by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(2k)$ circuit. Similarly to what was described earlier with S_1, we note that this identity might be a sum of multiple minimal identities. Notice that all linear forms, except those in the simple parts of the identities, will be present in both $C^{\min}_{S_{i+1}}$ and $C'_{S_{i+1}}$. Using Theorem 3.9, we can show the linear forms in the simple parts of the identities together span at most a $4k-2$ dimensional space (just as we had done with S_1). By iterating over all subsets of linear forms in $C'_{S_{i+1}}$ (just as we had done for C'_{S_1}), we will assume that we have correctly identified the subset corresponding to the gcd of all minimal identities. We look at the linear forms belonging to all the gcds. We discard all those linear forms that lie in W_0. All the remaining will also be contained in the circuit representation C and hence have bounded bit complexity $\mathrm{poly}(\tau)$ (or have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$). The resulting linear forms together with L_i will correspond to the new L_{i+1}. Before we add any linear form to L_{i+1}, we will normalize to make the leading coefficient 1. Once we "guess" the correct set L_{i+1}, we will also "guess" (by iterating) the correct gates of $C^{\min}_{S_{i+1}}$ it belongs to.

The remaining unlearned linear forms in C'_{S_1} must sit in the simple parts of the identities or in W_0. Let W_{i+1} be the space of dimension at most $4k-2$ which contains the linear forms in the simple parts of the identities.

Runtime analysis: Since each iteration grows the size of S by at least 1, there are at most $d \leq 5k$ iterations.

It remains to bound the total number of branches or "guesses" that need to be explored. Naively counting the branches seems to give a $k^{O(k^3)}$ bound, but we will do the counting more carefully. Note that in each iteration i we first "map" or identify each gate of $C^{\min}_{S_i}$ with a gate of C'_{S_i}. Once we did this mapping, we guessed which subset of linear forms from each gate of $C^{\min}_{S_i}$ were the "correctly" learned linear forms. Once we identify this subset, then the linear forms are recovered by looking at the mapped gate of C'_{S_i}. Once all the iterations are completed, the final set of correctly identified linear forms is the union of correctly identified linear forms over all iterations. Thus, to specify each correctly identified linear form, we just need to specify the iteration in which it was learned and the "mapping" or the corresponding gate of C'_{S_i} it came from. Thus, the total number of branches is specified by the total number of "mappings" of gates across all iterations multiplied by the number of ways of specifying, for each linear form, which round it got identified in. The number of mappings of gates in one iteration is clearly at most k^k and hence across all iterations is at most $k^{O(k^2)}$. For a single linear form, there are $O(k)$ choices for specifying which iteration it got learned in, and thus over all $O(k^2)$ possible linear forms, this gives $k^{O(k^2)}$ branches corresponding to learning linear forms from the learned circuits. There was further branching

which corresponded to "guessing" the values of the b_i's at each iteration and finding the correct substitution for the variable parts not in S_i using the hitting-set generators. Both had another $k^{O(k)}$ number of branches for each iteration and hence $k^{O(k^2)}$ over all iterations.

We will only bound the run-time and bit complexity (field extension degree of output coefficients) in the correct branch lemma. We will have a timeout/stop condition for all branches matching the upper bound on the running time of the correct branch.

Now for the correct guess branch, in each iteration we need to solve a system of polynomial equations with $k^{O(k)}$ equations, in $O(k^2)$ variables and of degree at most $O(k)$. This can all be done in time $\mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$. One might observe that these solutions have higher bit complexity than the input polynomials. But in the correct-guess branch, we only use the coefficients we found in correct linear forms, which are scalar multiples of the circuit's linear forms. Hence, when we divide by the first non-zero coefficient of the linear forms, the linear form gets scaled to almost the same bit complexity (having coefficients in $\mathbb{F}_q$) as the linear forms in the circuit. Thus, bit-size does not blow up, and coefficients continue to lie in $\mathbb{F}_q$, as we take recursive steps. This will work for all linear forms learnt except the ones learnt in the last iteration when $S_i = [d]$, as there we have no correspondence with the input circuit, but these coefficients are not used again in the system solving, just in the output.

Moreover, right at the beginning we used Lemma 3.17 to do a variable reduction, and this is a further $\mathrm{poly}(n,d) \cdot 2^{O(\log^2 k)}$.

Finally, we use A_j^{-1} to undo the variable reduction and also multiply back $\gcd(C)$ that was removed at the beginning. For every resulting circuit (over all branches), we run a black-box PIT against the input f. The correct branch will lead to the correctly learned circuit, and the PIT using Corollary 3.5 discards all incorrectly learned circuits.

Thus, the total running time is

$$\mathrm{poly}(n,d) \cdot k^{O(k^2)} \cdot \mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$$

□

6 Reconstructing High Degree Circuits

In this section, we will show how to reconstruct the high-degree circuits. As in Theorem 3.3, let $\mathrm{Sys}_{\mathbb{F}}(n,m,d)$ be the complexity of finding a solution of m polynomial equations over n variables of degree at most d. For $\mathbb{F} = \mathbb{R}$ and $\mathbb{C}$, $\mathrm{Sys}_{\mathbb{F}}(n,m,d) = \mathrm{poly}(m,\tau) \cdot d^{O(n)}$ deterministic time. For $\mathbb{F} = \mathbb{F}_q$, $\mathrm{Sys}_{\mathbb{F}}(n,m,d) = \mathrm{poly}(m, \log q) \cdot d^{O(n)}$ randomized time with solution in $\overline{\mathbb{F}}$.

Lemma 6.1. *Let $\mathbb{F} = \mathbb{R}$, $\mathbb{C}$, or $\mathbb{F}_q$. Let $f \in \mathbb{F}[X_1, \ldots, X_d]$ be a degree-d, n-variate set-multilinear polynomial ($n = \sum_{j\in[d]}(|X_j|)$) computable by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuit C such that $d > 5k$.*

- *If $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, and C has constants in $\mathbb{Q}$, with bit complexity upper bounded by τ, then there exists a deterministic algorithm that knows n, k, d, τ, makes blackbox queries to f, runs in time $\mathrm{poly}(n,d,\tau) \cdot k^{O(k^2)} \cdot \mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\mathbb{F}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\mathbb{F}$.*
- *If $\mathbb{F} = \mathbb{F}_q$ then there exists a randomized algorithm that knows n, k, d, makes blackbox queries to f, runs in time $\mathrm{poly}(n,d,\log q) \cdot k^{O(k^2)} \cdot \mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k))$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over $\overline{\mathbb{F}}$ computing f, where $k' \leq k$ is the smallest integer such that f has a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ representation over $\overline{\mathbb{F}}$.*

Proof. Again, observe that WLOG we can assume the algorithm knows k'. This is because the algorithm can iterate over $\mathbb{N}$, starting from $k' = 1$ and increasing k' until the algorithm is successful, which can be detected with a PIT from Corollary 3.5. So, now onwards, we assume that k' is known.

Let $C^{\min}$ be a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ computing f over $\mathbb{F}$ for $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, while $C^{\min}$ will be a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ computing f over $\overline{\mathbb{F}_q}$. WLOG, we will assume that C doesn't have subsets of gates that sum to zero either. If it did, we will just remove the subset of gates from C, and swap k with the smaller top fan-in.

As a first step, we will consider a projection of the circuit to only the first $5k$ variable parts by substituting appropriately chosen field constants into the other variable parts such that certain nonzeroness properties of the resulting circuit are preserved.

We will then reconstruct the resulting low-degree set-multilinear circuit using Lemma 5.1. We will show that the learned circuit will be "close" to the original circuit projected to the first $5k$ variable parts. We will use this closeness to show that we can recover *all* linear forms, which will have bit complexity $\mathrm{poly}(\tau)$ (are in $\mathbb{F}_q[x_1, \ldots, x_n]_1$, when $\mathbb{F} = \mathbb{F}_q$), from some $k+2$ of the variable parts, and then use the recovered linear forms to reduce the original reconstruction task to that of reconstructing circuits of smaller top fan-in.

Getting access to the simple circuit and variable reduction: Using Corollary 3.11, we can get blackbox access to $\mathrm{sim}(C^{\min})$ and the $\gcd(C^{\min})$ in time $k^{O(\log k)} \cdot \mathrm{poly}(n)$. So, we will only focus on reconstructing $\mathrm{sim}(C^{\min})$ and multiply the gcd at the end. Also, if $\deg(\mathrm{sim}(C)) \leq 5k$, we can use Lemma 5.1 to reconstruct the circuit. Therefore, moving forward, we will assume $C^{\min} = \mathrm{sim}(C^{\min})$, just as in Lemma 4.1. Let us fix a representation $C^{\min}$ as follows:

$$C^{\min} = \sum_{i \in [k']} T_i = \sum_{i \in [k']} \prod_{j \in [d]} \ell_{i,j},$$

where $\ell_{i,j} \in \mathbb{F}[X_j]_1$.

Using the set-multilinear essential variable computation of Lemma 3.17, we can compute $a_1, \ldots, a_d$, where a_i is the number of essential variables of f in X_i. For each variable part $j \in [d]$, it also gives us an invertible linear transformation M_j on X_j such that, after the transformation, f depends only on the first a_j variables in the j-th variable part. We apply this invertible transformation as in Lemma 4.1. Then the resulting polynomial will only depend on the a_j of the X_j variables (even though the individual linear forms may not), but we just set all the other variables in X_j to 0 and observe that it does not change the polynomial being computed (after the invertible transformation). At the end, after learning the circuit, we will undo the invertible transformation to recover the circuit computing the original polynomial. Thus, moving forward we will assume we have computed the invertible transformations for each variable part, set certain variables to 0, and hence now each variable part depends only on a_j variables. In particular, for every $j \in [d]$,

$$a_j = \dim\left(\mathrm{span}(\ell_{1,j}, \ldots, \ell_{k',j})\right).$$

Also, note that each $a_j \leq k' \leq k$.

Projecting to low degree circuit: Since $C^{\min}$ is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}$ representation of f with a minimum top fan-in, no strict subset of the product gates sums to 0.

We will find a substitution of field constants into all variables in the variable parts indexed $[d] \setminus [5k]$ which preserves this property. We actually find a set of evaluation points where one of the elements of the set does the job for us (and we can iterate over all possibilities), and we will assume for the purposes of exposition that we have found the "correct" substitution.

Note that such a substitution (or set of substitutions) can be obtained in $2^{O(k+\log^2 k)} \cdot \text{poly}(n,d)$ time via a hitting set generator for set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k)$ circuits as in Corollary 3.6. We will be recursively calling this algorithm for smaller fan-in and degree circuits, but we will use the projections of the same substitution, and therefore find it only in the first call. Let C_1 be the resulting circuit.

We can then use the algorithm for low-degree circuits (Lemma 5.1) to reconstruct a circuit C_1', computing the same polynomial as C_1. Note that the top fan-in of C_1' is at most k' and could be smaller. Let it be $\bar{k}$, s.t. $\bar{k} \leq k' \leq k$. We output fail if $\bar{k} > k'$. Thus C_1' has an expression of the following kind.

$$C_1' = \sum_{i \in [\bar{k}]} T_i' = \sum_{i \in [\bar{k}]} \prod_{j \in [5k]} \ell_{i,j}'$$

.

Learning some linear forms of $C^{\min}$ using C_1': We will first show that C_1' is "close" to C_1. To see this, observe that $C_1' - C_1$ is an identity computed by a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k+k')$ circuit. It might not be a minimal identity. Thus we decompose it as a sum of minimal identities, and analyze each identity respectively. These minimal identities are a partition of all the product gates, and let the partition be $D_1, \ldots, D_q$, and let r_h be the number of product gates in D_h. Let $S_h \subset [5k]$ be the set of variable parts on which $\text{sim}(D_h)$ depends. For each $h \in [q]$, Theorem 3.7 gives

$$|S_h| \leq r_h - 2,$$

i.e. the number of variable parts on which the simple part of the identity depends is at most $r_h - 2$. Note that such a bound can also be deduced from the rank bound (Theorem 3.9) but the degree bound directly gives slightly better parameters. Therefore, there are at least $5k - (r_h - 2)$ variable parts on which all gates in D_h are identical. Note that every product gate of C_1 belongs to some minimal identity D_h. Also, D_h must necessarily contain some gate of C_1'. This follows because no subset of gates vanishes in C_1, and C_1' has minimum top fan-in. Thus, every gate T_i from C_1 in this minimal identity will have another gate T_i' from C_1' such that they have the same linear forms on all except at most $r_h - 2$ variable parts. We do this for every minimal identity.

Notice that $|\cup S_h| \leq \sum_{h \in [q]} (r_h - 2) \leq (\sum_{h \in [q]} r_h) - 2 \leq 2k - 2$.

It follows that for any gate T_i in $C^{\min}$, if we only look at the product of linear forms in that gate corresponding to variable parts in $[5k] \setminus \cup S_h$, then this equals the corresponding product of linear forms from some gate T_j' in C_1'. Let $S = [5k] \setminus \cup S_h$. Observe that $|S| \geq 3k + 2$.

We observe that, as the circuit C with coefficients of bit-complexity τ (coefficients in $\mathbb{F}_q$) and $C^{\min}$ compute the same polynomial, $C - C^{\min}$ is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k+k')$ identity. Using the same arguments as above, we can conclude that the number of variable parts on which C and $C^{\min}$ didn't have the same linear forms (up to scalar factors) will be at most $k + k' - 2 \leq 2k - 2$. Removing these variable parts from S means that there are $k + 4 \geq k + 2$ variable parts in S. It follows that for any gate T_i in $C^{\min}$, if we only look at the product of linear forms in that gate corresponding to variable parts in S, then this equals the corresponding product of linear forms from some gate T_j' in C_1', all of which have bit complexity bounded by $\text{poly}(\tau)$ (have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$). Also, observe that because of our choice of the variable parts, these linear forms also divide gates in the circuit C.

In particular, for every gate from C_1 (or $C^{\min}$), there is another gate from C_1' such that restricted to the variable parts in S, the gates are identical (up to multiplication by a field constant), and the

constituent linear forms have poly(τ) bit complexity (have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$).

For the rest of the proof, we will assume we know this set S. We will also assume that we know for each T_i in C_1, which T_j in C_1' it corresponds to. In our actual implementation, we will have to iterate over all possible choices of subsets of $[5k]$ of size $k+2$, and all possible ways of finding the gate correspondences. There are at most $k^{O(k)}$ such choices to iterate over. The correct iteration would lead to the true reconstruction. The bad choices will get eliminated by a PIT check after the reconstruction. We will also scale all the linear forms by dividing by their first non-zero coefficients; therefore, all the correctly guessed linear forms will have coefficients of the same bit-complexity as the input circuit.

WLOG, let us say $S = [k+2]$.

Recall $C^{\min} = \sum_{i\in[k']} T_i = \sum_{i\in[k']} \prod_{j\in[d]} \ell_{i,j}$. For each $i \in [k']$, since we now assume we know the linear forms in gate T_i corresponding to the variable parts in $[k+2]$, let $A_i := \prod_{j\in[k+2]} \ell_{ij}$. Clearly, for all $i \in [k']$, $A_i | T_i$, and we have learned $A_1, \ldots, A_{k'}$. Also observe that for every gate $\overline{T}_j$ in C, there is an A_i, such that $A_i | \overline{T}_j$. As $\gcd(T_1, \ldots, T_{k'}) = 1$, it follows that $\gcd(A_1, \ldots, A_{k'}) = 1$ as well. Note that the A_i's need not be independent or even distinct.

Recursively regrouping the gates: We will recursively rewrite the circuit (by regrouping the gates) till we get it into a nice enough form to learn.

At each stage of the recursive procedure, we will have a representation of $C^{\min}$ with the following properties.

1. There is a partition of $[k']$ of the form $[k'] = D_1 \sqcup D_2 \sqcup \ldots \sqcup D_r$.
2. There is a set $S \subseteq [k+2]$ such that $|S| \geq r+2$.
3. There are polynomials $A_1, \ldots, A_r$, each a product of linear forms, such that A_i is known (or learned) by the algorithm. Each A_i is a product of $|S|$ linear forms, one from each variable part corresponding to S. Moreover for all $i \in [k']$ if $i \in D_j$ then A_j divides T_i. Thus $C^{\min}$ can be represented as
$$\sum_{i=1}^{r} A_i \cdot \left(\sum_{j\in D_i} (T_j / A_i) \right)$$
For $i \in [r]$, let $B_i := \left(\sum_{j\in D_i} (T_j/A_i) \right)$. Thus $C^{\min} = \sum_{i=1}^{r} A_i B_i$, where A_i are polynomials that have been learned and each B_i is a $\Sigma\Pi\Sigma_{\sqcup_j X_j}(|D_i|)$ circuit.

The high-level plan will be to recursively reduce r by combining some of the D_i and still maintaining the other required properties. In order to do this, we will have to modify the set S, and we will describe the details shortly. We will show how to do this r-reduction when, at the end of the previous iteration, the set $A_1, \ldots, A_r$ that is produced is linearly dependent. In the rest of the proof, there are two main components. When $A_1, \ldots, A_r$ are linearly dependent, we will show how to reduce r. When $A_1, \ldots, A_r$ are linearly independent, we will show how to use this independence to learn a $\Sigma\Pi\Sigma_{\sqcup_j X_j}[k']$ representation of C.

Note that we can never achieve the above 3 requirements with r being 1 (otherwise there will be a common GCD). Thus, the r-reduction process will eventually terminate with the linearly independent A_i. We now describe the process for reducing r.

The starting point of our recursive r-reduction process is with $r = k'$, $S = [k+2]$ and $D_i = \{i\}$. Now suppose that we have applied this procedure a few times and we have $[k'] = D_1 \sqcup D_2 \sqcup \ldots \sqcup D_r$, $S \subseteq [k+2], |S| \geq r+2$, and $C^{\min} = \sum_{i=1}^{r} A_i \cdot \left(\sum_{j\in D_i} (T_j/A_i) \right) = \sum_{i=1}^{r} A_i B_i$.

Also, we are in the situation where $A_1, \ldots, A_r$ are linearly dependent. Consider any minimal set $U \subseteq [r]$ such that A_i's for $i \in U$ are linearly dependent, i.e. there exists $c_i \in \mathbb{F}$, such that

$$\sum_{i \in U} c_i A_i = 0$$

and moreover any proper subset of U has independent A_i's. As $\sum_{i \in U} c_i A_i$ is a minimal identity, from Theorem 3.7 we know $\deg(\mathrm{sim}(\sum_{i \in U} c_i A_i)) \leq |U| - 2$. Therefore, $\deg(\gcd(\{A_i : i \in U\})) \geq |S| - |U| + 2$.

Let $S' \subseteq S$ correspond to the variable parts on which the $\gcd(\{A_i : i \in U\})$ depends. Now, for all $i \in U$, A_i restricted to only the linear forms in S' is the same. This S' will be our new set S. We also restrict all remaining unmerged A_j's to S', and move the removed factors to the respective B_j's. We thus merge all D_i for $i \in U$ to a larger set. Thus our partition now has $r - |U| + 1$ parts, which is strictly smaller than r. The new set S has size at least $r - |U| + 4$. We replace all the A_i for i in U by a single gate which equals $\gcd(\{A_i : i \in U\}$. This completes the r-reduction procedure.

We will iterate this process till we are in the *linearly independent* case, and we now describe what to do when we reach that case.

Learning the circuit when $A_1, \ldots, A_r$ are linearly independent. We now assume we have access to linearly independent $A_1, \ldots, A_r$ which depend on variable parts $S \subseteq [k+2]$, with corresponding partitioning of $[k'] := D_1 \sqcup D_2 \sqcup \ldots \sqcup D_r$ such that for all $i \in [r]$, $A_i | T_j$ for all $j \in D_i$. Let $k_i := |D_i|$. As $r \geq 2$, we have all $k_i \leq k' - 1 \leq k - 1$. Therefore, $C^{\min}$ can be written as $\sum_{i=1}^{r} A_i \cdot \left(\sum_{j \in D_i} (T_j / A_i)\right)$. For $i \in [r]$, let $B_i := \left(\sum_{j \in D_i} (T_j / A_i)\right)$, and thus $C^{\min} = \sum_{i=1}^{r} A_i B_i$. Observe that B_i's are computable by $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k_i)$ circuits.

As discussed earlier, for each gate in C, there is some A_i that divides it. This comes from our choice of variable parts. Therefore, C can also be written as $C = \sum_{i=1}^{r} A_i \cdot B_i'$. Now, B_i' can be computed by a $\Sigma\Pi\Sigma(k_i')$ circuit where $k_i' \leq k - 1$ as $r \geq 2$, and the bit-complexity of the coefficients in the circuit is at most $\mathrm{poly}(\tau)$ (have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$). More importantly, we want to argue that the polynomials B_i and B_i' are the same. We know C and $C^{\min}$ compute the same polynomial and therefore $0 = C^{\min} - C = \sum_{i=1}^{r} A_i \cdot (B_i - B_i')$. If any of the $B_i \neq B_i'$, then there exists a substitution α for the variable parts in $[d] \setminus S$ such that $B_i(\alpha) - B_i'(\alpha) \neq 0$ at least for some i's. This gives us a linear relation in the A_i's which contradicts the fact that $A_1, \ldots, A_r$ are linearly independent. Therefore, we know all the B_i's are also computable by $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k-1)$ circuits with bit complexity in linear forms at most $\mathrm{poly}(\tau)$ (have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$). Also observe that this poly increase comes from normalization and is the same polynomial across recursions and doesn't blow up.

We will now show how to simulate blackbox access to each B_i. Since the top fan-in of each B_i is smaller than k, we will inductively learn B_i.

Let $\alpha_1, \ldots, \alpha_r$ be the points in $\prod_{j \in S} \mathbb{F}^{|X_j|} \sim \mathbb{F}^{\sum_{j \in S} |X_j|}$. Consider the $r \times r$ matrix M such that

$$M_{i,j} = A_i(\alpha_j).$$

We want points such that $\det(M) \neq 0$. Let $\mathcal{H}$ be a hitting set for $\Sigma\Pi\Sigma_{\sqcup_j X_j}(r)$ circuits. Observe that as A_i's are factors of the product gates of $C^{\min}$, they are also set-multilinear products of $|S|$ linear forms over variables $\{X_j \mid j \in S\}$. As $A_1, \ldots, A_r$ are linearly independent, then by Lemma 3.14, $A_1|_{\mathcal{H}}, \ldots, A_r|_{\mathcal{H}}$ (vector of evaluations of A_i's on $\mathcal{H}$) are also linearly independent. Therefore, the matrix N with columns $A_1|_{\mathcal{H}}, \ldots, A_r|_{\mathcal{H}}$ has rank r. We can greedily find r independent rows of N, and that gives us $\alpha_1, \ldots, \alpha_r \in \mathcal{H}$ over which the corresponding M is full rank and

hence invertible. This can be done in $\text{poly}(r, |S|, |\mathcal{H}|)$ time. From Corollary 3.5, we have a hitting set of size $|\mathcal{H}| = r^{O(\log r)} \cdot \text{poly}(r, k, |S|) = k^{O(\log k)}$.

Let $\alpha_1, \ldots, \alpha_r$ be the points in $\prod_{j \in S} \mathbb{F}^{|X_j|} \sim \mathbb{F}^{\sum_{j \in S} |X_j|}$ from the hitting set, such that $\det(M) \neq 0$. Let $Y := \sqcup_{j \in ([d] \setminus S)} X_j$. As $f(\alpha_i, Y) = \sum_{j=1}^{r} A_j(\alpha_i) \cdot B_j(Y)$, we have for every j,

$$B_j(Y) = \sum_{i=1}^{r} (M^{-1})_{i,j} (f(\alpha_i, Y))$$

Thus, we can get access to smaller fan-in set-multilinear depth-3 circuits, which we can reconstruct recursively.

We recursively run the algorithm for reconstruction on B_i for all $i \in [r]$ and obtain the respective circuits C_i. We finally have our circuit as $C = \sum_{i=1}^{r} A_i \cdot (C_i)$. The base case of $k = 1$ is easy to reconstruct, as it is just obtaining linear factors, and comes from the factoring algorithm of Lemma 3.10.

Finally, we use M_j^{-1} to undo the variable reduction and also multiply back $\gcd(C)$ that was removed at the beginning. For every resulting circuit (over all branches), we run a black-box PIT against the input f. The correct branch will lead to the correctly learned circuit, and the PIT using Corollary 3.5 discards all incorrectly learned circuits.

Running time analysis: For a running time analysis, let $T_{k,n,d}$ be the running time of the entire algorithm. We see that learning C_1' takes time $\text{poly}(n, d) \cdot k^{O(k^2)} \cdot \text{Sys}(O(k^2), k^{O(k)}, O(k))$ from Lemma 5.1 or a recursive call to the current lemma with a smaller top fan-in. After that, we run $\binom{5k}{k+2}$ guesses to figure out which variable parts we learned perfectly, and then k^k guesses to learn $A_1, \ldots, A_r$. We will spend $k^{O(\log k)}$ time on getting access to the B_i's. Finally, there will be r recursive calls for B_i. As discussed B_i's can be computed by $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k-1)$ circuits with bit complexity $\text{poly}(\tau)$ (have coefficients in $\mathbb{F}_q$, if $\mathbb{F} = \mathbb{F}_q$). We know from our construction that $\sum_{i \in [r]} k_i = k' \leq k$. Therefore, we have

$$T(n,d,k) \leq \text{poly}(n,d) \cdot k^{O(k^2)} \cdot \text{Sys}(O(k^2), k^{O(k)}, O(k)) + k^{O(k)} \cdot \left(\sum_{i=1}^{r} T(n, d - |S|, k-1) + k^{O(\log k)} \right)$$

Simplifying, we have the recurrence as

$$T(n,d,k) \leq \text{poly}(n,d) \cdot k^{O(k^2)} \cdot \text{Sys}(O(k^2), k^{O(k)}, O(k)) + k^{O(k)} \cdot (T(n, d - |S|, k-1))$$

which gives us $T(n,d,k) = \text{poly}(n,d) \cdot k^{O(k^2)} \cdot \text{Sys}(O(k^2), k^{O(k)}, O(k))$. □

7 Proof of Theorem 1.1

Proof of Theorem 1.1. Our algorithm inspects the value of d for our input circuit, and if $d > 5k$, we use Lemma 6.1 to reconstruct our circuit. If $d \leq 5k$, then we run the appropriate algorithm from Lemma 5.1. The combined running time of all the algorithms will be

$$k^{O(k^2)} \cdot \text{poly}(n,d) \cdot \text{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k)).$$

For $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, and when constants appearing in the input circuit have bit complexity at most τ, Theorem 3.3 implies that $\text{Sys}(n, m, d) = \text{poly}(m, \tau) \cdot d^{O(n)}$ for deterministic algorithms. Hence,

$\mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k)) = k^{O(k^2)} \cdot \mathrm{poly}(\tau)$ for deterministic algorithms. Therefore, we have a deterministic reconstruction (proper-learning) algorithm that runs in time $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \tau)$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit for the minimal value of k'.

For $\mathbb{F} = \mathbb{F}_q$, Theorem 3.3 implies that $\mathrm{Sys}(n, m, d) = \mathrm{poly}(m, \log q) \cdot d^{O(n)}$ for randomized algorithms. Hence, $\mathrm{Sys}_{\mathbb{F}}(O(k^2), k^{O(k)}, O(k)) = k^{O(k^2)} \cdot \mathrm{poly}(\log q)$ for randomized algorithms. Therefore, we have a randomized reconstruction (proper-learning) algorithm that runs in time $k^{O(k^2)} \cdot \mathrm{poly}(n, d, \log q)$ and outputs a set-multilinear $\Sigma\Pi\Sigma_{\sqcup_j X_j}(k')$ circuit over an extension of degree-$k^{O(k^2)}$ of $\mathbb{F}_q$ for the minimal value of k' wrt $\overline{\mathbb{F}_q}$. □

8 Acknowledgments

The authors would like to thank Abhibhav Garg for many helpful discussions related to the complexity of solving systems of polynomial equations.

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 F. Pereira, C.J. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 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 Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 36–112. JMLR.org, 2015.

[AGS26] Robert Andrews, Abhibhav Garg, and Éric Schost. Hilbert's nullstellensatz is in the counting hierarchy. *arXiv preprint arXiv:2602.17904 (To Appear in FOCS2026)*, 2026.

[Ang87] Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1987.

[BBB+00] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano 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 David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 594–603. ACM, 2014.

[BGKS22] Vishwas Bhargava, Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning generalized depth three arithmetic circuits in the non-degenerate case. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, University of Illinois,*

Urbana-Champaign, USA (Virtual Conference), September 19-21, 2022, LIPIcs, pages 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*. Springer, 2006.

[BS25] Vishwas Bhargava and Devansh Shringi. Faster & deterministic FPT algorithm for worst-case tensor decomposition. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*, LIPIcs, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[BSS26] Vishwas Bhargava, Leonard J. Schulman, and Shiri Sivan. An algorithmic proof of kruskal's tensor decomposition theorem. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pages 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026.

[BSV20] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Reconstruction of depth-4 multilinear circuits. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2144–2160. SIAM, 2020.

[BSV21] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Reconstruction algorithms for low-rank tensors and depth-3 multilinear circuits. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 809–822. ACM, 2021.

[Car06] Enrico Carlini. Reducing the number of variables of a polynomial. In Mohamed Elkadi, Bernard Mourrain, and Ragni Piene, editors, *Algebraic Geometry and Geometric Modeling*, pages 237–247. Springer, Berlin, Heidelberg, 2006.

[CM20] Sitan Chen and Raghu 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.

[FS12] Michael A. Forbes and Amir Shpilka. On identity testing of tensors, low-rank recovery and compressed sensing. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 163–172. ACM, 2012.

[FS13] Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October, 26-29, 2013*, pages 243–252. IEEE Computer Society, 2013.

[GG20] Zeyu Guo and Rohit Gurjar. Improved Explicit Hitting-Sets for ROABPs. In Jarosław Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of

Leibniz International Proceedings in Informatics (LIPIcs), pages 4:1–4:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[GKL11] Ankit Gupta, Neeraj Kayal, and Satyanarayana V. Lokam. Efficient reconstruction of random multilinear formulas. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 778–787. IEEE Computer Society, 2011.

[GKL12] Ankit Gupta, Neeraj Kayal, and Satyanarayana V. Lokam. Reconstruction of depth-4 multilinear circuits with top fan-in 2. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 625–642. ACM, 2012.

[GKQ14] Ankit Gupta, Neeraj Kayal, and Youming Qiao. Random arithmetic formulas can be reconstructed efficiently. *Comput. Complex.*, 23(2):207–303, 2014.

[GKS20] Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 889–899. IEEE, 2020.

[GVX14] Navin Goyal, Santosh S. Vempala, and Ying Xiao. Fourier PCA and robust tensor decomposition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 584–593. ACM, 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. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, page 11–20, New York, NY, USA, 2013. Association for Computing Machinery.

[JLV23] Nathaniel Johnston, Benjamin Lovitz, and Aravindan Vijayaraghavan. Computing linear sections of varieties: quantum entanglement, tensor decompositions and beyond. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1316–1336. IEEE, 2023.

[JO14] Prateek Jain and Sewoong Oh. Learning mixtures of discrete product distributions using spectral decompositions. In Maria Florina Balcan, Vitaly Feldman, and Csaba Szepesvári, editors, *Proceedings of The 27th Conference on Learning Theory*, volume 35 of *Proceedings of Machine Learning Research*, pages 824–856, Barcelona, Spain, 13–15 Jun 2014. PMLR.

[Kay11] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1409–1421. SIAM, 2011.

[KMW25] Pravesh K. Kothari, Ankur Moitra, and Alexander S. Wein. Overcomplete tensor decomposition via koszul-young flattenings. In *66th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025*, pages 1871–1882. IEEE, 2025.

[KNS19] Neeraj Kayal, Vineet Nair, and Chandan Saha. Average-case linear matrix factorization and reconstruction of low width algebraic branching programs. *Comput. Complex.*, 28(4):749–828, 2019.

[KNST19] Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Trans. Comput. Theory*, 11(1):2:1–2:56, 2019.

[KO25] Pascal Koiran and Rafael Oliveira. Tensor decomposition beyond uniqueness, with an application to the minrank problem. *CoRR*, abs/2510.26587, 2025.

[Koi25] Pascal Koiran. An efficient uniqueness theorem for overcomplete tensor decomposition. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 1909–1932. SIAM, 2025.

[Kru77] Joseph B. Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2):95–138, 1977.

[KS01] Adam R. Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 216–223. ACM, 2001.

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

[KS09] Zohar Shay Karnin and Amir 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 2009, Paris, France, 15-18 July 2009*, pages 274–285. IEEE Computer Society, 2009.

[KS19] Neeraj Kayal and Chandan Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 413–424. ACM, 2019.

[Lan12] J. M. Landsberg. *Tensors: Geometry and Applications*, volume 128 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, Rhode Island, 2012.

[LP23] Benjamin Lovitz and Fedor Petrov. A generalization of kruskal's theorem on tensor decomposition. *Forum of Mathematics, Sigma*, 11:e27, 2023.

[LRA93] Sue E. Leurgans, R. T. Ross, and R. B. Abel. A decomposition for three-way arrays. *SIAM J. Matrix Anal. Appl.*, 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.

[Moi18] Ankur Moitra. *Algorithmic aspects of machine learning*. Cambridge University Press, 2018.

[MR05] Elchanan Mossel and Sébastien Roch. Learning nonsingular phylogenies and hidden markov models. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 366–375, New York, NY, USA, 2005. Association for Computing Machinery.

[MSS16] Tengyu Ma, Jonathan Shi, and David Steurer. Polynomial-time tensor decompositions with sum-of-squares. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*, pages 438–446. IEEE Computer Society, 2016.

[MW19] Ankur Moitra and Alexander S. Wein. Spectral methods from tensor networks. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 926–937, New York, NY, USA, 2019. Association for Computing Machinery.

[Ope26a] OpenAI. Deterministic Polynomial Factorization over Prime Fields. OpenAI Math Release preprint Link, 2026.

[Ope26b] OpenAI. Hilbert's tenth problem over the rational numbers. OpenAI Math Release preprint, October 2026. October 6, 2026 version.

[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.

[Roj99] J.Maurice Rojas. Solving degenerate sparse polynomial systems faster. *Journal of Symbolic Computation*, 28(1):155–186, 1999.

[Shp07] Amir Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 284–293. ACM, 2007.

[Sin16] Gaurav Sinha. Reconstruction of Real Depth-3 Circuits with Top Fan-In 2. In Ran Raz, editor, *31st Conference on Computational Complexity (CCC 2016)*, volume 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:53, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Sin22] Gaurav Sinha. Efficient reconstruction of depth three arithmetic circuits with top fan-in two. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, Berkeley, CA, USA, January 31 - February 3, 2022*, LIPIcs, pages 118:1–118:33. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

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

[SS25] Shubhangi Saraf and Devansh Shringi. Reconstruction of Depth 3 Arithmetic Circuits with Top Fan-In 3. In *40th Computational Complexity Conference (CCC 2025)*, volume 339, pages 21:1–21:22, 2025.

[SSV26] Shubhangi Saraf, Devansh Shringi, and Narmada Varadarajan. Reconstruction of depth-3 arithmetic circuits with constant top fan-in. In Aditya Bhaskara and Artur Czumaj, editors, *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC 2026, Salt Lake City, UT, USA, June 22-26, 2026*, pages 1704–1715. ACM, 2026.

[ST26] Amir Shpilka and Yann Tal. Polynomial identity testing and reconstruction for depth-4 powering circuits of high degree. *CoRR*, abs/2602.20832, 2026.

[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.

[SY10] A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[Wei23] Alexander S. Wein. Average-case complexity of tensor decomposition for low-degree polynomials. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1685–1698. ACM, 2023.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il