Electronic Colloquium on Computational Complexity, Report No. 127 (2021)

Small Circuits Imply Efficient Arthur-Merlin Protocols

Michael Ezra* Ron D. Rothblum*

August 30, 2021

Abstract

The inner product function $\langle x, y\rangle = \sum_i x_i y_i \bmod 2$ can be easily computed by a (linear-size) $\mathsf{AC}^0(\oplus)$ circuit: that is, a constant depth circuit with AND, OR and parity (XOR) gates. But what if we impose the restriction that the parity gates can only be on the bottom most layer (closest to the input)? Namely, can the inner product function be computed by an AC^0 circuit composed with a single layer of parity gates? This seemingly simple question is an important open question at the frontier of circuit lower bound research.

In this work, we focus on a minimalistic version of the above question. Namely, whether the inner product function cannot be *approximated* by a small DNF augmented with a single layer of parity gates. Our main result shows that the existence of such a circuit would have unexpected implications for interactive proofs, or more specifically, for interactive variants of the Data Streaming and Communication Complexity models. In particular, we show that the existence of such a small (i.e., polynomial-size) circuit yields:

1. An $O(d)$-message protocol in the Arthur-Merlin Data Streaming model for every n-variate, degree d polynomial (over $\mathbb{GF}(2)$), using only $\widetilde{O}(d) \cdot \log(n)$ communication and space complexity. In particular, this gives an $\mathsf{AM}[2]$ Data Streaming protocol for a variant of the well-studied triangle counting problem, with poly-logarithmic communication and space complexities.
2. A 2-message communication complexity protocol for any sparse (or low degree) polynomial, and for any function computable by an $\mathsf{AC}^0(\oplus)$ circuit. Specifically, for the latter, we obtain a protocol with communication complexity that is poly-logarithmic in the size of the $\mathsf{AC}^0(\oplus)$ circuit.

*Technion. Supported in part by a Milgrom family grant, by the Israeli Science Foundation (Grants No. 1262/18 and 2137/19), and grants from the Technion Hiroshi Fujiwara cyber security research center and Israel cyber directorate. Email: `{michaelezra,rothblum}@cs.technion.ac.il`

ISSN 1433-8092

1 Introduction

Understanding the expressive power of bounded depth circuits is a central goal in complexity theory, with the hope of eventually answering fundamental questions, such as $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ or $\mathsf{P} \not\subseteq \mathsf{NC_1}$. Seminal works from the 80's showed that the parity function cannot be computed by $\mathsf{AC^0}$ circuits - that is, constant-depth polynomial-size circuits with unbounded fan-in AND, OR and NOT gates [FSS81, Ajt83, Hås86]. Razborov and Smolensky [Raz87, Smo87] took the next step forward by considering the class $\mathsf{AC^0}(\oplus)$, which extends $\mathsf{AC^0}$ by allowing also (unbounded fan-in) parity gates, and showed that this class cannot compute the majority or $\bmod p$ functions. Most recently, Williams [Wil14] separated the class $\mathsf{ACC^0}$, in which the circuit is further allowed to use arbitrary $\bmod p$ gates, from the class NEXP of non-deterministic exponential-time computations (see also the recent exciting sequence of works [CW19, VW20, CLW20, CR20, MW20]).

Despite these results, we are still far from understanding the power of constant-depth circuits. For example, it is easy to construct an $\mathsf{AC^0}(\oplus)$ circuit for computing the inner product function: simply take the parity of the respective point-wise products. On the other hand, if we do not allow parity gates, then it is easy to show a lower bound. A natural question that arises is whether a similar lower bound holds if we augment the $\mathsf{AC^0}$ circuit with a single layer of parity gates immediately after the input layer. The resulting circuit class is called an $\mathsf{AC^0}$ of parities (and is sometimes denoted by $\mathsf{AC^0}_\oplus$). Recently, there has been growing interest in whether this circuit class can compute the inner product function [Juk06, Rot12, SV12, ABG$^+$14, CS16, CGJ$^+$18, BKT20, FIKK20].

Interestingly, an exponential lower bound for the inner product function *is* known [Jac97] for the special case in which the $\mathsf{AC^0}$ circuit has depth 2.[1] Namely, a DNF of parities, denoted by $\mathsf{DNF}_\oplus$. For depth 3 circuits (on top of the parity layer), only a relatively weak (quadratic) lower bound is known [CGJ$^+$18]. Lastly, for general $\mathsf{AC^0}_\oplus$ circuits, an exponential lower bound is known [FIKK20] only for the very restricted case in which the number of parity gates is linear.[2]

Even for the case of DNFs, the lower bound arising from the work of Jackson [Jac97] only rules out an (almost) *exact* $\mathsf{DNF}_\oplus$ for computing the inner product function. Thus, a question (posed explicitly by Cohen and Shinkar [CS16]) that seems just beyond the reach of current techniques is:

"Does there exist a small DNF of parities that approximates the inner product function?"

We refer to the assumption that a positive answer holds for this question as the $\mathsf{IP}_2\widetilde{\in}\mathsf{DNF}_\oplus$ hypothesis (In the actual theorem statements below we use a quantitatively precise version of the assumption). In this work, we study the ramifications of the $\mathsf{IP}_2\widetilde{\in}\mathsf{DNF}_\oplus$ hypothesis, with the belief that these results develop our understanding of the hypothesis or could even bring us closer to the eventual goal of *refuting* it.

1.1 Our Results

We show that a positive answer to the $\mathsf{IP}_2\widetilde{\in}\mathsf{DNF}_\oplus$ hypothesis implies (unexpected) efficient interactive (Arthur-Merlin) protocols for a large class of problems (in different models to be described below). We note that the quantitative parameters of the resulting protocols seem to be far more efficient than expected. Thus, these results fall in line with our belief that the $\mathsf{IP}_2\widetilde{\in}\mathsf{DNF}_\oplus$ hypothesis is false. Moreover, these results also form a new approach for *refuting* the $\mathsf{IP}_2\widetilde{\in}\mathsf{DNF}_\oplus$

[1]This bound was tightened by Cohen and Shinkar [CS16], who gave a lower bound that exactly matches the known upper bound.

[2]In fact, their result holds for the more general case in which an arbitrary (i.e., not necessarily linear) preprocessing step is done first on the two parts of the input separately.

hypothesis through Arthur-Merlin lower bounds, in the sense that progress in finding Arthur-Merlin lower bounds can be applied, using our results, to refute the $\mathsf{IP}_2 \widetilde{\in} \mathsf{DNF}_{\oplus}$ hypothesis. While finding lower bounds for Arthur-Merlin protocols is a notoriously difficult problem (e.g., in communication complexity), all of our protocols go through efficient *Holographic-Interactive protocols* (see Section 2.1), where Arthur-Merlin lower bounds *are* known [GR17].

The models that we consider are "Arthur-Merlin" variants of the standard Data Streaming and the Communication Complexity models. In order to describe these variants, we first recall the standard definitions of Data Streaming and Communication Complexity models and then explain how they are extended to Arthur-Merlin variants, by giving the relevant parties access to an all-powerful (but untrusted) prover.

Recall that in the standard Data Streaming Model (popularized by [AMS99]), a bounded space algorithm is required to compute a certain function of the inputs by using the least amount of space. The algorithm gets the input bits as a sequence of bits (stream), in the sense that after seeing a bit in the sequence, the algorithm no longer has access to the bits that preceded it (unless these were stored in its memory). In the standard Communication Complexity Model [Yao79], there are two parties, called Alice and Bob, who are trying to evaluate a function f on their joint input. That is, Alice and Bob are given inputs x and y, respectively, and need to jointly compute the value $f(x, y)$, while transmitting the least amount of bits.

We focus on the Arthur-Merlin (AM) variant of these models, where the parties are also assisted by an untrusted prover, often referred to as Merlin, who sees all inputs (and has unlimited computational resources). The parties are allowed to make a short *public coin* interaction with Merlin, *before* (deterministically) running the standard protocol. The interaction is of the AM (Arthur-Merlin) type in the sense that the messages to Merlin consist of only fresh random coins, and in particular, do not depend on the input bits nor on previous messages that were exchanged. Beyond the coins that were revealed to Merlin in the interaction, the parties are not allowed to toss any additional coins. Throughout this work we use the notation $\mathsf{AM}[k]$ to refer to AM protocols with k messages exchanged between the parties.

1.1.1 The AM Data Streaming Model

In the AM Data Streaming Model [CTY11, CCM+13, GR13, CCGT14, CCMT14, CCM+15, Tha16, CGT20], we allow the bounded space algorithm processing the stream, to interact with the untrusted prover Merlin, who sees the entire input (and is not space bounded). Many of these works differ in the exact form of the interaction. For example, does the small-space verifier get full access to messages sent by the prover, or merely streaming access? We elaborate on these differences in Section 2.2.2. In this work we consider the following natural model, which we refer to as the $\mathsf{AM}[k]$ Data Streaming model:

1. In the first phase, the verifier engages in a k-message public-coin interactive protocol with the prover (starting with a verifier message). At the end of this phase the verifier holds a transcript τ.

2. In the second phase, the verifier is allowed to process the input stream in a bit-by-bit manner. The verifier's computation in this phase is allowed to depend on the transcript τ that it saw. We emphasize that the verifier in this phase is deterministic.[3]

3. After processing the stream, the verifier decides whether to accept or reject.

[3]The verifier could in principle toss additional coins in the first phase to be used in the second phase, but we count this as an additional message. This is motivated by the definition of the classical complexity class AM which does not allow Arthur additional coin tosses after seeing Merlin's message.

As usual, we require that there is a strategy for Merlin to convince the streaming verifier to accept true statements, but the verifier rejects any false statements (with high probability) even if Merlin cheats. Naturally, we require the space complexity of the verifier to be small and the communication with the prover to be short as well (since otherwise Merlin can provide the entire input!).

As our first result, assuming the existence of a small DNF of parities for the inner product, we construct (multi-round) AM Data Streaming protocol for *any* function f that can be computed by a low-degree polynomial (over $\mathbb{GF}(2)$).

Theorem 1 (Informally stated (see Theorem 9)). *Assume that there exists a* DNF *of parities of size S, that computes the inner product function on $\frac{5}{6}+\epsilon$ fraction of the inputs, for some constant $\epsilon > 0$. Then, there exists an* AM$[2d]$ *Data Streaming protocol with $\widetilde{O}(d) \cdot \log(S)$ proof length, space complexity and randomness complexity, for every degree d polynomial over $\mathbb{GF}(2)$.*

When S is polynomial and d is constant, the protocol has poly-logarithmic proof and space complexities. This should be contrasted with approaches based on constant-round versions of the celebrated sumcheck protocol [LFKN92], which have polynomial proof-length.

We also emphasize that (as usual in this context) here and throughout this work, we do not consider the *computational complexity* of the verifier and only focus on the space and communication complexities.

Application: A Streaming Protocol for Counting Triangles Mod 2. We also point out an interesting implication of Theorem 1 to a variant of the well studied Triangle-Count problem. In the Triangle-Count problem, a streaming algorithm is required to count (or sometimes just approximate) the number of triangles (i.e., cliques of three vertices) in an undirected (simple) graph. A large body of work has studied this problem in the streaming context in general [BKS02, JG05, BFL+06, KMPT12, KMSS12, JSP13, MVV16, BC17, KMPV19], and in particular when the streaming algorithm is assisted by a prover [CCMT14, Tha16, CGT20]. There are two main variants of the Triangle-Count problem, which differ in the exact form that the input is given to the streaming algorithm. In the first variant, studied in [BKS02, BFL+06, KMPT12, MVV16, KMPV19], the edges are given in an adjacency-list format. Namely, first, the edges connected to the first vertex appear in the stream, then the edges that are connected to the second vertex, and so on. In the second variant (also referred to as the dynamic updates variant), studied in [BKS02, JG05, BFL+06, KMSS12, JSP13, MVV16, BC17], the stream consists of dynamic additions (and sometime also deletions) of edges, in an arbitrary order.

We consider a variant of the Triangle-Count problem, denoted by $\bigoplus$Triangle, where the goal is to compute the *parity* of the number of triangles in the graph. We consider in which the graph is given as a stream of its edges, where each edge appears in the stream exactly once. We note that the MA complexity[4] of $\bigoplus$Triangle is well understood: for every proof length p and verifier space complexity s, it holds that $s \cdot p = \Omega(n^2)$ [CCMT14, Tha16].[5] Also, a matching quadratic upper bound is known for (almost) any combination of $s \cdot p = \widetilde{O}(n^2)$ [Tha16, CGT20]. On the other hand, this problem has no known (non-trivial) upper or lower bounds in the AM setting.

Assuming the $\mathsf{IP}_2 \widetilde{\in} \mathsf{DNF}_{\oplus}$ hypothesis, we show an efficient AM protocol for $\bigoplus$Triangle. Our protocol has space complexity and proof length that are poly-logarithmic in the circuit size (regardless of the specific order of the edges in the stream).

[4]Loosely speaking, in an MA model, first the prover sends a proof message. Then, the verifier gets the input as a stream, and conducts a (randomized) streaming computation.

[5]The lower bound is not stated explicitly for this problem, but follows from the fact that it holds for the case that the graph is promised to contain exactly one triangle or be triangle-free.

Theorem 2 (AM Streaming for $\bigoplus$Triangle, informally stated (see Theorem 11)**.** *Assume that there exists a* DNF *of parities of size S that computes the inner product function on $\frac{5}{6}+\epsilon$ fraction of the inputs, for some constant $\epsilon > 0$. Then, there exists an* AM[2] *Data Streaming protocol for* $\bigoplus$Triangle *with* $\text{polylog}\left(S\left(n^3\right)\right)$ *proof length and space complexity.*

Indeed, assuming that S is polynomial, the protocol of Theorem 2 has poly-logarithmic proof length and space complexity.

1.1.2 AM Communication Complexity

We next describe our results in the (AM) Communication Complexity model. In the AM Communication Complexity Model [Kla03, AW09, GS10, Kla11, GPW16], Alice and Bob are allowed to also conduct a public-coin interaction with the prover Merlin, who sees both of their inputs, but is non trustworthy. The parties communicate using a broadcast channel, namely, each of the parties is exposed to all the messages sent by Merlin, and all the random coins tossed by Alice and Bob. For sake of simplicity, we can assume that Alice and Bob do *not* interact, since Merlin can simply provide all messages that they would have exchanged had they interacted (and the two parties can check that the communication is consistent with what they would have sent). As above, we require that Merlin will convince *both* Alice and Bob of the correctness of true statements, but no matter what Merlin does, with high probability either Alice or Bob will reject false statements.

It is not hard to show that any Data Streaming protocol can be transformed into a Communication Complexity protocol, for the same problem, as follows: Alice starts running the data streaming algorithm until the algorithm finishes processing her portion of the input (i.e., at the midpoint). She then transmits to Bob her memory state. Bob continues the emulation using his portion of the input. The communication complexity of the resulting protocol is therefore at most the space complexity of the streaming algorithm.

Thus, Theorem 1 immediately implies an AM communication complexity protocol for low-degree polynomials as well. Interestingly, however, we are able to achieve significantly better parameters by constructing an AM Communication Complexity protocol directly. In particular, we construct a one-round protocol, which can also be extended to a protocol for any function that is decidable by an $\mathsf{AC}^0(\oplus)$ circuit. Lastly, we also note that while the protocol in Theorem 1 depends on the degree of the polynomial, the protocol in Theorem 3 depends only on the number of monomials, and therefore can also be applied to high-degree, but sparse, polynomials.

Theorem 3 (Informally stated (see Theorem 8 and Corollary 10))**.** *Assume that there exists a* DNF *of parities of size S that computes the inner product function on $\frac{5}{6}+\epsilon$ fraction of the inputs, for some constant $\epsilon > 0$. Then, there exist:*

- *An* AM[2] *Communication Complexity protocol with $O\left(\log\left(S(2N)\right)\right)$ communication complexity, for every function f that can be expressed as a polynomial (over $\mathbb{GF}(2)$) with N monomials.*

  In particular, if f is a degree d polynomial over $2n$ input bits, the AM[2] *protocol has communication complexity $O\left(\log\left(S\left(2\cdot(2n)^d\right)\right)\right)$.*

- *An* AM[2] *Communication Complexity protocol with $O\left(\log\left(S\left(2^{\text{polylog}(T)}\right)\right)\right)$ communication complexity, for any function that is decidable by an $\mathsf{AC}^0(\oplus)$ circuit of size T.*

Note that the protocols in Theorem 3 are 2-message protocols, whereas the protocol in Theorem 1 require a large number of rounds of interaction. One could potentially reduce the number of rounds using (a suitable variant of) the round collapse theorem [BM88] (see also, Claim 5.2). However, we emphasize that Theorem 3 gives significantly better parameters than round collapsing the protocol of Theorem 1. For example, if $S = \mathrm{poly}(n)$, by applying a round collapse to our data streaming results, we get an $\mathsf{AM}[2]$ Communicating Complexity protocol with $O\left(\log^d n\right)$ communication complexity for degree d polynomials. In contrast, our explicit protocol of Theorem 3 has a *linear* (rather than exponential) dependence on the degree d. This improvement allows us to extend the explicit Communication Complexity protocol for low degree (or sparse) polynomials, also for any function that is decidable by an $\mathsf{AC}^0(\oplus)$ circuit. Interestingly, we do not know how to obtain a non-trivial result of the same flavor from the protocol in Theorem 1.

1.2 Technical Overview

Our main technical step is to construct, assuming that the $\mathsf{IP}_2 \widetilde{\in} \mathsf{DNF}_{\oplus}$ hypothesis holds, a special type of proof-system for computing the inner product function, called a *Holographic Interactive Proof* (HIP) - a notion introduced in the work of Gur and Rothblum [GR17] (inspired by a similar model for PCPs, introduced by Babai et al. [BFLS91]). An HIP is defined similarly to a standard interactive proof, except that the verifier, rather than being given access to the main input explicitly, is given oracle access to an *encoding* of the input. The hope is that the redundancy provided by the encoding will allow the verifier to run in *sub-linear* time. Hence, the main complexity resources that we focus on are the *query complexity*, which is the number of bits that the verifier reads from the encoding, and the *communication complexity*, which is the total number of bits exchanged with the prover. We focus specifically on an $\mathsf{AM}[2]$ variant, where the verifier first sends random coins r to the prover, who responds with a message π, called the proof. The verifier then decides deterministically, based on the input queries, random string r and proof π, whether to accept or reject.

Let us assume therefore that there exists a DNF of parities C of size S that approximates the inner product function. We use C to design an $\mathsf{AM}[2]$ HIP for verifying inner product claims. The input encoding that we will use in the HIP corresponds to the parity layer of the circuit C, and is therefore a *linear* function. This point is crucial for our results.

1.2.1 An $\mathsf{AM}[2]$-HIP for Inner Product Claims

As our first step, we construct a simple HIP for verifying that the inner product of two strings is equal to 1 and which only works for *most* inputs. This falls short of our eventual goal which is to check general inner products and over *worst-case* inputs. Nevertheless, we present this HIP as it will serve as an important ingredient in our construction.

Step 1: Verifying one-sided claims, on the average. Recall that C is a DNF of parities that approximates the inner product function. A simple one-round HIP protocol for verifying whether $f(x) = 1$ on a given input x can be established as follows: the prover sends an index of a satisfied clause (such an index exists if and only if $f(x) = 1$), and the verifier checks whether the clause is indeed satisfied, by reading the bits in the clause from the input's encoding. Note that the proof-system is holographic as the verifier reads each bit in the clause by making a single query to the output of the parity layer. The communication is $\log(S)$ and the query complexity is bounded by the maximal width of the clauses.

Since we seek small query complexity, we would like to ensure that the DNF has small width. To do so we observe that each clause in a DNF of parities can be viewed as a system of linear equations. Also, note that with probability at most $\frac{1}{2^r}$, a random input satisfies a linear system with rank at least r. Therefore, a natural idea is to remove all of the wide clauses. When doing so one should first make sure that the equations forming the clause are linearly independent, which can be easily done (by choosing a maximal set of linearly independent equations). Thus, after eliminating linear dependencies, we remove all clauses with width $\Omega(\log S)$. Since we only removed clauses, the new circuit disagrees with $f(x)$ only if x satisfies one of the removed clauses. Since we only removed clauses of rank greater than $O(\log S)$, by the union bound and setting the constant in the big-O to be large enough, the probability that an input x satisfies one of the removed clauses is at most $\frac{S}{2^{O(\log S)}} = \frac{1}{\text{poly}(S)} = o(1)$.

Overall we have constructed an HIP that can verify whether an inner product of two strings is 1 on most inputs, with $O(\log S)$ proof length, and $O(\log S)$ query complexity. As noted before, our next step is to convert this protocol – which works in the average case – into a protocol that can verify *any* inner product claim.

Self-correction of the inner product function. As an initial observation, we observe that the self-correction property of linear functions can be extended also to the inner product function (this can also be viewed as a special case of locally decoding the Reed-Muller code over $\mathbb{GF}(2)$, see [GKZ08]). For any input strings x, y, and vectors u, v' which are taken at random, it holds that

$$\langle x, y\rangle = \langle x \oplus u, y \oplus v\rangle \oplus \langle x \oplus u, v\rangle \oplus \langle u, y \oplus v\rangle \oplus \langle u, v\rangle, \tag{1}$$

where $\langle a, b\rangle$ denotes the inner product of strings $a, b \in \{0,1\}^n$. Note that the terms on the right-hand side of Eq. (1), are inner product over different (correlated) random inputs. Also, recall that the "simple" protocol that was described previously, can verify inner product claims about random inputs with high probability over the inputs. Thus, at first glance it may seem sufficient to use *Eq.* (1), and verify the random claims using our average-case protocol. Unfortunately, by moving to claims over inner products of random inputs we will also need the ability to verify whether an inner product is 0, while so far we only have an HIP for "1-claims". Therefore, instead of using Eq. (1) directly, we present a generic compiler that extends the self-correction property of Eq. (1) also to the case where there is a protocol that can only verify *most* of the 1-claims (a similar idea was used also in the works of Shaltiel and Umans [SU05, SU06]). In this compiler, we rely on the fact that in our HIP the prover can't convince the verifier to accept a false 1-claim (with high probability over the inputs). For simplicity, we outline this compiler with respect to protocols for the inner product function but in the technical sections, we extend this argument to any *homogeneous multilinear mapping* (see Lemma 3.5).

Step 2: Self-correction with one-sided errors. In order to use Eq. (1), we need to verify also the 0-claims on the right-hand side of Eq. (1). Observe, that since Eq. (1) gives inner product claims of (individually) random inputs, then, in expectation, about half will be 0's and half will be 1's.

Thus, since (with high probability) a cheating prover cannot lie on false 1-claim, it will likely have to generate false 0-claims and therefore skew the distribution of 0 vs. 1 claims. In order to detect this, we simply repeat the experiment sufficiently many times (using independent coin tosses) and checking the empirical average value of the prover's claims. To sum up, given a ground protocol that works only on most 1-claims, the compiler produces the following protocol: the verifier uses Eq. (1) several times, each time with fresh random strings. At each iteration, the prover sends the values of the random inner products on the right-hand side of Eq. (1), while having the verifier check only the 1-claims, by using the ground protocol, and blindly accepting the 0-claims. At the end of the interaction, the verifier checks whether the average value of all the prover's claims is close enough to the expectation of the inner product function. If the average is close enough, the verifier infers that the prover is honest. Otherwise, the verifier infers that the prover lies, and thus it rejects.

Lastly, in order to reduce the randomness complexity to $O(\log n)$ randomness complexity, we use a standard technique, due to Newman [New91], for reducing the randomness complexity (using non-uniformity). We show that this technique works also in the context of AM-HIPs.

An alternate approach. We find it also instructive to describe another approach for dealing with the 0-claims in Eq. (1), and explain the reason we decided not to use it. The idea here is to show a random self-reduction from a 0-claim to a 1-claim. This can be done by embedding the input strings x and y into longer random string strings x' and y', while ensuring that the $\langle x', y' \rangle = 1 \oplus \langle x, y \rangle$.

The reason we decided not to follows this approach is that it changes the input size. In particular, it would mean that the verifier in the HIP would need to access a different linear transformation then that in the bottom layer of the DNF.

1.2.2 From HIP to Communication Complexity (Proving Theorem 3)

Our key idea in proving to construct an AM Communication Complexity protocol for sparse polynomials is the observation that a polynomial can be viewed as a linear combination of its monomials. In the communication complexity setting, each monomial is a product between a subset of Alice's input bits, and a subset of Bob's input bits. Thus, we can view the evaluation of the polynomial $f(x,y) = \sum_{\text{monomial } (\alpha,\beta)} x_\alpha \cdot y_\beta$, where $x_\alpha = \prod_{i \in \alpha} x_i$ and $y_\beta = \prod_{j \in \beta} y_j$, as an inner product between the strings (x_α) and (y_β).

Thus, in order to solve the problem, it suffices to construct an AM Communication Complexity protocol for computing the inner product function. Such a protocol follows easily from our HIP for inner products - since each query that the HIP verifier makes, is a *linear* query to the joint input (x,y), it can be emulated by having Alice and Bob compute and share their individual contributions.

The second part of Theorem 3 now follows easily by observing that every degree d polynomial over $\mathbb{GF}(2)$ can have at most n^d monomials, and by applying the polynomial approximation method of Razborov and Smolensky [Raz87, Smo87] (where the choice of the random polynomial can be made by the verifier as part of its first step in the protocol).

1.2.3 From HIP to Data Streaming (Proving Theorem 1)

Unfortunately, our approach for computing sparse polynomials that worked in the communication complexity setting, fails in the *streaming* setting. The issue is that each monomial consists of a product of *multiple* input bits. Therefore, the polynomial's monomials induce an inner product between a coefficient vector, and a *tensor* of the input, rather than the input in its basic form. While it is relatively easy to make queries to an encoding of the input by a streaming verifier, it is not clear at all how to make queries to an encoding of a *tensor* of the input.

Nevertheless, our starting point is the above observation that a polynomial can be expressed as a certain inner product. In more details, a degree d polynomial $\mathcal{P} : \{0,1\}^n \to \{0,1\}$ (over the field $\mathbb{GF}(2)$) can be viewed as a linear combination of its monomials, each of which is a product between a coefficient and a product of d input bits. Therefore, there exists a function $\mathsf{Coef}_{\mathcal{P}} : [n]^d \to \{0,1\}$ that depends only on $\mathcal{P}$, such that:

$$\mathcal{P}(x) = \bigoplus_{j_1,\ldots,j_d \in [n]} x_{j_1} \cdot x_{j_2} \cdots x_{j_d} \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d). \tag{2}$$

The basic idea of the protocol is to iteratively use the HIP protocol for inner product claims (henceforth, the ground HIP protocol), to gradually reduce a claim about the right-hand side of Eq. (2), to claims that don't depend on the inputs - that is, claims that depend only the structure of the specific code that the HIP uses, and the structure of the polynomial $\mathcal{P}$. Since the resulting claims do not depend on the input, the verifier will be able to check them without additional communication or queries.

Inspired by the celebrated sumcheck protocol of Lund *et al.* [LFKN92], we construct a d-round AM-HIP protocol, so that in the i-th round we reduce a set of claims over $d-(i-1)$ input variables, to a set of claims that depend on only $d-i$ input variables. The i-th round starts with claims of the form:

$$\bigoplus_{j_1,\ldots,j_d\in[n]} \left(\hat{\beta}^{(i-1)}\left(j_1,\ldots,j_{i-1}\right)\cdot x_{j_i}\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right) = b^{(i-1)}, \tag{3}$$

where $\hat{\beta}^{(i-1)}$ is a function that depends only on the structure of the linear code that the base HIP protocol uses. Our goal is to end the round with multiple claims of the form:

$$\bigoplus_{j_1,\ldots,j_d\in[n]} \hat{\beta}^{(i)}(j_1,\ldots,j_i)\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d) = b^{(i)}.$$

Observe that by changing the order of summation in Eq. (3), we can rewrite each claim as:

$$\bigoplus_{j_i\in[n]} x_{j_i}\cdot\left(\bigoplus_{\substack{j_1,\ldots,j_{i-1}\in[n],\\ j_{i+1},\ldots,j_d\in[n]}} \left(\hat{\beta}^{(i-1)}\left(j_1,\ldots,j_{i-1}\right)\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right)\right) = b^{(i-1)}. \tag{4}$$

The claims in Eq. (4) are an inner product between x and the truth table of a function that depends on only $d-i$ inputs variables. Thus, by applying the ground HIP protocol, we get multiple claims on the encoding of x, and multiple claims on the encoding of the truth table of a function that depends on $d-i$ variables. The claims on the encoding of x can be verified using the HIP verifier's oracle queries. Regarding the second class of queries, since the code is linear, the t-th claim is of the form of

$$\bigoplus_{j_i\in[n]} \gamma_{t,z}(j_i)\cdot \bigoplus_{\substack{j_1,\ldots,j_{i-1}\in[n],\\ j_{i+1},\ldots,j_d\in[n]}} \left(\hat{\beta}^{(i-1)}\left(j_1,\ldots,j_{i-1}\right)\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right) = b^{(i)}_{t,z}.$$

where the $(\gamma_{t,z})$'s correspond to the coefficients of the base code T. By changing the order of summation again, and defining $\hat{\beta}^{(i)}_{z,t}\left(j_1,\ldots,j_i\right) = \gamma_{t,z}(j_i)\cdot\hat{\beta}^{(i-1)}_{t}\left(j_1,\ldots,j_{i-1}\right)$ we get claims of the form:

$$\bigoplus_{j_1,\ldots j_d\in[n]} \left(\hat{\beta}^{(i)}_{t,z}\left(j_1,\ldots,j_i\right)\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right) = b^{(i)}_{t,z},$$

where the $\hat{\beta}^{(i)}_{t,z}\left(j_1,\ldots,j_i\right)$ don't depend on the input variables. Note that we got new claims that depend on $d-i$ input variables, as required.

Avoiding a complexity blowup. Although the above idea seems promising, we have one additional issue to deal with. In contrast to the traditional sumcheck protocol which generates a single claim in the end of each round, our ground HIP protocol produces multiple claims.[6] As a result, the number of times we need to use the ground HIP protocol grows by a at least a constant factor in each round, and overall becomes (at least) exponential in d. In order to reduce the dependence on d to *linear*, at the beginning of each round we combine claims together by

[6]Recall that the query complexity is roughly logarithmic in the size of our DNF of parities.

taking random linear combinations. This method lets us preserve the number of queries after each round, and thus achieve a linear dependence on d.

On the approximation factor. The (roughly) $\frac{5}{6}$ approximation factor required in all of our results, stems from the use of Eq. (1). Recall that the verifier needs to check all the 1-claims on the right-hand side of Eq. (1) with a success probability greater than $\frac{1}{2}$. Leveraging the fact that one of the terms on the right-hand side of Eq. (1) is independent of the input strings, we only have three terms to check. As a result, we must have a circuit that computes all the three terms correctly with probability greater than $\frac{1}{2}$. By union bounding over these three terms, we get that the circuit must compute a random input incorrectly with probability at most $\frac{1}{6}$, which sets the approximation limitation to $\frac{5}{6}$. A potential approach for improving the approximation factor is to rely on locally list, and we leave this possibility to future work.

1.2.4 Counting Triangles Mod 2 (Theorem 2)

Lastly, we give the outline of the streaming protocol for the $\bigoplus$Triangle problem. To do so we leverage the fact that $\bigoplus$Triangle can be expressed as a degree 3 polynomial over $\mathbb{GF}(2)$ and apply the streaming protocol of Theorem 1.

In more detail, let $\{I_{(u,v)}\}_{u,v}$ be a set of indicator variables where $I_{(u,v)}$ is 1 if and only if the edge (u,v) appears in the graph. We can express the parity of the number of triangles in the graph, by evaluating the following degree 3 polynomial:

$$\mathcal{P}_{\oplus\textsc{Tri}}(I_{e_1},\ldots,I_{e_{n^2}}) = \bigoplus_{v<u<w\in[n]} I_{(v,u)} \cdot I_{(u,w)} \cdot I_{(w,v)}.$$

Thus, by applying our streaming protocol for low degree polynomials from Theorem 1, we derive an AM[6] Data streaming protocol for $\bigoplus$Triangle. Lastly, in order to derive a one-round protocol, we use the round collapsing technique of Babai and Moran [BM88] for reducing the number of rounds in public coin interactions.

1.3 Organization

In Section 2 we provide definitions and notations that are used throughout our work. In Section 3 we construct the HIP for inner product claims, which serves as a basis for all our other results. Next, in Section 4, we present the Communication Complexity and the Streaming protocols for any problem that is decidable by a low degree polynomial. In the end of Section 4, we also extend the Communication Complexity result for any polynomial that is decidable by a constant depth circuit. Lastly, in Section 5, we study the $\bigoplus$Triangle problem as a concrete application of our streaming protocol. Appendices A and B contain some standard proofs that were deferred from the main sections.

Acknowledgments

We thank Yuval Ishai, Eyal Kushilevitz, Or Meir and the anonymous reviewers for useful comments.

2 Preliminaries

All the logarithms in this work are in base 2, and by default operations are over $\mathbb{GF}(2)$, unless stated otherwise. For $k \in \mathbb{N}$ we use $[k]$ to denote the set $\{1,2,\ldots,k\}$, and $\mathsf{IP}_2 : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ to denote the inner product function (over the field $\mathbb{GF}(2)$), that is, $\mathsf{IP}_2(x,y) = \oplus_{i\in[n]} x_i \cdot y_i$.

The relative Hamming distance of two Boolean strings $x, y \in \{0,1\}^n$ is defined as $\frac{|\{i : x_i \neq y_i\}|}{n}$. The relative distance of two Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ is defined as the relative distance of their truth tables, namely $\Pr_x[f(x) \neq g(x)]$. We say that f is δ-close to g if their relative distance is at most δ. Otherwise, we say that g and f are δ-far.

Error Correcting Codes. An error-correcting code over a field $\mathbb{F}$ is an injective function $C : \mathbb{F}^k \to \mathbb{F}^n$, where $n \geq k$. Strings in the domain are called *messages* and string in the image are called *codewords*. We say that C is linear if the function is a linear map (over $\mathbb{F}$). Two important parameters of an error-correcting code are the *rate* of the code, which is defined as $\frac{k}{n}$, and the minimal distance, which is defined as the minimal relative Hamming distance of any two distinct codewords.

2.1 Holographic Interactive Proofs

We first recall the definition of an interactive proof. Then, we define *holographic* interactive proofs (HIP), which are similar to interactive proofs, except that the verifier has oracle access to an *encoding* of the input via some error-correcting code, rather than direct access as in classical interactive proofs.

Definition 2.1 (Interactive Proof). *An* interactive proof *for a language $\mathcal{L}$ is a pair of two algorithms: a computationally unbounded prover P and a probabilistic polynomial-time verifier V. Given common input x, V and P exchange messages, and at the of the interaction the verifier decides whether to accept or reject. The interaction must satisfy the following requirements:*

- ***Completeness:*** *if $x \in \mathcal{L}$, then, when V interacts with P, with probability of at least $2/3$ it accepts.*
- ***Soundness:*** *if $x \notin \mathcal{L}$, then for every prover strategy P^*, when V interacts with P^*, it accepts with probability of at most $1/3$.*

Complexity. An interactive proof has several complexity parameters that we care about: the number of rounds in the interaction (round complexity); the number of bits transferred between the prover and the verifier (communication complexity); the number of random coins the verifier tosses (randomness complexity); and the runtime of the verifier (time complexity) as well as the runtime of the honest prover.

In this work, we focus on a variant of interactive proofs, where the verifier can access the input only through oracle access to an *encoding* of the input via some error-correcting code. This variant of interactive proof introduced in [GR17] (following [BFLS91]), is denoted as a Holographic Interactive Proof (HIP) and it is defined as follows:

Definition 2.2. (Holographic Interactive Proof [GR17]). *A* Holographic Interactive Proof (HIP) *for a language $\mathcal{L}$ with respect to an error-correcting code C, is defined similarly to an interactive proof except that the verifier, rather than having direct access to x, has oracle access to $C(x)$.*

An HIP has an additional complexity parameter that we shall care about, which is the total number of queries that the verifier makes to the encoded input during the interaction. In addition, we say that an HIP verifier is *non-adaptive* if each query does not depend on answers to past queries.

Arthur-Merlin (Public Coins) Games. Public-coin interactive proofs (also known as Arthur-Merlin proofs) are interactive proofs in which the verifier sends all its random coins to the prover immediately as they are tossed. At each round, the verifier sends a single message that consists of only fresh random coins, and in particular, doesn't depend on the input, prover's messages, or on the previous coins. At the end of the interaction, the verifier, which is not allowed to toss any other random coins, decides whether to accept or reject by running a deterministic computation that depends on the tossed random coins, the input and the prover's messages. These protocols are denoted $\mathsf{AM}[k]$, where k indicates the maximal number of messages exchanged between the verifier and the prover.

Similarly to $\mathsf{AM}[k]$, we define the Holographic Interactive Proof variant, denoted $\mathsf{AM}[k]$-HIP, where the verifier, rather than having direct access to the input, has oracle access to an encoding of the input via some error-correcting code. In our discussion about $\mathsf{AM}[k]$-HIP, we separate the communication complexity into randomness complexity and proof length. The randomness complexity is the overall number of bits the verifier sends, and the proof length is the overall number of bits the prover sends.

2.2 AM Data Streaming and Communication Complexity

2.2.1 AM Communication Complexity

Communication Complexity, introduced by Yao [Yao79], studies the amount of communication required for a number of (computationally unbounded) parties to compute a certain function f on an input which is distributed among them. In the standard setting, there are two parties, Alice, who is given as input a string x, and Bob, who is given a string y. The goal is for Alice and Bob to compute $f(x, y)$ with the least amount of communication between them.

Babai et al. [BFS86] considered the analogs of complexity classes, such as P, NP and Arthur-Merlin games, in the field of communication complexity. Loosely speaking, in an AM (Arthur-Merlin) communication complexity variant, in addition to Alice and Bob, there is also a third party (called Merlin), which gets both of their inputs. Alice and Bob first interact with Merlin via multiple rounds of *public coin* interaction, such that at each round Alice and Bob send a message that consists of only fresh random coins, and Merlin responds with a proof message. After the interaction is over, each of the parties gets the transcript of the interaction (all Merlin's messages and all the tossed coins), and deterministically decides whether to accept or reject the transcript. We say that the protocol accepts if and only if *both* Alice and Bob decide to accept. We remark that the parties communicate using a broadcast channel. Namely, each of the parties is exposed to all the messages sent by Merlin, and all the random coins. In addition, in contrast to the traditional communication complexity model, Alice and Bob are not allowed to interact with each other. Yet, a similar model in which Alice and Bob are allowed to interact is expressively equal since Merlin can always send also the messages that Alice and Bob would have exchanged.

Definition 2.3. $\mathsf{AM}[k]$-Communication Complexity). *Let $f : X \times Y \to \{0, 1\}$ be a function for some (finite) sets X and Y. An $\mathsf{AM}[k]$-Communication Complexity ($\mathsf{AM}[k]$-CC) protocol for f is a protocol between three deterministic algorithms: A (Alice) which gets an input $x \in X$, B (Bob) which gets an input $y \in Y$, and a prover P (Merlin) which gets both x and y. The protocol proceeds as follows: first, a* public-coin *interactive protocol with k exchanged messages is conducted, where Merlin gets fresh random coins as messages and responds with proof messages. Then, Alice and Bob get all the tossed coins and Merlin's messages, and each of them deterministically decides whether to accept or reject. The protocol must satisfy the following requirements:*

- ***Completeness:*** *if $f(x, y) = 1$, then, when Alice and Bob interact with P, with probability of at least $2/3$ they* both *accept.*

- ***Soundness:*** *if $f(x, y) = 0$, then for every prover strategy P^*, when Alice and Bob interact with P^*, the probability they* both *accept is at most $1/3$.*

Complexity. The communication complexity of the protocol is defined to be the maximal number of bits that are exchanged between the *three* parties, over all inputs $x \in X, y \in Y$. That is, the maximum, over $x \in X$ and $y \in Y$, of the sum of the number of random coins and the overall length of Merlin's messages.

2.2.2 AM Data Streaming

In our work, we consider another model, called the Data Streaming Model. In this model, there is a limited space algorithm that gets the input as a sequence of bits (stream). The access to the input is read-once in the sense that after seeing a bit in the sequence, the algorithm no longer has access to the bits that preceded it (unless these were stored in its memory). The main complexity measure is the amount of space used by the algorithm.

Previous works already studied the advantage of adding an unbounded (but untrusted) prover with which the limited-space algorithm (henceforth, the verifier) can interact. Prior works focused on the following models:

1. Online Schemes [CCGT14, CCMT14, Tha16, CGT20] where the verifier and a (randomized) prover *simultaneously* read each bit of the stream. The prover is allowed to send a proof message after each bit that is read.

2. Prescient Schemes [GR13, CCGT14, CCMT14, Tha16] which are similar to Online Schemes except that the prover knows all the bits in the stream in advance.

3. AMA Schemes [GR13, CCGT14] where there is a shared public source of randomness to which both the verifier and the prover have access. In addition, the verifier has its own private coins which are hidden from the prover. The prover, which knows the input stream in advance, is allowed to send a single short proof message to the verifier.

4. SIP - Streaming Interactive Proof [CTY11, CCM$^+$13, CCM$^+$15] where the parties are allowed to make multiple rounds of interaction. However, they both must first get and process the entire input stream and only then conduct the interaction.

In all these models the verifier also has private coins that allow it to keep some random secret hidden from the prover. We suggest a fully *public coin* model, which we refer to as the AM[k]-Data Streaming Model (AM[k]-DS), where all the random coins are known to the prover. In more details, the parties first exchange k messages via a public coin interaction. Then, given the interaction's transcript, the verifier deterministically[7] processes the input stream and decides whether to accept or reject. In this work, we focus on a non-uniform variant of the AM[k]-Data Streaming model, that we define using a collection of *Ordered Binary Decision Diagrams* (OBDD). Loosely speaking, fixing an input length n, each OBDD in the collection corresponds to a fixed transcript of the interaction and, given the input stream as an input, determines the output of the protocol. Before giving the formal definition, let us recall the definition of an OBDD.

Ordered Binary Decision Diagrams. An Ordered Binary Decision Diagram (OBDD) is similar to a Binary Decision tree with two modifications: redundant nodes are omitted, and different nodes are allowed to share the same subtree as long as at any route, the order of the input variables is the same. Also, any input variable appears at each route at most once. See Fig. 1 for an example.

[7]We note that the verifier can specify random coins by adding an additional dummy round at the end of the interaction. However, this increases the number of rounds of interaction by one.

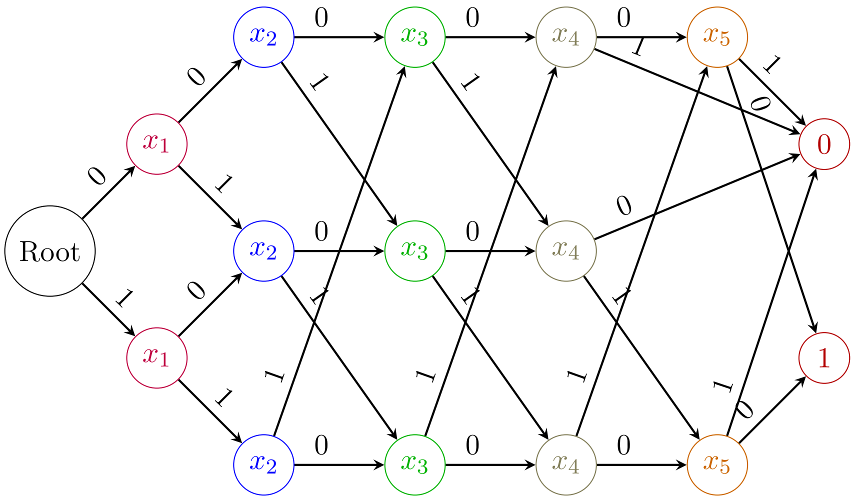

Figure 1: An OBDD for the claim $\sum_{i=1}^{5} x_i \bmod 3 = 0$, with the input-order $(x_1, x_2, x_3, x_4, x_5)$.

Definition 2.4 (Ordered Binary Decision Diagram)**.** *An* Ordered Binary Decision Diagram (OBDD) *is a directed acyclic graph* (DAG) *with the following properties:*

1. ***Structure:*** *There are two types of nodes: leaves and testing nodes. Each leaf is associated with the label* 0 *or the label* 1*, and doesn't have any outgoing edges. Each testing node is associated with an input variable* x_i *and has two outgoing edges that are labeled with the constants* 0 *and* 1*. In addition, we also have a single testing node that doesn't have incoming edges which we refer to as the root.*

2. ***Computation:*** *Given the input* $x = x_1, \ldots, x_n \in \{0,1\}^n$*, the output of the OBDD is defined to be the label of the leaf that is reached by a route that starts from the root, and at each node labeled with an input variable* x_i*, continues in the direction of the edge that is labeled with the value of* x_i*.*

3. ***Order:*** *There exists an ordering* π *of the input variables* $x_1, \ldots, x_n$*, such that each variable appears at most once in the order. At any route from the root to one of the leaves, the order of the input variables must be equal to* π*.*

Width, layers and space complexity. We say that two nodes are in the same layer if they test the same input variable. We define the width of a layer to be the number of nodes in the layer, and the width of an OBDD to be the width of the widest layer. Note that OBDDs capture read-once bounded space algorithms (i.e. streaming algorithms), where the space complexity is logarithmic in the width of the corresponding OBDD.

Arthur-Merlin Data Streaming. An *Arthur-Merlin* Data Streaming protocol consists of an interactive public-coin protocol, and a collection of OBDDs. Each OBDD is defined with respect to a fixed transcript of the public-coin interaction (i.e. randomness and prover's messages). The protocol starts with multiple rounds of public-coin interaction that generate a fixed transcript z. Then, the OBDD corresponds to z is executed. The chosen OBDD gets the input stream as an input and determines the output of the protocol.

Definition 2.5 (AM$[k]$-Data-Streaming)**.** *Fixing an input length* n*, an* AM$[2k]$-Data Streaming

(AM[$2k$]-DS) *protocol for a language*[8] $\mathcal{L} \subseteq \{0,1\}^n$, *with* $\rho(n)$ *randomness and* $\ell(n)$ *proof length, consists of a collection of OBDDs* $\left\{ f_{r,\pi} : \{0,1\}^n \to \{0,1\} \;\middle|\; r \in \{0,1\}^{\rho(n)}, \pi \in \{0,1\}^{\ell(n)} \right\}$ *such that first a* public-coin *interactive proof with* $2k$ *exchanged messages is conducted, where the verifier sends fresh random coins and the (computationally unbounded) prover, which gets the input* x *in advance, responds with proof messages. Then, denoting the prover's messages by* $\pi = (\pi_1, \ldots, \pi_k)$ *and the verifier's random messages by* $r = (r_1, \ldots r_k)$, *the output of the protocol is defined to be* $f_{r,\pi}(x)$. *The protocol must satisfy the following requirements:*

- ***Completeness:*** *if* $x \in \mathcal{L}$, *then, with probability of at least* $2/3$, *the transcript* (r, π) *generated in the interaction with the prover satisfies* $f_{r,\pi}(x) = 1$.
- ***Soundness:*** *if* $x \notin \mathcal{L}$, *then for every prover strategy* P^*, *with probability of at most* $1/3$, *the transcript* (r, π) *generated in the interaction with* P^* *satisfies* $f_{r,\pi}(x) = 1$.

Complexity. We focus on three parameters of interest: $\ell(n)$, which is the maximal total number of bits the prover sends during the interaction (proof length); $\rho(n)$, which is the maximal number of random coins that are used in the protocol (randomness complexity); and the verifier space complexity which is defined as $\lceil \log(\mathrm{W}) \rceil$, where W is the width of the widest OBDD in the collection.

A note about the input-order. We emphasize the importance of the precise order of the inputs for the OBDDs. For example, it is easy to compute the inner product of Boolean strings $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n)$ by an OBDD that first gets x_1, y_1, then x_2, y_2, and so on. But, using simple counting arguments, it can be shown that an OBDD that first gets all the $x's$ and then all the y's must have an exponential width for computing the inner product function (given that there wasn't a prior interaction with Merlin). Furthermore, the order between the prover's messages and the input stream is also important. Note that in the SIP model, which is the only other streaming scheme that allows multiple rounds of interaction, the interaction is conducted *after* the verifier processes the input stream, while we define the opposite scenario where first the transcript of the interaction is received and only then the verifier gets the input stream. The reason for that is that in the SIP scheme, the verifier also has private coins that allow it to keep some random secret hidden from the prover, which cannot be done in our model where all the random coins are public.

2.2.3 Relations Between HIP, Streaming and Communication Complexity

The Data Streaming, Communication Complexity and Holographic Interactive Proof models are all related in the sense that there are transformations from AM[k]-Holographic Interactive Proofs to AM[k]-Data Streaming, and from AM[k]-Data Streaming to AM[k]-Communication Complexity.

Fact 4. (From AM[k]-HIP to AM[k]-DS) *Fix an input length* n. *Let* $\mathbb{F}$ *be some finite field, and let* $C : \mathbb{F}^n \to \mathbb{F}^{n'}$ *be a linear code over* $\mathbb{F}$. *Suppose there exists an* AM[k]-HIP *protocol with respect to the code* C *for a language* $\mathcal{L} \subseteq \mathbb{F}^n$, *using* q *queries,* ρ *random bits and proof length* ℓ. *Then, there exists an* AM[k]-DS *protocol for* $\mathcal{L}$ *with* $\ell + O(q \cdot \log n)$ *proof length,* ρ *randomness complexity, and* $q \cdot \lceil \log |\mathbb{F}| \rceil$ *verifier space complexity, for any order of the input in the stream. Moreover, if the verifier is non-adaptive then the proof length is reduced to* ℓ.

Loosely speaking, in order to convert an AM[k]-HIP protocol to an AM[k]-DS protocol, we only need to compute the queries of the AM[k]-HIP in small space, where the input is read as

[8]We define the model for languages over the $\{0,1\}$ alphabet. However, it can naturally be defined over any alphabet.

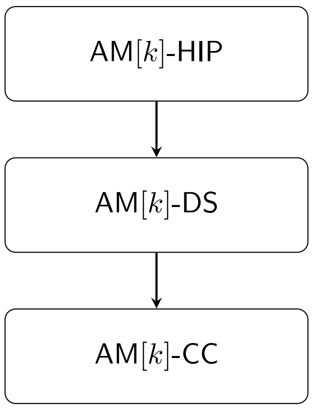

Figure 2: Relationship between the different models.

a stream. That is done by observing that a query to a linear code is a linear combination of the bits in the stream. Thus, if the verifier knows which queries it would make, it can compute each query in advance, by summing the bits in the linear combination that form the query using only $\lceil \log |\mathbb{F}| \rceil$ space for each query. Therefore, the AM[k]-DS protocol will be the same as the AM[k]-HIP protocol except that at its final message, the prover sends also the locations of the queries that the verifier is going to make in its verification after receiving the last message from the prover. Then, the OBDD that corresponds to the transcript of the interaction, computes the values of these queries by reading the input stream, and then output the result according to the output of the AM[k]-HIP verifier. Note that the output of the AM[k]-HIP protocol depends only on the queries' values (that have already been computed) and the interaction's transcript, and thus the OBDD, after computing the queries, can output the result without additional computation. For the formal details, and proof of Fact 4, see Appendix A.

Fact 5. (From AM[k]-DS to AM[k]-CC). *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, and let $\mathcal{L}_f = \{(x,y) \mid f(x,y) = 1\}$. Suppose there exists an* AM[k]*-Data Streaming protocol for $\mathcal{L}_f$, where x precedes y in the input stream, with proof length ℓ, randomness complexity ρ and verifier space complexity s. Then, there exists an* AM[k]-CC *protocol for f with $(\ell + s + \rho)$ communication complexity, where Alice gets x and Bob gets y.*

Loosely speaking, the transformation from AM[k]-DS to AM[k]-CC is done by splitting the input stream into two halves, the first (x) is given to Alice and the second (y) is given to Bob. The parties simulate the streaming protocol as follows: Alice, which gets the first half of the stream, simulates the streaming protocol from the beginning and up to the moment half of the stream is read. Then Bob, which gets the second half of the stream, continues the simulation by asking the prover to send the state at which Alice stopped. Alice's goal is to verify that she indeed stops at the state Merlin claimed, and Bob's goal is to use that state in order to check the output of the OBDD. The key observation is that since the streaming protocol uses small space, the additional message Merlin sends to tell in which state Alice will stop, is also small. The formal details of the transformation are given in Appendix A.

2.3 Homogeneous Multilinear Mapping

Recall that a linear mapping f is a mapping between two vector spaces with the linear property that for any vectors x, y and scalar α:

$$f(x + \alpha \cdot y) = f(x) + \alpha \cdot f(y)$$

We extend this definition to multivariable functions as follows:

Definition 2.6. (See also [Lan93, Page 511]). *Let $V_1, \dots, V_k$, W be vector spaces over some field $\mathbb{F}$. A map*

$$f : V_1 \times \cdots \times V_k \to W$$

is said to be a homogeneous k-linear mapping *(or homogeneous multilinear) if it is linear in each variable, i.e., if for every index $i \in [k]$ and for every fixed $\{x_t\}_{t \neq i}$ elements, the mapping $g(x) \stackrel{\text{def}}{=} f(x_1, \dots, x_{i-1}, x, x_{i+1}, \dots x_k)$ is linear.*

In particular, when $k = 1$ we get the standard notation of a linear mapping, and when $k = 2$ we get a bilinear mapping (e.g. the inner product function). We emphasize that the definition of multilinearity requires the restriction to every axis to be *linear* rather than just *affine* (which is why we refer to it as a *homogeneous* multilinear map). For example, the Boolean function $f(x_1, x_2, x_3) = x_1 \cdot x_2 + x_2 \cdot x_3 + x_1 \cdot x_3$ is an affine function in each variable, but it is *not* a homogeneous 3-linear mapping since $f(1, 1, 0) + f(1, 1, 1) = 0$, while $f(1, 1, 0 + 1) = 1$.

In our work, we focus on the inner product function (over the field $\mathbb{GF}(2)$), which is a homogeneous 2-linear mapping.

Fact 6. *Tthe inner product function $\mathsf{IP}_2(x, y) \stackrel{\text{def}}{=} \langle x, y \rangle = \bigoplus_i (x_i \cdot y_i)$ is a homogeneous 2-linear mapping.*

We next show that the self-correction property of *linear* mappings extends to homogeneous multilinear mapping.

Claim 2.7 (Self correction of homogeneous multilinear mappings)**.** *Let $f : V_1 \times \cdots \times V_k \to W$ be a homogeneous k-linear mapping. For any $(x^{(1)}, \dots, x^{(k)}) \in V_1 \times \cdots \times V_k$ and for any $(r^{(1)}, \dots, r^{(k)}) \in V_1 \times \cdots \times V_k$:*

$$f\left(x^{(1)}, \dots, x^{(k)}\right) = \sum_{s_1, \dots, s_k \in \{0,1\}} f\Big((-1)^{1-s_1} r^{(1)} + x^{(1)} s_1, \dots, (-1)^{1-s_k} r^{(k)} + x^{(k)} s_k\Big).$$

Thus, the value of $f\left(x^{(1)}, \dots, x^{(k)}\right)$ can be computed by choosing $r^{(1)}, \dots, r^{(k)}$ at random and taking the sum of the output of f on 2^k different (correlated) random inputs.

Remark 2.8 (Self-correction of polynomials)**.** *We remark that in contrast to the self-correction of polynomials which requires the polynomials to have degree lower than the field size, Claim 2.7 doesn't have requirements on the degree or on the field size. For example, in the inner product function over $\mathbb{GF}(2)$, the degree is the same as the field size. Yet, we can use Claim 2.7 by evaluating the function on only 4 random inputs, since its a homogeneous 2-linear mapping.*

In particular, in this work we focus on the case the mapping is over $\mathbb{GF}(2)$, and thus Claim 2.7 reduces to:

$$f(x^{(1)}, \dots, x^{(k)}) = \bigoplus_{s_1, \dots, s_k \in \{0,1\}} f\Big(r^{(1)} \oplus (x^{(1)} \cdot s_1), \dots, r^{(k)} \oplus (x^{(k)} \cdot s_k)\Big). \qquad (5)$$

The proof of Claim 2.7 is deferred to Appendix A.1.

2.4 Boolean Circuits

Another complexity model we will consider in this work is that of Boolean Circuits. We give here the basic notations and definitions.

Definition 2.9. (Boolean Circuit). *A* Boolean circuit *with n inputs and m outputs, is a directed acyclic graph* (DAG) *where there are n nodes without incoming edges called "input nodes", and m nodes without outgoing edges called "output nodes". The other nodes are called "gates". Each gate v has $i \geq 1$ incoming edges and is associated with a Boolean function $g_v : \{0,1\}^i \to \{0,1\}$. The computation of the circuit on a given input $x \in \{0,1\}^n$ is defined recursively:*

- ***Input:*** *The i-th input node is labeled with $x_i \in \{0,1\}$.*
- ***Gates:*** *Any node u with i incoming edges from nodes with labels $l_1, ..., l_i$, is labeled with $g_u(l_1, ..., l_i)$.*
- ***Output:*** *The output of the computation is defined as the labels of the output nodes.*

Three standard gates are the NOT, AND and OR gates (De-Morgan gates). Another important gate is the parity (XOR) gate.

Circuit size and depth. Two important parameters of a Boolean circuit are its size and its depth. Intuitively, the first quantifies how much time a single computation takes, while the second quantifies the time of parallel computation. Formally, the size of a circuit is defined as the number of nodes[9] (inputs, gates and outputs) in the circuit, and the depth of a circuit is defined as the length of the longest directed path from an input node to an output node.

Circuit composition. The composition of circuit C_1 (having n inputs and k outputs) with circuit C_2 (having k inputs and m outputs), is the circuit $C = C_2 \circ C_1$ consisting of all the nodes of C_1 and C_2 as well as all their edges, combined with additional edges that connect the output nodes of C_1 with the input nodes of C_2. The input nodes of C are defined to be the n input nodes of C_1. As a result, the output nodes of C are defined to be the m output nodes of C_2. Note that if C_1 computes a function f_1 and C_2 computes a function f_2, then $C_2 \circ C_1$ computes the composed function $f_2 \circ f_1$.

$\mathsf{DNF} \circ \mathsf{T}$ **Circuits.** For a linear transformation T, we denote by $\mathsf{DNF} \circ \mathsf{T}$ the circuit that is a composition of some circuit that computes the transformation T, with a DNF circuit (disjunction of clauses). We note that since T is linear, these circuits are a particular type[10] of a DNF of parties, where the parity gates are only allowed to compute the function T.

Definition 2.10 ($\mathsf{DNF} \circ \mathsf{T}$ circuit)**.** *Let T be a linear transformation $\mathsf{T} : \{0,1\}^n \to \{0,1\}^m$. A circuit C is called a* $\mathsf{DNF} \circ \mathsf{T}$ circuit *if there is a circuit $C_1 : \{0,1\}^n \to \{0,1\}^m$ that computes the transformation T, and another* DNF *circuit $C_2 : \{0,1\}^m \to \{0,1\}$ such that $C = C_2 \circ C_1$.*

We define the width of a clause (i.e., a conjunction of literals) as the number of literals in the clause, and the width of a DNF circuit, as well as the width of a $\mathsf{DNF} \circ \mathsf{T}$ circuit, as the width of the widest clause in the DNF.

$\mathsf{AC}^0(\oplus)$ **Circuits.** Lastly, we define the class of $\mathsf{AC}^0(\oplus)$ circuits as consisting of all constant depth circuits with the De-Morgan (AND, OR, NOT) gates, as well as parity gates which can

[9]Note that under this definition, the circuit size is at least n.

[10]Namely, DNF circuits with an additional layer of parity (XOR) gates which can be applied only directly on the input gates.

be used *anywhere* in the circuit. Note that this class is more general than DNF of parities or $\mathsf{DNF} \circ \mathsf{T}$ circuits

2.5 A Concentration Bound

Finally, we recall the (non-Boolean version of the) Chernoff Bound, which is used in our analysis in Section 3:

Lemma 2.11. (Chernoff Bound for non-Boolean variables. See, e.g., [Goe15, Theorem 4]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that $a \leq X_i \leq b$ for all i. Let $X = \sum_{i=1}^n X_i$ and set $\mu = \mathbf{E}[X]$. Then, for all $\delta > 0$:*

- Upper Tail: $\Pr\left[X \geq (1+\delta)\mu\right] \leq e^{-\frac{2\delta^2\mu^2}{n(b-a)^2}}$.
- Lower Tail: $\Pr\left[X \leq (1-\delta)\mu\right] \leq e^{-\frac{\delta^2\mu^2}{n(b-\alpha)^2}}$.

3 From Circuits to $\mathsf{AM}[2]$-HIP, $\mathsf{AM}[2]$-DS and $\mathsf{AM}[2]$-CC

In this section, we show that the existence of a sufficiently small $\mathsf{DNF} \circ \mathsf{T}$ circuit that approximates the inner product function, yields an efficient (non-uniform) $\mathsf{AM}[2]$-HIP protocol for checking inner product claims. Using Facts 4 and 5, we will later use this result to derive $\mathsf{AM}[2]$-DS protocols (for any order of the input bits in the stream) and an $\mathsf{AM}[2]$-CC protocols from the same assumption (see Sections 4 and 5).

Definition 3.1. (L_{IP} language). *We define the language L_{IP} as*

$$L_{\mathsf{IP}} \overset{def}{=} \Big\{(x,y,b) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\} \ : \ \mathsf{IP}_2(x,y) = b\Big\}.$$

The main result that we prove in this section is reducing the construction of an $\mathsf{AM}[2]$-HIP for L_{IP}, with respect to a linear code T, to constructing a $\mathsf{DNF} \circ \mathsf{T}$ circuit for approximating the inner product function.

Lemma 3.2. ($\mathsf{AM}[2]$-HIP for L_{IP}). *Fix an integer n, and a parameter $\epsilon \in (0, 1/6]$. Let $\mathsf{T} : \{0,1\}^{2n} \to \{0,1\}^{n'}$ be some linear code. Suppose there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size S that computes $\mathsf{IP}_2(x,y)$ on at least $\frac{5}{6} + \epsilon$ fraction of the inputs. Then, there exists an $\mathsf{AM}[2]$-HIP protocol for L_{IP}, with proof length $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, randomness complexity $O(\log n)$ and $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ queries to the bits of $\mathsf{T}(x,y)$.*

Using Facts 4 and 5, we derive similar results also in the $\mathsf{AM}[2]$-DS and $\mathsf{AM}[2]$-CC models:

Corollary 7. ($\mathsf{AM}[2]$-DS and $\mathsf{AM}[2]$-CC for L_{IP}). *Let $n, n' \in \mathbb{N}$, and $\epsilon \in (0, 1/6]$. Let $\mathsf{T} : \{0,1\}^{2n} \to \{0,1\}^{n'}$ be some linear code. Suppose there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size S that computes IP_2 on at least a $\frac{5}{6} + \epsilon$ fraction of the inputs. Then,*

1. *There exists an $\mathsf{AM}[2]$-DS protocol for L_{IP}, with proof length $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, randomness complexity $O(\log n)$ and verifier space complexity $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$.*
2. *There exists an $\mathsf{AM}[2]$-CC protocol for the function $\mathsf{IP}_2(x,y)$ with $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ communication complexity.*

Proof organization and outline (for a more detailed outline see Section 1.2). The proof of Lemma 3.2 goes through three main steps. In the first step, described in Section 3.1, we prune all the "wide" clauses in the $\mathsf{DNF} \circ \mathsf{T}$ circuit, and show that this gives a good approximation of the original circuit. In the second step, described in Section 3.2, we present an extremely simple holographic NP-proof for verifying whether an input x satisfies a $\mathsf{DNF} \circ \mathsf{T}$ circuit, where the length of the witness is logarithmic in the size of the circuit. The proof-system is holographic in the sense that the verifier only needs to look at bits in the encoding $\mathsf{T}(x)$, and in particular, the number of queries to the bits of $\mathsf{T}(x)$ is equal to the DNF's width.

Thus, we wish to apply the holographic NP-proof on the circuit of narrow wide from Step 1 to check inner product claims using a small number of queries and a short proof. However, the narrow circuit only computes the inner product correctly on *most* inputs. Recall that by Claim 2.7, which we referred to as the self correction property of homogeneous multilinear mappings, the inner product of two certain input strings can be expressed as a sum of random inner products. Hence, in order to construct a randomized protocol that works on all inputs, a natural idea would be to use Claim 2.7 in order to move from a certain inner product claim to multiple claims about inner products of random inputs.

Unfortunately, by moving to claims over inner products of random inputs, we may need the ability to verify also whether an inner product is 0, whereas the holographic NP-proof from our second step can only verify that an inner product is 1. Therefore, in our third step, described in Section 3.3, instead of using the straightforward self-correction property, we present a more general compiler (see Lemma 3.5) that extends the self-correction property also to the case where there is a protocol that can only verify *most* of the 1-claims and *none* of the 0-claims. That is, given a protocol that verifies whether an inner product is 1 on most inputs, the compiler gives a probabilistic protocol that can verify *any* inner product claim, on all inputs (w.h.p over random coins tosses). We remark that this result generalizes to any homogeneous multilinear mapping and the inner product result follows as a special case. Combining these results, we conclude with the proof of Lemma 3.2 in Section 3.4.

3.1 Eliminating Wide Clauses

Our first step is reducing the width of the given circuit (which approximates the inner product function). In more detail, given a $\mathsf{DNF} \circ \mathsf{T}$ circuit C with respect to a linear transformation T, we construct a new $\mathsf{DNF} \circ \mathsf{T}$ circuit that agrees with C on most inputs and consists of only clauses with a narrow width. The construction is straightforward: first get rid of all linear dependencies that are internal to a clause, and then remove all the remaining wide clauses.

Proposition 3.3. *Let T be a linear transformation. For any $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size S, and a parameter $\tau \in (0,1)$, there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C'_τ of size at most S, and clauses of width $w = \left\lceil \log\left(\frac{S}{\tau}\right) \right\rceil$, such that $\Pr_x\left[C'_\tau(x) = C(x)\right] \geq 1 - \tau$. Furthermore, for every x such that $C(x) = 0$ it holds that $C'_\tau(x) = 0$.*

Proof. We construct the new circuit C'_τ by making two modifications on the original circuit C:

1. First, we eliminate linear dependencies within each clause.
2. Then, we remove clauses with width greater than $w = \left\lceil \log\left(\frac{S}{\tau}\right) \right\rceil$.

For Step 1, observe that each DNF clause is a conjunction of the form $(\mathsf{T}_{i_1}(x) = b_1) \wedge (\mathsf{T}_{i_2}(x) = b_2) \wedge \cdots \wedge (\mathsf{T}_{i_k}(x) = b_k)$, where $\mathsf{T}_i(x)$ is the i-th bit of $\mathsf{T}(x)$. Since T is a linear transformation, each $\mathsf{T}_i(x)$ is a linear combination of the bits of x. Therefore, each DNF clause can be viewed as a system of linear equations. The clause is satisfied if and only if all the equations are satisfied. As noted above, our first modification is removing the dependencies between elements in each clause. Formally, for each clause, if there is a solution to the equations that form the clause, then we select a maximal subset of linearly independent equations for that clause and discard the rest. If the equation-system is unsatisfiable (and therefore the clause can never be satisfied), then we can remove the entire clause altogether. This procedure gives us a new $\mathsf{DNF} \circ \mathsf{T}$ circuit, where each clause consists of only satisfiable linearly independent equations. Note that this procedure does not change the input/output behavior of the circuit, as we removed only linearly dependent equations, or clauses that were already unsatisfiable.

As our second modification we remove all clauses whose width is greater than $w = \lceil \log(\frac{S}{\tau}) \rceil$. By applying these two modifications, we get a new circuit C'_τ of size of at most S, where the width of each clause is at most w.

Note that $C'_\tau(x) \neq C(x)$ only if we removed a clause that x satisfies. For each clause we removed, the probability (over a random input x) that the clause is satisfied, is equal to the probability that all the equations that form the clause are satisfied. By the first modification, all the equations are linearly independent. By the second modification, each clause we removed is a linear system with rank at least w. Thus, each such system is satisfied with probability at most 2^{-w}. Thus, by the union bound, the probability that there is a removed clause that is satisfied can be bounded as:

$$\Pr_x \left[C'_\tau(x) \neq C(x)\right] \leq \Pr_x \left[\exists \text{ removed clause that } x \text{ satisfies}\right] \leq S \cdot 2^{-w}.$$

The furthermore part of Proposition 3.3 follows from the fact that the first modification does not affect the circuit's output at all, and in the second modification we only *remove* clauses and therefore can only change the circuit's output from 1 to 0, but never from 0 to 1.

■

3.2 Holographic NP proof for $\mathsf{DNF} \circ \mathsf{T}$ Circuits

Our second step is constructing for any $\mathsf{DNF} \circ \mathsf{T}$ circuit, a holographic NP proof for one-sided claims, that is, for proving that the circuit outputs 1. The proof-system is holographic in the sense that the verification of the proof only requires to make queries to the bits of $\mathsf{T}(x)$, rather than having direct access to the input. In particular, we show a straightforward holographic NP proof with a $\lceil \log S \rceil$-bit length proof, where the number of queries is equal to the width of the DNF.

Proposition 3.4. *Let* T *be a linear transformation, and* C *be a* $\mathsf{DNF} \circ \mathsf{T}$ *circuit of size* S *and width* w*. There exists an* NP *proof for the language* $L = \{x \mid C(x) = 1\}$*, where the witness length is* $\lceil \log S \rceil$*. Verification of the proof requires* w *queries to the bits of* $\mathsf{T}(x)$ *(rather than reading the input directly).*

Proof. The witness is an index of a satisfied clause in C. The verifier reads the w bits of $\mathsf{T}(x)$ that are in the specified clause to verify that the clause is indeed satisfied. Proposition 3.4 follows immediately from the construction and the definition of $\mathsf{DNF} \circ \mathsf{T}$ circuits. ■

Inability of verifying 0-claims. Note, that while it is easy to have a short proof for checking whether a $\mathsf{DNF} \circ \mathsf{T}$ circuit is *satisfied*, it is not clear how to prove that it is not (i.e., that $C(x) = 0$ for a given input x). Since the narrow circuit that is obtained from Proposition 3.3 computes the inner product function correctly only on a fraction of the inputs, we would like to use the self-correction of homogeneous bilinear mappings (see Claim 2.7) to derive an AM[2]-HIP protocol for inner product claims, that works on all inputs. However, the straightforward self-correction requires to verify also whether an inner product is 0, which we don't know how to do efficiently. We deal with this difficulty next.

3.3 From One-Sided Average-Case to Two-Sided Worst-Case

Recall that for a proximity parameter ϵ and a linear code T, our goal is a reduction from an $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocol (w.r.t to T) for L_{IP}, to computing the inner product function on $\frac{5}{6}+\epsilon$ fraction of the inputs by a $\mathsf{DNF}\circ\mathsf{T}$ circuit. By Proposition 3.3, we have a $\mathsf{DNF}\circ\mathsf{T}$ circuit of width at most $w = O\left(\frac{\log S}{\tau}\right)$ that agrees with the inner product function on at least $\frac{5}{6}+\epsilon-\tau$ fraction of the inputs. By Proposition 3.4, we can exploit the narrow clauses in C'_τ, and verify inner product claims by verifying the output of C'_τ with only $O(\log S)$ proof length and $w = O\left(\frac{\log S}{\tau}\right)$ queries to the encoding of the input via T. However, that gives us a solution that works for *most* inputs, rather than *all* inputs, and also works only for claims of the form $\mathsf{IP}_2(x,y) = 1$ and does not extend also to claims of the form $\mathsf{IP}_2(x,y) = 0$. In addition, note that using the straightforward self-correction property of the inner product function (see Claim 2.7) also raises a difficulty: the straightforward self-correction reduces an inner product claim to multiple inner product claims over random strings, which may be equal to 0.

Thus, our next step is presenting a compiler that generalizes the self-correction property also for the case we have a protocol for verifying only one-sided claims (i.e. only verifies whether an inner product is 1). The compiler is given an $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocol that can only verify claims of the form $\mathsf{IP}_2(x,y) = 1$, and only on most inputs, and outputs an $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocol that can verify any inner product claim, on *all* inputs. Actually, this follows from a more general result, established in Lemma 3.5, that gives a compiler for any homogeneous multilinear mapping, of which the inner product is a special case (see Fact 6).

We remark that the compiler has a $O(\frac{n}{\epsilon^2})$ overhead to the randomness complexity, whereas we seek randomness complexity $O(\log n)$. We overcome this issue by employing a generic technique, due to [New91], for reducing randomness complexity using non-uniformity. We observe that this technique can also be applied on $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocols (see Proposition 3.6 below).

Lemma 3.5 (From one-sided avg-case, to two-sided worst-case)**.** *Let $k \geq 1$ be an integer, let $\epsilon \in \left(0, \frac{1}{2\cdot(2^k-1)}\right]$ and let $\mathsf{T} : \left(\{0,1\}^n\right)^k \to \{0,1\}^{n'}$ be a linear map. Also, let $g : \left(\{0,1\}^n\right)^k \to \{0,1\}$ be a non-zero function that is $(1-\frac{1}{2(2^k-1)}+\epsilon)$-close to a homogeneous k-linear mapping f. Suppose that there exists an $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocol for the language $\left\{(x^{(1)},\ldots,x^{(k)}) : g\left(x^{(1)},\ldots,x^{(k)}\right) = 1\right\}$ with proof length ℓ, randomness complexity r, and q queries to the bits of $\mathsf{T}\left(x^{(1)},\ldots,x^{(k)}\right)$. Then, there exists an $\mathsf{AM}[2]\text{-}\mathsf{HIP}$ protocol for the language $\left\{(x^{(1)},\ldots,x^{(k)},b) : f\left(x^{(1)},\ldots,x^{(k)}\right) = b\right\}$ with queries to the bits of $\mathsf{T}\left(x^{(1)},\ldots,x^{(k)}\right)$, and with the following parameters:*

$$\textit{Query Complexity: } O\left(\frac{q\cdot 2^k\cdot\left(k+\log\left(\frac{1}{\epsilon}\right)\right)}{\epsilon^2}\right).$$

$$\textit{Proof Length: } O\left(\frac{\ell\cdot 2^k\cdot\left(k+\log\left(\frac{1}{\epsilon}\right)\right)}{\epsilon^2}\right).$$

$$\textit{Randomness Complexity: } O\left(\frac{n\cdot k + r\cdot 2^k\cdot\left(k+\log\left(\frac{1}{\epsilon}\right)\right)}{\epsilon^2}\right).$$

Notations. A few notations that will be used throughout the proof. By a "b-claim" we refer to a claim of the form "g outputs the bit b" on a certain input. Also, recall that by the self-correction property of homogeneous multilinear mappings (see Claim 2.7), it holds that for any

$(r^{(1)}, \ldots, r^{(k)}) \in \{0,1\}^n$:

$$f(x^{(1)}, \ldots, x^{(k)}) = \bigoplus_{s_1,\ldots,s_k \in \{0,1\}} f\Big(r^{(1)} \oplus (x^{(1)} \cdot s_1), \ldots, r^{(k)} \oplus (x^{(k)} \cdot s_k)\Big). \tag{6}$$

We refer to the term in the sum in Eq. (6) that corresponds to $(s_1, \ldots, s_k) = (0, \ldots, 0)$ as the first term, and note that it doesn't depend on the input $(x^{(1)}, \ldots, x^{(k)})$.

Proof Outline. The basic idea in the proof of Lemma 3.5 (which is similar to the idea that was used in the works of Shaltiel and Umans [SU05, SU06]) is observing that all of the inputs to f on the right-hand side in Eq. (6) are (individually) random. Thus, we can utilize our average-case HIP in order to prove correctness. The major difficulty is that some of the claimed values may be 0, and we have an HIP for 1-claims. We cope with the 0-claims by observing that the prover can only convince the verifier that g outputs 0 when it actually outputs 1, but not the opposite (that is because the verifier can verify all the 1-claims). Therefore, by having the verifier check only 1-claims and blindly accept 0-claims, we get that if a malicious prover sends a false claim, then it must be a 0-claim. A key observation is that this makes the expected value of its claims lower than the expected value of g on a random input. By repeating the experiment several times, each time with fresh random strings, we get a significant statistical gap between the case that the prover lies and the case it is honest. Thus, by adding an accepting criterion on the average value of the prover's claims, we can detect when the prover is lying. To sum up, the new protocol proceeds as follows: the parties repeat the self-correction procedure of Eq. (6) for a large enough number of times, each time with fresh random vectors. For each iteration, the prover sends the values of the terms on the right-hand side of Eq. (6), and the verifier checks only the 1-claims by using the original protocol. Finally, the verifier accepts if all the following conditions are satisfied:

1. The original protocol accepts all the 1-claims.
2. The average value of all the prover's claims is close enough to the expectation.
3. The value of f in most of the iterations is b.

Proof. The rest of the proof refers to the formal protocol, which is described in Fig. 3.

First, let us show that all the steps in the protocol are well defined. Note, that using oracle access to the bits of $\mathsf{T}\left(x^{(1)}, \ldots, x^{(k)}\right)$, the verifier can compute all the queries that V requires by computing $\mathsf{T}\left(r^{(1,1)}, \ldots, r^{(1,k)}, \ldots, r^{(m,k)}\right)$ on its own and using the linearity of T. Therefore, the execution of V in Step 3(c) is well defined. In addition, since V is an AM[2]-HIP verifier, the prover can send its claims in Step 3 together with the first message in the simulation[11] of V. Moreover, the parties can run Step 3 on all $i \in [m]$ in parallel, resulting in an AM[2]-HIP protocol as required.

Communication and query complexity. V is obtained from the original protocol by $O(k+\log m)$ parallel repetitions. Thus, V makes $O((k+\log m)\cdot q)$ queries, uses $O((k+\log m)\cdot r)$ random coins, and has $O((k+\log m)\cdot \ell)$ proof length. Since the protocol consists of running V at most $m \cdot 2^k$ times and sending additional $m \cdot 2^k$ bits for the prover's claims, the query complexity is $O\left((k+\log m)\cdot q \cdot 2^k \cdot m\right)$ and the proof length is $O\left((k+\log m)\cdot \ell \cdot 2^k \cdot m\right)$. The randomness complexity consists of the cost of running V at most $m \cdot 2^k$ times and the additional $m \cdot k$ random n-bit length Boolean strings the verifier tosses in Step 2. Therefore, the total

[11]Formally, since the verifier runs V only for 1-claims, it will send in advance the random coins for V and uses them only if the prover will indeed send 1-claims.

Statement: $g\left(x^{(1)},\ldots,x^{(k)}\right)=b$, for $b\in\{0,1\}$ and $x^{(1)},\ldots,x^{(k)}\in\{0,1\}^n$

Parameters: Let $\mu\stackrel{\text{def}}{=}\mathbf{E}[g]$, let $\delta\stackrel{\text{def}}{=}\frac{\epsilon}{4\cdot\mu}$, and let $m\stackrel{\text{def}}{=}\frac{64}{\epsilon^2}$. Note that since $0\le\epsilon<\frac{1}{2^k}$, then[a] $0\le\delta<1$, and since g is not identically zero, then $\mu>0$.

Protocol:

1. Reduce the completeness and the soundness errors of the original $\mathsf{AM}[2]$-HIP protocol to be at most $(\frac{1}{100\cdot m\cdot(2^k-1)})$ using parallel repetition (see [Gol98, Appendix C.1]). Denote the new protocol by V.

2. For every $i\in[m]$ and $j\in[k]$, select a random vector $r^{(i,j)}\in\{0,1\}^n$ and send $r^{(i,j)}$ to the prover.

3. For every $i\in[m]$, define

$$R_i=\bigoplus_{s_1,\ldots,s_k\in\{0,1\}} g\Big(r^{(i,1)}\oplus(x^{(1)}\cdot s_1),r^{(i,2)}\oplus(x^{(2)}\cdot s_2),\ldots,r^{(i,k)}\oplus(x^{(k)}\cdot s_k)\Big).$$

   The verifier computes the value of each R_i as follows:

   (a) **Claims**: The prover sends the value of all the terms in the sum that forms R_i, except for the first term[b] which is computed by the verifier on its own, as it depends only on the random coins and not also on the inputs $x^{(1)},\ldots,x^{(k)}$.

   (b) **Computing** R_i: The verifier computes the sum of all the prover's claims and the computed value of the first term, and considers the sum as the value of R_i.

   (c) **Verifying 1-claims**: For each prover's claim from $3(a)$, if it is a 0-claim, then the verifier considers it blindly as 0. Otherwise, the verifier runs V to verify that the value is indeed 1 and rejects if V rejected.

4. **Average criterion**: If the average value of *all* the prover's claims in the interaction is less than $(1-\delta)\cdot\mu$, then the verifier rejects.

5. Else, the verifier accepts if more than $m/2$ of the R_i-s were computed as b, and otherwise it rejects.

[a] Clearly $\delta\ge 0$. To see that $\delta<1$, note that otherwise we get that $\mu\le\frac{1}{2^{k+2}}$, which by Claim 2.7 means that for any *fixed* input z, $g(z)$ has probability higher than 0 to be zero, and thus must be identically 0.

[b] It is an optimization we used to reduce the number of claims that the verifier has to check, and thus pushing the limitation on ϵ to $\frac{1}{2\cdot(2^k-1)}$. When k is large, it does not really matter. However, when $k=2$ (e.g, in the inner product function), it is the difference between a $\frac{7}{8}$ approximation limit and a $\frac{5}{6}$ approximation limit.

Figure 3: Protocol of Lemma 3.5.

randomness complexity is $O\left(m \cdot 2^k \cdot (k + \log m) \cdot r + m \cdot k \cdot n\right)$. By setting $m = O\left(\frac{1}{\epsilon^2}\right)$ we get the required proof length, query and randomness complexities.

Soundness and completeness error. Before analyzing the completeness and the soundness of the protocol, we define two sets of random variables, $\{A_i\}_{i \in [m]}$ and $\{W_i\}_{i \in [m]}$, as well as an indicator I, that will be used in the analysis of the protocol. Consider the 2^k-size i-th sum that the verifier tries to compute in Step 3:

$$R_i = \bigoplus_{s_1, \ldots, s_k \in \{0,1\}} g\Big(r^{(i,1)} \oplus (x^{(1)} \cdot s_1), r^{(i,2)} \oplus (x^{(2)} \cdot s_2), \ldots, r^{(i,k)} \oplus (x^{(k)} \cdot s_k)\Big).$$

We define the random variables as follows:

- $0 \le A_i \le 1$ is defined to be the average of the *real* values of all the terms in the sum that forms R_i.
- $W_i \in \{0,1\}$ is defined to be an indicator which equals 1 if and only if g and f disagree on at least one of the $2^k - 1$ prover's claims about terms in the i-th sum.
- $I \in \{0,1\}$ is defined to be 1 if and only if V failed to verify at least one (true or false) 1-claim during the protocol.

Note, that since the random strings $\{r^{(i,j)}\}_{(i,j) \in [m] \times [k]}$ are independent, then all the A_i are independent and identically distributed, ans similarly for the W_i-s. Thus, we denote

$$\mu_w \overset{def}{=} \mathbf{E}[W_1] = \cdots = \mathbf{E}[W_m], \text{ and,}$$
$$\mu_A \overset{def}{=} \mathbf{E}[A_1] = \cdots = \mathbf{E}[A_m].$$

Also, note that for any $s_1, \ldots, s_k \in \{0,1\}$ and $i \in [m]$, the strings $\left\{r^{(i,t)} \oplus (x^{(t)} \cdot s_t)\right\}_{t \in [k]}$ are independent random strings. Therefore, $\mu_A = \mu$, and so

$$\mathbf{E}\left[\sum_{i=1}^{m} A_i\right] = \mu_A \cdot m = \mu \cdot m.$$

In addition, since g is $(1 - \frac{1}{2(2^k-1)} + \epsilon)$-close to the function f, by the union bound f and g agree on all of the $2^k - 1$ prover's claims about the i-th sum with all but $\left(\frac{1}{2(2^k-1)} - \epsilon\right) \cdot (2^k - 1) = \frac{1}{2} - \epsilon \cdot (2^k - 1)$ probability. Thus, $\mu_W \le \frac{1}{2} - \epsilon \cdot (2^k - 1) < \frac{1}{2}$. By linearity of expectation, we get that

$$\mathbf{E}\left[\sum_{i=1}^{m} W_i\right] = m \cdot \mu_w < \frac{m}{2}.$$

Lastly, regarding I, since the completeness and the soundness errors of V are at most $\frac{1}{100 \cdot m \cdot (2^k - 1)}$, by union bounding over all the $m \cdot (2^k - 1)$ prover's claims during the protocol, we get that

$$\Pr\Big[I = 1\Big] \le \frac{1}{100}.$$

With these random variables in hand, we are ready to analyze the completeness and the soundness of the protocol.

Completeness. Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ such that $f\left(x^{(1)}, \ldots, x^{(k)}\right) = b$, we need to show that the verifier rejects with probability at most $\frac{1}{3}$. The verifier rejects only if one of the following three events happens: V rejects some 1-claim; the average value of the prover's claims is lower than $(1-\delta)\cdot\mu$; or, most of the R_i-s have been computed by the verifier wrongly as $1-b$, rather than as b. The probability that the first event occurs is bounded by the probability that $I = 1$, which we already bounded by $\frac{1}{100}$.

For the second event, note that the average value of the prover's claims when the prover is honest, is exactly $\frac{\sum A_i}{m}$ (the average of the real values). Hence, the second event occurs only if $\frac{\sum A_i}{m} < (1-\delta)\cdot\mu$. We can bound the probability for this event using the Chernoff Bound (Lemma 2.11):

$$\Pr\left[\frac{\sum_{i=1}^m A_i}{m} < (1-\delta)\cdot\mu\right] \le e^{-\delta^2\mu^2 m} = e^{-2\cdot\frac{\epsilon^2}{8}\cdot m} < e^{-2}.$$

For the third event ("most of the R_i-s were computed wrongly"), observe that by Eq. (6) and the construction of the protocol, for any $i \in [m]$, if f and g agree on all the prover's claims about the terms that forms R_i, then the verifier computes R_i correctly as b. Thus, the probability that most of the R_i-s were computed *wrongly* is bounded by the probability that on at least $\frac{m}{2}$ of them, f and g disagree on at least one of the prover's claims, namely, the probability that $\sum_{i=1}^m W_i \ge \frac{m}{2}$. Therefore, to bound the probability that the third event occurs, it suffices to bound the probability that $\sum_{i=1}^m W_i \ge \frac{m}{2}$. If $\mu_W = 0$, then the probability is zero, and if $\mu_W > 0$, we get that:

$$\begin{aligned}
\Pr\left[\sum_{i=1}^m W_i \ge \frac{m}{2}\right] &= \Pr\left[\sum_{i=1}^m W_i \ge \left(1 + \left(\frac{1}{2\cdot\mu_W} - 1\right)\right)\cdot\mu_W\cdot m\right] \\
&\le e^{-\frac{2\cdot\left(\frac{1}{2\cdot\mu_W}-1\right)^2\cdot{\mu_W}^2\cdot m^2}{m}} \\
&= e^{-2\cdot\left(\frac{1}{2}-\mu_W\right)^2\cdot m} \\
&\le e^{-2\epsilon^2\cdot(2^k-1)^2\cdot m} \\
&< e^{-2\epsilon^2\cdot m} \\
&< e^{-2}.
\end{aligned}$$

The second inequality follows by the Chernoff bound (Lemma 2.11) and that $\mu_w < \frac{1}{2}$. To sum up, by adding the probabilities of these three events, we get that

$$\Pr\left[\text{V rejects}\right] < \frac{1}{100} + 2e^{-2} < \frac{1}{3}.$$

Soundness. Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ such that $f\left(x^{(1)}, \ldots, x^{(k)}\right) = 1-b$, we need to show that for any prover strategy, the verifier accepts with probability at most $\frac{1}{3}$.

Let us fix some prover strategy P^*. We distinguish between the following two cases:[12] the case the event $\Big(I = 1\Big) \vee \left(\frac{1}{m}\cdot\sum_{i=1}^m W_i \ge \frac{\frac{1}{2}+\mu_w}{2}\right)$ occurs, and the case the complement event

[12]An intuition for why we compare $\frac{1}{m}\sum_i W_i$ to $\frac{\frac{1}{2}+\mu_W}{2}$: we want to show that with *high* probability f and g completely agree in *most* of the iterations, and thus the prover must send false claims in order to make the verifier accept. The "most" requires us to pick a fraction smaller than $\frac{1}{2}$, and the "high probability" requires us to pick a fraction greater than μ_w. Thus, we choose the average of these two.

$\Big(I=0\Big) \wedge \Big(\frac{1}{m}\cdot\sum_{i=1}^m W_i < \frac{\frac{1}{2}+\mu_w}{2}\Big)$ occurs. By the law of total probability, the probability that the verifier accepts can be expressed as

$$\Pr\left[\Big(I=1\Big) \vee \Big(\frac{1}{m}\cdot\sum_{i=1}^m W_i \geq \frac{\frac{1}{2}+\mu_w}{2}\Big)\right] + \Pr\left[\text{V accepts } \wedge \Big(I=0\Big) \wedge \Big(\frac{1}{m}\cdot\sum_{i=1}^m W_i < \frac{\frac{1}{2}+\mu_w}{2}\Big)\right]. \tag{7}$$

Let us start by bounding the first expression. We get that

$$\begin{aligned}
\Pr\left[\Big(I=1\Big) \vee \Big(\frac{1}{m}\cdot\sum_{i=1}^m W_i \geq \frac{\frac{1}{2}+\mu_w}{2}\Big)\right] &\leq \Pr\Big[I=1\Big] + \Pr\left[\frac{1}{m}\cdot\sum_{i=1}^m W_i \geq \frac{\frac{1}{2}+\mu_w}{2}\right] \\
&\leq \frac{1}{100} + \Pr\left[\sum_{i=1}^m W_i \geq \frac{m\cdot(\frac{1}{2}+\mu_w)}{2}\right] \\
&= \frac{1}{100} + \Pr\left[\sum_{i=1}^m W_i \geq \Big(1+(\frac{1}{4\mu_w}-\frac{1}{2})\Big)\cdot m\cdot\mu_W\right] \\
&\leq \frac{1}{100} + e^{\frac{-2\cdot\left(\frac{1}{4\mu_w}-\frac{1}{2}\right)^2\cdot{\mu_W}^2\cdot m^2}{m}} \\
&= \frac{1}{100} + e^{-2\left(\frac{1}{4}-\frac{\mu_w}{2}\right)^2\cdot m} \\
&\leq \frac{1}{100} + e^{-2\cdot\frac{\epsilon^2(2^k-1)^2}{4}\cdot m} \\
&< \frac{1}{100} + e^{-2\cdot\frac{\epsilon^2}{4}\cdot m} \\
&< \frac{1}{100} + e^{-2}.
\end{aligned}$$

Where the fourth inequality is by the Chernoff bound (Lemma 2.11) and the fact that $\mu_W \leq \frac{1}{2}$. For the sake of completeness, also note that by their definition, all the W_i-s are independent with each other, and with the prover strategy.

For the second expression in Eq. (7), recall that in order to make the verifier accept, the prover must make the verifier compute more than $\frac{m}{2}$ of the R_i-s as b. As a result, since the second expression in Eq. (7) requires that $\frac{1}{m}\cdot\sum_{i=1}^m W_i < \frac{\frac{1}{2}+\mu_w}{2}$, the prover must send at least $\frac{m\cdot(\frac{1}{2}-\mu_w)}{2}$ false claims in order to make the verifier accept. Since $I=0$, the prover can only send false 0-*claims* without making the verifier reject, and by doing that, the average value of its claims reduces to $\frac{\sum_{i=1}^m A_i}{m} - \frac{1}{m}\cdot\frac{\#\text{false 0-claims}}{2^k-1} \leq \frac{\sum_{i=1}^m A_i}{m} - \frac{1}{m}\cdot\frac{m\cdot(\frac{1}{2}-\mu_w)}{2\cdot(2^k-1)}$. Recall, that the verifier rejects also if the average value of the prover's claims is less than $(1-\delta)\cdot\mu$. Therefore, in order to bound the second expression of Eq. (7), it suffices to bound the probability that $\frac{\sum_{i=1}^m A_i}{m} - \frac{1}{m}\cdot\frac{m\cdot(\frac{1}{2}-\mu_w)}{2\cdot(2^k-1)} \geq (1-\delta)\cdot\mu$, as it contains the event that is computed in the second expression of Eq. (1). We note that by their definition, all the A_i-s are independent with the prover strategy, and thus we can use the Chernoff bound and get:

$$\begin{aligned}\Pr\left[\frac{1}{m}\cdot\sum_{i=1}^{m}A_i-\frac{1}{m}\cdot\frac{m\cdot(\frac{1}{2}-\mu_w)}{2\cdot(2^k-1)}\geq(1-\delta)\cdot\mu\right]&=\Pr\left[\sum_{i=1}^{m}A_i\geq\Big(1+\big(\frac{\frac{1}{2}-\mu_w}{2\cdot\mu\cdot(2^k-1)}-\delta\big)\Big)\cdot m\cdot\mu\right]\\&\leq\Pr\left[\sum_{i=1}^{m}A_i\geq\Big(1+\big(\frac{\epsilon\cdot(2^k-1)}{2\cdot\mu\cdot(2^k-1)}-\delta\big)\Big)\cdot m\cdot\mu\right]\\&=\Pr\left[\sum_{i=1}^{m}A_i\geq\Big(1+\frac{\delta}{2}\Big)\cdot m\cdot\mu\right]\\&\leq e^{-2\cdot\frac{\delta^2}{4}\cdot\mu^2\cdot m}\\&\leq e^{-2}.\end{aligned}$$

To sum up, we get that

$$\Pr\left[\text{V accepts}\right]<\frac{1}{100}+2e^{-2}<\frac{1}{3}.$$

■

3.4 An AM[2]-HIP for L_{IP}

We complete the proof of Lemma 3.2 by showing that it can be derived directly from the results established in Sections 3.1 to 3.3. For convenience, we restate Lemma 3.2:

Lemma 3.2. (AM[2]-HIP for L_{IP}). *Fix an integer n, and a parameter $\epsilon\in(0,1/6]$. Let $\mathsf{T}:\{0,1\}^{2n}\to\{0,1\}^{n'}$ be some linear code. Suppose there exists a DNF $\circ$ T circuit C of size S that computes $\mathsf{IP}_2(x,y)$ on at least $\frac{5}{6}+\epsilon$ fraction of the inputs. Then, there exists an AM[2]-HIP protocol for L_{IP}, with proof length $\log(S)\cdot\widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, randomness complexity $O(\log n)$ and $\log(S)\cdot\widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ queries to the bits of $\mathsf{T}(x,y)$.*

Proof. First, we observe that it suffices to prove a result similar to Lemma 3.2 but with $O(\frac{n}{\epsilon^2})$ randomness complexity, rather than $O(\log n)$, by using the following proposition:

Proposition 3.6. (Reducing randomness). *Fix an input length n, and let $\mathcal{L}\subseteq\{0,1\}^n$. For any AM[2]-HIP protocol for $\mathcal{L}$ with proof length c and query complexity q, there exists an AM[2]-HIP protocol for $\mathcal{L}$ with proof length $O(c)$, query complexity $O(q)$, and randomness complexity $O(\log n)$.*

The proof of Proposition 3.6 is mainly based on a general technique of [New91] for reducing the randomness complexity of non-uniform probabilistic proofs. We remark that [New91] shows his technique in the context of (public-coin) randomized communication complexity protocols, but it can readily adapt to public-coin interactive proofs, as well as to AM[2]-HIP. The proof of Proposition 3.6 is deferred to Appendix A.3.

Let C'_τ be the narrow circuit that is obtained from C using Proposition 3.3 and a parameter $\tau>0$. Recall that C'_τ has width $w=O\left(\frac{\log S}{\tau}\right)$, and it agrees with C on $(1-\tau)$ fraction of the inputs. Since C is $\left(\frac{5}{6}+\epsilon\right)$-close to the inner product function, then C'_τ is $\left(\frac{5}{6}+\epsilon-\tau\right)$-close to the inner product function. We choose τ to be $\frac{\epsilon}{2}$ and get that C'_τ is $\left(\frac{5}{6}+\frac{\epsilon}{2}\right)$-close to the inner product function. In addition, since the expectation of the inner product function is $\frac{1}{2}-\frac{1}{2^{n+1}}$, then C'_τ doesn't compute the zero function. By Proposition 3.4, there is an AM[2]-HIP protocol (in particular, a Holographic NP proof) for claims of the form $C'_\tau(x,y)=1$, with zero random bits, $O\left(\frac{\log S}{\tau}\right)$ proof length and $O\left(\frac{\log S}{\tau}\right)$ queries to the bits of $\mathsf{T}_1(x)$ and $\mathsf{T}_2(y)$. Hence, by Lemma 3.5 with the parameters $k = 2$, $f = \mathsf{IP}_2(x, y)$, $g = C'_\tau(x, y)$, and $\mu = \mathbf{E}_{x,y}\left[C'_\tau(x, y)\right] > 0$, we get an $\mathsf{AM}[2]$-HIP protocol for L_{IP} with $O\left(\frac{n}{\epsilon^2}\right)$ randomness complexity, $O\left(\frac{\log S}{\epsilon^3} \cdot \log\left(\frac{1}{\epsilon}\right)\right)$ proof length and $O\left(\frac{\log S}{\epsilon^3} \cdot \log\left(\frac{1}{\epsilon}\right)\right)$ queries.

■

4 Verifying Low Degree Polynomials

In this section, we continue to study the ramifications of the existence of (relatively) small $\mathsf{DNF} \circ \mathsf{T}$ circuits for computing the inner product function on most inputs. We show that such circuits yield an efficient (multiple round) Arthur-Merlin Streaming protocol and a (one round) Arthur-Merlin Communication Complexity protocol for problems that are decidable by low degree polynomials (over $\mathbb{GF}(2)$).

While a Communication Complexity result follows from the streaming result (see Fact 5), we achieve better parameters by showing an explicit Communication Complexity protocol. In particular, the Communication Complexity protocol has only a one round of interaction, which lets us extend the protocol for any function that is decidable by an $\mathsf{AC}^0(\oplus)$ circuit (see Section 4.3 for the details). In Table 1 we compare the complexity parameters of the explicit $\mathsf{AM}[2]$-CC protocol and of the $\mathsf{AM}[k]$-CC protocol that is obtained from the streaming protocol by applying Fact 5. The table shows the number of rounds and the amount of communication in these two protocols as a function of the size of the $\mathsf{DNF} \circ \mathsf{T}$ circuit for approximating the inner product function, to a factor of $\frac{5}{6} + \epsilon$ (for some $\epsilon > 0$).

In additoin, we also remark that the $\mathsf{AM}[2]$-CC protocol is simpler than the streaming protocol, and can be viewed as a warmup for the $\mathsf{AM}[2]$-DS protocol.

Circuit Size	Using $\mathsf{AM}[2d]$-$\mathsf{DS} \to \mathsf{AM}[k]$-$\mathsf{CC}$ transformation.	An explicit $\mathsf{AM}[2]$-CC protocol
$S(m)$	$\log(S(m)) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$, d rounds.	$\log\left(S(2 \cdot m^d)\right) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, 1 round
2^{m^δ}	$m^\delta \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$, d rounds.	$m^{d \cdot \delta} \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, 1 round
$2^{\mathrm{polylog}\,(m)}$	$\mathrm{polylog}(m) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$, d rounds.	$\mathrm{poly}(d) \cdot \mathrm{polylog}\,(m) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, 1 round
$\mathrm{poly}(m)$	$\log(m) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$, d rounds.	$d \cdot \log m \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$, 1 round

Table 1: $\mathsf{AM}[k]$-CC communication and rounds for d degree polynomial, and input size $m = 2n$.

4.1 An $\mathsf{AM}[2]$ Communication Complexity Protocol

Theorem 8. *Fix integers n, N and a parameter $\epsilon \in (0, 1/6]$. Let $\mathsf{T} : \{0,1\}^{2N} \to \{0,1\}^{n'}$ be some linear code. Suppose there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size $S = S(2N)$ that computes the function $\mathsf{IP}_2(x, y)$ on at least a $\frac{5}{6} + \epsilon$ of the N-bit length inputs. Then, for any polynomial $\mathcal{P} : \{0,1\}^n \to \{0,1\}$ with N monomials, and for any $b \in \{0,1\}$, there exists an $\mathsf{AM}[2]$-CC protocol for the function $\mathcal{P}_b(x, y) \overset{def}{=} \begin{Bmatrix} 1 & \mathcal{P}(x, y) = b \\ 0 & \text{o/w} \end{Bmatrix}$ with $\log\left(S(2N)\right) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ communication* *complexity, where Merlin (the prover) gets the inputs $x, y \in \{0,1\}^n$, Alice gets x and Bob gets y.*

Note that N may be polynomial or even exponential in n. In particular, if $\mathcal{P}$ is a d degree polynomial, then $N \leq (2n)^d$. By denoting $m = 2n$, we get $\log\left(S\big(2 \cdot m^d\big)\right) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ communication complexity, as in Table 1.

Proof. We aim to use the $\mathsf{AM}[2]\text{-}\mathsf{CC}$ protocol for the inner product function from Corollary 7, by reducing the task of evaluating $\mathcal{P}$ to the task of computing the inner product of the truth tables of two certain functions. In fact, in order to apply Corollary 7 we need an inner product between the truth table of a function that depends only on Alice's inputs and the truth table of a function that depends only on Bob's inputs. Observe, that the output of $\mathcal{P}$ can be viewed as a linear combination of its monomials, each of which is a product between a subset of the input variables and a coefficient (which is also either 0 or 1, since $\mathcal{P}$ is defined over $\mathbb{GF}(2)$). For any monomial $\alpha \in [N]$ and inputs $x, y \in \{0,1\}^n$, we denote by $I_x^{(\alpha)}$ and $I_y^{(\alpha)}$ the indices sets of the input variables that appear in the α-th monomial, where $I_x^{(\alpha)}$ is the indices of x, and $I_y^{(\alpha)}$ is the indices of y. Therefore, the output of $\mathcal{P}$ on a given input $z = (x, y)$ can be written as the following expression:

$$\mathcal{P}(x,y) = \bigoplus_{\alpha \in [N]} \Big(\prod_{i \in I_x^{(\alpha)}} x_i\Big) \cdot \Big(\prod_{i \in I_y^{(\alpha)}} y_i\Big) \cdot \mathsf{Coef}_{\mathcal{P}}(\alpha),$$

where $Coef_{\mathcal{P}}(\alpha)$ outputs the coefficient in the α-th monomial, and an empty product is defined to be 1. Lastly, by defining $X_{\mathcal{P}}(x,\alpha) \overset{def}{=} \left(\prod_{i \in I_x^{(\alpha)}} x_i\right)$ and $Y_{\mathcal{P}}(y,\alpha) \overset{def}{=} \left(\prod_{i \in I_y^{(\alpha)}} y_i\right)$, and noticing that the coefficients are known to all the parties, we get an inner product between a function that depends only on Alice's inputs, and a function that depends only on Bob's inputs

$$\mathcal{P}(x,y) = \bigoplus_{\alpha \in [N]} X_{\mathcal{P}}(x,\alpha) \cdot Y_{\mathcal{P}}(y,\alpha) \cdot \mathsf{Coef}_{\mathcal{P}}(\alpha). \tag{8}$$

Using these notations, the $\mathsf{AM}[2]\text{-}\mathsf{CC}$ protocol is described in Fig. 4.

First, we show that the protocol is well defined. Observe, that the computation of each party during the preprocessing step depends only on the input they get or on the polynomial's structure, which is known to all the parties. Hence, the preprocessing step of the protocol is well defined. Also, note that the domain of $\mathcal{A}$ and $\mathcal{B}$ is $[N]$, which means that their truth tables contain exactly N entries, and thus we can apply the $\mathsf{AM}[2]\text{-}\mathsf{CC}$ protocol for the inner product function in Step 2, which is defined for N-bit length inputs.

Communication complexity. The preprocessing step does not require any communication since all the computations of the parties are local. Thus, the communication complexity is equal to the communication complexity of Step 2, which is $\log\big(S(2N)\big) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ by Corollary 7.

Completeness and soundness. From Eq. (8) and the definitions of $\mathcal{A}$ and $\mathcal{B}$ we get that

$$\mathcal{P}(x,y) = \bigoplus_{\alpha \in [N]} \mathcal{A}(\alpha) \cdot \mathcal{B}(\alpha)$$

Thus, the completeness and the soundness follow immediately from the completeness and soundness of the $\mathsf{AM}[2]\text{-}\mathsf{CC}$ protocol in the second step, which follow from Corollary 7. ∎

Inputs:

- Alice's input: a polynomial $\mathcal{P}$ with N monomials, a bit $b \in \{0,1\}$ and $x \in \{0,1\}^n$.
- Bob's input: $y \in \{0,1\}^n$ and the same $\mathcal{P}$ and b.
- Merlin's input: The same $\mathcal{P}$, x, y and b.

Protocol:

1. Preprocessing:
   (a) Alice constructs $\mathcal{A}(\alpha) \in \{0,1\}^N$, defined as $\mathcal{A}(\alpha) = X_{\mathcal{P}}(x,\alpha)$.
   (b) Bob constructs $\mathcal{B}(\alpha) \in \{0,1\}^N$, defined as $\mathcal{B}(\alpha) = Y_{\mathcal{P}}(y,\alpha) \cdot \mathsf{Coef}_{\mathcal{P}}(\alpha)$.
   (c) Merlin constructs $\mathcal{A}$ and $\mathcal{B}$.
2. The parties run the $\mathsf{AM}[2]\text{-}\mathsf{CC}$ protocol that is obtained from Corollary 7 with respect to the circuit C, for N-bit length inputs, on the truth tables of $\mathcal{A}$ and $\mathcal{B}$.

Figure 4: ($\mathsf{AM}[2]\text{-}\mathsf{CC}$ for $\mathcal{P}$)

4.2 An $\mathsf{AM}[2d]$ Streaming Protocol

Theorem 9. *Fix an integer n, and let $\epsilon \in (0, 1/6]$. Let $\mathsf{T} : \{0,1\}^{2n} \to \{0,1\}^{n'}$ be some linear code. Suppose there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size S that computes the function $\mathsf{IP}_2(x,y)$ on at least a $\frac{5}{6} + \epsilon$ fraction of the n-bit length inputs. Then, for any d degree polynomial $\mathcal{P} : \{0,1\}^n \to \{0,1\}$, and for any $b \in \{0,1\}$, there exists an $\mathsf{AM}[2d]\text{-}\mathsf{DS}$ protocol for the language $L \overset{def}{=} \{x \in \{0,1\}^n \mid \mathcal{P}(x) = b\}$ with $\log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$ randomness complexity, proof length and space complexity.*

By Fact 4, it suffices to show a *non-adaptive* $\mathsf{AM}[2d]\text{-}\mathsf{HIP}$ protocol for the language L, with $\log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$ randomness, communication and query complexities. Thus, the proof of Theorem 9 follows from the following lemma:

Lemma 4.1. *Let $T : \{0,1\}^{2n} \to \{0,1\}^{n'}$ be some linear code. Suppose there exists a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size S that computes the function $\mathsf{IP}_2(x,y)$ on at least a $\frac{5}{6} + \epsilon$ fraction of the n-bit length inputs. Then, for any d degree polynomial $\mathcal{P} : \{0,1\}^n \to \{0,1\}$, and for any $b \in \{0,1\}$, there exists a non-adaptive $\mathsf{AM}[2d]\text{-}\mathsf{HIP}$ protocol for the language $L \overset{def}{=} \{x \in \{0,1\}^n \mid \mathcal{P}(x) = b\}$ with $\log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)$ randomness, communication and query complexities.*

Similar to the proof of Theorem 8, since $\mathcal{P}$ is a polynomial (over $\mathbb{GF}(2)$), its output can be viewed as a linear combination of its monomials. Since $\mathcal{P}$ is a degree d polynomial, any monomial is a product between a subset of d input variables, and a Binary coefficient. Therefore, there exists a function $\mathsf{Coef}_{\mathcal{P}} : [n]^d \to \{0,1\}$ that depends only on $\mathcal{P}$, such that:

$$\mathcal{P}(x) = \bigoplus_{j_1,\ldots,j_d \in [n]} x_{j_1} \cdot x_{j_2} \cdot \cdots \cdot x_{j_d} \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d). \tag{9}$$

The goal of the protocol is to convert a claim about the right-hand side of Eq. (9), to claims that don't depend on the inputs (i.e. depend only on T and $\mathcal{P}$), and thus can be verified without additional communication or queries. We do so, by showing a d-round $\mathsf{AM}\text{-}\mathsf{HIP}$ protocol that gets a claim on $\mathcal{P}(x)$, and by using queries to $\mathsf{T}(x)$, converts it to multiple claims that don't depend on the input. In particular, the t-th claim will be of the form

$$\bigoplus_{j_1,\dots,j_d\in[n]} \left(\hat{\beta}_t(i_1,\dots,i_d)\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\dots,j_d)\right) = b'_t,$$

where $\hat{\beta}_t$ is a function that depends only on T.

The protocol consists of d rounds, and is loosely inspired by the sumcheck protocol [LFKN92]. At each round, we use the $\mathsf{AM}[2]$-HIP protocol for L_{IP} from Lemma 3.2, to eliminate a single input variable. That is, after the i-th round, we are left with claims the depend on only $d-i$ input variables, where the t-th claim is of the form

$$\bigoplus_{j_1,\dots,j_d\in[n]} \hat{\beta}^{(i)}_t(j_1,\dots,j_i)\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\dots,j_d) = b^{(i)}_t$$

In addition, we will also use the linearity of T to merge claims at the beginning of each round. It will be important in order to keep the same number of claims at the end of each round, and in particular, avoiding an exponential blowup.

Proof. Denote $d' = \max\{100, d\}$. Let (P', V') be the $\mathsf{AM}[2]$-HIP protocol for L_{IP} that is obtained from Lemma 3.2 after reducing the soundness error and the completeness error to d'^{-2} using $O(\log d)$ *parallel* repetitions (see [Gol98, Appendix C.1]). Note that by Lemma 3.2, (P', V') has $r_0 \overset{def}{=} O(\log n\cdot \log d)$ randomness complexity, $c_0 \overset{\text{def}}{=} \log(S)\cdot\log(d)\cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ communication complexity and $q_0 \overset{\text{def}}{=} \log(S)\cdot\log(d)\cdot\widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ query complexity.

Also, denote $\ell_{\text{merge}} \overset{def}{=} \lceil 2\cdot \log d'\rceil$, and $\ell \overset{def}{=} q_0\cdot \ell_{\text{merge}}$. Throughout the protocol, ℓ will be the maximal number of claims at the end and at the beginning of each round, and ℓ_{merge} will be the number of claims after we merge claims. The i-th round of the protocol proceeds as follows:

Round i: The round starts with at most ℓ claims of the form

$$\left\{\bigoplus_{j_1,\dots,j_d\in[n]} \left(\hat{\beta}^{(i-1)}_t(j_1,\dots,j_{i-1})\cdot x_{j_i}\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\dots,j_d)\right) = b^{(i-1)}_t\right\}_{t\in[\ell]},$$

where $\hat{\beta}^{(i-1)}_t$ doesn't depend on the input, for any $t\in[\ell]$. The round proceeds in three steps:

1. **Reducing the number of claims:** First, we reduce the number of claims from ℓ to ℓ_{merge}. The verifier chooses at random ℓ_{merge} subsets $\left\{S_t \subseteq [\ell]\right\}_{t\in[\ell_{\text{merge}}]}$, and for each S_t, takes the XOR of the claims that corresponds to the indices of S_t. Thus, for each $t\in[\ell_{\text{merge}}]$, the verifier gets the following new claim

$$\bigoplus_{k\in S_t}\left(\bigoplus_{j_1,\dots,j_d\in[n]} \left(\hat{\beta}^{(i-1)}_k(j_1,\dots,j_{i-1})\cdot x_{j_i}\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\dots,j_d)\right)\right) = \bigoplus_{k\in S_t} b^{(i-1)}_t,$$

or equivalently,

$$\bigoplus_{j_1,\dots,j_d\in[n]}\left(\bigoplus_{k\in S_t}\left(\hat{\beta}^{(i-1)}_k(j_1,\dots,j_{i-1})\right)\cdot x_{j_i}\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\dots,j_d)\right) = \bigoplus_{k\in S_t} b^{(i-1)}_t.$$

By defining $\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1}) \overset{def}{=} \bigoplus_{k\in S_t}\left(\hat{\beta}_k^{(i-1)}(j_1,\ldots,j_{i-1})\right)$, and $b_t'^{(i-1)} = \bigoplus_{k\in S_t} b_k^{(i-1)}$, we get the following ℓ_{merge} claims:

$$\left\{\bigoplus_{j_1,\ldots,j_d\in[n]}\left(\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1})\cdot x_{j_i}\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right) = b_t'^{(i-1)}\right\}_{t\in[\ell_{\text{merge}}]},$$

where $\hat{\gamma}_t^{(i-1)}$ doesn't depend on the input variables. Note, that if we start with true claims then all the new ℓ_{merge} claims that are obtained by selecting the random subsets are true with probability 1, and if we start with at least one false claim, then the probability that all the new claims are true is $2^{-\ell_{\text{merge}}}$. Thus, we reduce the number of claims to ℓ_{merge} by adding $2^{-\ell_{\text{merge}}}$ error. Our next step is to convert these claims to claims that depend on only $d-i$ input variables.

2. **Rephrasing claims as an inner product:** By changing the order in the summation, we can convert our claims into the following form:

$$\left\{\bigoplus_{j_i\in[n]} x_{j_i}\cdot\left(\bigoplus_{\substack{j_1,\ldots,j_{i-1}\in[n],\\ j_{i+1},\ldots,j_d\in[n]}}\left(\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1})\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d)\right)\right) = b_t'^{(i-1)}\right\}_{t\in[\ell_{\text{merge}}]}.$$

Observe, that the left-hand side of each claim is actually an inner product over $\mathbb{GF}(2)$. Therefore, for any $t\in[\ell_{\text{merge}}]$, the t-th claim can be rephrased as the following claim:

$$\left\langle x, \bigoplus_{\substack{j_1,\ldots,j_{i-1}\in[n],\\ j_{i+1},\ldots,j_d\in[n]}}\left(\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1})\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_{i-1},*,j_{i+1},\ldots,j_d)\right)\right\rangle = b_t'^{(i-1)}$$

3. **Eliminating one variable using $(\mathbf{P}',\mathbf{V}')$:** Next, the parties run (P',V') on each of the ℓ_{merge} claims in parallel. However, rather than making the queries, the verifier asks the prover for their values. Since the prover may be malicious, each query becomes a claim that should be verified. For each $t\in[\ell_{\text{merge}}]$, if V' rejects in the t-th interaction, then the verifier rejects. Otherwise, the verifier gets q_0 claims on the encoding of x by T, and additional q_0 claims on the encoding of truth table of the function

$$\bigoplus_{\substack{j_1,\ldots,j_{i-1}\in[n],\\ j_{i+1},\ldots,j_d\in[n]}}\left(\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1})\cdot x_{j_{i+1}}\cdots x_{j_d}\cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_{i-1},*,j_i,\ldots,j_d)\right).$$

Note that we have in total $\ell' = q_0\cdot\ell_{\text{merge}}$ claims of each type. The claims on the encoding of x can be verified at the end of the protocol by just querying $\mathsf{T}(x)$. For the second type of the claims, observe that since T is a linear code, the encoding of T is a linear combination over $\mathbb{GF}(2)$. Thus, there are Boolean coefficients $\left\{\gamma_{t,z}(j_i)\right\}_{t\in[\ell_{\text{merge}}],z\in[q_0],j_i\in[n]}$ that depend only on T, such that these $q_0\cdot\ell$ claims can be rewritten as:

$$\left\{ \bigoplus_{j_i \in [n]} \gamma_{t,z}(j_i) \cdot \bigoplus_{\substack{j_1,\ldots,j_{i-1} \in [n], \\ j_{i+1},\ldots,j_d \in [n]}} \left(\hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1}) \cdot x_{j_{i+1}} \cdots x_{j_d} \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d) \right) = b''_{t,z} \right\}_{\substack{t \in [\ell_{\text{merge}}] \\ z \in [q_0]}} .$$

By changing the order of the summation, we get:

$$\left\{ \bigoplus_{j_1,\ldots j_d \in [n]} \left(\gamma_{t,z}(j_i) \cdot \hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1}) \cdot x_{j_{i+1}} \cdots x_{j_d} \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d) \right) = b''_{t,z} \right\}_{\substack{t \in [\ell_{\text{merge}}] \\ z \in [q_0]}} .$$

Finally, by defining $\hat{\beta}_{z,t}^{(i)}(j_1,\ldots,j_i) = \gamma_{t,z}(j_i) \cdot \hat{\gamma}_t^{(i-1)}(j_1,\ldots,j_{i-1})$ we get the following $\ell = q_0 \cdot \ell_{\text{merge}}$ claims:

$$\left\{ \bigoplus_{j_1,\ldots j_d \in [n]} \left(\hat{\beta}_{t,z}^{(i)}(j_1,\ldots,j_i) \cdot x_{j_{i+1}} \cdots x_{j_d} \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d) \right) = b'_{t,z} \right\}_{\substack{t \in [\ell_{\text{merge}}] \\ z \in [q_0]}} ,$$

where the $\hat{\beta}_{t,z}^{(i)}(j_1,\ldots,j_i)$ don't depend on the input variables. Note that we converted at most $\ell = q_0 \cdot \ell_{\text{merge}}$ claims that depend on $d-(i-1)$ input variables, to the same number of claims, but that depend on only $d-i$ input variables, as required.

At the end of the d-th round: After the final (d-th) round, the verifier is left with two types of claims. Claims on $\mathsf{T}(x)$ from the previous d-round, which can can be verified by simply making queries to $\mathsf{T}(x)$, and the following claims:

$$\left\{ \bigoplus_{j_1,\ldots j_d \in [n]} \hat{\beta}_t^{(d)}(j_1,\ldots,j_d) \cdot \mathsf{Coef}_{\mathcal{P}}(j_1,\ldots,j_d) = b_t^{(d)} \right\}_{t \in [\ell]} .$$

Each of these claims doesn't depend on the input, and in particular, depends only on T and the structure of $\mathcal{P}$. Therefore, the verifier can verify these claims easily on its own, without making any other queries or communication with the prover.

Completeness. Recall that $d' = \max\{100, d\}, \ell_{\text{merge}} = \lceil 2 \cdot \log d' \rceil, \ell = q_0 \cdot \ell_{\text{merge}}$ and the completeness and the soundness errors of (P', V') are at most d'^{-2}. Since the random subsets technique for reducing the number of claims doesn't convert true claims into false claims, then, for each round that starts with true claims, the probability that at the end of the protocol we get at least one false claim is bounded by the completeness error of (P', V') multiplied by the number of claims it verifies in the round. Thus, a round converts true claims into false claims with probability of at most $d'^{-2} \cdot \ell_{\text{merge}}$. By the union bound over the d rounds, we get that the completeness error is at most

$$d \cdot d'^{-2} \cdot \ell_{\text{merge}} = \frac{d}{d'^2} \cdot \lceil 2 \cdot \log d' \rceil \leq \frac{\lceil 2 \cdot \log d' \rceil}{d'} < \frac{1}{3}.$$

Soundness. Suppose we start with a false claim. By the construction of the protocol, in order to make the verifier accept, there should be at least one round which starts with at least one false claim, but ends with only true claims. Since the soundness error of (P', V') is at most d'^{-2}, and the random subsets technique for reducing the number of claims, introduces $2^{-\ell_{\text{merge}}}$ error, then for each round that starts with at least one false claim, the probability there exists a prover strategy such that at the end of the round we get only true claims is at most $d'^{-2} \cdot \ell + 2^{-\ell_{\text{merge}}}$. Thus, by the union bound over the d rounds, we get that the soundness error is at most

$$d \cdot (d'^{-2} \cdot \ell_{\text{merge}} + 2^{-\ell_{\text{merge}}}) = \frac{d}{d'^{-2}} \cdot \lceil 2 \log d' \rceil + d \cdot 2^{-\lceil 2 \log d' \rceil} \leq \frac{\lceil 2 \cdot \log d' \rceil}{d'} + \frac{1}{d'} < \frac{1}{3}.$$

Complexity. At the beginning of each of the d rounds, the verifier sends the description of ℓ_{merge} subsets of $[\ell]$, and thus needs to toss and send $\ell \cdot \ell_{\text{merge}} = q_0 \cdot \ell_{\text{merge}}^2$ random coins. In addition, at each round (P', V') is run ℓ_{merge} times, and thus requires additional $\ell_{\text{merge}} \cdot r_0$ random coins. To sum up, the total randomness complexity of the entire protocol is

$$d \cdot (q_0 \cdot \ell_{\text{merge}}^2 + r_0 \cdot \ell_{\text{merge}}) = \log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right).$$

In addition, by the construction of the protocol, each round produces $q_0 \cdot \ell$ queries to $\mathsf{T}(x)$, and thus the query complexity is

$$d \cdot q_0 \cdot \ell = \log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right)..$$

Lastly, regarding the proof length. At each round the protocol (P', V') is run ℓ times, and for each run the prover sends also q Boolean claims. Therefore, the prover sends at each round $c_0 + q_0 \cdot \ell$ bits, and thus the total proof length is

$$d \cdot (c_0 + q_0 \cdot \ell) = \log(S) \cdot \widetilde{O}\left(\frac{d}{\epsilon^3}\right).$$

■

4.3 Protocols for $\mathsf{AC}^0(\oplus)$

In their celebrated works, Razborov and Smolensky [Raz87, Smo87] showed a general technique to approximate $\mathsf{AC}^0(\oplus)$ circuit, by a distribution of randomized low degree polynomials, in the sense that for any input, with high probability, a random polynomial agrees with the circuit on the output.

Lemma 4.2. (Circuit Approximation by polynomials). *For any parameter $\epsilon > 0$, there exists a probabilistic algorithm that takes as input an $\mathsf{AC}^0(\oplus)$ Boolean circuit $C : \{0,1\}^{2n} \to \{0,1\}$ of size S and depth d. It uses $O\left(\log\left(\frac{1}{\epsilon}\right) \cdot \log^2(S)\right)$ random bits, and outputs a polynomial $\mathcal{P}$ of total degree $O\left(\left(\log\left(\frac{1}{\epsilon}\right) + \log S\right)^d\right)$, such that for every $x, y \in \{0,1\}^n$:*

$$\Pr\Big[\mathcal{P}(x,y) = C(x,y)\Big] \geq 1 - \epsilon.$$

For sake of completeness, we give the proof of Lemma 4.2 in Appendix B.

Thus, the result of Theorem 8 can be extended also for every problem that is decidable by an $\mathsf{AC}^0(\oplus)$ circuit.

Corollary 10. *Fix an integer n, and let $\epsilon \in (0, 1/6]$. Suppose that for any k there exists a linear transformation $\mathsf{T} : \{0,1\}^{2k} \to \{0,1\}^{\ell(k)}$ and a $\mathsf{DNF} \circ \mathsf{T}$ circuit C of size $S_0(2k)$ that computes the function $\mathsf{IP}_2(x,y)$ on at least a $\frac{5}{6} + \epsilon$ of the k-bit length inputs. Then, there exists a constant c such that for any function $f : \{0,1\}^{2n} \to \{0,1\}$ that is computed by an $\mathsf{AC}^0(\oplus)$ circuit of size S and depth $d \geq 2$, there exists an $\mathsf{AM}[2]$-CC protocol for f with $\log\left(S_0(2^{(c \cdot \log n \cdot \log^d S)})\right) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ communication complexity.*

Remark 4.3. *We note that we could have use Lemma 4.2 also in order to get a streaming protocol for* $\mathsf{AC}^0(\oplus)$ *circuits. However, this yields a protocol with too many rounds, which can also be derived by the GKR protocol [GKR08].*

5 The $\bigoplus$Triangle Problem

In the Triangle-Count problem, a streaming algorithm is required to count (or sometimes just approximate) the number of triangles (i.e. cliques with three vertices) in an undirected (simple) graph $G = (V, E)$. We focus on a setting where the stream consists of the edges in the graph, where each edge appears in the stream exactly once. We consider a variant of the Triangle-Count problem, where the goal is to compute the *parity* of the number of triangles.

Definition 5.1. *Let* $G = (V, E)$ *be an undirected simple graph such that* $V \subseteq [n]$ *and* $E \subseteq [n] \times [n]$*. In the* $\bigoplus$Triangle *problem, the edges in* E *are given as a stream in some arbitrary order, where each edge appears in the stream exactly once. The goal is to output the parity of the number of triangles (i.e. cliques of size* 3*) in* G*.*

We show that the existence of a sufficiently small $\mathsf{DNF} \circ \mathsf{T}$ circuit that approximates the inner product function, yields an efficient $\mathsf{AM}[2]$-DS protocol for $\bigoplus$Triangle.

Theorem 11. *Fix an integer* n *and a parameter* $\epsilon \in (0, 1/6]$*. Let* $\mathsf{T} : \{0,1\}^{n^3} \to \{0,1\}^{n'}$ *be a linear code. Suppose there exists a* $\mathsf{DNF} \circ \mathsf{T}$ *circuit* C *of size* S *that computes the function* $\mathsf{IP}_2(x, y)$ *on at least a* $\frac{5}{6} + \epsilon$ *fraction of the* n^3*-bit length inputs. Then, there exists an* $\mathsf{AM}[2]$-DS *protocol for* $\bigoplus$Triangle *with* $O(\log n)$ *randomness, and* $\log^3(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ *verifier space complexity and proof length.*

Proof. Consider the following degree 3 polynomial $\mathcal{P}_{\oplus\textsc{Tri}} : \{0,1\}^{n^2} \to \{0,1\}$:

$$\mathcal{P}_{\oplus\textsc{Tri}}(I_{e_1}, \ldots, I_{e_{n^2}}) = \bigoplus_{v<u<w\in[n]} I_{(v,u)} \cdot I_{(u,w)} \cdot I_{(w,v)}, \tag{10}$$

where I_{e_i} is an indicator variable which represents whether the edge e_i appears in G. Observe that by the definition of a clique, any distinct vertices $v < u < w \in [n]$ induce a triangle in G if and only if $I_{(v,u)} \cdot I_{(u,w)} \cdot I_{(w,v)} = 1$. Thus, $\mathcal{P}_{\oplus\textsc{Tri}}(I_{e_1}, \ldots, I_{e_{n^2}})$ is equal to the parity of the number of triangles in G. Since $\mathcal{P}_{\oplus\textsc{Tri}}$ is a 3 degree polynomial, then by Lemma 4.1, there exists an $\mathsf{AM}[6]$-HIP protocol for computing $\mathcal{P}_{\oplus\textsc{Tri}}$ with $\log(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ proof length and communication complexity. However, this HIP has 3 rounds, while we seek to have a one-round protocol.

To obtain a one-round protocol, we first rely on the classical round collapsing result of Babai and Moran [BM88] for public-coin interactive proofs. We remark that their result also holds for public coin *holographic* interactive proofs, as their transformation preserves the holographic property of the proof-system (see also [RVW13, Lemma 4.6]).

Claim 5.2. (Round collapsing, see also [RVW13, Lemma 4.6]). *Let* $k > 1$ *be a* constant. *Suppose there exists an* $\mathsf{AM}[2k]$-HIP *protocol for a language* $\mathcal{L} \subseteq \{0,1\}^n$ *with proof length* ℓ*, and query complexity* q*. Then, there exists an* $\mathsf{AM}[2]$-HIP *protocol for* $\mathcal{L}$ *with proof length* $\ell \cdot t$ *and query complexity* $q \cdot t$*, where* $t = k^{O(k)} \cdot \ell^{k-1}$*.*

Therefore, by applying Claim 5.2, we derive an $\mathsf{AM}[2]$-HIP protocol for $\bigoplus$Triangle with $\log^3(S) \cdot \widetilde{O}\left(\frac{1}{\epsilon^3}\right)$ query complexity and proof length. By applying Proposition 3.6, we reduce the randomness complexity to $O(\log n)$.

At this point we would like to apply Fact 4 to obtain a streaming protocol. A major issue that raises is that in Fact 4, the verifier needs to get access to all the indicators, whereas in

$\bigoplus$ Triangle the verifier only gets the edges that appear in G (or equivalently, only the indicators that are equal to 1). However, also recall that in Fact 4 the verifier actually uses the input only to compute linear combinations, and thus it can ignore input bits that are equal to 0. Therefore, the protocol that is obtained from Fact 4 can be easily adapted to support also the case that only the variables with the value 1 are given, as in $\bigoplus$ Triangle. ■

Remark 5.3. *The reason we start the proof of Theorem 11 with the* HIP *protocol for inner product claims, rather than starting directly with the streaming protocol of Theorem 9, is because we seek a one-round protocol. By starting with an* HIP *protocol, we can apply the round collapse theorem of [BM88] to reduce the number of rounds, and just then transform the resulting protocol into a streaming one.*

References

[ABG+14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. Candidate weak pseudorandom functions in $AC^0 \circ mod_2$. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 251–260. ACM, 2014.

[Ajt83] Miklós Ajtai. $\sum^1{}_1$-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009.

[BC17] Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BFL+06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 253–262. ACM, 2006.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking Computations in Polylogarithmic Time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31. ACM, 1991.

[BFS86] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347. IEEE Computer Society, 1986.

[BKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632. ACM/SIAM, 2002.

[BKT20] Mark Bun, Robin Kothari, and Justin Thaler. Quantum algorithms and approximating polynomials for composed functions with shared inputs, 2020.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[CCGT14] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 687–706. SIAM, 2014.

[CCM$^+$13] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. On interactivity in Arthur-Merlin communication and stream computation. *Electron. Colloquium Comput. Complex.*, 20:180, 2013.

[CCM$^+$15] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable Stream Computation and Arthur-Merlin communication. In David Zuckerman, editor, *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, volume 33 of *LIPIcs*, pages 217–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[CCMT14] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Algorithms*, 11(1):7:1–7:30, 2014.

[CGJ$^+$18] Mahdi Cheraghchi, Elena Grigorescu, Brendan Juba, Karl Wimmer, and Ning Xie. $AC^0 \circ mod_2$ lower bounds for the boolean inner product. *J. Comput. Syst. Sci.*, 97:45–59, 2018.

[CGT20] Amit Chakrabarti, Prantar Ghosh, and Justin Thaler. Streaming verification for graph problems: Optimal tradeoffs and nonlinear sketches. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1–12. IEEE, 2020.

[CR20] Lijie Chen and Hanlin Ren. Strong average-case lower bounds from non-trivial derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1327–1334. ACM, 2020.

[CS16] Gil Cohen and Igor Shinkar. The complexity of DNF of parities. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical*

Computer Science, Cambridge, MA, USA, January 14-16, 2016, pages 47–58. ACM, 2016.

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, 2011.

[CW19] Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 19:1–19:43. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[FIKK20] Yuval Filmus, Yuval Ishai, Avi Kaplan, and Guy Kindler. Limits of Preprocessing. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 17:1–17:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[FSS81] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 260–270. IEEE Computer Society, 1981.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008.

[GKZ08] Parikshit Gopalan, Adam R. Klivans, and David Zuckerman. List-decoding Reed-Muller codes over small fields. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 265–274. ACM, 2008.

[Goe15] Michel Goemans. Chernoff bounds, and some applications. http://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf, 2015.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer, 1998.

[GPW16] Mika Göös, Toniann Pitassi, and Thomas Watson. The landscape of communication complexity classes. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 86:1–86:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[GR13] Tom Gur and Ran Raz. Arthur-Merlin streaming complexity. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 528–539. Springer, 2013.

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer*

Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA, volume 67 of *LIPIcs*, pages 39:1–39:43. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GS10] Dmitry Gavinsky and Alexander A. Sherstov. A separation of NP and conp in multiparty communication complexity. *Theory Comput.*, 6(1):227–245, 2010.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986.

[Jac97] Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *J. Comput. Syst. Sci.*, 55(3):414–440, 1997.

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 710–716. Springer, 2005.

[JSP13] Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 589–597. ACM, 2013.

[Juk06] Stasys Jukna. On graph complexity. *Comb. Probab. Comput.*, 15(6):855–876, 2006.

[Kla03] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 118–134. IEEE Computer Society, 2003.

[Kla11] Hartmut Klauck. On arthur merlin games in communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 189–199. IEEE Computer Society, 2011.

[KMPT12] Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Math.*, 8(1-2):161–185, 2012.

[KMPV19] John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 119–133. ACM, 2019.

[KMSS12] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming -*

39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012.

[Lan93] Serge Lang. *Algebra (3. ed.)*. Addison-Wesley, 1993.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[MVV16] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 401–411. ACM, 2016.

[MW20] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.

[Raz87] Alexander A Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. *Math. notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Rot12] Guy N. Rothblum. How to compute under AC^0 leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 552–569. Springer, 2012.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802. ACM, 2013.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Comput. Complex.*, 15(4):298–341, 2006.

[SV12] Rocco A. Servedio and Emanuele Viola. On a special case of rigidity. *Electron. Colloquium Comput. Complex.*, 19:144, 2012.

[Tha16] Justin Thaler. Semi-streaming algorithms for annotated graph streams. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 59:1–59:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[VW20] Nikhil Vyas and R. Ryan Williams. Lower bounds against sparse symmetric functions of ACC circuits: Expanding the reach of #SAT algorithms. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Wil14] Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213. ACM, 1979.

A Missing Proofs

A.1 Homogeneous Multilinear Mapping (Proof of Claim 2.7)

We give here the proof of Claim 2.7. Recall that our goal is, for a homogeneous k-linear mapping $f : V_1 \times \cdots \times V_k \to W$ and $x^{(1)} \in V_1, \ldots, x^{(k)} \in V_k$, show that for any $r^{(1)} \in V_1, \ldots, r^{(k)} \in V_k$ it holds that:

$$f(x^{(1)}, \ldots, x^{(k)}) = \sum_{s_1,\ldots,s_k \in \{0,1\}} f\Big((-1)^{1-s_1} r^{(1)} + x^{(1)} s_1, \ldots, (-1)^{1-s_k} r^{(k)} + x^{(k)} s_k\Big).$$

Proof. (By induction on k)

For the base case ($k = 1$), the claim follows immediately from the definition of linear mapping:

$$f(x^{(1)}) = f(r^{(1)} + x^{(1)}) + f(-r^{(1)}).$$

For the inductive step, note that by the definition of homogeneous k-linear:

$$f(x^{(1)}, x^{(2)}, \ldots, x^{(k)}) = f(r^{(1)} + x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(k)}) + f(-r^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(k)})$$

By fixing the first variable, each of the two functions on the right-hand side is a homogeneous $k-1$ linear mapping, and thus by the inductive hypothesis we get that:

$$f(r^{(1)} + x^{(1)}, x^{(2)}, \ldots, x^{(k)}) =$$
$$\sum_{s_2,\ldots,s_k \in \{0,1\}} f\Big(r^{(1)} + x^{(1)}, (-1)^{1-s_2} r^{(2)} + x^{(2)} s_2, \ldots, (-1)^{1-s_k} r^{(k)} + x^{(k)} s_k\Big)$$

and,

$$f(-r^{(1)}, x^{(2)}, \ldots, x^{(k)}) =$$
$$\sum_{s_2,\ldots,s_k \in \{0,1\}} f\Big(-r^{(1)}, (-1)^{1-s_2} r^{(2)} + x^{(2)} s_2, \ldots, (-1)^{1-s_k} r^{(k)} + x^{(k)} s_k\Big).$$

As a result, by taking the addition of these two, we get that

$$f(x^{(1)}, \ldots, x^{(k)}) = \sum_{s_1,\ldots,s_k \in \{0,1\}} f\Big((-1)^{1-s_1} r^{(1)} + x^{(1)} s_1, \ldots, (-1)^{1-s_k} r^{(k)} + x^{(k)} s_k\Big).$$

■

A.2 Relationship Between HIP, DS and CC (Proofs of Facts 4 and 5)

We give here the proof of Facts 4 and 5.

Fact 4. (From AM[k]-HIP to AM[k]-DS) *Fix an input length n. Let $\mathbb{F}$ be some finite field, and let $C : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a linear code over $\mathbb{F}$. Suppose there exists an* AM[k]-HIP *protocol with respect to the code C for a language $\mathcal{L} \subseteq \mathbb{F}^n$, using q queries, ρ random bits and proof length ℓ. Then, there exists an* AM[k]-DS *protocol for $\mathcal{L}$ with $\ell + O(q \cdot \log n)$ proof length, ρ randomness complexity, and $q \cdot \lceil \log |\mathbb{F}| \rceil$ verifier space complexity, for any order of the input in the stream. Moreover, if the verifier is non-adaptive then the proof length is reduced to ℓ.*

Proof. Denote by (P, V) the $\mathsf{AM}[k]$-HIP protocol. Note that when P sends its last message, it knows to which locations V will make its queries. Thus, a similar $\mathsf{AM}[k]$-HIP protocol can be established where the prover adds to its last message the locations of the queries that the verifier will make, and the verifier first makes the queries to these locations, store their values in its memory, and only then simulates the computation of V. For each query that V makes, the verifier checks the value that has already been computed, or rejects if V tries to make a query to a location that didn't have been specified before by the prover. Note that the new protocol has the same completeness and soundness errors as the completeness and soundness errors of (P, V). Also, note the randomness complexity the query complexity didn't change as well. The only overhead is on the proof length, which consists of the original proof plus q indices of locations in the encoding of the input, and thus is increased to $\ell + O\left(q \cdot \log n\right)$. Let us denote this new protocol by (P, V'). We will show that (P, V') can be converted into an $\mathsf{AM}[k]$-DS protocol.

Let $\mathsf{Val}(x)$ be the queries values that were computed by V'. We define the value of tuple $(r, \pi, \mathsf{Val}(x))$ to be the unique output of V', when it gets an input x, uses a sequence of random coins r, and receives a proof π. We will show that for every r and π, there is an OBDD with $|\mathbb{F}|^q$ width, such that for every input x, it outputs the value of $(r, \pi, \mathsf{Val}(x))$. Observe that by doing so, we will finish the proof of Fact 4.

Let r be a sequence of coins, and π be a proof. Consider the following OBDD $f_{r,\pi}$:

- **Layers:** The OBDD consists of n layers and $|\mathbb{F}|^q$ leaves. Each leaf is associated with a single possible *distinct* value of $\{\mathsf{Val}(\mathsf{x}) \mid x \in \{0,1\}^n\}$. Note, that since there are q queries, then there are $|\mathbb{F}|^q$ distinct values for $\mathsf{Val}(\mathsf{x})$. Since V' makes queries to a linear code, each query is a linear combination of the input stream. Thus, the queries can be computed by maintaining q accumulator variables, each takes $|\mathbb{F}|$ values. Or equivalently, $|\mathbb{F}|^q$ states for each layer. At the i-th layer, the OBDD reads the i-th token in the stream and updates the state of the accumulator variables by virtually adding its value to the variables that the token form their linear combination. Note that it can be done by simply moving to the corresponding state in the $(i + 1)$=layer since the locations of the queries are fixed in the OBDD.

- **Leaves:** Finally, for any possible value $v \in \mathsf{Val}(\mathsf{x})$, we define that label of the leaf which is associated with v, to be the value of $(r, \pi, \mathsf{Val}(\mathsf{x}))$.

The correctness follows from the construction and from the completeness and soundness errors of (P', V').

The moreover part of Fact 4 follows from the fact that if the original $\mathsf{AM}[k]$-HIP verifier is non-adaptive, then the locations of the queries depend only on the transcript of the original protocol. Hence, P' doesn't need to attach to the original proof also the indices of the queries, and thus the proof length reduces to ℓ.

■

Fact 5. (From $\mathsf{AM}[k]$-DS to $\mathsf{AM}[k]$-CC). *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, and let $\mathcal{L}_f = \{(x, y) \mid f(x, y) = 1\}$. Suppose there exists an $\mathsf{AM}[k]$-Data Streaming protocol for $\mathcal{L}_f$, where x precedes y in the input stream, with proof length ℓ, randomness complexity ρ and verifier space complexity s. Then, there exists an $\mathsf{AM}[k]$-CC protocol for f with $(\ell + s + \rho)$ communication complexity, where Alice gets x and Bob gets y.*

Proof. The idea is to have Alice and Bob simulate the streaming protocol. Alice simulates the first part of the computation of the corresponding OBDD, which depends only on the input x. Bob continues the simulation from the place Alice has stopped, and he simulates the second part which depends only on the input y.

By the definition of the $\mathsf{AM}[k]$-DS, in each OBDD in the protocol, the nodes in the first n layers test bits in x, and the nodes in the last n layers test bits in y. Thus, Alice knows the bits that are tested in the first n layers, and Bob knows the bits that are tested in the rest n layers. Therefore, Alice and Bob can simulate the $\mathsf{AM}[k]$-DS protocol as follows:

Protocol: Let $x, y \in \{0,1\}^n$, and let r be a sequence of random coins. Denote by π the proof of a honest prover in the $\mathsf{AM}[k]$-DS protocol when getting the input (x, y) and tossing the coins r. Let $f_{r,\pi}$ be the corresponding OBDD in the $\mathsf{AM}[k]$-DS protocol. The $\mathsf{AM}[k]$-CC protocol proceeds as follows:

1. **Prover:** Let $\mathcal{S}$ be the state at which $f_{r,\pi}(x, y)$ reaches at the $n+1$ layer. The prover sends to Alice and Bob the original proof π, and the state $\mathcal{S}$.
2. **Alice:** Simulates the computation of the OBDD $f_{r,\pi}$, from the beginning and until she reaches to the $|n|+1$ layer. If she stopped at the state $\mathcal{S}$, she accepts. Otherwise, she rejects.
3. **Bob:** Simulates the computation of the OBDD $f_{r,\pi}$, but by starting from the state $\mathcal{S}$ at the $|n|+1$ layer, and continuing until the end of the computation. Bob accepts if and only of the OBDD outputs 1.

Completeness, soundness and complexity. Note that the computations of the OBDD in the first n layers depend only on Alice's local input, and the computations in the rest n layer depend only on Bob's local input. Thus, the protocol is well defined. The completeness and the soundness of the protocol follows immediately from those of the original streaming protocol.

Regarding the communication complexity, Alice and Bob send ρ random coins to the prover, which responds with the proof of the original streaming protocol and an OBDD state. From the proof length and the space complexity of the original streaming protocol, sending the original proof takes at most l bits, and sending the state takes at most s bits. Therefore, the total communication complexity is $\ell + s + \rho$. ■

A.3 Reducing Randomness in HIPs (Proof of Proposition 3.6)

In this section we prove Proposition 3.6. The proof is based on the technique due to Newman [New91], which showed a similar result in the context of (standard) public-coin Communication Complexity protocols.

Proposition 3.6. (Reducing randomness). *Fix an input length n, and let $\mathcal{L} \subseteq \{0,1\}^n$. For any $\mathsf{AM}[2]$-HIP protocol for $\mathcal{L}$ with proof length c and query complexity q, there exists an $\mathsf{AM}[2]$-HIP protocol for $\mathcal{L}$ with proof length $O(c)$, query complexity $O(q)$, and randomness complexity $O(\log n)$.*

Proof. First, using $O(1)$ parallel repetitions, we reduce the completeness and the soundness errors of the $\mathsf{AM}[2]$-HIP protocol to a sufficiently small constant, say 0.001. Let V be the resulting protocol, and let D be the multiset from which the verifier tosses *uniformly* its random messages. Also, let us denote $t \overset{\text{def}}{=} O(n)$, the exact value will be chosen later. For any (multiset) subset $S \subseteq D$ of size t, we denote by V_S the protocol that is obtained by restricting V to choose messages only from the subset S (uniformly). We will show using the probabilistic method, that with non-zero probability, there is a subset S of size t, such that the completeness and the soundness errors of V_S are at most $\frac{1}{3}$. Note, that choosing a message at random from S requires only $\lceil \log t \rceil = O(\log(n))$ randomness complexity, and thus it will finish the proof.

Let $S \subseteq D$ be a random (multiset) subset of size t. For any $i \in [t]$ and $x \in \{0,1\}^n$, we define $I_i(S,x)$ to be an indicator that is equal to 1 if and only if V_S failed[13] to verify whether $x \in \mathcal{L}$, when it selects the i-th message from S. Note that $I_i(S,x)$ is a random variable with a probability over the choice of S. Also, note that by the definition of V_S, for any S, the completeness and the soundness errors of V_S are bounded by the maximal value of $\frac{1}{t}\sum\limits_{i\in[t]} I_i(S,x)$, where the maximum is taken over all the inputs $x \in \{0,1\}^n$. By the completeness and the soundness of V, for any i it holds that $\mathbf{E}\left[I_i(S,x)\right] \le 0.001$. Therefore, using the Chernoff bound (see Lemma 2.11), we get that:

$$\Pr_S\left[\frac{1}{t}\sum_{i\in S} I_i(S,x) \ge \frac{1}{3}\right] \le e^{-t}.$$

By union bounding over all the inputs, we get that

$$\Pr_S\left[\exists x \in \{0,1\}^n \ s.t \ \frac{1}{t}\sum_{i\in S} I_i(S,x) \ge \frac{1}{3}\right] \le 2^{n-t}.$$

The proof follows by selecting $t = O(\log n)$. ∎

B $\mathsf{AC}^0(\oplus)$ Polynomial Approximation

Lemma 4.2. (Circuit Approximation by polynomials). *For any parameter $\epsilon > 0$, there exists a probabilistic algorithm that takes as input an $\mathsf{AC}^0(\oplus)$ Boolean circuit $C : \{0,1\}^{2n} \to \{0,1\}$ of size S and depth d. It uses $O\left(\log\left(\frac{1}{\epsilon}\right) \cdot \log^2(S)\right)$ random bits, and outputs a polynomial $\mathcal{P}$ of total degree $O\left(\left(\log\left(\frac{1}{\epsilon}\right) + \log S\right)^d\right)$, such that for every $x, y \in \{0,1\}^n$:*

$$\Pr\Big[\mathcal{P}(x,y) = C(x,y)\Big] \ge 1 - \epsilon.$$

Proof. Let $C : \{0.1\}^{2n} \to \{0,1\}$ be a depth d Boolean circuit of size S. Denote by $w \le S$ the fan-in of the largest gate in C (i.e. the number of inputs to the gate). Consider the following algorithm:

1. First, convert $C : \{0,1\}^{2n} \to \{0,1\}$ to an equivalent circuit $C_1 : \{0,1\}^{2n} \to \{0,1\}$ such that C_1 has only $\vee$ and $\oplus$ gates which all have the same w fan-in, as follows:

   (a) Replace any $\wedge$ gate with $\vee$ and $\neg$ gates by using De-Morgan's law.

   (b) Replace any $\neg b$ (where b is the gate's input) by the gate $(b \oplus 1)$.

   (c) For any gate ($\oplus$ or $\vee$) with t inputs, add additional $m - t$ zeros (constants) to fix the fan-in to w.

   Observe that the aforementioned transformations introduce zero error. Moreover, the depth may increase up to three times. Yet, since we will care only on the number of $\vee$ gates, we note that any path from an input to an output node still goes through at most d $\vee$ gates.

[13] Since V_S is one round protocol, the indicator variable is well defined: for $x \in \mathcal{L}$, a failure when choosing the i-th message from S is defined to be the case where there is not any proof that makes the verifier accept; and when $x \neq \mathcal{L}$, a failure is defined to be the case where there is at least one proof that makes the verifier accept.

2. Next, approximate C_1 by a polynomial $\mathcal{P} : \{0,1\}^{2n} \to \{0,1\}$, by converting $\oplus$ gates and $\vee$ gates to arithmetic operations over $\mathbb{GF}(2)$, as follows:

   (a) Replace any $\bigoplus(b_1, \ldots, b_w)$ gate by the addition operation $(b_1 + \cdots + b_w)$ over $\mathbb{GF}(2)$.

   (b) For some *constant* $0 < \delta < 1$, let $E : \{0,1\}^w \to \{0,1\}^{\text{poly}(w)}$ be a *linear* error-correcting code with $1 - \delta$ relative distance. Let us denote by m the number of $\vee$ gates in C_1 (note that $m \le S$), and define $l = O\left(\log_{\frac{1}{\delta}}\left(\frac{1}{\epsilon}\right) + \log_{\frac{1}{\delta}}(m)\right)$ to be a fixed integer such that:
   $$(\delta)^l \le \frac{\epsilon}{m}.$$
   Then, choose l indices $i_1, \ldots, i_l$ in E at random, and replace any $\bigvee(b_1, \ldots, b_w)$ gate, with the following $\eta(b_1, \ldots, b_w)$ arithmetic operation:
   $$\eta(b_1, \ldots, b_w) \equiv 1 - \prod_{j=1}^{l}\Big(1 - E(b_1, \ldots, b_w)_{i_j}\Big).$$

Let us show that $\mathcal{P}$ satisfies the requirements of Lemma 4.2:

- **Randomness.** The only randomized part is the tossing of l indices in the codewords of E. Thus, the total randomness is:
  $$l \cdot \log\Big(\text{poly}(w)\Big) = O\left(\left(\log_{\frac{1}{\delta}}\left(\frac{1}{\epsilon}\right) + \log_{\frac{1}{\delta}}(m)\right) \cdot \log w\right) = O\Big(\log\left(\frac{1}{\epsilon}\right) \cdot \log^2(S)\Big),$$
  where the last is due to the fact that $w, m \le S$.

- **Degree.** First, observe that addition operations don't increase the degree of $\mathcal{P}$. Moreover, since E is a linear code, it can be computed by a polynomial with total degree 1, and therefore the operation η can be computed by a polynomial with total degree l. Furthermore, as mentioned before, any path from an input to an output node in C_1 goes through at most d $\vee$ gates, and hence goes through at most d η operations in the corresponding polynomial $\mathcal{P}$. Therefore, the total degree of the polynomial is $l^d = O\left(\left(\log\left(\frac{1}{\epsilon}\right) + \log S\right)^d\right)$.

- **Correctness.** Note that C_1 is equivalent to C, and that converting $\oplus$ gates by the addition operations introduces zero error. Hence, the only error can be due to a wrong result in an η operation. Let $\vee_1, \ldots, \vee_m$ be some topology sorting of $\vee$ gates[14] in C_1, and $\eta_1, \ldots, \eta_m$ be their corresponding η operations in $\mathcal{P}$. For the i-th gate, define A_i to be the event that $\vee_i$ and η_i output the same result. Thus,
  $$\Pr\Big[\mathcal{P}(x,y) = C(x,y)\Big] \ge \Pr\big[A_1\big] \cdot \Pr\big[A_2 \mid A_1\big] \cdot \Pr\big[A_3 \mid A_1 \wedge A_2\big] \cdots \Pr\big[A_m \mid A_1 \wedge \cdots \wedge A_{m-1}\big]$$
  For an i-th gate, the probability that η_i and $\vee_i$ output the same bit, given that the previous gates in the topology sorting also output the same bit, is at least $1 - \delta^l$. That is because if the inputs to the $\vee$ gate are all zeros ($\bigvee = 0$), then E outputs the zero codeword and therefore η outputs 0 as required. On the other hand, if there is at least one input bit that is equal to one ($\bigvee = 1$), then E outputs a codeword with at least $1 - \delta$ non-zero bits.

[14] I.e., for any $\vee_i$ and $\vee_j$, if $\vee_j$ is a descendant of $\vee_i$ then $i < j$.

The probability that η outputs zero (instead of 1) is the probability that the bits in all the chosen indices are zero, which is equal to δ^l. As a result we get that:

$$\Pr\left[A_1\right] = \Pr\left[A_2 \mid A_1\right] = \cdots = \Pr\left[A_m \mid A_1 \wedge \cdots \wedge A_{m-1}\right] \geq 1 - (\delta)^l.$$

And hence,

$$\Pr\left[\mathcal{P}(x,y) = C(x,y)\right] \geq \left(1 - \delta^l\right)^m \geq \left(1 - \frac{\epsilon}{m}\right)^m \geq 1 - \epsilon.$$

$\blacksquare$