ETH-Hardness of Learning Monotone Circuits and Approximating Their Size

Bruno Cavalar
University of Oxford, UK

Susanna F. de Rezende
Lund University, Sweden

Matthew Gray
University of Oxford, UK

Rahul Santhanam
University of Oxford, UK

Abstract

We show the following hardness results for monotone learning and approximation of monotone circuit size:

1. Under the Randomised Exponential-Time Hypothesis (rETH), it requires time $n^{\Omega(\log n)}$ to PAC-learn monotone formulas with n input bits and size $s(n) = n$ by monotone circuits of size $n^{(\log n)^{1-\epsilon}}$, for every $\epsilon > 0$.
2. Under the Randomised Exponential-Time Hypothesis (rETH), for any $\delta > 0$, there is a polynomially bounded function m such that $m^{1-\delta}$-multiplicatively approximating the minimum monotone circuit size of a monotone function consistent with a sequence of $m(n)$ labelled examples $\{(x_i, b_i)\}$ over n-bit inputs requires time $m^{\Omega(\log(m))}$.

Our results are shown by a novel application of lifting arguments in proof and communication complexity to hardness of monotone learning, by building on the seminal result of Atserias and Müller [6] on hardness of automating Resolution proofs.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Machine learning theory; Theory of computation → Proof complexity

Keywords and phrases monotone circuit complexity, PAC learning, lifting theorems, Resolution, meta-complexity, hardness of approximation, minimum circuit size problem

Digital Object Identifier 10.4230/LIPIcs.CCC.2026.40

Related Version *Full Version*: `https://arxiv.org/abs/2607.12331`

Funding Rahul Santhanam and Bruno Cavalar acknowledge support from the EPSRC project EP/Z534158/1 on "Integrated Approach to Computational Complexity: Structure, Self-Reference and Lower Bounds". Susanna de Rezende received funding from the Knut and Alice Wallenberg Foundation grant KAW 2023.0116, ELLIIT, and the Swedish Research Council grant 2021-05104.

Acknowledgements We thank Noel Arteche for helpful discussions, and the anonymous reviewers for their valuable comments and suggestions.

1 Introduction

1.1 Motivation from Meta-Complexity and Learning

Meta-Complexity. One of the main goals of the area of *meta-complexity* is to prove strong hardness results for computing complexity measures such as circuit size, formula size and monotone circuit size. While such hardness results can be derived in a standard way from cryptographic assumptions such as the existence of one-way functions [43, 32], it has proven much more challenging to show hardness results under *worst-case assumptions*, such as NP-hardness results. A good reason for this difficulty was identified by Kabanets and Cai [32],

41st Computational Complexity Conference (CCC 2026).
Editor: Dana Moshkovitz; Article No. 40; pp. 40:1–40:25
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

who showed that NP-hardness of the Minimum Circuit Size Problem (which asks whether a Boolean function F given by its truth table has circuits of size at most s, for a given parameter s) via polynomial-time honest reductions would imply hard-to-prove circuit lower bounds such as super-polynomial circuit lower bounds for EXP.

Despite this obstacle, there are now several hardness results for various notions of circuit size under worst-case hardness assumptions, including DNF size, Boolean formula size (under ETH), multi-output circuit size, oracle circuit size, etc., [38, 2, 24, 30, 27, 28, 29, 22, 26]. A common feature of these results, except for the recent inapproximability result of [26] for oracle circuit size, is that they do not show strong inapproximability results for circuit size. Here, by a strong inapproximability result, we mean the hardness of approximating the circuit size within a $N^{\Omega(1)}$ multiplicative factor, where N is the size of the function representation.

A major motivation for showing strong inapproximability result for circuit size under worst-case assumptions is that they offer a pathway to worst-case to average-case reductions for NP. It was shown in [44, 23], building on work of [8], that there are non-black-box reductions from strong inapproximability of problems such as circuit size and average circuit size to average-case hardness for these problems. Thus, NP-hardness of strong inapproximability for either of these problems would rule out Heuristica, the complexity world imagined by Impagliazzo [31] where NP is hard in the worst case but easy on average. A recent result of Mazor and Pass [39] rules out strong inapproximability of circuit size under Levin reductions from SAT, assuming a standard cryptographic assumption, but strong inapproximability might still hold under a more general notion of reduction such as Turing reductions. The strong inapproximability result of [26] for oracle circuit size is based on cryptographic techniques and seems difficult to apply to standard circuit size measures. It could potentially be very useful to find other techniques yielding strong inapproximability.

In terms of circuit size measures for which hardness under worst-case assumptions is known, monotone circuit size has been little-studied despite its central role in complexity theory. Showing NP-hardness for the Minimum Monotone Circuit Size Problem (which asks whether a monotone Boolean function given by its truth table has monotone circuit size at most s, for a given parameter s) would be a significant step towards NP-hardness for the Minimum Circuit Size Problem, as there is a well-known reduction from Minimum Circuit Size to Minimum Monotone Circuit Size.

Learning. One of the main goals of the area of *learning theory* is to identify which learning problems are hard in the PAC-learning model. When PAC-learning a concept class $\mathcal{C}$, we are given samples $(x, f(x))$ from a distribution $\mathcal{D}$, where f belongs to $\mathcal{C}$, and our task is to output with high probability a hypothesis c that approximates f well according to the distribution. If the output hypothesis c is in $\mathcal{C}$, the learning algorithm is said to be *proper*, otherwise it is said to be *improper*. We also consider a *semi-proper* version of learning $\mathcal{C}$-circuits by $\mathcal{C}'$-circuits, where $\mathcal{C}' \supseteq \mathcal{C}$.

It is known how to rule out even *improper learning* of various concept classes under cryptographic assumptions [33, 34, 9]. More recently, there has been exciting work on ruling out improper learning even of weak concept classes such as DNFs [10, 11, 12] based on average-case assumptions such as Feige's hypothesis [16] and its variants. However, just as in the case of meta-complexity problems, showing hardness under worst-case assumptions remains challenging. This was explained in a work of Applebaum, Barak and Xiao [3], who showed that NP-hardness of learning general Boolean circuits under Karp reductions or non-adaptive reductions would imply the collapse of the Polynomial Hierarchy. This leaves open the possibility of using more general reductions such as Turing reductions to show hardness.

Known worst-case hardness results are typically for proper learning of weak circuit classes such as DNFs, or for semi-proper learning where the output hypothesis class $\mathcal{D}$ is not too much more powerful than the concept class [1]. While hardness of learning even restricted subclasses of monotone circuits under the uniform distribution is known under cryptographic assumptions [9], and unconditional hardness results are known in the statistical query learning framework [17], very little seems to be known about the worst-case hardness of learning monotone circuits.

As can be seen from the discussion above, there are several parallels between the situation with hardness of meta-complexity problems and the situation with hardness of learning. These parallels are not coincidental: learning for a circuit class $\mathcal{C}$ can be seen as a *search version* of the problem of determining minimum $\mathcal{C}$-size of a Boolean function. Indeed, given access to a Boolean function f, outputting a circuit of size s that computes f (perhaps approximately) via a learning algorithm witnesses that f has (perhaps approximate) circuit size at most s. This connection is exploited to derive new learning algorithms in [8].

In this paper, we aim to establish strong results on the hardness of monotone learning based on worst-case assumption, and then to leverage the connection between monotone learning and meta-complexity to get strong inapproximability of monotone circuit size.

The meta-complexity problem we consider is the following: given $m = \mathsf{poly}(n)$ labelled examples (x_i, b_i), where each x_i is an n-bit string and b_i a bit, and a parameter s, is there a monotone function f of monotone circuit size at most s that is consistent with the labelled examples, i.e., $b_i = f(x_i)$ for each i. In the context of learning, this is known as the Occam Learning problem, and the non-monotone analogue of this problem was called Partial Minimum Circuit Size Problem in [30], where it was shown to be NP-hard. We are interested here in strong inapproximability results, which the techniques of [30] are incapable of proving, since they reduce from the well-approximable Set Cover Problem.

1.2 Our Results

Our first main result is on the hardness of PAC-learning monotone formulas. Proving hardness of improperly PAC-learning monotone formulas is at least as hard as showing NP-hardness of learning general Boolean circuits, and hence beyond reach at this point in time. However, we are able to prove a strong result in the *semi-proper* setting under worst-case assumptions, where our concept class is the class of monotone formulas, but our hypothesis class is the class of polynomially larger monotone circuits. In this setting, we are able to get a quasipolynomial lower bound on the running time of any PAC-learning algorithm, assuming the randomised Exponential Time Hypothesis. Recall that the randomised Exponential Time Hypothesis (rETH) postulates that any algorithm solving 3SAT on formulas of n variables must take time $2^{\Omega(n)}$. This is a standard hardness assumption in fine-grained complexity and parameterized complexity.

▶ **Theorem 1.1** (Informal). *Under rETH, for every $\alpha > 0$, monotone formulas of size n on n variables require time $n^{\Omega(\log n)}$ to be PAC-learned by monotone circuits of size $n^{(\log n)^{1-\alpha}}$.*

The formal statement of Theorem 1.1 is in the first item of Theorem 4.11 in Section 4.4. To the best of our knowledge, it was not known previously even how to get hardness of learning monotone circuits of size n by monotone circuits of size $n^{1+\alpha}$ under standard worst-case assumptions. In fact, our techniques yield a tradeoff, where we can show hardness even for learning monotone formulas of even slightly super-polylogarithmic monotone formula size by monotone circuits of polynomial size. This result is stated in the second item of Theorem 4.11.

Our second main result shows the strong inapproximability of monotone circuit size for the partial version of the Minimum Monotone Circuit Size Problem, where we are given a sequence of labelled examples as input. The assumption used here is again rETH.

▶ **Theorem 1.2.** *For each $c > 1$ the following holds under rETH: there is no algorithm running in time $N^{o(\log N)}$ which, given as input a sequence of $O(n^c \log n)$ labelled examples (x_i, b_i) representable by N bits in all, where each x_i is an n-bit string and b_i a bit, can distinguish between the case that these labelled examples are consistent with a monotone formula of size n (i.e., there is f of monotone formula size n for which $f(x_i) = b_i$ for each i) and the case that these labelled examples are inconsistent with a monotone circuit of size n^c. In particular, for arbitrary $\epsilon > 0$ there is no $N^{o(\log N)}$ time algorithm which $N^{1-\epsilon}$-multiplicatively approximates the partial monotone formula size or the partial monotone circuit size.*

The formal statement of Theorem 1.2 is in the fourth item of Corollary 4.9.

Both Theorem 1.1 and Theorem 1.2 are consequences of a more general hardness result for a monotone version of the Computational Gap Learning problem studied in [3]. In our problem, which we call Monotone Circuit-Formula Gap Learning, we are given a circuit which samples a distribution of labelled examples (x_i, b_i) where x_i is of length n and b_i a bit, and are asked to distinguish between 2 cases: the first where the distribution is completely consistent with a formula of size at most s_1, and the second where the distribution does not even non-trivially correlate with any circuit of size at most s_2. Here $s_1, s_2 : \mathbb{N} \to \mathbb{N}$ are parameters of the problem, which we are able to choose so that $s_2 \gg s_1$. Theorem 1.1 follows immediately from our result about Monotone Circuit-Formula Gap Learning, while Theorem 1.2 follows by a standard sampling argument together with a union bound.

To show our central result about Monotone Circuit-Formula Gap Learning, we use a novel technique in the domain of hardness of learning: *lifting*, together with known strong inapproximability results for Resolution proof size [6, 14, 20, 13, 7]. One of the main applications of lifting is to obtain monotone circuit size lower bounds in a generic way from Resolution proof width lower bounds [18, 19]. The generic nature of these reductions opens the possibility of using them to derive *reductions* from approximating proof size for various proof systems to approximating circuit size for various circuit models. This is yet another instance of the connection between learning and automatability of proof systems proposed in [1].

It may be useful to compare our results with the work of Hirahara [22]. Hirahara shows that a variant of the Computational Gap Learning problem is NP-hard (under randomised reductions), where the task is to distinguish between linear programs (Turing machines computing a linear function) of size s_1 and programs of size $s_2 = s_1 \cdot n^{o(1)}$. Besides the obvious difference that we consider monotone circuits vs. formulas whereas [22] considers programs, crucially our results allow for a superpolynomial gap between circuit size, whereas [22] obtain only a sublinear gap between program size (though under the weaker assumption of NP-hardness). Hirahara also shows NP-hardness of approximating partial circuit size, when one is given the entire truth table of a partial function (referred to as MCSP^*), again only with a $n^{o(1)}$ approximation gap. In Theorem 1.2, however, we consider a *succinct* version of this problem, in which we are only given inputs where the function is defined; we also obtain a stronger hardness of approximation gap here.

We next give a technical overview of the ideas behind our proofs.

1.3 Technical Overview

Overall approach. The main insight of the paper is to convert the hardness of *automating* Resolution [6] into the hardness of approximating monotone circuit size, using results from the literature (called *lifting theorems*) that connect the complexity of Resolution refutations with the monotone complexity of Boolean functions [18].

To understand how this works, let us review how the seminal paper of Atserias and Müller [6] proved that automating Resolution is NP-hard. Recall that an algorithm that automates Resolution must, given as input an unsatisfiable CNF formula ϕ, construct a Resolution refutation of ϕ in time $\mathsf{poly}(|\phi| + s)$, where s is the size of the shortest Resolution refutation of ϕ. Given a 3-CNF formula F on n variables as input, the proof of [6] consists in constructing in polynomial-time a formula $\mathsf{Ref}(F)$ with the following properties:

1. If F is satisfiable, then $\mathsf{Ref}(F)$ admits a polynomial-size Resolution refutation;
2. If F is unsatisfiable, then any Resolution refutation of $\mathsf{Ref}(F)$ must have size $2^{n^{\Omega(1)}}$.

In turn, this implies that if we can *automate* Resolution, then we can determine in which of the two cases we are in, and thus decide the satisfiability of F.

Another significant ingredient in our approach is the *DAG-like lifting theorem*, which converts Resolution *width* lower bounds for a formula ϕ into monotone circuit size lower bounds for a related monotone function f_ϕ [18]. Importantly, the monotone complexity of the resulting function f_ϕ is tightly connected to the Resolution width of ϕ, so that f_ϕ has large monotone complexity iff ϕ requires large Resolution width.

Ideally, we would now like to combine the properties of $\mathsf{Ref}(F)$ with the lifting theorem to reduce the satisfiability of F to deciding the monotone complexity of $f := f_{\mathsf{Ref}(F)}$ when given access to some succinct representation of f (e.g., access to random samples of f, or a succinct representation of its truth-table). Unfortunately, the connection between monotone complexity and Resolution goes only via Resolution *width*, whereas the complexity gap obtained by Atserias-Müller is only for Resolution *size*. Though Resolution size does indeed give *upper bounds* on the monotone complexity of a related function via *feasible interpolation* [36, 35, 42], the converse is not known to hold.

A potential way out would be to consider the notion of *block-width* of proofs [6], a generalisation of Resolution-width to the setting when the variables are split into blocks, as is the case with $\mathsf{Ref}(F)$. The proof of Atserias-Müller can be rephrased as first showing that the block-width of $\mathsf{Ref}(F)$ is $O(1)$ when F is satisfiable, and $n^{\Omega(1)}$ when F is unsatisfiable [20]. However, though the notion of block-width can be lifted to Resolution size, it's not known if it can be lifted to monotone circuit size, and indeed significant technical challenges have been found when trying to do so in at least one previous work [20].

A more promising approach comes from considering a different encoding of $\mathsf{Ref}(F)$ found in [14]. Their encoding admits a Resolution-width gap of $\mathsf{Ref}(F)$ of the following form[1]:

1. If F is satisfiable, $\mathsf{Ref}(F)$ has Resolution-width $O(n)$;
2. If F is unsatisfiable, $\mathsf{Ref}(F)$ has Resolution-width $\Omega(n^3)$.

This seems to be what we are aiming at: an efficiently constructible formula with a noticeable gap in Resolution-width. However, lifting gives us that the monotone complexity of f_ϕ is roughly $2^{\Theta(\mathsf{ResWidth}(\phi))} \cdot \mathsf{Res}(\phi)$. So we obtain a function on $n^{O(1)}$ input bits with $2^{O(n)}$ monotone complexity when F is satisfiable, and $2^{\Omega(n^3)}$ when F is unsatisfiable. Unfortunately, it is not clear how to distinguish between these two cases other than by guessing the *exponential size* circuit that computes the function. In other words, we have obtained a reduction from an NP problem to an NEXP problem, which is hardly surprising. Moreover, the reduction is ultimately useless, as we could just solve SAT directly in exponential time.

[1] The encoding of [6] cannot achieve this gap because it uses unary pointers.

We overcome this hurdle by constructing a different formula $\mathsf{Ref}^*(F)$ with the following properties:[2]

1. $\mathsf{Ref}^*(F)$ has $2^{O(\sqrt{n})}$ variables and $2^{O(\sqrt{n})}$ clauses.
2. $\mathsf{Ref}^*(F)$ admits Resolution proofs of depth at most $O(\sqrt{n})$ when F is satisfiable;
3. $\mathsf{Ref}^*(F)$ requires Resolution proofs of width at least $2^{\Omega(\sqrt{n})}$ when F is unsatisfiable.

Note that we obtain not just a Resolution width upper bound in the satisfiable case, but the strictly more powerful upper bound on Resolution *depth*, which is strongly tied to the size of monotone *formulas* [41, 21]. This is how we obtain a gap between monotone formula and circuit sizes in the final result. Note moreover that the resulting formula has *subexponential* size. This is a careful tradeoff that we exploit: by increasing the size of the formula, we are able to reduce its Resolution-width (and even its Resolution-depth). We now describe the construction of the formula in more detail.

The $\mathsf{Ref}^*(F)$ Formula. The formula $\mathsf{Ref}(F)$ essentially encodes the statement "F has a Resolution proof of small size s", where s is a fixed polynomial in the number of variables of F. When F is satisfiable, the formula $\mathsf{Ref}(F)$ is *unsatisfiable*, and this formula can be refuted with low *block-width* [6, 40].

Their strategy can be seen as low memory technique to achieve the following. Given any proposed proof for F, a *refuter*[3] navigates through it while maintaining the guarantee that the current clause is unsatisfied by the known satisfying assignment x^*. Beginning at $\bot$, eventually such a refuter will make it to a line of the proof that is a weakening of an axiom. Since the line must be unsatisfied by x^*, it cannot be a weakening of any axiom and a contradiction has been found. While this technique can be used to show an $O(1)$ block-width upper bound, the depth of the proof strategy is $\Omega(s \cdot n)$, since it will involve learning every variable of $\Omega(s)$ lines of the proposed proof. Adapting this technique to achieve a $\sqrt{n}$ depth upper bound seems like a nearly hopeless endeavour on first blush. Achieving depth $\sqrt{n}$ requires us to refute $\mathsf{Ref}(F)$ while querying in total fewer variables than the number of variables of F. This presents two barriers:

- The refuter needs to get to a line of the proof that is claimed to be a weakening of an axiom while only seeing variables from $\sqrt{n}$ total blocks, and
- The refuter needs to learn some information about all n variables while being able to query far fewer than n variables.

To overcome the first of these issues, we can force the proof to have depth $\sqrt{n}$, as done in [13]. However, if we are only resolving over one literal at a time, the clauses at depth-$\sqrt{n}$, which are weakening of axioms of F, would have width $\sqrt{n}$. This would break the lower bound of Atserias and Müller because their strategy requires that the clauses in the purported proof that are claimed to be a weakening of an axiom must have width n.

To retrieve the lower bound, we can change the underlying proof system to one which resolves on $\sqrt{n}$ variables at a time. This results in each block, instead of having just a left and a right child – which are the two clauses from which it was derived – now having $2^{\sqrt{n}}$ children, and thus having one pointer point_z for each possible setting $z \in \{0,1\}^{\sqrt{n}}$ of the variables being resolved over (we also importantly fix the order in which variables are resolved

[2] For simplicity of exposition, at this introduction we only explain the construction with the parameters that give a polynomial gap between linear-size formulas and monotone circuits. The construction for "juntas" (i.e., functions that do not depend on all inputs) is the same, but with a different parametrisation.

[3] The "Refuter" is typically referred to as "Prover" in a Prover-Adversary game. Since F is satisfiable, the goal of the Prover is to find a contradiction to the statement that F admits a Resolution refutation, and so its task can be referred to as "refuting" a proposed proof of unsatisfiability of F.

over, with the first segment of $\sqrt{n}$ variables, then the second segment, etc). Unfortunately, this immediately means that the resulting $\mathsf{Ref}(F)$ formula will have to be of size at least $2^{\sqrt{n}}$ just to manage these pointer variables. On the positive side, this gives us a hint for how to overcome the second barrier.

In standard resolution refutations, if you know that a clause B' is the left child of B which is the result of resolving over x_i, you know that x_i is in B', and more importantly that $\neg x_i$ is *not* in B'; and if B' is the right child you know that $\neg x_i$ is in B', and more importantly that x_i is *not* in B'. Consequently, knowing that x^* is a satisfying assignment, if the prover always queries whichever child is not satisfied by x^*, they will eventually find themselves in a leaf which cannot be a valid weakening. In our new multi-variable resolution system, if you know that B' is the z^{th} child of B, you know $\sqrt{n}$ bits of information about the clause that B' claims to have. Using x^* as above, one could in principle get to a leaf which cannot be a valid weakening in depth $O(\sqrt{n})$.

We can extend this density of information from the pointers to a new class of summary variables $\mathsf{summ}^B_{i,z}$ which give us information about all the variables from $x_{(i-1)\sqrt{n}}$ to $x_{i\sqrt{n}-1}$. This idea of using summary variables appeared before in [7]. It is important that our summary variables encode which literals are *not* present (which is ultimately what we are concerned with for the upper bound). Formally, $\mathsf{summ\text{-}unlit}^B_{i,z}$ is set equal to 1 to indicate that for each $z_j = 1$, literal $x_{(i-1)d+j}$ is not present in block B, and that for each $z_j = 0$, literal $\neg x_{(i-1)d+j}$ is not present in block B. We can ensure that a Prover can learn what all $\sqrt{n}$ of the $\mathsf{summ\text{-}unlit}^B_{i,z}$ variables for a leaf node should be while only querying the right set of $O(\sqrt{n})$ variables on their way from the root to a leaf.

To make our bounds as tight as possible, we add some additional complexity, including binary pointers for axioms, split variables to reduce clause width, and a low degree expander between each layer of the proof. This low-degree expander, which was previously used in [13, 7], allows us to keep down to a constant the number of $\mathsf{point}^B_{B',z}$ variables a prover needs to query for each B, z before finding one that is set to one. While the expander does somewhat complicate the proof of the lower bound, we are able to derive it by reducing the retraction pigeonhole principle (rPHP) instance to finding a violated clause, similarly to what was done in [13].

Hardness of approximating circuit size and learning. To see how to join our new Ref^* formula with a lifting theorem to obtain the claimed hardness results, it will be helpful to understand the function that comes from lifting. Roughly speaking, "lifting" usually consists in composing some original computational object (e.g., a propositional formula or a function) with a gadget, so as to hide the original object in a stronger computational model in such a way that the only way to compute/solve the composed problem is by solving the original problem.

In our case, given a formula F with n variables and width k, together with a parameter m, the lifted function $f := f_{F,m}$ from [18] will have $|F| \cdot m^k$ input bits, where $|F|$ is the number of clauses of F. To each $i \in [|F| \cdot m^k]$, we associate a pair (C, α), where C is a clause of F and $\alpha \in [m]^k$. We interpret (C, α) as the k-width clause over the variables $\{y_{ij} : i \in [n], j \in [m]\}$ obtained by replacing the variable $x_i \in \mathsf{vars}(C)$ in C with $y_{i,\alpha(i)}$. We can thus see each input to f as a collection of clauses (C, α). The task of the partial function $f_{F,m}$ is to distinguish between the following two cases:

1. Accept (Output 1): there exists $x : [n] \to [m]$ such that the input bit corresponding to $(C, x^{\mathsf{vars}(C)})$ is set to 1 for every clause C of F, and everything else is set to 0;
2. Reject (Output 0): there exists $y \in \{0,1\}^{mn}$ such that all input bits associated with clauses (C, α) satisfied by y are set to 1, and everything else is set to 0.

The partial function is clearly well-defined, as in the first case the collection of clauses is unsatisfiable, and in the second case they are satisfiable. For this same reason, no input to the function which contains all the clauses of Case (1) will ever be in Case (2), so the function is monotone. One can think of the function f as "hiding" several variable renamings of the the formula F in its input bits; to compute f is to "sniff out" if F has been completely embedded among the renamed clauses.

Most importantly for our purposes, we can easily sample from the distribution which half of the time gives a uniform accepting input of f, and in the other half gives a uniform rejecting input. Indeed, first sample a bit $b \in \{0,1\}$, then, if $b = 1$, sample $n \log m$ bits and interpret it as $x : [n] \to [m]$, otherwise sample a string $y \in \{0,1\}^{mn}$. With x (resp. y), it's then easy to set the relevant bits so as to output an accepting (resp. rejecting) input of $f_{F,m}$. Let us abuse notation and denote by $\mathcal{D}^{F,m}$ both a circuit that samples from this distribution of binary strings (together with the bit b as a prefix) and the distribution itself. We thus see samples from $\mathcal{D}^{F,m}$ as a pair (x, b) where $x \in \{0,1\}^{|F|m^k}$ and $b \in \{0,1\}$. Given F and m as input, it's also easy to see that we can construct the circuit sampling from $\mathcal{D}^{F,m}$ in time $\mathsf{poly}(|F|, m^k)$.

Going back to the approach above, given an input F to 3SAT, our reduction will first construct $\mathcal{D} := \mathcal{D}^{\mathsf{Ref}^*(F),m}$, so as to obtain a distribution with the following properties. If F is satisfiable, then

$$\text{there is a monotone formula } L \text{ of size} \leq s_1 \text{ s.t. } L(x) = b \text{ for every } (x,b) \in \mathrm{supp}(\mathcal{D}); \quad (1)$$

if F is unsatisfiable, then

$$\text{every monotone circuit } K \text{ of size} \leq s_2 \text{ satisfies } \mathop{\mathbb{P}}_{(x,b)\leftarrow\mathcal{D}}[K(x) = b] < 1/2 + \gamma. \quad (2)$$

The value of s_1 is $2^{O(\sqrt{n}\log m)}$, and the value of s_2 is $2^{\Omega(m/\sqrt{n})}$; these come from a refined version of the DAG-like lifting theorem [15] with "small gadgets" that allows us to essentially pick any value of m we would like. However, lifting theorems typically are only stated for $\gamma = 1/2$ (corresponding to a worst-case lower bound). Fortunately, generalising to $\gamma \approx m^{-1}$ is an easy task, requiring only the modification of certain numbers and parameters in the proof of [15]. This is essential for our purposes, as in a learning problem we only expect to approximate the target concept.

We are now ready to state the computational problem for which we show a lower bound. The *Monotone Circuit-Formula Gap Learning problem* $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ is the problem of distinguishing between cases (1) and (2) when given $\mathcal{D}$ as input. Given that our $\mathsf{Ref}^*(F)$ formula has size $2^{O(\sqrt{n})}$, the reduction from SAT to this problem takes this time. Under ETH, we thus obtain an $N^{\Omega(\log N)}$ lower bound for this problem, since the size N of the input is $2^{O(\sqrt{n})}$. To reduce to the case $s_1 = n$, we pad the sampled strings; this takes time $2^{\widetilde{O}(\sqrt{n})}$, so our final lower bound is $N^{\widetilde{\Omega}(\log N)}$.

We can further reduce this problem to a gap version of Partial-Monotone-MCSP we call $\mathsf{mMCFSP}^{s_2}_{s_1}[\gamma]$[4]. In this problem, instead of $\mathcal{D}$, the algorithm is given t examples $(x_1, b_1), \ldots, (x_t, b_t)$, and must distinguish whether there is a monotone formula L of size s_1 such that $L(x_i) = b_i$ for all $i \in [t]$, or whether every monotone circuit C of size s_2 is such that $|\{i \in [t] : C(x_i) \neq b_i\}| \geq (1/2 + \gamma)t$. The reduction is randomised, naturally coming from sampling the distribution $\mathcal{D}$. Moreover, we need to sample $O(s_2 \log s_2)$ examples so

[4] Sometimes the literature reserves the term "Partial-MCSP" to the problem where one is given the entire truth table of a partial function, but we follow here the convention of [30].

that, with high probability, any circuit of size at most s_2 will err on a proportion larger than $1/2 + \gamma$ of the examples (in case (2)), and so the reduction takes time $O(s_2 \log s_2)$. This means that we get a tradeoff between the choice of s_2 and the runtime lower bound coming from rETH. The same reduction also gives us lower bounds on PAC-learning.

2 Preliminaries

For $f, g : \mathbb{N} \to \mathbb{N}$, we write $f \sim g$ if $\lim f/g \to 1$, and $f \asymp g$ if $f = \Theta(g)$. We say that a function Boolean f is γ-approximated by g over a distribution $\mathcal{D}$ if $\mathbb{P}_{x \leftarrow \mathcal{D}}[f(x) = g(x)] \geq 1/2 + \gamma$.

2.1 Basic complexity

Monotone circuits are Boolean circuits without $\neg$ gates. *Formulas* are Boolean circuits whose gates have fan-out 1. *Monotone formulas* are analogously defined.

The *Exponential-Time Hypothesis (ETH)* postulates that there exists a constant $\varepsilon > 0$ such that any algorithm solving 3SAT on n-variate formulas must take time $2^{\varepsilon n}$. The *randomised* ETH *(rETH)* postulates the same lower bound for randomised algorithms.

2.2 Expander graphs

A (m, n, δ, r, e)-*bipartite expander graph* is a bipartite graph $G = ((U \cup V), E)$ with $|U| = m$, $|V| = n$, with every vertex in U having degree at most δ, and where for every subset $S \subseteq [m]$ such that $|S| \leq r$, the neighbourhood of S, denoted $N(S)$, satisfies $|N(S)| \geq e|S|$.

▶ Remark 2.1. A complete bipartite graph $G = ((U \cup V), E)$ with $|U| = m$, $|V| = n$, is a (m, n, δ, r, c)-bipartite expander graph for, e.g., $\delta = n$, $r = n/2$ and $c = 2$.

A standard calculation [25] shows that random graphs are good expanders. We record one parameter regime that we use in our applications.

▶ **Lemma 2.2.** *Let $m, n \in \mathbb{N}$ be such that $m = n^2$. Then there exists $\delta = O(1)$ and $r = \Omega(\sqrt{n})$ such that with high probability a random graph $G \sim \mathbf{G}(m, n, \delta)$ is an $(m, n, \delta, r, \delta/2)$-bipartite expander graph.*

2.3 Proof complexity

Resolution. A *literal* is a propositional atom or its negation, a *clause* is a disjunction of literals, and a *CNF formula* is a conjunction of clauses. We see clauses as sets of literals, and write simply $C \subseteq D$ to express that C is a subclause of D. We denote the set of variables in a clause C by $\mathsf{vars}(C)$ (if x_i or $\neg x_i$ are present in C, we have $x_i \in \mathsf{vars}(C)$). A *Resolution refutation* of an unsatisfiable CNF formula $\varphi = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$ is a sequence $D_1, \ldots, D_s$ of clauses over $x_1, \ldots, x_n$ such that $D_s = \bot$, denoting the empty clause, and for every $i \in [s-1]$, the clause D_i either (a) is one of the clauses $C_1, \ldots, C_m$ of φ, or (b) is a *weakening* of a previous clause, meaning that $D_i \supseteq D_j$ for some $1 \leq j < i$, or (c) has been obtained from two previous clauses $D_j = A \vee x$ and $D_k = B \vee \neg x$, for $j, k < i$, by an application of the *Resolution rule*:

$$\frac{A \vee x \qquad B \vee \neg x}{A \vee B}$$

We say that $A \vee B$ was obtained by *resolving* over x. The *size* of the refutation is s, the number of clauses appearing in the refutation.

To every Resolution refutation we can associate a directed acyclic graph (dag) in a natural way. We define the *depth* of the proof as the length of the longest path in the dag, starting from the root labeled by the empty clause $\perp$. The proof is said to be *tree-like* if the associated graph is a tree rooted at $\perp$.

Let F be an unsatisfiable k-CNF formula. We denote by $\mathsf{ResWidth}(F)$ the minimum integer such that F admits a Resolution refutation whose clauses all have width at most $\mathsf{ResWidth}(F)$. Let $\mathsf{ResDepth}(F)$ be the minimum integer such that F admits a Resolution refutation with depth at most $\mathsf{ResDepth}(F)$. We denote by $\mathsf{Res}(F)$ the size of the smallest Resolution refutation of F, and by $\mathsf{TreeRes}(F)$ the size of the smallest tree-like such refutation. We denote by $|F|$ the number of clauses in F.

Prover-adversary game. An equivalent characterisation of Resolution width and Resolution depth can be given by notions from *query complexity*. Suppose that F is an n-variate unsatisfiable CNF formula. Given any assignment $x \in \{0,1\}^n$ to the variables of F, some clause of C is necessarily falsified. The minimum number k of queries to the variables of F needed to discover one such clause is equivalent to $\mathsf{ResDepth}(F)$.

To characterise $\mathsf{ResWidth}$, it is helpful to modify the above setup in what is commonly called a "Prover-Adversary game". As before, the Prover is querying variables, but now the Prover is allowed to forget the value of some queried variables. The adversary is answering the queries, but may respond with different values when a variable whose value was forgotten is queried again. The minimum number k of variables that the Prover needs to store in order to discover a falsified clause is equal to $\mathsf{ResWidth}(F) \pm 1$ [5].

2.4 Lifting for average-case lower bounds

Given a *gadget* $g : \mathcal{X} \times \mathcal{Y}$, let $g^n : \mathcal{X}^n \times \mathcal{Y}^n$ denote the function

$$g^n(x, y) = (g(x_1, y_1), \ldots, g(x_n, y_n)).$$

Let $\mathsf{Ind}_m : [m] \times \{0,1\}^m \to \{0,1\}$ denote the "index gadget", defined as the Boolean function such that $\mathsf{Ind}_m(x, y) = y[x]$.

▶ **Definition 2.3.** *Let F be an unsatisfiable formula with n variables. Let $f_{F,m}$ be the partial Boolean function over $|F| \cdot m^k$ bits defined as follows. To each $i \in [|F| \cdot m^k]$, associate a pair (C, α), where C is a clause of F and $\alpha \in [m]^k$. We interpret (C, α) as the k-width clause over the variables $\{y_{ij} : i \in [n], j \in [m]\}$ obtained by replacing the variable $x_i \in \mathsf{vars}(C)$ in C with $y_{i,\alpha(i)}$. Each binary string of length $|F| \cdot m^k$ is thus in 1-to-1 correspondence with the set*

$$F \circ \mathsf{Ind}_m^n := \{(C, \alpha) : C \text{ is a clause of } F,\ \alpha : \mathsf{vars}(C) \to [m]\}.$$

The function $f_{F,m}$ has the following behaviour:

1. *Accept the binary string whose support is $\left\{(C, \alpha) \in F \circ \mathsf{Ind}_m^n : x^{\mathsf{vars}(C)} = \alpha\right\}$, for every $x : [n] \to [m]$.*
2. *Reject the binary string whose support is $\{(C, \alpha) \in F \circ \mathsf{Ind}_m^n : (C, \alpha)(y) = 1\}$, for every $y \in \{0,1\}^{nm}$.*

The following theorem shows that the monotone circuit complexity of $f_{F,m}$ is roughly $m^{\Theta(\mathsf{ResWidth}(F))}$ for some $m \gg \mathsf{ResWidth}(F)$; moreover, a formula upper bound of the form $m^{O(\mathsf{ResDepth}(F))}$ also holds. We say that a circuit $D : \{0,1\}^r \to \{0,1\}^N$ *represents* a distribution $\mathcal{D}$ over N-input bits if the random binary string $D(\mathbf{r})$ where $\mathbf{r} \leftarrow \{0,1\}^r$ is distributed according to $\mathcal{D}$.

▶ **Theorem 2.4** ([15, Theorem 11, modified]). *There exists an algorithm $A(F, 1^m)$ which receives as input an unsatisfiable k-CNF formula F on N variables, runs in time $\mathsf{poly}(|F|\cdot m^k)$, and outputs a circuit representing a distribution $\mathcal{D}^{F,m} = (\mathcal{D}_1 + \mathcal{D}_0)/2$ over $|F| \cdot m^k$ bits such that:*

1. *There is a monotone circuit K of size $m^{O(\mathsf{ResWidth}(F))} \cdot \mathsf{Res}(F)$ and a monotone formula L of size $m^{O(\mathsf{ResDepth}(F))}$ such that $K(x) = L(x) = b$ for every $x \in \mathrm{supp}(\mathcal{D}_b)$ and $b \in \{0,1\}$.*
2. *There exists an absolute constant c such that for any $\ell \leq \mathsf{ResWidth}(F)$, any monotone circuit K that satisfies $\mathbb{P}_{b\leftarrow\{0,1\},x\leftarrow\mathcal{D}_b}[K(x) = b] \geq 1/2 + \gamma$, for $\gamma \geq c \cdot \ell \cdot \log(mN)/m$, has size at least*

$$\left(\frac{m}{c \cdot \ell \cdot \log(mN)}\right)^{\ell-1}.$$

Moreover, the distribution $\mathcal{D}_0$ (resp. $\mathcal{D}_1$) is uniformly distributed over $f^{-1}_{F,m}(0)$ (resp. $f^{-1}_{F,m}(1)$).

Proof sketch. The algorithm A constructs a circuit $\mathcal{D}^{F,m} : \{0,1\}^r \to \{0,1\}^{|F|\cdot m^k}$ where $r = mN + 1$ that behaves in the following way. If the first bit of r is 0, then let $y \in \{0,1\}^{mN}$ be the remaining mN bits. In the output bit (C, α), the circuit outputs 1 iff y satisfies the clause (C, α). Moreover, if the first bit of r is 1, then let $x \in [m]^N$ be given the first $m \log n$ of the remaining bits. In the output bit (C, α), the circuit outputs 1 if $x^{\mathsf{vars}(C)} = \alpha$. The construction of D, and D itself, can be clearly be done in time $\mathsf{poly}(|F| \cdot m^k)$.

Property (1) of the resulting distribution is folklore (see, e.g., [37]). Property (2) is proved in [15, Theorem 11] with $\gamma = 1/2$; the result with smaller γ can be easily obtained by tweaking a few parameters. We sketch how to do this in the full version of this paper. ◀

3 Shallow Refutation formula

Let F be a 3-CNF formula over n variables. The $\mathsf{Ref}_s(F)$ formula encodes the statement "F has a resolution proof of length s". Slightly more formally, the variables of $\mathsf{Ref}_s(F)$ are divided into s blocks, each of which encodes a potential clause in the purported refutation of F and how this clause was derived, and the clauses of $\mathsf{Ref}_s(F)$ enforce that all derivation steps are sound.

The known resolution refutation of $\mathsf{Ref}_s(F)$ when F is satisfiable, given by Pudlák's upper bound [40], has large width $\Omega(n)$. Under ETH, a large width is unavoidable, since it is possible to find width w refutations of an unsatisfiable N-variate CNF formula in time $N^{O(w)}$. While it is possible to show a gap between the Resolution width of the formula $\mathsf{Ref}_s(F)$ in the satisfiable and unsatisfiable cases of F [6, 14] ($O(n)$ vs. $\Omega(s/n)$), for our applications we need a better upper bound. In this section, we define a formula related to $\mathsf{Ref}_s(F)$ which substantially decrease the width upper bound at the cost of an increase in the size of the formula. Moreover, we show that our new formula has small Resolution *depth* when F is satisfiable.

3.1 The new formula $\mathsf{Ref}^G_d(F)$

Our new formula is denoted by $\mathsf{Ref}^G_d(F)$, and it depends on a bipartite expander graph G and an integer parameter d. We somewhat follow the notation established in "The Proof Analysis Problem" by Arteche et al. [4], but modify the structure of the purported resolution refutation and introduce some extension variables, similar to what was done in [13] and [7].

As was done in [13] we structure the blocks in d layers (instead of n), and as in [7], we add "summary" variables that give information about segments of n/d variables. These changes allow us to turn the $O(1)$ block-width upper bound and corresponding $O(n)$ resolution width upper bound in [40] into a $O(d)$ upper bound on depth, at the cost of increasing the number of variables and clauses in $\mathsf{Ref}_d^G(F)$ to $2^{\Theta(n/d)}$.

Since it is crucial that these summary variables encode that literals are absent, we also switch focus and define "un-literal" variables encoding that single literals are *not present* in a block as opposed to previous encoding of Ref which has variables encoding the *inclusions* of literals. This is done mostly for exposition purposes since our unlit variables are just the negation of the usual literal variables. We also include binary variable for pointers (as done in [14]) to make it possible to query pointers more efficiently.

Finally, we also include "split" variables to turn our formula into a 3-CNF formula using a standard transformation. This is done in order to minimise the blow-up in size of the function we obtain after lifting $\mathsf{Ref}_d^G(F)$.

We start by explaining the structure of the formula $\mathsf{Ref}_d^G(F)$. We then introduce the main variables of $\mathsf{Ref}_d^G(F)$ and the clauses enforcing that the purported proof is sound and is structured as we would like. Finally, we introduce the split variables and explain how they are used to transform the formula into a 3-CNF formula.

Structure of $\mathsf{Ref}_d^G(F)$ and notation. Let F be a CNF formula over n variables and $\mathsf{poly}(n)$ clauses, let d be an integer, and let G be a bipartite graph with $t2^{n/d}$ nodes on left side, t nodes on the right, and outdegree δ from each node on the left. We define $s := t \cdot d$. For the sake of convenience in our presentation, we assume that d divides n, and that $|F|$ (the number of clauses of F) and δ, the left outdegree of the expander G, are powers of 2, so they can be cleanly indexed into by binary pointers. We later explain why we can make these assumptions without loss of generality (see Remark 3.2).

The formula $\mathsf{Ref}_d^G(F)$ is defined over $s+1$ blocks arranged in $d+1$ layers. The block $s+1$ is the root and it is alone in layer 0. The other s blocks are arranged in decreasing order in layers 1 to d with s/d blocks per layer. Formally, for $B \in [s]$, we have $\mathsf{layer}(B) = 1+d-\lceil B/t \rceil$ while $\mathsf{col}(B) = B - (\lceil B/t \rceil - 1)t$. So for $B = s$ we get that $\mathsf{layer}(s) = 1, \mathsf{col}(s) = t$, while for $B = 1$ we get $\mathsf{layer}(1) = d, \mathsf{col}(1) = 1$.

Given an integer S which is a power of 2 and an integer $X \in [S]$, we denote by $\mathsf{bin}_S(X)$ the binary string of length $\log S$ encoding X. When S is clear from the context, we simply write $\mathsf{bin}(X)$. For instance, for $A \in [|F|]$, we write $\mathsf{bin}(A) \in \{0,1\}^{\log |F|}$ to be the binary string of length $\log |F|$ encoding A.

Finally, we need a way of managing binary indices for the at most δ valid z^{th} children B' of B. To do this, we need a surjection (which we will label $\mathsf{sur}_{z,\mathsf{col}(B)}$) from the set of at most δ valid z^{th} children to $[\delta]$. It is easy to see that, if $\mathsf{sur}_{z,\mathsf{col}(B)}$ is the lexicographical order over the valid B's, then $\mathsf{sur}_{z,\mathsf{col}(B)}$ can be efficiently computed when needed while constructing $\mathsf{Ref}_d^G(F)$ in all the parameter regimes considered in this paper. Note that there may be parameter regimes in which this is not the case.

The variables of $\mathsf{Ref}_d^G(F)$. Let $x_1, .., x_n$ be the variables of F. For any variable x_i, the variable $(x_i)^0$ corresponds to $\neg x_i$ and $(x_i)^1$ corresponds to x_i. We define the set $\mathsf{Unlit}_n = \{x_1, ..., x_n, \bar{x}_1, ..., \bar{x}_n\}$. Let $\mathsf{Leaves} := [s/d]$ be the set of blocks at layer d. Some variables (like $\mathsf{weak}, \mathsf{weakbin}$, which encode what axiom in F the block is a weakening of, in unary and in binary, respectively) are only defined for blocks in Leaves and some ($\mathsf{point}, \mathsf{pointbin}$, which encode the blocks that were used to derive the current block, also in unary and in binary, respectively) are only defined for blocks that are not in Leaves.

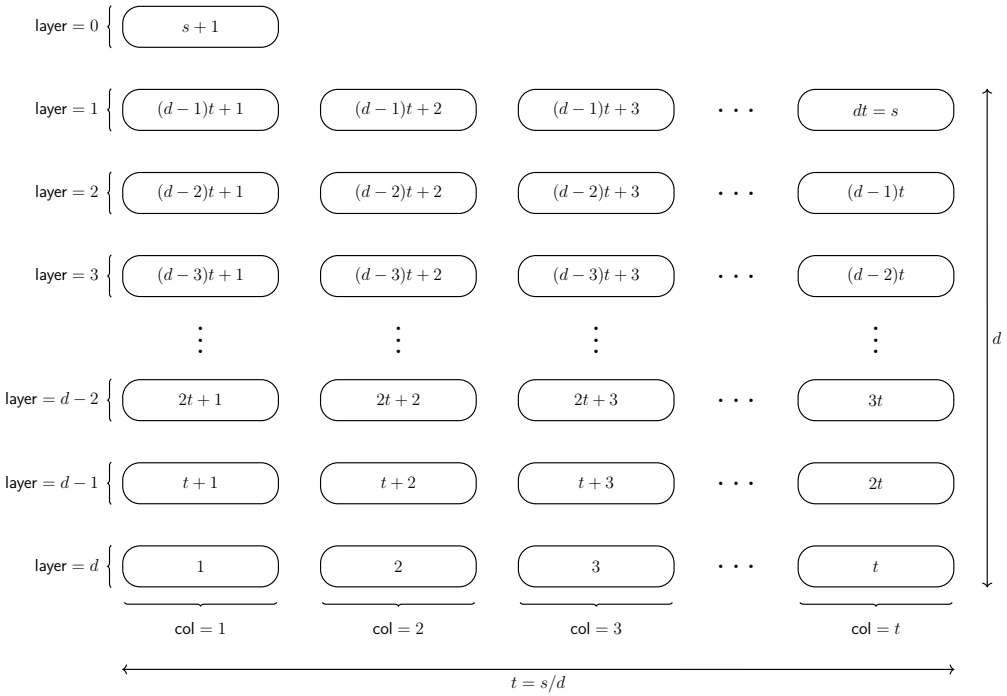

Figure 1 The layout of blocks within the purported refutation, demonstrating the columns and layers.

Recall that, in our setting, a block is obtained by resolving n/d variables at once. Therefore, the z^{th} child, for $z \in \{0,1\}^{n/d}$, of a block at layer i should not contain literals x_{id+j} for each $z_j = 1$, nor literals $\neg x_{id+j}$ for each $z_j = 0$. This will be enforced by using the variables summ-unlit. In fact, except for blocks that are leaves, the only information about what literals are contained in each block is given by these summ-unlit variables. Since the layer of a block determines what variables can be present in the block, only the relevant summ-unlit variables are defined for each block. In particular, the root does not contain any summ-unlit variable and the leaves contain all summ-unlit variables. Moreover, the leaves are the only blocks that contain the unlit variables, as only then are these needed to connect the clause in the block with the axiom of F that it is supposedly a weakening of.

We are now ready to introduce the variables of our formula.

▶ Remark 3.1. For each class of variables listed above there are fewer than $O\left(|F| \cdot n \cdot s^2 \cdot 2^{n/d}\right)$ variables of the class. Assuming $|F| = n^{O(1)}$, $d \leq n$, and $s = 2^{O(n/d)}$, this is $2^{O(n/d)} \cdot n^{O(1)}$.

▶ Remark 3.2. We now explain why we can make the divisibility assumptions we made about $|F|, d, n$ and δ.

- When d does not divide n, we can add one final layer with a corresponding set of summary and pointer variables to manage those x_i with $i > \lfloor n/d \rfloor d$.
- When $|F|$ and δ are not powers of two, we can add clauses which rule out each invalid binary pointer or interpret each invalid pointer as some canonical value.

The clauses of $\mathsf{Ref}_d^G(F)$. We first introduce the unsplit versions of the clauses which make up $\mathsf{Ref}_d^G(F)$. These clauses enforce the basic structure of the Shallow-Resolution formula ensuring particularly that our summary and pointer variables interact correctly. The clauses are defined for arbitrary G but in our proofs we focus on two cases for G: either G is a bipartite expander with t nodes on the left side, $t \cdot 2^{n/d}$ on the right side, and outdegree δ from each node the left side; or G is a complete graph such that all $(\mathsf{col}(B), (\mathsf{col}(B'), z)) \in E(G)$.

unlit^B_ℓ : literal $\ell \in \mathsf{Unlit}_n$ is not present in block $B \in \mathsf{Leaves}$;

weak^B_A : block $B \in \mathsf{Leaves}$ was obtained by weakening axiom $A \in [|F|]$;

$\mathsf{weakbin}^B_j$: for $B \in \mathsf{Leaves}$ and $j \in [\log |F|]$, the value of the j^{th} bit of $\mathsf{bin}(A) \in \{0,1\}^{\log |F|}$, where $A \in [|F|]$ is the axiom that block B was obtained from by weakening;

$\mathsf{summ\text{-}unlit}^B_{i,z}$: for block $B \in [s]$, $i \in [\mathsf{layer}(B)]$, and $z \in \{0,1\}^{n/d}$, for each $z_j = 1$, literal $x_{(i-1)d+j}$ is not present in block B, and for each $z_j = 0$ literal $\neg x_{(i-1)d+j}$ is not present in block B;

$\mathsf{point}^B_{B',z}$: for blocks $B, B' \in [s+1]$, the z^{th} child of block B is B' (and therefore $\mathsf{summ\text{-}unlit}^{B'}_{i,z} = 1$ for $i = \mathsf{layer}(B') = \mathsf{layer}(B) + 1$,)

$\mathsf{pointbin}^B_{z,j}$: for block $B \in [s+1]$, $j \in [\log \delta]$, and $z \in \{0,1\}^{n/d}$, the value of the j^{th} bit of $\mathsf{bin}(\mathsf{sur}_{z,\mathsf{col}(B)}(B')) \in \{0,1\}^{\log \delta}$ where B' is the z^{th} child of B (and, in particular, $\mathsf{layer}(B') = \mathsf{layer}(B) + 1$);

enable^B : block $B \in [s+1]$ is enabled.

We divide the clauses into three categories depending on whether the block in the superscript is in layer 0 (meaning the block is the root), in layer d (meaning the block is a leaf), or between layers 1 and $d-1$ (meaning the block is an internal block).

Root case – final block is enabled.

$$\mathsf{enable}^{s+1} \tag{Ref-1}$$

Leaf case – managing axioms. The following clauses are defined for each $B \in \mathsf{Leaf}$.

$$\left(\mathsf{enable}^B \wedge \mathsf{weak}^B_A\right) \to \neg \mathsf{unlit}^B_\ell \qquad \text{for } A \in [|F|],\ \ell, \text{ in clause } A; \tag{Ref-2}$$

$$\left(\bigwedge_{j \in [\log |F|]} \left(\mathsf{weakbin}^B_j\right)^{\mathsf{bin}(A)_j}\right) \to \mathsf{weak}^B_A \qquad \text{for } A \in [|F|]; \tag{Ref-3}$$

$$\left(\mathsf{enable}^B \wedge \mathsf{summ\text{-}unlit}^B_{i,z}\right) \to \mathsf{unlit}^B_{(x_k)^{z_j}} \qquad \text{for } i \in [d], j \in [n/d], \ k = (i-1)n/d + j\,. \tag{Ref-4}$$

Internal Step – pointer and summary enforcement. The following clauses are defined for each B that is not a leaf, i.e., for B such that $\mathsf{layer}(B) < d$; for each $z \in \{0,1\}^{n/d}$; and for each B' which is a possible zth child for B, i.e., for $B' \in P_B \coloneqq \{B' : \mathsf{layer}(B') = \mathsf{layer}(B) + 1$ and $((\mathsf{col}(B), z), \mathsf{col}(B')) \in E(G)\}$.

$$\left(\bigwedge_{j \in [\log \delta]} \left(\mathsf{pointbin}^B_{z,j}\right)^{\mathsf{bin}(\mathsf{sur}(B'))_j}\right) \to \mathsf{point}^B_{B',z} \tag{Ref-5}$$

$$\left(\mathsf{enable}^B \wedge \mathsf{point}^B_{B',z}\right) \to \mathsf{enable}^{B'} \tag{Ref-6}$$

$$\left(\mathsf{enable}^B \wedge \mathsf{point}^B_{B',z'} \wedge \mathsf{summ\text{-}unlit}^B_{i,z}\right) \to \mathsf{summ\text{-}unlit}^{B'}_{i,z} \qquad \text{for } z' \in \{0,1\}^{n/d},\ i \le \mathsf{layer}(B); \tag{Ref-7}$$

$$\left(\mathsf{enable}^B \wedge \mathsf{point}^B_{B',z}\right) \to \mathsf{summ\text{-}unlit}^{B'}_{i+1,z} \qquad \text{for } i = \mathsf{layer}(B). \tag{Ref-8}$$

▶ Remark 3.3. For each class of clauses listed above, there are fewer than $O(|F| \cdot n \cdot s^2 \cdot 2^{2n/d})$ clauses of that class, and each clause has fewer than $O(\log(|F|) + \log(\delta))$ literals in them.

We now introduce a final set of variables which refer to as "split variables". These are used to transform the formula into 3-CNF using a standard transformation. For every clause $C_i = v^i_1 \vee v^i_2 \vee ... \vee v^i_{|C_i|}$ so far introduced we introduce a collection of split variables $\mathsf{split}^j_{C_i}$ for each $j \in [|C_i| - 3]$ and replace C_i with a collection of clauses $(v^i_1 \vee v^i_2 \vee \mathsf{split}^1_{C_i}) \wedge (\neg\mathsf{split}^1_{C_i} \vee v^i_3 \vee \mathsf{split}^2_{C_i}) \wedge ... \wedge (\neg\mathsf{split}^{|C_i|-3}_{C_i} \vee v^i_{|C_i|-1} \vee v^i_{|C_i|})$.

▶ Remark 3.4. By Remark 3.3 if $s = 2^{O(n/d)}$, $d \leq n$, $|F| = n^{O(1)}$, and $\delta = O(s)$ then before splitting, the formula has $2^{O(n/d)} \cdot n^{O(1)}$ clauses containing at most $O(\log n + \log s)$ variables each. As a consequence, after splitting, $\mathsf{Ref}^G_d(F)$ has at most $O(\log n + \log s) \cdot 2^{O(n/d)} \cdot n^{O(1)} = 2^{O(n/d)} \cdot n^{O(1)}$ clauses, each of which contains at most 3 variables.

3.2 Main Theorem

We now show the following theorem.

▶ **Theorem 3.5.** *Given a 3-CNF formula F on n variables, with $\mathsf{poly}(n)$ clauses, and positive integers $d \leq n$ and $t \leq 2^{n/d}$, and a $(t \cdot 2^{n/d}, 2^{n/d}, \delta, r, c)$-bipartite expander graph G there is a deterministic algorithm running in time $2^{O(n/d)} \cdot n^{O(1)}$ that constructs a 3-CNF formula $\mathsf{Ref}^G_d(F)$ which satisfies the following properties:*

1. *$\mathsf{Ref}^G_d(F)$ has $2^{\Theta(n/d)} \cdot n^{\Theta(1)}$ variables and $2^{\Theta(n/d)} \cdot n^{\Theta(1)}$ clauses.*
2. *$\mathsf{Ref}^G_d(F)$ has width $O(1)$;*
3. *$\mathsf{Ref}^G_d(F)$ admits Resolution proofs of depth at most $O(d \log \delta + \log n)$ when F is satisfiable;*
4. *$\mathsf{Ref}^G_d(F)$ requires Resolution proofs of width at least $2^{\Omega(r(c-1)/n)}$ when F is unsatisfiable.*

Proof. The first two claims follow directly from Remark 3.4. The fourth claim follows from essentially the same argument given in [7] (reducing a hard for resolution pigeonhole principle to finding a collision on the expander G), which we defer to the full-length version of this paper.

The third claim follows from the guaranteed success of the following $O(d \log \delta + \log n)$ depth prover in the standard Prover-Delayer game (see Section 2.3) used to analyse bounds for resolution proofs. We lay out the strategy the prover takes in each of the three cases: root where $B = s + 1$, internal where $B \leq s$ and $\mathsf{layer}(B) < d$, leaf where $\mathsf{layer}(B) = d$, and backchecking which is done after finding a leaf which is a valid weakening. For each of these, we specify the types of clauses the prover could identify as being violated if the delayer gave answers that do not allow the prover to proceed in each step. In all cases it should be clear that the prover can identify a violated clause of one of the listed classes Ref-1–Ref-8 immediately. Moreover, since all non-split clauses have at most $O(\log \delta + \log |F|) = O(\log \delta + \log n)$ variables, finding a split clause which is violated can be done in additional depth $O(\log \delta + \log n)$ by querying all the variables in the clause and all the split variables associated with the clause.

The prover critically knows an assignment x^* which satisfies F and will use this knowledge in implementing their strategy. In discussing the strategy below, we apply subscripts to assignment strings x^*_i and summary strings z^{j+1}_i with these subscripts serving as indices specifying the i^{th} bit in the string.

Invariance condition. After applying the root case and k internal steps of the strategy (but not having yet applied the leaf case or the backchecking step), the following invariance conditions are maintained:

- for all $j \in 0 \cup [k]$ the prover knows the variables enable^{B_j} and $\mathsf{point}^{B_j}_{B_{j+1},z^{j+1}}$ which are all equal to 1;
- $B_0 = s+1$;
- $\forall j \in [d], \forall i \in [n/d] : z_i^{j+1} = x^*_{jn/d+i}$.

Root Case. Learn that $s+1$ is enabled by querying enable^{s+1} which will equal 1, otherwise (Ref-1) is violated. Learn the index B_1 of the child which is unsatisfied by x^* by querying the $\log\delta$ variables $\mathsf{pointbin}^{s+1}_{z^1,j}$ where $z_i^1 = x_i^*$. Finally, query the variable $\mathsf{point}^{s+1}_{B_1,z}$ which must be 1 otherwise (Ref-5) is violated.

Internal Step. Learn that B_j is enabled by querying enable^{B_j} which will equal 1 (Ref-6). Learn the index B_{j+1} of the child of B_j which is unsatisfied by x^* by querying the $\log\delta$ variables $\mathsf{pointbin}^{B_j}_{z^{j+1},j'}$ where $z_i^{j+1} = x^*_{jn/d+i}$. Finally, query the variable $\mathsf{point}^{B_j}_{B_{j+1},z}$ which must be 1 otherwise right side of (Ref-5) is violated.

Leaf Case. Learn that B_j is enabled by querying enable^{B_j} which will equal 1, otherwise (Ref-6) is violated. Query all $\mathsf{summ\text{-}unlit}^{B_j}_{i,z^i}$ where each z^i is specified according to the system in the internal step. If they are all 1 then, learn the axiom $A \in [|F|]$ which B_j is a weakening of by querying the $\log|F|$ variables $\mathsf{weakbin}^{B_j}_{j'}$. Then query the variable $\mathsf{weak}^{B_j}_A$ which must be 1 otherwise it would violate (Ref-3). Since x^* is a satisfying assignment, one of (Ref-2) or (Ref-4) will be violated. We are left with the case where at least one of the $\mathsf{summ\text{-}unlit}^{B_j}_{i,z^i}$ variables is 0.

Backchecking. For the indices i and j for which $\mathsf{summ\text{-}unlit}^{B_j}_{i,z^i}$ was found to be 0, query all $\mathsf{summ\text{-}unlit}^{B_\kappa}_{i,z^i}$ for $\kappa \in [i+1, j-1]$. If $\mathsf{summ\text{-}unlit}^{B_{i+1}}_{i,z^i} = 0$ then the appropriate (Ref-8) is violated. Otherwise the prover finds a κ such that $\mathsf{summ\text{-}unlit}^{B_\kappa}_{i,z^i} = 1$ but $\mathsf{summ\text{-}unlit}^{B_{\kappa+1}}_{i,z^i} = 0$, which implies the appropriate (Ref-7) is violated.

In the root case, and each of the d internal steps, the prover queries $O(\log\delta)$ variables; in the leaf case, the prover queries $O(\log|F|)$ variables; and in the backchecking case, the prover queries at most d variables. In total, it makes at most $O(d\log\delta + \log|F|)$ queries and is guaranteed to find a violated clause. ◀

From Theorem 3.5 we obtain two corollaries. First, using the fact that an unbalanced complete bipartite graph is an expander (Remark 2.1), we obtain the following.

▶ **Corollary 3.6.** *Given a 3-CNF formula F on n variables, with $\mathsf{poly}(n)$ clauses, and positive integers $d \le n$ and $t \le 2^{n/d}$, there is a deterministic algorithm running in time $2^{O(n/d)} \cdot n^{O(1)}$ that constructs a 3-CNF formula $\mathsf{Ref}^G_d(F)$ which satisfies the following properties:*

1. *$\mathsf{Ref}^G_d(F)$ has $2^{O(n/d)} \cdot n^{O(1)}$ variables and $2^{O(n/d)} \cdot n^{O(1)}$ clauses;*
2. *$\mathsf{Ref}^G_d(F)$ has width $O(1)$;*
3. *$\mathsf{Ref}^G_d(F)$ admits Resolution proofs of depth at most $O(d\log t + \log n)$ when F is satisfiable;*
4. *$\mathsf{Ref}^G_d(F)$ requires Resolution proofs of width at least $2^{\Omega(t/n)}$ when F is unsatisfiable.*

Using a randomised construction of expanders (Lemma 2.2), we can improve the above construction in the following way.

▶ **Corollary 3.7.** *Given a 3-CNF formula F on n variables, with $\mathsf{poly}(n)$ clauses, and a positive integer $d \le n$, there is a randomized algorithm running in time $2^{O(n/d)} \cdot n^{O(1)}$ that constructs a 3-CNF formula $\mathsf{Ref}^G_d(F)$ which satisfies the following properties:*

1. $\mathsf{Ref}_d^G(F)$ *has* $2^{O(n/d)} \cdot n^{O(1)}$ *variables and* $2^{O(n/d)} \cdot n^{O(1)}$ *clauses;*
2. $\mathsf{Ref}_d^G(F)$ *has width* $O(1)$*;*
3. $\mathsf{Ref}_d^G(F)$ *admits Resolution proofs of depth at most* $O(d + \log n)$ *when* F *is satisfiable;*
4. $\mathsf{Ref}_d^G(F)$ *requires Resolution proofs of width at least* $2^{2^{\Omega(n/d)}}$ *when* F *is unsatisfiable.*

4 Learning lower bounds under ETH

In this section, we formally prove Theorems 1.1 and 1.2. We begin by defining a gap learning decision problem to which we will reduce the SAT problem, and from which we will be able to reduce to other learning problems.

▶ **Definition 4.1.** *Let* $s_1, s_2, \varepsilon : \mathbb{N} \to \mathbb{N}$ *be time-constructible functions. The computational problem* $\mathsf{mCFGL}_{s_1}^{s_2}[\varepsilon]$ *(referred to as* Monotone Circuit-Formula Gap Learning*) receives as input* $(1^n, \mathcal{D})$*, where* $\mathcal{D}$ *is a circuit with* $n+1$ *output bits and size* n^2 *(here, we identify the circuit with its representation for simplicity of notation). We interpret* $\mathcal{D}$ *as a distribution of pairs* $(x, b) \in \{0,1\}^n \times \{0,1\}$*, obtained by sampling a uniformly random input* r *and outputting* $\mathcal{D}(r)$*. The task is to decide if there exists a monotone formula* C *(with* n *input bits and single output bit) of size* s_1 *such that* $\mathbb{P}_{(x,b)\leftarrow\mathcal{D}}[C(x) = b] = 1$*, or if every monotone circuit* C' *such that* $\mathbb{P}_{(x,b)\leftarrow\mathcal{D}}[C'(x) = b] \geq 1/2 + \varepsilon$ *must have size at least* s_2*. The number* n *is called the* dimension *of the input.*

▶ Remark 4.2. We observe that the size of the sampling circuit $\mathcal{D}$ is chosen to be n^2 arbitrarily but without loss of generality. Indeed, given any family of inputs where the n-bit samplers have size $s = n^{O(1)}$, if $s < n^2$ we can just add dummy gates to the circuit; if $s > n^2$ we can pad the output x with $\sqrt{s} - n$ 0's; in both cases, we reduce to the case $s = n^2$ in polynomial-time.

We now implement the strategy outlined in the introduction. We take a 3CNF formula F on n variables, construct the formula $\mathsf{Ref}_d^G(F)$ satisfying either Corollary 3.6 (in case of a deterministic reduction) or Corollary 3.7 (in case of a randomised reduction). Using a deterministic reduction, taking m such that $n \leq m \ll 2^{n/d}$, $t \leq 2^{n/d}$ and $\ell = \varepsilon dm/n \ll 2^{t/n}$ for small enough ε in Theorem 2.4 gives that the function $f_{\mathsf{Ref}_s^G(F),m}$ has $N = 2^{O(n/d)} \cdot n^{O(1)}$ input bits and the following monotone complexity:

1. At most $m^{O(d \log t + \log n)} = 2^{O((d \log t + \log n) \log m)}$ when F is satisfiable, and
2. At least $2^{\Omega(\ell)} \geq 2^{\Omega(m/\log N)}$ when F is unsatisfiable.

Using a randomised reduction, the value of t is irrelevant and the resulting monotone complexity is

1. At most $m^{O(d+\log n)} = 2^{O((d+\log n)\log m)}$ when F is satisfiable, and
2. At least $2^{\Omega(\ell)} \geq 2^{\Omega(m/\log N)}$ when F is unsatisfiable.

Since we are reducing from ETH, the value $m = \mathsf{poly}(n)$ is already good enough, as a monotone circuit lower bound beyond $2^{\Omega(n)}$ (for an N-variate function) is not of much use for reductions to Partial-MCSP or PAC-learning (since we would need to output a circuit of this size or generate an exponential number of examples). The value d is then the main parameter we choose. There are two parameter regimes we will care about:

1. *($d = \log n$).* We obtain a formula upper bound which is smaller than the number N of input bits (a "junta"). The runtime lower bound based on ETH becomes $N^{\Omega(\log \log N)}$.
2. *($d \approx \sqrt{n}$).* This gives us a polynomial vs. quasipolynomial gap. The runtime lower bound becomes $N^{\widetilde{\Omega}(\log N)}$.

4.1 ETH-hardness of Monotone Circuit-Formula Gap Learning

We first show the result for juntas.

▶ **Theorem 4.3** (Junta Gap-Learning). *Let $\gamma > 0$ be a constant. For large enough $n \in \mathbb{N}$, the 3SAT problem on n variables reduces to $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ in time $2^{O(n/\log n)}$, where $s_1(N) = 2^{O(\log\log N)^3}$ and $s_2(N) = N^{\Omega(\log N)}$. Consequently, the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ needs time $N^{\Omega(\log\log N)}$ to be solved under ETH.*

Proof. Let F be a given 3-CNF over n variables. Fix $d = \log n$. Note that $\mathsf{Ref}^G_s(F)$ is a formula of width $O(1)$ over $N = 2^{\Theta(n/\log n)}$ variables and with $N^{O(1)}$ clauses. Let

$$m := n^3/(\log n)^3, \quad t := n^2.$$

and let $\ell = \varepsilon dm/n$ for a sufficiently small constant $\varepsilon > 0$. The function $f^* := f_{\mathsf{Ref}^G_d(F),m}$ is defined on $n_0 := N^{O(1)}m^{O(1)} = 2^{O(n/\log n)}$ variables (Definition 2.3). Theorems 2.4 and 3.5 imply that f^* can be computed with monotone formulas of size $s := m^{O(d\log t+\log n)} = 2^{O(\log^3 n)} = 2^{O(\log\log N)^3}$ when F is satisfiable. Moreover, when F is unsatisfiable, the function requires monotone circuits of size at least

$$s' := 2^{\Omega(\ell)} = N^{\Omega(\log N)}$$

to be γ-approximated over the distribution $\mathcal{D} := \mathcal{D}^{\mathsf{Ref}^G_s(F),m}$ given by Theorem 2.4, since we can take ε sufficiently smaller than γ. This distribution can be constructed in time $2^{O(n/\log n)}$ given F and m. A postprocessing taking $2^{O(n/\log n)}$-time may be needed to adjust the description of the circuit to the required length (see Remark 4.2). The ETH lower bound follows by noticing that $N^{\log\log N} = 2^{O(n)}$. ◀

The polynomial vs. quasipolynomial gap follows similarly.

▶ **Theorem 4.4** (Polynomial vs. quasipolynomial gap). *Let $\gamma > 0$ be a constant. For large enough $n \in \mathbb{N}$, the 3SAT problem on n variables reduces to $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ in time $2^{O(\sqrt{n}\log n)}$, where $s_1(N) = N$ and $s_2(N) = N^{\Omega(\log N)}$. Consequently, the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ needs time $N^{\Omega((\log N)/(\log\log N)^2)}$ to be solved under ETH.*

Proof. Let F be a given 3-CNF over n variables. Fix $d = \alpha\sqrt{n}/\log n$ where $\alpha > 0$ is small enough constant. Note that $\mathsf{Ref}^G_s(F)$ is a formula of width $O(1)$ over $N = 2^{\Theta(\sqrt{n}\log n)}$ variables and with $N^{O(1)}$ clauses. Let

$$m := (\log N)^2 \asymp n\log^2 n, \quad t := n^2.$$

and let $\ell = \varepsilon dm/n$ for a sufficiently small constant $\varepsilon > 0$. The function $f^* := f_{\mathsf{Ref}^G_d(F),m}$ is defined on $n_0 := N^{\Theta(1)}m^{\Theta(1)} = 2^{\Theta(\sqrt{n}\log n)} = N^{\Theta(1)}$ variables (Definition 2.3). Theorems 2.4 and 3.5 imply that f^* can be computed with monotone formulas of size

$$s := m^{O(d\log t+\log n)} = 2^{O(d\log^2 n+\log^2 n)} = 2^{O(\alpha\sqrt{n}\log n)} \le n_0,$$

when F is satisfiable and α is small enough. Moreover, when F is unsatisfiable, the function requires monotone circuits of size at least

$$s' := 2^{\Omega(\ell)} = N^{\Omega(\log N)}$$

to be γ-approximated over the distribution $\mathcal{D} := \mathcal{D}^{\mathsf{Ref}^G_s(F),m}$ given by Theorem 2.4, since we can take ε sufficiently smaller than γ. This distribution can be constructed in time $2^{O(\sqrt{n}\log n)}$ given F and m. The ETH lower bound follows by noticing that $N^{(\log N)/(\log\log N)^2} = 2^{O(n)}$. ◀

4.2 rETH-hardness of Monotone Circuit-Formula Gap Learning

We now show slightly stronger results but under rETH.

▶ **Theorem 4.5** (Junta Gap-Learning under rETH). *Let $\gamma > 0$ be a constant. For large enough $n \in \mathbb{N}$, the 3SAT problem on n variables randomly reduces to $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ in time $2^{O(n/\log n)}$, where $s_1(N) = (\log N)^{O(\log\log N)}$ and $s_2(N) = N^{\Omega(\log N)}$. Consequently, the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ needs time $N^{\Omega(\log\log N)}$ to be solved under rETH.*

Proof. Let F be a given 3-CNF over n variables. Fix $d = \log n$. Note that $\mathsf{Ref}^G_s(F)$ is a formula of width $O(1)$ over $N = 2^{\Theta(n/\log n)}$ variables and with $N^{O(1)}$ clauses. Let

$$m := n^3/(\log n)^3,$$

and let $\ell = \varepsilon dm/n$ for a sufficiently small constant $\varepsilon > 0$. The function $f^* := f_{\mathsf{Ref}^G_d(F),m}$ is defined on $n_0 := N^{O(1)}m^{O(1)} = 2^{O(n/\log n)}$ variables (Definition 2.3). Theorems 2.4 and 3.5 imply that f^* can be computed with monotone formulas of size $s := m^{O(d+\log n)} = 2^{O(\log^2 n)} = 2^{O(\log\log N)^2}$. when F is satisfiable. The rest of the proof is the same as in Theorem 4.3. ◀

▶ **Theorem 4.6** (Polynomial vs. quasipolynomial gap under rETH). *Let $\gamma > 0$ be a constant. For large enough $n \in \mathbb{N}$, the 3SAT problem on n variables reduces to $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ in time $2^{O(\sqrt{n})}$, where $s_1(N) = N$ and $s_2(N) = N^{\Omega(\log N)}$. Consequently, the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ needs time $N^{\Omega(\log N)}$ to be solved under rETH.*

Proof. Let F be a given 3-CNF over n variables. Fix $d = \alpha\sqrt{n}$ where $\alpha > 0$ is small enough constant. Note that $\mathsf{Ref}^G_s(F)$ is a formula of width $O(1)$ over $N = 2^{\Theta(\sqrt{n})}$ variables and with $N^{O(1)}$ clauses. Let

$$m := (\log N)^2 \asymp n,$$

and let $\ell = \varepsilon dm/n$ for a sufficiently small constant $\varepsilon > 0$. The function $f^* := f_{\mathsf{Ref}^G_d(F),m}$ is defined on $n_0 := N^{\Theta(1)}m^{\Theta(1)} = 2^{\Theta(\sqrt{n}\log n)}$ variables (Definition 2.3). Theorems 2.4 and 3.5 imply that f^* can be computed with monotone formulas of size

$$s := m^{O(d+\log n)} = 2^{O(d)} = 2^{O(\alpha\sqrt{n})} \le n_0,$$

when F is satisfiable and α is small enough. Moreover, when F is unsatisfiable, the function requires monotone circuits of size at least

$$s' := 2^{\Omega(\ell)} = N^{\Omega(\log N)}$$

to be γ-approximated over the distribution $\mathcal{D} := \mathcal{D}^{\mathsf{Ref}^G_s(F),m}$ given by Theorem 2.4, since we can take ε sufficiently smaller than γ. This distribution can be constructed in time $2^{O(\sqrt{n})}$ given F and m. The ETH lower bound follows by noticing that $N^{\log N} = 2^{O(n)}$. ◀

4.3 From Gap Learning to Partial-Gap-MCFSP

We now define the partial MCSP problem we will consider. Instead of the entire truth table of a partial function, we consider a more succinct encoding where we are only given locations where the function is defined.

▶ **Definition 4.7.** *Let $s_1, s_2, \varepsilon : \mathbb{N} \to \mathbb{N}$ be time-constructible functions. Let $\mathsf{mMCFSP}^{s_2}_{s_1}[\varepsilon]$ be the promise problem which receives as input 1^n and a list of m pairs $((x_1, b_1), \ldots, (x_m, b_m))$ (called* examples*), where $x_i \in \{0,1\}^n$ and $b_i \in \{0,1\}$ for all $i \in [m]$, and must distinguish between the following two cases:*

- *there is a monotone formula C of size at most s_1 such that, for all $i \in [m]$, $C(x_i) = b_i$;*
- *for all circuits C' of size less than s_2, we have $|\{i \in [m] : C'(x_i) \neq b_i\}| \geq m(1/2 - \varepsilon)$.*

The number n is called the dimension *of the input.*

To show hardness of mMCFSP, it suffices to reduce from mCFGL. As described in the introduction, it suffices to sample from the distribution $\mathcal{D}$ given as input to mCFGL, In the small formula case, we clearly generate examples which can be computed by a small formula. In the large circuit case, with enough examples we can conclude by the Chernoff inequality that no circuit of small size can compute most of those examples correctly.

▶ **Theorem 4.8.** *For every growing functions $s_1, s_2 : \mathbb{N} \to \mathbb{N}$ such that $s_1 \leq s_1' < s_2' \leq s_2$ (for large enough n) and $\gamma : \mathbb{N} \to [0,1]$, the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$ with input $(1^n, \mathcal{D})$ randomly reduces to $\mathsf{mMCFSP}^{s_2'}_{s_1'}[2\gamma]$ in time $\mathsf{poly}(n) + O(\frac{1}{\gamma^2} \cdot s_2' \log s_2')$, producing $O(\frac{1}{\gamma^2} \cdot s_2' \log s_2')$ examples of dimension n.*

Proof. Let $\mathcal{D}$ be given as input for $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma]$. Take $m = \frac{4}{\gamma^2} \cdot K s_2' \log s_2'$ samples from $\mathcal{D}$, and call them $(x_1, b_1), ..., (x_m, b_m)$.

In the case that there exists a formula of size $s_1 < s_1'$ computing $\mathcal{D}$ then that same small circuit will agree with $(x_1, b_1), ..., (x_m, b_m)$ for all i. So "small" instances map to "small" instances. It will then suffice to prove the theorem to show that "large" instances map to "large" instances. In particular, it suffices to show that, with all but negligible probability, for every circuit of C' of size less than s_2', we have $|\{i \in [m] : C'(x_i) \neq b_i\}| \geq m(1/2 - \varepsilon - \gamma)$.

Let $a_{C'}$ be the number of $i \in [m]$ such that $C'(x_i) = b_i$. By definition of mCFGL, we know $\mathbb{P}_{(x,b) \leftarrow \mathcal{D}}[C'(x) = b] \leq 1/2 + \gamma$, and thus $\mathbb{E}[a_{C'}] \leq m(1/2 + \gamma)$. Therefore, for each circuit C', by Chernoff-Hoeffding's inequality we know that

$$\mathop{\mathbb{P}}_{(x_1,b_1),\ldots,(x_m,b_m) \leftarrow \mathcal{D}}[a_{C'} - m(1/2 + \gamma) \geq m\gamma] \leq e^{-2m^2\gamma^2/m} = e^{-2m\gamma^2} = (s_2')^{-4s_2'}, \tag{9}$$

or, equivalently,

$$\mathop{\mathbb{P}}_{(x_1,b_1),\ldots,(x_m,b_m) \leftarrow \mathcal{D}}[a_{C'} \geq m(1/2 + 2\gamma)] \leq (s_2')^{-4s_2'}.$$

There are less than $(s_2')^{3s_2'}$ monotone circuits with n bit inputs, s_2' gates, fan-in 2, and arbitrary fan-out. By union bounding over these circuits, we can see that the probability of any circuit satisfying $a_{C'} \geq m(1 + 2\gamma)$ is less than $1/3$ for large enough n. This completes the proof. ◀

We now employ the hardness results obtained for mCFGL under rETH in Section 4.2 to obtain hardness of $\mathsf{mMCFSP}^{s_2}_{s_1}$ using the reduction of the previous theorem. We consider 4 choice of parameters which are in a way best possible:

1. *Junta vs. quasipolynomial.* Here we take the largest gap possible between formula and circuit size. This gives us a weaker runtime lower bound of $N^{\Omega(\log\log N)^{1-\varepsilon}}$. This is the weakest runtime lower bound we obtain.
2. *Linear vs. quasipolynomial.* Here we obtain a weaker gap but against superpolynomial-size circuits. This gives us a better runtime lower bound of $N^{\Omega(\log N)^{1-\varepsilon}}$.

3. *Junta vs. polynomial.* We then consider the setting where the gap is largest possible subject to $s_2 = \mathsf{poly}(n)$. This gives us a runtime lower bound of $N^{\Omega(\log\log N)}$, slightly better than (1).
4. *Linear vs. polynomial.* We weaken the gap to an arbitrary polynomial n^c. This gives us a better runtime lower bound of $N^{\Omega(\log N)}$, slightly better than (2). This is the strongest runtime lower bound we obtain.

▶ **Corollary 4.9.** *Let $\gamma > 0$ be a constant. Under rETH, the following holds:*

1. ***(Junta vs. quasipolynomial)*** *For every constant $\varepsilon > 0$, the problem $\mathsf{mMCFSP}^{s_2}_{s_1}[\gamma]$ requires time $N^{\Omega(\log\log N)^{\varepsilon}}$ to be decided, where $s_1(n) = (\log N)^{\log\log N}$ and $s_2(n) = n^{(\log\log n)^{1-\varepsilon}}$. The hard instances consist of $m = \Theta(s_2 \log s_2)$ examples, and thus have size $N = nm = n^{\Theta(\log\log n)^{1-\varepsilon}}$.*
2. ***(Linear vs. quasipolynomial)*** *For every constant $\varepsilon \in (0,1)$, the problem $\mathsf{mMCFSP}^{s_2}_{s_1}[\gamma]$ requires time $N^{\Omega((\log N)^{2/(2-\varepsilon)})}$ to be decided, where $s_1(n) = n$, and $s_2(n) = n^{(\log n)^{1-\varepsilon}}$. The hard instances consist of $m = \Theta(s_2 \log s_2)$ examples, and thus have size $N = nm = n^{\Theta(\log n)^{1-\varepsilon}}$.*
3. ***(Junta vs. polynomial)*** *For every constant $\varepsilon > 0$, the problem $\mathsf{mMCFSP}^{s_2}_{s_1}[\gamma]$ requires time $N^{\Omega(\log\log N)}$ to be decided, where $s_1(n) = (\log N)^{\log\log N}$ and $s_2(n) = n^c$. The hard instances consist of $m = \Theta(s_2 \log s_2)$ examples, and thus have size $N = \Theta(nm) = \mathsf{poly}(n)$.*
4. ***(Linear vs. polynomial)*** *For every constant $c > 0$, the problem $\mathsf{mMCFSP}^{s_2}_{s_1}[\gamma]$ requires time $N^{\Omega(\log N)}$ to be decided, where $s_1(n) = n$, and $s_2(n) = n^c$. The hard instances consist of $m = \Theta(s_2 \log s_2)$ examples, and thus have size $N = \Theta(nm) = \mathsf{poly}(n)$.*

Proof. We first prove Item (1). By Theorem 4.5, for large enough $n \in \mathbb{N}$ the problem $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma/2]$ requires time $n^{\Omega(\log n)}$ to be computed under ETH for some $s_1 = 2^{O(\log\log n)^2}$ and $s_2 = n^{\Omega(\log n)}$. Take $s_1' = s_1$ and $s_2' = n^{(\log\log n)^{1-\varepsilon}}$, thus satisfying $s_1 < s_1' < s_2' < s_2$.

If $\mathsf{mMCFSP}^{s_2'}_{s_1'}[\gamma]$ can be computed in time $T(N)$ on an instance of $m = Ks_2' \log s_2' = n^{O(\log\log n)^{1-\varepsilon}}$ examples and dimension n (and thus input size $N = nm$), where K is a large enough constant, we obtain by Theorem 4.8 that $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma/2]$ can be solved in randomised time $\mathsf{poly}(n) + O(m) + T(N)$. This must be at least $n^{\Omega(\log\log n)}$ which implies $T(N) = n^{\Omega(\log\log n)}$. Since $N = n^{\Theta(\log n \log n)^{1-\varepsilon}}$, we obtain $\log n \log n = (1 - o(1)) \log\log N$ and

$$T(N) \geq N^{\Omega(\log\log N)^{\varepsilon}}.$$

We now prove Item (2). The proof is similar. Let $s_1' = n$ and $s_2' = n^{(\log n)^{1-\varepsilon}}$ in the argument above. We have that $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma/2]$ requires time $n^{\Omega(\log n)}$ to be computed by Theorem 4.6. Since $N \asymp ns_2' \log s_2' = n^{\Theta(\log n)^{1-\varepsilon}}$, we obtain $\log n \asymp (\log N)^{1/(2-\varepsilon)}$ and thus

$$T(N) \geq 2^{\Omega(\log n)^2} \geq 2^{\Omega(\log N)^{2/(2-\varepsilon)}}.$$

We repeat the argument for Item (3). Take $s_1' = (\log n)^{\log\log n}$ and $s_2' = n^c$. We have that $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma/2]$ requires time $n^{\Omega(\log\log n)}$ to be computed by Theorem 4.5. Since $N \asymp ns_2' \log s_2' \asymp n^{c+1} \log n$, we obtain $\log N \asymp \log n$ and $\log\log N \sim \log\log n$, and thus

$$T(N) \geq n^{\Omega(\log\log n)} \geq N^{\Omega(\log\log N)}.$$

Finally, we prove Item (4). Let $s_1' = n$ and let $s_2' = n^c$ in the argument above. We have that $\mathsf{mCFGL}^{s_2}_{s_1}[\gamma/2]$ requires time $n^{\Omega(\log n)}$ to be computed by Theorem 4.6. Since $N \asymp ns_2' \log s_2' \asymp n^{c+1} \log n$, we obtain $\log N \asymp \log n$ and thus

$$T(N) \geq 2^{\Omega(\log n)^2} \geq 2^{\Omega(\log N)^2}.$$

◀

4.4 From Gap Learning to Improper PAC-Learning

Given a function $c : \{0,1\}^n \to \{0,1\}$ and a distribution $\mathcal{D}$ with support in $\{0,1\}^n$, we denote by $\mathsf{Ex}(c, \mathcal{D})$ the oracle that runs in unit time and outputs a sample (x, b) where $x \leftarrow \mathcal{D}$ and $b = c(x)$. We also define $\mathsf{err}_{c,\mathcal{D}}(h) := \mathbb{P}_{x \in \mathcal{D}}[h(x) \neq c(x)]$.

▶ **Definition 4.10.** *Let $\mathcal{C}_n, \mathcal{H}_n$ be sets of Boolean functions over $\{0,1\}^n$, and let $\mathcal{C} = \bigcup_{n\geq 1} \mathcal{C}_n$ and $\mathcal{H} = \bigcup_{n\geq 1} \mathcal{H}_n$ be such that $\mathcal{C} \subseteq \mathcal{H}$. Let $T : \mathbb{N} \to \mathbb{N}$. We say that the* concept class $\mathcal{C}$ *is* PAC-learnable in time T *by the* hypothesis class $\mathcal{H}$ *if there is a randomised algorithm such that, for every $c \in \mathcal{C}$, given $(1^n, 1^{1/\varepsilon}, 1^{1/\delta})$ and oracle access to $\mathsf{Ex}(c, \mathcal{D})$, outputs in time $T(n + 1/\varepsilon + 1/\delta)$, with probability $1 - \delta$, a concept $h \in \mathcal{H}$ such that $\mathsf{err}_{c,\mathcal{D}}(h) \leq \varepsilon$.*

For $s : \mathbb{N} \to \mathbb{N}$, let $\mathcal{C}_s$ (resp. $\mathcal{L}_s$) be the set of monotone Boolean circuits (resp. formulas) on n bits of size $s(n)$. Let $\mathcal{C} = \bigcup_{n\in\mathbb{N}} \mathcal{C}_{s(n)}$ and $\mathcal{L} = \bigcup_{n\in\mathbb{N}} \mathcal{L}_{s(n)}$. Hardness of PAC-learning $\mathcal{L}$ by $\mathcal{C}$ follows from the results of Section 4.2 by a proof similar to the one employed in the previous section.

▶ **Theorem 4.11.** *Under rETH, we have:*

1. *For every constant $\alpha > 0$, monotone formulas of size n require time $n^{\Omega(\log n)}$ to be PAC-learned by monotone circuits of size $s(n)$, for any $s(n)$ such that $n \ll s(n) \leq n^{(\log n)^{1-\alpha}}$.*
2. *For every constant $\alpha > 0$, monotone formulas of size $(\log n)^{O(\log\log n)}$ require time $n^{\Omega(\log\log n)}$ to be PAC-learned by monotone circuits of size $s(n)$, for any $s(n)$ such that $n \ll s(n) \leq n^{(\log\log n)^{1-\alpha}}$.*

Proof. We first prove the first item. The proof of the second item is the same, so we omit it. Let $\varepsilon = \delta = 1/3$. Let $(1^n, \mathcal{D})$ be an input to $\mathsf{mCFGL}^{s_2}_{s_1}[\varepsilon]$, where s_1, s_2 are as in Theorem 4.4. Let $s'_1 = n$ and $s'_2 = s(n)$. Suppose there is an PAC-learner $\mathcal{A}$ for $\mathcal{L}_n$ outputting a hypothesis in $\mathcal{C}_{s'_2}$ in time $T = T(n + 1/\varepsilon + 1/\delta)$.

Run the algorithm $\mathcal{A}$, replacing oracle calls to Ex with samples from $\mathcal{D}$, and obtaining a circuit C' at the end. Since $|\mathcal{D}| = \mathsf{poly}(n)$, this can be done in time $T \cdot \mathsf{poly}(n) + O(s'_2 \log s'_2)$, as we also need $O(s'_2 \log s'_2)$ bits to describe C'. We now take $O(\frac{1}{\gamma^2})$ samples from $\mathcal{D}'$, and accept $\mathcal{D}$ if C' computes correctly $2/3$ of the examples. In the YES case, we will be correct with probability $2/3$ since there is a formula of size s_1 computing correctly with probability 1. In the NO case, we know that $\mathbb{P}_{(x,b)\leftarrow\mathcal{D}}[C'(x) = b] \leq 1/2+\gamma$, and thus the number of examples $a_{C'}$ that C' computes correctly is less than $2/3$ with high probability by inequality (9). The total runtime is $T\mathsf{poly}(n) + O(s'_2 \log s'_2 + 1/\gamma^2)$, implying that $T = n^{\Omega(\log n)}$. ◀

References

1 Michael Alekhnovich, Mark Braverman, Vitaly Feldman, Adam R. Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. *J. Comput. Syst. Sci.*, 74(1):16–34, 2008. doi:10.1016/J.JCSS.2007.04.011.

2 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

3 Benny Applebaum, Boaz Barak, and David Xiao. On basing lower-bounds for learning on worst-case assumptions. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science FOCS '08*, pages 211–220, 2008. doi:10.1109/FOCS.2008.35.

4 Noel Arteche, Albert Atserias, Susanna F. de Rezende, and Erfan Khaniki. The proof analysis problem. In *Proceedings of the 66th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 2555–2576, 2025. doi:10.1109/focs63196.2025.00133.

5 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, 2008. Preliminary version in *CCC '03*. doi:10.1016/J.JCSS.2007.06.025.

6 Albert Atserias and Moritz Müller. Automating resolution is NP-hard. *Journal of the ACM*, 67(5), 2020. Preliminary version in *FOCS '19*. doi:10.1145/3409472.

7 Gaia Carenini and Susanna F. de Rezende. On the automatability of tree-like k-dnf resolution. In *Proceedings of the 40th Computational Complexity Conference (CCC '25)*, volume 339 of *LIPIcs*, pages 14:1–14:21, 2025. doi:10.4230/LIPIcs.CCC.2025.14.

8 Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *Proceedings of the 31st Conference on Computational Complexity (CCC '16)*, volume 50 of *LIPIcs*, pages 10:1–10:24, 2016. doi:10.4230/LIPIcs.CCC.2016.10.

9 Dana Dachman-Soled, Homin K. Lee, Tal Malkin, Rocco A. Servedio, Andrew Wan, and Hoeteck Wee. Optimal cryptographic hardness of learning monotone functions. *Theory Comput.*, 5(1):257–282, 2009. doi:10.4086/TOC.2009.V005A013.

10 Amit Daniely. Complexity theoretic limitations on learning halfspaces. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 16)*, pages 105–117, 2016. doi:10.1145/2897518.2897520.

11 Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning DNF's. In *Proceedings of the 29th Conference on Learning Theory (COLT '16)*, pages 815–830, 2016. URL: http://proceedings.mlr.press/v49/daniely16.html.

12 Amit Daniely and Gal Vardi. From local pseudorandom generators to hardness of learning. In *Proceedings of the 34th Conference on Learning Theory (COLT '21)*, pages 1358–1394. PMLR, 2021. URL: http://proceedings.mlr.press/v134/daniely21a.html.

13 Susanna F. de Rezende. Automating tree-like resolution in time $n^{o(\log n)}$ is ETH-hard. In *Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium, (LAGOS '21)*, volume 195, pages 152–162, 2021. doi:10.1016/J.PROCS.2021.11.021.

14 Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21)*, pages 209–222, 2021. doi:10.1145/3406325.3451080.

15 Susanna F. de Rezende and Marc Vinyals. Lifting with colourful sunflowers. In *Proceedings of the 40th Computational Complexity Conference (CCC 25)*, pages 36:1–36:19, 2025. doi:10.4230/LIPIcs.CCC.2025.36.

16 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC 02)*, pages 534–543, 2002. doi:10.1145/509907.509985.

17 Vitaly Feldman, Homin K. Lee, and Rocco A. Servedio. Lower bounds and hardness amplification for learning shallow monotone formulas. In *Proceedings of the 24th Conference on Learning Theory (COLT '11)*, pages 273–292, 2011. URL: http://proceedings.mlr.press/v19/feldman11a/feldman11a.html.

18 Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. *Theory of Computing*, 16(1):1–30, 2020. Preliminary version in *STOC '18*. doi:10.4086/toc.2020.v016a013.

19 Mika Göös, Pritish Kamath, Robert Robere, and Dmitry Sokolov. Adventures in monotone complexity and TFNP. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 38:1–38:19, 2019. doi:10.4230/LIPIcs.ITCS.2019.38.

20 Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 20)*, pages 68–77, 2020. doi:10.1145/3357713.3384248.

21 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018. Preliminary version in FOCS '15. doi:10.1137/16M1059369.

22 Shuichi Hirahara. NP-hardness of learning programs and partial MCSP. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science FOCS 22*, pages 968–979, 2022. doi:10.1109/FOCS54457.2022.00095.

23 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. *SIAM J. Comput.*, 52(6):S18–349, 2023. doi:10.1137/19M124705X.

24 Shuichi Hirahara, Igor C. Oliveira, and Rahul Santhanam. NP-hardness of minimum circuit size problem for OR-AND-MOD circuits. In *Proceedings of the 33rd Computational Complexity Conference (CCC 18)*, pages 5:1–5:31, 2018. doi:10.4230/LIPIcs.CCC.2018.5.

25 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(04):439–562, August 2006. doi:10.1090/s0273-0979-06-01126-8.

26 Yizhi Huang, Rahul Ilango, and Hanlin Ren. NP-hardness of approximating meta-complexity: A cryptographic approach. *SIAM J. Comput.*, 54(4):819–886, 2025. doi:10.1137/23M1608483.

27 Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and $\mathrm{AC}^0[p]$. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS '20)*, pages 34:1–34:26, 2020. doi:10.4230/LIPIcs.ITCS.2020.34.

28 Rahul Ilango. Constant depth formula and partial function versions of MCSP are hard. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS '20)*, pages 424–433, 2020. doi:10.1109/FOCS46700.2020.00047.

29 Rahul Ilango. The minimum formula size problem is (ETH) hard. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS '21)*, pages 427–432, 2021. doi:10.1109/FOCS52979.2021.00050.

30 Rahul Ilango, Bruno Loff, and Igor C. Oliveira. NP-hardness of circuit minimization for multi-output functions. In *Proceedings of the 35th Computational Complexity Conference (CCC '20)*, pages 22:1–22:36, 2020. doi:10.4230/LIPIcs.CCC.2020.22.

31 Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th Annual IEEE Structure in Complexity Theory Conference (SCT 95)*, pages 134–147, 1995. doi:10.1109/SCT.1995.514853.

32 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 73–79, 2000. doi:10.1145/335305.335314.

33 Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994. doi:10.1145/174644.174647.

34 Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC '93)*, pages 372–381, 1993. doi:10.1145/167088.167197.

35 Jan Kraíček. Interpolation by a game. *Mathematical Logic Quarterly*, 44(4):450–458, 1998. doi:10.1002/MALQ.19980440403.

36 Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997. doi:10.2307/2275541.

37 Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. In *Proceedings of 13th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 104:1–104:24, 2022. doi:10.4230/LIPIcs.ITCS.2022.104.

38 William Masek. Some NP-complete set covering problems. Unpublished, 1979.

39 Noam Mazor and Rafael Pass. Gap MCSP is not (Levin) NP-complete in Obfustopia. In *Proceedings of the 39th Computational Complexity Conference, (CCC '24)*, pages 36:1–36:21, 2024. doi:10.4230/LIPIcs.CCC.2024.36.

40 Pavel Pudlák. On reducibility and symmetry of disjoint NP pairs. *Theoretical Computer Science*, 295:323–339, 2003. doi:10.1016/S0304-3975(02)00411-5.

41 Ran Raz and Pierre McKenzie. Separation of the Monotone NC Hierarchy. *Combinatorica*, 19(3):403–435, 1999. Preliminary version in *STOC* 97. doi:10.1007/S004930050062.

42 Alexander A Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya: mathematics*, 59(1):205, 1995.

43 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997. doi:10.1006/JCSS.1997.1494.

44 Rahul Santhanam. Pseudorandomness and the minimum circuit size problem. In Thomas Vidick, editor, *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS '20)*, pages 68:1–68:26, 2020. doi:10.4230/LIPIcs.ITCS.2020.68.