Interactive Proofs of Proximity: Delegating Computation in Sublinear Time

Guy N. Rothblum*
Microsoft Research, Silicon Valley

Salil Vadhan†
Harvard University

Avi Wigderson‡
Institute for Advanced Study

Abstract

We initiate the study of interactive proofs with sublinear-time verifiers. These proof systems can be used to ensure approximate correctness for the results of computations delegated to an untrusted server. Following the literature on property testing, we seek proof systems where with high probability the verifier accepts every input in the language, and rejects every input that is *far* from the language. The verifier's query complexity (and computation complexity), as well as the communication, should all be sublinear. We call such a proof system an *Interactive Proof of Proximity* (IPP).

- On the positive side, our main result is that all languages in $\mathcal{NC}$ have Interactive Proofs of Proximity with roughly $\sqrt{n}$ query and communication and complexities, and polylog(n) communication rounds.

  This is achieved by identifying a natural language, membership in an affine subspace (for a structured class of subspaces), that is complete for constructing interactive proofs of proximity, and providing efficient protocols for it. In building an IPP for this complete language, we show a tradeoff between the query and communication complexity and the number of rounds. For example, we give a 2-round protocol with roughly $n^{3/4}$ queries and communication.

- On the negative side, we show that there exist natural languages in $\mathcal{NC}^1$, for which the sum of queries and communication in any constant-round interactive proof of proximity must be polynomially related to n. In particular, for any 2-round protocol, the sum of queries and communication must be at least $\tilde{\Omega}(\sqrt{n})$.

- Finally, we construct much better IPPs for specific functions, such as bipartiteness on random or well-mixing graphs, and the majority function. The query complexities of these protocols are provably better (by exponential or polynomial factors) than what is possible in the standard property testing model, i.e. without a prover.

*Email: rothblum@alum.mit.edu.

†Center for Research on Computation and Society (CRCS) and School of Engineering and Applied Sciences (SEAS), Harvard University, Cambridge, MA USA. Work done in part while on leave as a Visiting Researcher at Microsoft Research SVC and a Visiting Scholar at Stanford University. Also supported by NSF grant CNS-1237235. E-mail: salil@seas.harvard.edu.

‡Research partially supported by NSF grant CCF-0832797. Email: avi@ias.edu.

1 Introduction

The power of efficiently verifiable proof systems is a central question in the study of computation. In this work, we study the power, and the limitations, of interactive proof systems with very efficient *sublinear time* verifiers.

An interactive proof system [GMR89, BM88] is an interactive protocol between a prover and a randomized verifier, in which the prover convinces the verifier of the validity of a computational statement. The computational statement is usually the membership of an input x, known both to the prover and to the verifier, in a language L. The famous $\mathcal{IP} = \mathcal{PSPACE}$ Theorem [LFKN92, Sha92] established that interactive proofs with *polynomial-time* verifiers are remarkably powerful: they can be used to prove any language/statement computable in polynomial space. In this work, we follow the quest for efficient proof verification to its extreme. We ask:

Which computational statements have an interactive proof that can be verified in sublinear time?

Since the verifier runs in sublinear time, it cannot even read the input in its entirety. Following work on sublinear time algorithms and property testing [RS96, GGR98], we consider verifiers that have query access to the input: the input x is treated as an oracle, and on query i the verifier receives $x[i]$. Our goal is sublinear verifier running time, but we also consider the *query complexity*, the *communication complexity*, and the *round complexity* of the protocol, as well as the running time of the honest prover: in different settings, queries, communication, and rounds of interaction may come at different costs. We emphasize that, as in the standard interactive proof model, *the verifier's input is reliable* and fixed throughout the protocol, whereas *the prover is unreliable and untrusted*, and might deviate from the protocol in an adaptive and arbitrary manner. We find this to be the most natural model for sublinear-time proof verification, but we also provide some motivating applications below. Finally, throughout this work we consider computationally unbounded cheating provers, and make no cryptographic assumptions.

On reflection, it quickly becomes apparent that there exist simple and natural languages, for which membership cannot be verified in sublinear time. For example, there is no way for a prover to convince the verifier that the parity of a string is 0, unless the verifier reads the entire string. Thus, drawing further inspiration from property testing, we relax the requirement: the verifier should still accept inputs in the language, but it is only required to reject (with high probability) inputs that are *far from the language*, say at (fractional) Hamming distance at least $\varepsilon \in (0, 1)$ from every input in the language.[1] We call such a protocol an *Interactive Proof of ε-Proximity*. We define and study a new complexity class: $\mathcal{IPP}$, the class of languages that have interactive proofs of proximity with sublinear-time verification. See Section 2 for formal definitions of these proof systems and the complexity class $\mathcal{IPP}$ (parameterized with the distance, query, communication and round complexities). For clarity, throughout the introduction we often ignore many of the proof system parameters. In particular, we refer loosely to the complexity class $\mathcal{IPP}$ as the class of languages that have Interactive Proofs of ε-Proximity, with sublinear query and communication complexities, for some non-trivial ε.

A Motivating Application: Delegating Computation in Sublinear Time. In addition to being a foundational issue in the theory of computation, proof verification is also motivated by

[1] More generally, one could think of other distance measures or norms. We will (mostly) focus our attention on fractional Hamming distance throughout this work.

real-world applications, such as delegating computation: a powerful server can run a computation for a weaker client, and provide an interactive proof of the output's correctness [GKR08]. The study of Interactive Proofs of Proximity with sublinear time verification is also motivated by such applications: namely, *delegating computation in sublinear time*. Consider a client that has reliable query access to an input x, but is very computationally restricted: it can only run in sublinear time. Still, this client wants to approximate a function f on x. A more powerful server can also access x, and can run in polynomial time, but it might cheat. The server can compute $y = f(x)$ and send it to the client. The client receives some (potentially incorrect) output y', and can use a sublinear time Interactive Proof of Proximity to verify that y' is "not too far" from $f(x)$. By "not too far" here, we mean that there exists some x' at Hamming distance ε from x s.t. $y' = f(x')$. Thus, even a sublinear time client can reliably delegate its computations, and verify the approximate accuracy of answers.

We emphasize again that we assume that the client's access to the input x is reliable: x is fixed in advance, or at the very least x does not change adaptively depending on the client's random coins. On the other hand, we assume nothing about the untrusted prover's behavior: it might cheat arbitrarily and adaptively, and it is not computationally bounded. The assumption that x is reliable warrants further discussion and motivation. For example, x might be a large dataset that both the client and the server can access (e.g. the Twitter graph). Alternatively, x might be real-time data that can be reliably measured by both the client and the server, e.g. network traffic or climate data. The client might only have the resources to measure (and store) x at a few points in time.[2] Finally, we note that even if the client's input is stored on an *untrusted* server, a memory checker [BEG+94] could be combined with an Interactive Proof of Proximity. The memory checker would detect any cheating on the client's queries to the input. Known constructions of memory checkers would maintain information-theoretic security, at only a sublinear overhead in terms of client-server communication.

Comparison with Prior Work on Sublinear Time Computation and Verification. A rich body of work within the literature on sublinear-time algorithms focuses on *property testing* [RS96, GGR98]. There, a randomized tester has query access to the input, and needs to distinguish whether the input is in a language or far from the language. See also [Gol10b] and the excellent survey in [Ron09]. We extend this model by also providing interaction with a more powerful prover, who can read the input in its entirety, but might cheat. Of course, we assume more here than what is needed for standard property testing: namely, the existence of such a prover. However, we leverage this assumption to obtain more powerful and general results than what is possible in the property testing model. Most significantly, we tackle an inherent limitation of property testing: research on property testing has focused on constructing testers for languages based on their combinatorial or algebraic structure. This limitation seems inherent, because there exist simple and natural languages for which (provably) no sublinear time property testers exist (e.g. parity, see also [GKNR12]). In contrast, looking ahead, we show that IPPs allow general results for a rich class of computations. Further, we will show that even for specific problems, interaction with an untrusted prover can give a (provable) exponential improvement in query complexity compared to what is possible in property testing without a prover (even with only a single round of interaction

[2] In this scenario, the protocol might only be run later, after the real-time data has been observed. When this is the case, we may want an Interactive Proof where the verifier's queries can be made ahead of time, and are independent of the prover's messages. The verifiers in all of our protocols have this *"oblivious query"* property.

and logarithmic communication).

Another beautiful line of research, starting with the work of Babai *et al.* on Holographic Proofs [BFLS91], has focused on "PCP-like" proof systems with sublinear-time verifiers. They constructed such proof systems under the assumption that the input is given in encoded form. Ergun, Kumar and Rubinfeld [EKR04] constructed PCP-like proofs for several natural languages. The work of Ben-Sasson *et al.* on PCPs of Proximity [BGH$^+$06], and the work of Dinur and Reingold on Assignment Testers [DR06], give general constructions using PCP machinery. These works can be viewed as extending the property testing model by giving the verifier query access to a *fixed* proof string. While that proof string may be wrong, it is nonetheless fixed and does not change with verifier's queries to it. In our model (and in the delegating computation motivation) the prover can *adaptively* change its strategy and answers, as a function of subsequent verifier's messages. As discussed in [GKR08], handling adaptive cheating provers is essential for applications to delegating computation, and "PCP-like" proof systems are not directly suited for these applications. Further, this difference in the power of the cheating prover is driven home by the fact that in the standard setting, with polynomial time verifiers, the setting of a *fixed* proof string gives rise to the class $\mathcal{MIP}$ [BGKW88], and allows verification of languages in $\mathcal{NEXP}$ [BFL91], whereas the setting of an *adaptive* prover gives rise to the class $\mathcal{IP}$ [GMR89, BM88], and allows only verification of languages in $\mathcal{PSPACE}$ [LFKN92, Sha92]. Thus, we expect to be able to prove less than in the "PCP-like" setting, and indeed prove lower bounds to that effect.

1.1 The Power of IPP

In this section we outline our positive results. First, we show a general result, Theorem 1.1 below: every language computable by circuits of low depth,[3] has a sublinear-time Interactive Proof of Proximity. To prove this general result we identify a "complete problem" for constructing Interactive Proofs of Proximity: testing proximity to an affine subspace (form a structured class having to do with polynomial interpolation, see below). We provide an IPP for the complete problem, and then use completeness to derive IPPs for low-depth languages. In addition, we also mention our results for specific languages: IPPs for testing bipartiteness in the bounded degree model, and for testing Hamming weight. We find constructions for specific languages to be interesting even given the general result in Theorem 1.1. First, these constructions improve the communication and round complexities obtained there. The improvements can be quite dramatic: for bipartiteness we obtain a provable exponential improvement in the query complexity needed for IPPs compared with standard property testing (without a prover). Moreover, even if the communication or round complexities are not improved, construction for specific languages avoid the overhead induced by the completeness reduction (which requires going through a circuit representation of the problem and the arithmetization machinery used in the protocol of [GKR08]).

IPPs for Low-Depth Computation. We show that there exists a universal constant $\gamma > 0$, s.t. every language in $Depth(n^\gamma)$ has a sublinear-complexity IPP.

Theorem 1.1 ($Depth(n^\gamma) \subseteq \mathcal{IPP}$)**.** *For every language L computable by log-space uniform circuits*

[3]Throughout, when we say a language is computable in "low depth" or $Depth(n^\gamma)$, we mean that it can be computed by a log-space uniform family of Boolean circuit with fan-in 2 and depth $O(n^\gamma)$, for a universal constant $\gamma > 0$.

of depth $D = D(n)$, size $S = S(n)$ and fan-in 2, and every distance $\varepsilon = \varepsilon(n)$,[4] *there exists an Interactive Proof of ε-Proximity for L.*

The proof has perfect completeness and soundness error $1/2$. For an input of length n, the query complexity is $(1/\varepsilon)^{1+o(1)}$, the communication complexity is $(\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)} \cdot \mathrm{poly}(D))$, and the number of rounds is $O(\log n + D \cdot \log S)$. The honest Prover $\mathcal{P}$ runs in time $\mathrm{poly}(S)$, and the verifier $\mathcal{V}$ runs in time $O(((1/\varepsilon) + (\varepsilon \cdot n))^{1+o(1)} \cdot \mathrm{poly}(D) \cdot \log S)$.

The proof is in Section 3. We note that Theorem 1.1 gives sublinear-copmlexity IPPs even for very small values of ε,[5] e.g. for $\varepsilon = 1/n^{1-\delta}$ for any constant $\delta > 0$. In interpreting this result, one useful setting to consider is that of an $\mathcal{NC}$ language (computable in depth $D(n) = \mathrm{polylog}(n)$ and size $S(n) = \mathrm{poly}(n)$). Here Theorem 1.1 gives a tradeoff between the query complexity q and communication complexity c, where $q \times c = n^{1+o(1)}$. The number of rounds is poly-logarithmic, and the honest prover runs in polynomial time. In particular, for any $\varepsilon(n) \geq n^{-1/2}$, the queries, communication, and verifier runtime can all be be $n^{1/2+o(1)}$. This applies to varied problems in $\mathcal{NC}$: for example, graph functions such as planarity testing, connectivity, finding a perfect matching, or algebraic problems such as computing matrix rank..

Remark 1.2 (Nearly-Optimal Query Complexity)**.** *In general, query complexity $\Omega(1/\varepsilon)$ is essential for IPPs: a cheating prover can always choose $x \in L$ and flip $(\varepsilon \cdot n)$ uniformly random coordinates. Suppose that w.h.p this gives x' that is ε-far from L (e.g. L is an error-correcting code, so no two strings in L are close). When running the protocol on x', the cheating prover can follow the honest prover's strategy for input x, and unless the verifier queries one of the coordinates on which x and x' differ it will accept (by completeness). Since x and x' differ only on $(\varepsilon \cdot n)$ uniformly random coordinates, the verifier must make $\Omega(1/\varepsilon)$ queries in order to reject x' (as needed for soundness).*

A "Complete" Problem: Testing Membership in an Affine Subspace. For input length n, an integer $t \ll n$, and a finite field F, an affine subspace is specified by a matrix $A \in F^{t \times n}$ and a "target" vector $v \in F^t$. The language *AffineMem*(A, v), of checking membership in the affine subspace defined by A and v, is the set of inputs $x \in F^n$ for which $A \cdot x = v$.[6] An IPP for *AffineMem* allows a verifier to check whether its input $x \in F^n$ is close to a vector $x' \in F^n$ such that $A \cdot x' = v$, using only a sublinear (in n) number of queries to x, and sublinear communication. Looking ahead, we will focus on testing membership in a specific structured family of affine subspaces. In particular, the matrix A will have a succinct representation. See further details below.

We provide a reduction, via an interactive protocol, from proving proximity to any language L computable in low depth, to proving proximity to *AffineMem* (for a restricted class of matrices A). This reduction uses the "Interactive Proofs for Muggles" of [GKR08] (see Section 3).

Theorem 1.3 (Informal; *AffineMem* is "complete")**.** *For every distance bound $\varepsilon = \varepsilon(n)$, and every language L computable by log-space uniform boolean circuits of depth $D = D(n)$, size $S = S(n)$, and fan-in 2, for $t = t(n) = O(\varepsilon \cdot n \cdot \log n)$ and a finite field F, there exists an interactive protocol between a prover and a verifier as follows. On input x, the output is (an implicit description of) a matrix $A \in F^{t \times n}$ and an (explicit) vector $v \in F^t$, such that:*

[4] Here and throughout, we assume that all complexity and distance parameters such as $D(n)$, $S(n)$, $\varepsilon(n)$ are computable in space $O(\log n)$.

[5] The smaller ε is, the "harder" it is to prove ε-proximity. In particular an Interactive Proof of ε-Proximity, is also an interactive proof of ε'-proximity for any $\varepsilon' \geq \varepsilon$.

[6] Or rather, the language is specified by an ensemble of matrices and target vectors, one for each input length.

- *Completeness. If $x \in L$ and the prover honestly follows the protocol, then $x \in AffineMem(A, v)$.*
- *Soundness. If x is ε-far from L, then for any cheating prover, with all but $1/2$ probability over the verifier's coins, x is also ε-far from $AffineMem(A, v)$.*

The prover runs in time $\text{poly}(S)$*, and the verifier runs in time* $\varepsilon \cdot n \cdot \text{poly}(D)$*. The communication complexity is* $\varepsilon \cdot n \cdot \text{poly}(D)$*, and the number of rounds is* $O(D \cdot \log S)$*. The verifier makes no queries to the input x during the protocol.*

As noted above, we focus our attention on a structured variant of *AffineMem*, the problem *PVAL* of evaluating (or interpolating) bounded-degree multivariate polynomials. In fact, the formal statement of Theorem 1.3 shows completeness for this structured variant, and we use the additional structure to directly construct an IPP for *PVAL*. This, in turn, yields (via completeness) an IPP for any low-depth language, and in particular also for the more general *AffineMem* problem (where the matrix A has any implicit representation computable in low depth).

The problem *PVAL* is defined as follows. Each input x specifies a multivariate bounded-degree polynomial P_x over a field F. An affine subspace is now defined by a size-t set $J \subseteq F^m$ of "evaluation points", and a "target" vector $v \in F^t$ of values. The language $PVAL(J, v)$ is the set of inputs x s.t. $\forall j \in J, P_x(j) = v[j]$ (where $v[j]$ denotes entry of v corresponding to item j).[7] One difference from the more general *AffineMem* problem, is that here the (large) matrix A (of size $t \cdot n$) is implicitly specified by the set J (of size t).

In more detail: the input x specifies the polynomial P_x as a list of its evaluations on a canonical subset $S \subseteq F^m$ (we call these the "interpolation points"). E.g., for polynomials of degree less than k in each variable, we can take $S = H^m$ for any fixed $H \subseteq F$ of size k (usually F will be an extension field of H). Taking $n = k^m$, the input x gives an evaluation for each interpolation point, yielding a unique polynomial P_x of degree less than k in each variable that agrees with these evaluations. For any point $j \in F^m$ and a "target value" $v[j] \in F$, the constraint $P_x(j) = v[j]$ is a linear constraint over P_x's evaluations on the interpolation points (using the linearity of polynomial interpolation). I.e., we can express this constraint as $A_j \cdot x = v[j]$, where $\mathcal{A}_j \in F^n$ is a (fixed) function of j. The language $PVAL(J, v)$ is exactly $AffineMem(A_J, v)$, where for each $j \in J$ the matrix A_J includes the row A_j defined above.

An IPP for *PVAL*. The goal of a sublinear verifier for *PVAL*, is to determine whether the given evaluations of the polynomial on evaluation points in J are correct, while only reading a sublinear number of interpolation points. We note that the polynomial's evaluation on each point in J might well depend on most (if not all) of the evaluations on interpolation points. Thus, the verifier cannot (on its own) verify any of the claimed evaluations on J without reading $\Omega(n)$ coordinates of x. Moreover, for natural choices of J and v the distribution generated by drawing a uniformly random input $x \leftarrow PVAL(J, v)$ can be $\Omega(n)$-wise independent: In particular, for any $n/10$ queries made by the verifier into the input, the values it sees are uniformly random. A uniformly random input, however, is $(1/10)$-far from $PVAL(J, v)$ with all but negligible probability. We are asking the verifier to distinguish these two distributions using a sublinear number of queries! Indeed, this is impossible in the standard property testing model (without a prover), because the verifier should

[7] To be more precise, *PVAL* is an infinite ensemble of finite languages indexed by the field F, the number of variables, the degree bound, the subset J of points, and the vector v of values on J.

accept a random input in the language, and reject a uniformly random input, but its views of the input in both cases are statistically close.

In the presence of a prover, however, we can overcome these difficulties, and construct an Interactive Proof of Proximity for *PVAL*.

Theorem 1.4 (Informal; IPP for *PVAL*)**.** *For every $n \in \mathbb{N}$ and $\varepsilon > 0$, for J, v, F, k, m as generated in the reduction of Theorem 1.3, there exists an ε-IPP for $PVAL(J, v)$. The IPP has perfect completeness, soundness $1/2$, query complexity $(1/\varepsilon)^{1+o(1)}$, and $O(\log(1/\varepsilon))$ rounds. The communication complexity is $\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)}$, and the verifier runtime is $((1/\varepsilon) + \varepsilon \cdot n)^{1+o(1)}$. The honest prover runtime is* $\mathrm{poly}(n)$.

More generally, we get a tradeoff between the communication complexity and round complexity. The theorem statement above is one extreme of this tradeoff (optimized for small communication). We can also optimize for the round complexity, and obtain a 3-message protocol with $\sqrt{\varepsilon} \cdot n \cdot (1/\varepsilon)^{o(1)}$ communication. See Corollaries 3.2 and 3.3 in Section 3 for formal statements.

Remark 1.5. *We emphasize that the IPP setting is different from the setting of local testing or decoding, and from the setting of PCP-like proofs, in that a malicious prover can be completely adaptive in its cheating. Thus, it is not obvious how to use the strong local testing and decoding properties of polynomials. In particular, one initial approach might be to have the prover perform polynomial evaluations for the verifier, and use local testing/decoding to detect cheating, as is done throughout the PCP literature and (in particular) in the work of [BGH+06]. The difficulty with this approach, is that local testing and decoding procedures are only sound against non-adaptive (static) corruptions, and are easily fooled by adaptive corruptions.*

A Taste of Techniques. We proceed with intuition for the proof of Theorem 1.4. Consider the case where the input x specifies a 2-variate polynomial $P_x(i_1, i_2)$ of degree less than $k = \sqrt{n}$ in each variable. x includes the evaluations of P_x on all the specification points H^2, where $H \subseteq F$ is of size $\sqrt{n}$. We can "break up" $P_x(i_1, i_2)$ into H univariate polynomials $\{Q_{i_1}(i_2) = P_x(i_1, i_2)\}_{i_1 \in H}$. Each univariate Q_{i_1} is specified by $\sqrt{n}$ bits of the input. Moreover, for any $j = (j_1, j_2) \in F$, we can use polynomial interpolation to express $P_x(j)$ as a linear combination of $\{Q_{i_1}(j_2)\}_{i_1 \in H}$, where the coefficients in the linear combination depend only on j_1.

With this view in mind, consider an evaluation point $j = (j_1, j_2) \in J$. The prover claims that $P_x(j) = v[j]$. In the IPP, for each $j \in J$, the prover sends $\sqrt{n}$ (alleged) values $\{Q_{i_1}(j_2)\}_{i_1 \in H}$. The verifier checks (via polynomial interpolation, as above), that these values are consistent with the "j-th claim" $P_x(j) = v[j]$. If the j-th claim was false, and the values are consistent with it, then there must be some $i_1 \in H$ for which the prover sent a false value v_{i_1, j_2} for $Q_{i_1}(j_2)$. We have gone from $|J|$ claims about the "large polynomial" P_x (specified by n variables), to $|J| \cdot \sqrt{n}$ claims about $\sqrt{n}$ "small polynomials" $\{Q_{i_1}\}_{i_1 \in H}$. While each small claim depends only on $\sqrt{n}$ input bits, it is not clear whether we have made any progress: verifying all of the new claims still requires reading every bit of the input x.

Suppose, however, that the prover has been careless in its cheating: in particular, for each $j = (j_1, j_2) \in J$, the prover lies about the value of $Q_{i_1}(j_2)$ for a different $i_1 \in H$. In this case, the verifier can pick a uniformly random $i_1 \in H$, and check all $|J|$ claims on Q_{i_1}. Observe that this only requires reading $\sqrt{n}$ input bits of x: the evaluations of P_x on points in $\{i_1\} \times H$. Since we assumed (with loss of generality) that for $|J|$ of the choices of i_1 there exists some claim on Q_{i_1} where the prover is cheating, the verifier will catch the prover with probability $|J|/|H|$. This would already

imply an IPP with sublinear complexity (for the appropriate choice of $|J|$). The main challenge is that a cheating prover need not be so careless—in particular, the prover may focus all of its false claims on a single small polynomial Q_{i_1}. In this case, the verifier cannot detect the cheating by testing randomly chosen Q's. This is the main technical challenge for constructing a non-trivial IPP. Our first observation is that, in this case, Q_{i_1} is "far from" the prover's claims: the $\sqrt{n}$ input bits that specify Q_{i_1} are very far from any $\sqrt{n}$ bits that specify a polynomial Q'_{i_1} that agrees with the prover's claims. In the IPP, the verifier combines the prover's claims about the different Q's by taking a random linear combination of them $\hat{Q} = \sum_{i_1 \in H} \alpha_{i_1} \cdot Q_{i_1}$. This induces $|J|$ new claims about the values of $\hat{Q}$ at $|J|$ points in F. We show that if there is a Q_{i_1} that is "far from" the prover's claims, then, with high probability over the linear combination, the new polynomial $\hat{Q}$ will be "far from" the induced claims. This is shown using a new lemma about distances between linear subspaces, which may be of independent interest:

Lemma 1.6. *Let S and T be two linear subspaces of F^n (for any finite field F and $n \in \mathbb{N}$). Suppose that there exists some point $s \in S$ such that s is at (fractional) Hamming distance at least ε from T (i.e., at Hamming distance at least ε from every $t \in T$).*

Then, with all but $1/(|F|-1)$ probability over the choice of a uniformly random point $r \in_R S$, the distance between r and T is at least $\varepsilon/2$.

Omitting many non-trivial details, the final step in the IPP is for the verifier to check consistency of the induced claims about $\hat{Q}$ viz a viz the input x. We show that this can be done using sublinear query and communication complexities. Note that the above was all for the case of 2-variate polynomials, and required only a constant number of rounds. In the full protocol we can handle m-variate polynomials by composing a protocol based on these ideas to repeatedly reduce the number of variables (using $O(m)$ rounds of communication). We conclude the technical overview by remarking that the protocol for *PVAL* can be generalized to a natural problem of evaluating tensor powers of a linear code. We take this more general view throughout Section 3 In this we follow, and are inspired by, the work of Meir [Mei10], who illuminated the central role of tensor codes in the construction of standard interactive proofs. Here too, tensors of linear codes play an essential role in constructing interactive proofs of proximity. See Section 3 for further details.

1.1.1 IPPs for Specific Languages

We now move to specific languages, for which we provide tailored IPPs that are more efficient (and simple) than the general construction of Theorem 1.1.

An IPP for Graph Bipartiteness. We construct an Interactive Proof of proximity for bipartiteness on random or well-mixing graphs in the bounded degree model (see e.g. [Gol10a]). The protocol accepts every bipartite graph. For a random graph of bounded degree, or for any graph that is both far from bipartite and well mixing, the verifier will reject with high probability. That is, this is a protocol for a promise problem, where YES instances are bipartite, and NO instances are far from bipartite and well-mixing (thus a random bounded-degree graph will be a NO instance with high probability). For constant distance parameter $\varepsilon = \Theta(1)$, the protocol has $O(\log n)$ query and communication complexities, and only a single round of communication. This is a particularly sharp illustration of the benefits of interaction with a prover, as Goldreich and Ron [GR02] proved a $\Omega(\sqrt{n})$ lower bound on the query complexity of any property tester for this problem (i.e. without

the help of a prover). The IPP uses ideas from Goldreich and Ron [GR99], but leverages the presence of a prover to obtain (provably) an exponential improvement in the query complexity (using only a single round of interaction, and logarithmic communication). An interesting open problem is extending this IPP to apply to all graphs (not just rapidly-mixing ones), as was done by [GR99] in the standard property testing model (they build a $\tilde{O}(\sqrt{n})$-query tester for all graphs). See Section 5.1 for further details.

An IPP for Majority/Hamming Weight. We also consider the language *Hamming*(w), of vectors in $\{0,1\}^n$ with Hamming weight w. We give a direct Interactive Proof of ε-Proximity for this problem, with query and communication complexities $\tilde{O}(1/\varepsilon)$, and logarithmically many rounds. This protocol can be used by a sublinear-time verifier, with the help of a more powerful prover, to approximate the Hamming weight of a given vector to within distance ε. In particular, it can also be used to prove that the (approximate) majority of bits in a boolean vector are 1 (where the approximation is up to fractional additive error ε). Here too, there is a provable improvement on what can be obtained without a prover: by sampling lower bounds, approximating the Hamming weight to within additive fractional error ε requires $\Omega(1/\varepsilon^2)$ queries. We note that, similarly to remark 1.2, $\Omega(1/\varepsilon)$ queries are necessary, and so this result is essentially tight. See Section 5.2 for further details.

1.2 The Limits of IPP

We prove lower bounds on the tradeoffs between query complexity, communication complexity, and round complexity, showing that constant-round interactive proofs of proximity must have query or communication complexity polynomially related to n. Specifically, we exhibit a uniform $\mathcal{NC}^1$ language $L_n \subseteq \{0,1\}^n$ (in fact an $\mathbb{F}_2$-linear subspace) for which any $O(1)$-round interactive proof of (1/8)-proximity with communication complexity $c(n)$ and query complexity $q(n)$ must have $\max\{c(n), q(n)\} = n^{\Omega(1/r(n))}$. So achieving polynomially small complexity (as we do) is the best we can hope for in bounded-round protocols. (It remains an intriguing open problem whether using many rounds, polylogarithmic communication and query complexity is possible.)

To prove our lower bound, we first use standard transformations of interactive proofs [GS86, BM88] to convert any bounded-round, private-coin IPP into an "AM-type" IPP with related complexity. By an "AM-type" IPP, we mean a 2-message public-coin protocol, where the verifier sends its coin tosses to the prover, receives a message from the prover, and then deterministically queries its input oracle and decides whether to accept or reject. For AM-type protocols, we obtain a lower bound of $\max\{c(n), q(n)\} = \tilde{\Omega}(\sqrt{n})$ by using a novel connection with pseudorandomness for CNF formulas. Specifically, we show that the maximum acceptance probability of the verifier in an AM-type IPP protocol can be computed as the probability that the input x satisfies a random CNF formula of size at most $2^{c(n)+q(n)}$ chosen according to the following distribution: choose the coins v of the verifier at random, and then consider the CNF formula that takes an OR over the $2^{c(n)}$ possible prover messages p and checks whether the input x yields any of the $\leq 2^{q(n)}$ accepting sequences of responses to the verifier's oracle queries. Thus, we can take our language L_n to be the image of a pseudorandom generator for CNF formulas of size $2^{c(n)+q(n)}$ with seed length $O((c(n)+q(n))^2 \log n)$, e.g. as constructed in [Nis91, Baz09]. The protocol will have roughly the same acceptance probability on a random element of L_n as it does on a uniformly random string, which will be far from L_n with high probability unless $\max\{c(n), q(n)\} = \tilde{\Omega}(\sqrt{n})$ (since otherwise L_n has exponentially small support). See Section 4 for further details.

1.3 Further Remarks and Related Work

See above for the relationship with property testing [RS96, GGR98] and with sublinear time proof verification in "PCP-like" models [BFLS91, EKR04, BGH+06, DR06].

Related Work on Interactive Proofs. Interactive Proofs were introduced by Goldwasser, Micali and Rackoff [GMR89], and independently by Babai and Moran [BM88]. The expressive power of $\mathcal{IP}$ was determined in a beautiful sequence of works by Lund, Fortnow, Karloff and Nissan [LFKN92] and Shamir [Sha92], who showed that every language in $\mathcal{PSPACE}$ has an interactive proof. While prior works have considered various restrictions on the verifier in an interactive proof (see below), the verifiers in these works all need to read their input in its entirety, and so their running time is at least *linear*. Dwork and Stockmeyer [DS92a, DS92b] investigated the power of finite state verifiers. Condon and Ladner [CL88], Condon and Lipton [CL89], and Condon [Con91] studied space (and time) bounded verifiers. Goldwasser *et al.* [GGH+07] considered the parallel running time of the verifier, and showed results for $\mathcal{NC}^0$ verifiers. Goldwasser, Kalai and Rothblum [GKR08] explored the power of interactive proofs with efficient (polynomial-time) honest provers and *linear-time* verifiers. Cormode, Mitzenmacher and Thaler [CMT10] and Cormode, Thaler and Yi [CTY11] studied interactive proofs with streaming verifiers: the verifier runs in logarithmic space and linear time, and only has single-pass streaming access to the input.

In terms of the running time of the *honest* prover, we follow [GKR08] in focusing on polynomial-time honest provers, and on interactive proofs for tractable languages. All of our protocols for specific languages have efficient honest provers. The general result of Theorem 1.1 applies to every language that is computable in low-depth (even with super-polynomial size circuits), but we emphasize that the honest prover running time in that protocol is polynomial in the circuit size. If the circuit is polynomial size, then the honest prover runs in polynomial time.

Computationally Sound Proof Systems. We note that *under cryptographic assumptions*, if we only require security against polynomial-time cheating provers, then even more powerful general results are possible (see below). In contrast, all of our protocols are unconditionally secure, and make no assumptions about the power of a cheating prover. We further remark that our negative results show a provable separation between what can be obtained by unconditionally sound versus computationally sound protocols (under cryptographic assumptions).

We briefly elaborate on computationally sound interactive proofs of proximity. These can be obtained by combining PCPs of Proximity (or Assignment Testers) [BGH+06, DR06], with the techniques of Kilian and Micali [Kil88, Mic94] for ensuring non-adaptive behavior of the (polynomial-time) cheating prover. The idea is for the prover to construct a PCP of Proximity, and then Merkle-hash it down to a short string that is sent to the verifier. The Merkle hash forces non-adaptive behavior later, when the prover answers the verifier's queries. These techniques require assuming a family of collision-resistant hash functions (CRHs). The end result is a 2-round computationally sound interactive proof of ε-proximity for any language in $\mathcal{EXP}$. The prover's running time is polynomial in the running time needed to decide the language, the verifier's running time is polylogarithmic in this running time (and polynomial in the security parameter). The query complexity is $(1/\varepsilon) \cdot \text{polylog}(n)$, and the communication complexity is polylogarithmic in n and polynomial in the security parameter. Micali's construction can be used to further reduce the round complexity to a single round in the random oracle model.

2 Definitions and Preliminaries

Notation. For a finite field F and a vector $\vec{x} \in \Sigma^n$, we use $|\vec{x}|$ to denote the length of $\vec{x}$. We use $\vec{x}[i]$ to refer to the i-th element of $\vec{x}$. For a subset $I \subseteq [n]$, $\vec{x}_{|I} \in F^{|I|}$ is the projection of $\vec{x}$ onto coordinates in I. The (absolute) Hamming weight $\|\vec{x}\|$ is the number of non-zero coordinates in $\vec{x}$. For two vectors $\vec{x}, \vec{y}$ of the same length, the (fractional) Hamming distance between $\vec{x}, \vec{y} \in F^n$, which we denote $\Delta(\vec{x}, \vec{y})$, is the fraction of coordinates on which $\vec{x}$ and $\vec{y}$ disagree ($\|\vec{x} - \vec{y}\|/n$). We sometimes refer to the *absolute* Hamming distance $\Delta_{absolute}(\vec{x}, \vec{y}) = \Delta(\vec{x}, \vec{y}) \cdot n$. Throughout this work, when we refer to Hamming distance we mean *fractional* distance unless we explicitly note otherwise.

For a matrix $X \in F^{k \times \ell}$, and $(i, j) \in [k] \times [\ell]$, we use $X[i][j]$ to refer to the item in the i-th row and j-th column of X. We use $X[i, *] \in F^{\ell}$ to refer to the i-th row, and $X[*, j] \in F^k$ to refer to the j-th column. For a subset of coordinates $I \subseteq [k] \times [\ell]$, we use $X_{|I}$ to denote the *vector* of X's values in coordinates in I (in lexicographic order). We define $|X|$, $\|X\|$, and the Hamming distance between matrices analogously to vectors.

In several places throughout this work, we view vectors as matrices. For $\vec{x} \in F^n$, we specify k and ℓ such that $n = k \cdot \ell$, and associate every coordinate in $[n]$ with a pair $(i, j) \in [k] \times [\ell]$ in the natural way. We view $\vec{x}$ as a matrix $X \in F^{k \times \ell}$, where every element of $\vec{x}$ is mapped to the appropriate coordinate in X.

For a distribution $\mathcal{A}$ over a finite domain S, and an element $s \in S$, $\mathcal{A}(s)$ denotes the probability of s by $\mathcal{A}$. For two distributions $\mathcal{A}$ and $\mathcal{B}$ over the same support S, $\Delta(\mathcal{A}, \mathcal{B})$ denotes the statistical distance between $\mathcal{A}$ and $\mathcal{B}$:

$$\Delta(\mathcal{A}, \mathcal{B}) = \frac{1}{2} \sum_{s \in S} |A(s) - \mathcal{B}(s)|$$

We use $negl(n) = n^{-\omega(1)}$ to denote a negligible function that is asymptotically smaller than the inverse of any polynomial.

Interactive Proofs of Proximity. These are protocols used for verifying that an input is *close to* a language L. The goal is achieving this with a verifier that runs in sublinear time, without even reading the input in its entirety. The input to the proof system is shared by the prover and the verifier. For greater generality, as in [BGH+06], we divide the input into an implicit or oracle part x, and an explicit part w. The prover gets (x, w) as a (standard) input. The verifier gets $n = |x|$ ($O(\log n)$ bits) and w as a standard input, and gets oracle access to x. For a language L over pairs (x, w), we refer to the language $L(w) = \{x : (x, w) \in L\}$. For example, the $PVAL$ language has an explicit input $(F, k, m, J, \vec{v})$, where F is a finite field, k an individual degree bound, m a number of variables, J a set of evaluation coordinates, and $\vec{v}$ a vector of values. The implicit input x is the description of a multivariate polynomial p_x. An input $(x, (F, k, m, J, \vec{v}))$ is in $PVAL$ if p_x and $\vec{v}$ agree on the evaluation points in J. Alternatively, x is in $PVAL(F, k, m, J, \vec{v})$ iff $(x, (F, k, m, J, \vec{v})) \in PVAL$.

Definition 2.1 (Interactive Proof of Proximity (IPP))**.** For a language L over alphabet $\Sigma = \Sigma(n)$, and distance $\varepsilon = \varepsilon(n)$, a proof system $(\mathcal{P}, \mathcal{V})$ is an *Interactive Proof of ε-Proximity* for a language L on pairs (x, w), with $\alpha = \alpha(n)$ and $\beta = \beta(n)$ completeness and soundness errors (respectively), if the following conditions hold for all x of length n and strings w,

- α-Completeness: if $(x, w) \in L$, then with all but α probability over the coins of $\mathcal{V}$, it will accept in the protocol execution $(\mathcal{P}(x, w), \mathcal{V}^x(n, w))$.
- β-Soundness: if x is at fractional Hamming distance ε or more from L, then for every cheating prover $\mathcal{P}'$, with all but β probability over the coins of $\mathcal{V}$, it will reject in the protocol execution $(\mathcal{P}'(x, w), \mathcal{V}^x(n, w))$

We call x the *implicit input*, and w the *explicit input* (which may be empty). A *public-coin protocol* is one in which the verifier's messages are simply random coin tosses. A protocol with completeness error 0 is said to have *perfect completeness*. The *query complexity* $q = q(n)$ is the number of queries $\mathcal{V}$ makes into x (each query returns an element in the alphabet Σ). The *communication complexity* $c = c(n)$ is the number of bits exchanged. The *round complexity* $r = r(n)$ is the number of communication rounds (with two messages per round).

Definition 2.2 ($\mathcal{IPP}$ Complexity Class)**.** The class $\mathcal{IPP}(\varepsilon, q, c, r, v, p)$ includes all languages that have Interactive Proofs of $\varepsilon(n)$-proximity, with query, communication, and round complexities $q(n)$, $c(n)$, and $r(n)$ (respectively), where the verifier running time is $v(n)$ and the honest prover's running time is $p(n)$.

For ease of notation, we also define $\mathcal{IPP}(\varepsilon, q, c, r)$, the class of languages as above, where the verifier's running time is $\text{poly}(q(n), c(n), \log n)$ and the prover's running time is $\text{poly}(n)$. Finally, on occasion we refer to $\mathcal{IPP}(\varepsilon)$, the class of languages as above, where the query and communication complexities, and the verifier's running time, are all sublinear in n, and the honest prover's running time is $\text{poly}(n)$.

3 Proving Proximity to *PVAL* and Low-Depth Languages

In this section we describe our positive results on Interactive Proofs of Proximity for Low-Depth Languages. These are obtained by first identifying the complete language *PVAL*: we give a reduction from proving proximity to any language computable in low depth, to proving proximity to *PVAL*. Our main technical contribution is an Interactive Proof of Proximity for *PVAL*. By completeness, this protocol immediately implies an Interactive Proof of Proximity for every language computable in low depth (see Theorem 1.1 in the introduction).

As discussed in the introduction, the implicit input X to *PVAL* specifies an m-variate polynomial over a finite field F. The language is specified by a set of points and values, and a polynomial X is in the language if it agrees with the given values on the given points. Equivalently, we can think of *PVAL* as the language of evaluating coordinates in a Reed-Muller encoding of the input X (a message for the code). A message X is in the language if its encoding, projected onto the given coordinates, agrees with the given values.

Definition 3.1 (*PVAL*)**.** *PVAL* is a language on pairs: X and $(F, k, m, J, \vec{v})$, where X is the implicit input, and the other inputs are explicit. $X \in F^{k^m}$ is the description m-variate polynomial p_X of degree at most k in each variable. Taking $[k]$ to be the lexicographically first k elements of F, X specifies p_X by its evaluations on all points in $[k]^m$ (we refer to $[k]^m$ as the "interpolation set"). $J \subseteq F^m$ is a set of evaluation points, and $\vec{v} \in F^{|J|}$ a vector of values. The input is in *PVAL* if and only if the polynomial p_X takes the values specified by $\vec{v}$ on the coordinates in J.

Equivalently, taking C^m to be an m-variate Reed-Muller code over F (with degree parameter k in each variable), the input is in $PVAL$ if and only if the encoding $C^m(X)$ takes the values specified by $\vec{v}$ on the coordinates in J: $C^m(X)_{|J} = \vec{v}$.

We say that $(X, J, \vec{v})$ is an instance of $PVAL$ on (F, k, m) or on C^m, if and only if it is the case that $(X, (F, k, m, J, \vec{v})) \in PVAL$. We say that $(X, J, \vec{v})$ is d-far from $PVAL$ (on (F, k, m) or on C^m) if and only if X is d-far from $PVAL(F, k, m, J, \vec{v})$).

We begin with an overview of our contributions: the completeness reduction to $PVAL$, and the Interactive Proof of Proximity for $PVAL$, highlighting the main technical contributions along the way. The full details follow.

Overview: $PVAL$ is "Complete". We show a reduction from proving proximity to any language L computable in (log-space uniform) size $S = S(n)$ and depth $D = D(n) = n^\gamma$, to proving proximity to $PVAL$. This uses the "Interactive Proofs for Muggles" of [GKR08]. They showed an interactive proof for any such L, where the prover runs in time $\text{poly}(S)$, and the round complexity, communication complexity, and verifier running time are all polynomial in D. That Interactive Proof, when run on an input $X \in \{0,1\}^n$, outputs a claim about an encoding of X at a uniformly random coordinate j. The encoding is performed using an m-variate Reed-Muller code C^m. The protocol's output is a pair (j, v), specifying the claim $C^m(X)[j] = v$. Completeness means that, when $X \in L$ and the prover honestly follows the protocol, the claim specified by (j, v) is indeed true. Soundness means that, for $X \notin L$, for any cheating prover strategy, with probability at least $1/2$ over the verifier's coins, the claim specified by (j, v) is false, i.e. $C^m(X)[j] \neq v$.

Our reduction runs the GKR interactive proof t times in parallel. This yields t pairs of the form (j, v), which we aggregate into a collection J of coordinates in $C^m(X)$, and a vector $\vec{v}$ of claimed values for those coordinates. Completeness means that, when $X \in L$ and the prover honestly follows the protocol, the claim specified by $(J, \vec{v})$ is true, i.e. $C^m(X)_{|J} = \vec{v}$. Soundness means that when $X \notin L$, for any cheating prover strategy, w.h.p. the claim specified by $(J, \vec{v})$ is false, i.e. $C^m(X)_{|J} \neq \vec{v}$. On closer inspection, soundness actually implies a stronger property: If X is at (fractional) Hamming distance ε from L, where $t \geq 2\varepsilon \cdot n \cdot \log n$ from L, then w.h.p. over the verifier's coins, $C^m(X')_{|J} \neq \vec{v}$ *for all X' at distance less than ε from X (simultaneously).* To see this, observe that every X' at distance less than ε from X is also not in L (because X is ε-far from L). By soundness of each invocation of the GKR protocol, with probability at least $1/2$, the claim on X' specified by that invocation's output is false, i.e. $C^m(X')[j] \neq v$. The probability that, over t independent parallel runs of the GKR protocol, $C^m(X')_{|J} = \vec{v}$ is at most $2^{-t} \leq n^{-2\varepsilon \cdot n}$. The number of inputs at (fractional) distance less than ε from X is at most $n^{\varepsilon \cdot n}$. Taking a Union Bound over all inputs X' at distance less than ε from X, we get that w.h.p. there exists no such X' for which the claim specified by $(J, \vec{v})$ is true.

We conclude that there is a reduction from any language $L \in Depth(n^\gamma)$ to $PVAL$ on the code C^m used by the GKR protocol. We note that the reduction specifies an instance X for $PVAL(J, \vec{v})$, where $|J| = O(\varepsilon \cdot n \cdot \log n)$. In our use of the GKR verifier, it sees J and $\vec{v}$, but never accesses the input X.[8] The completeness of $PVAL$ motivates our main contribution: an Interactive Proof of Proximity for this problem (and through it, for any language in $Depth(n^\gamma)$).

[8] In its original setting, the GKR verifier runs an additional test to check that the prover's claim about the input encoding in true. This requires (quasi)-linear time, and so we avoid this step.

Overview: Interactive Proof of Proximity for $PVAL$. As discussed in the introduction, the difficulty in proving proximity to $PVAL$, is that each coordinate in the encoding $C^m(X)$ may well depend on most (if not all) of X's coordinates. The verifier cannot (on his own) verify any of the prover's claims about coordinates in $C(X)$ without reading $\Omega(n)$ coordinates of X. Indeed, we do note know how to directly build an Interactive Proof of Proximity for verifying the values of an *arbitrary* code C^m in a specified set of coordinates.[9]

Luckily, C^m is not an arbitrary code: it is a Reed-Muller code, which (among its many useful properties) is a Linear Tensor Code (see Section 3.1 for background on Tensor codes). This is the main structural property of $PVAL$ that we will exploit. More generally, we will define the language $TVAL$ for evaluating any tensor code C^m of a linear code C. Analogously to $PVAL$, $TVAL$ (on C^m) is specified by a set of codeword coordinates J and values $\vec{v}$. A message X is in $TVAL$ if $C^m(X)_{|J} = \vec{v}$ (see Definition 3.6 for further details).

Our main result in this section is an Interactive Proof of Proximity for $TVAL$ on any tensor of a linear code. In particular, this immediately gives an interactive proof for $PVAL$ (which is $TVAL$ on a Reed-Muller code). This is stated in Theorem 3.12, which appears in Section 3.3. This theorem gives a tradeoff between the number of messages exchanged and the communication complexity. Rather than give the more general statement, we focus for now on the two corollaries outlined in the introduction. We consider the case $|J| = O(\varepsilon \cdot n \cdot \log n)$ (as in instances produced by the reduction from $Depth(n^\gamma)$ to $PVAL$).

Corollary 3.2. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0,1]$, take $k = \sqrt{1/\varepsilon}$ and $m = \log_k n$. For any linear code $C : F^k \to F^\ell$, where $\ell = \text{poly}(k)$, $|F| = \text{polylog}(n)$. There exists an Interactive Proof of ε-Proximity for $TVAL$ on C^m, with perfect completeness, soundness $1/2$, query complexity $(\varepsilon)^{1+o(1)}$ and 3 messages. For $|J| = O(\varepsilon \cdot n \cdot \log n)$, the communication complexity is $(n \cdot \sqrt{\varepsilon}) \cdot (1/\varepsilon)^{o(1)}$, and the verifier runtime is $((1/\varepsilon) + (n \cdot \sqrt{\varepsilon}))^{1+o(1)}$. The honest prover runtime is $\text{poly}(n)$.*

Corollary 3.3. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0,1]$, take $k = \log n$ and $m = \log_k n$. For any linear code $C : F^k \to F^\ell$, where $\ell = \text{poly}(k)$, $|F| = \text{polylog}(n)$, there exists an Interactive Proof of ε-Proximity for $TVAL$ on C^m, with perfect completeness, soundness $1/2$, query complexity $(1/\varepsilon)^{1+o(1)}$, and $O(\log(1/\varepsilon)/\log\log n)$ messages. For $|J| = O(\varepsilon \cdot n \cdot \log n)$, the communication complexity is $\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)}$, and the verifier runtime is $((1/\varepsilon) + (\varepsilon \cdot n))^{1+o(1)}$. The honest prover runtime is $\text{poly}(n)$.*

Protocol Intuition. We highlight some of the main ideas from this protocol, see also the discussion in the introduction (further details are in Sections 3.2 and 3.3). We begin with some high-level background. Let F be a finite field. Let $C : F^k \to F^\ell$ be a linear code (where $\ell = \text{poly}(k)$). The (linear) Tensor Code $C^2 : F^{k^2} \to F^{\ell^2}$ is obtained by treating the input message as a matrix $X \in F^{k \times k}$, encoding each of X's k rows using the base code C, and then encoding each of the ℓ resulting columns using C. This yields a codeword $C^2(X) \in F^{\ell \times \ell}$, which we also view as a matrix. We will use the code C^2 for our high level overview here. More generally, the m-th tensor C^m is obtained by treating the input message as a matrix $X \in F^{k \times k^{m-1}}$, encoding each of X's k rows using the $(m-1)$-th tensor C^{m-1}, and then encoding each of the resulting ℓ^{m-1} columns using C (see Section 3.1 for more background on Tensor Codes).

[9] We can use our general results to obtain an Interactive Proof of Proximity to verify the values of any code C whose encoding can be computed in low depth.

The Interactive Proof of Proximity, on input X, should accept if $C^2(X)_{|J} = \vec{v}$, and reject if X is at Hamming distance at least ε from every X' s.t. $C^2(X')_{|J} = \vec{v}$. For this informal exposition, we take $\varepsilon = k^{1/2}/n$ and $|J| = t = O(\varepsilon \cdot n \cdot \log n) = \tilde{O}(k^{1/2})$. For convenience, we'll use $d = \varepsilon \cdot n$ to denote the *absolute* Hamming distance. We aim for $o(k^2)$ query and communication complexities (sublinear in the input size k^2).

Viewing $C^2(X)$ as an $\ell \times \ell$ matrix, for each $j \in J$, the (honest) prover sends to the verifier the entire column of $C^2(X)$ that contains coordinate j. In fact, it suffices to send the first k coordinates of each such column, as the remaining column coordinates are simply the encoding of the first k coordinates using the code C. Let $Y \in F^{k \times |J|}$ be the resulting matrix (as computed by an honest prover). The verifier receives from the prover some $Y' \in F^{k \times |J|}$ (possibly the prover is cheating and $Y' \neq Y$), and checks that Y' is consistent with the values claimed in $\vec{v}$; namely that for each $j \in J$, the encoding of the first k coordinates in the column of Y' that contains coordinate j, is consistent with the value that $\vec{v}$ gives for coordinate j.

For completeness, when $Y = Y'$ and $X \in TVAL(J, \vec{v})$, the verifier will accept in the above checks. When X is ε-far from $TVAL(J, \vec{v})$, however, the prover must "cheat" in many of Y''s columns in order to avoid detection. Observe that the i-th row of Y contains the encoding of the i-th row of X in a coordinate set $J[1]$.[10] Thus, from Y' we get k "new" claims about the encodings of X's rows in a subset of coordinates. Each of these claims is an instance of $TVAL$ on the base code C: Using $X[i, *]$ and $Y'[i, *]$ to denote the i-th rows of X and Y' (respectively), for each $i \in [k]$ we get a new instance $(X[i, *], J[1], Y'[i, *])$ of $TVAL$ on C.

When the original instance X is not in $TVAL(J, \vec{v})$ on C^2, at least one of the k new instances will not be in $TVAL$ on C (because Y' is consistent with the values claimed in $\vec{v}$). Let d_i denote the (absolute) distance of the i-th new instance from $TVAL$ on C. Since the original instance is at absolute distance $d = \varepsilon \cdot n = k^{1/2}$ from $TVAL$ on C^2, we get that $\sum_{i \in [k]} d_i \geq d = k^{1/2}$. We want the verifier to efficiently test the k new instances, in query complexity that is smaller than the query complexity needed to test the original instance. One immediate challenge is that it might be the case that *all but one* of the instances are "kosher" (i.e. in $TVAL$), and the prover's cheating is concentrated in a single instance specified by some row $i^* \in [k]$. The verifier doesn't know i^*, and so it must test the validity of all k of the new claims. We deal with this challenge using new ideas described below.

When The Prover Cheats on a Single Row. First, assume that the prover's cheating is concentrated in a single row i^* with $d_{i^*} = d$ (we discuss the more general case later). The verifier sends to the prover coefficients specifying a random linear combination $\vec{z} \in F^k$ of X's rows. The (honest) prover sends back $\vec{x} = \vec{z} \cdot X \in F^k$. The verifier receives some $\vec{x}'$, and verifies $C(\vec{x}')_{J^{-(0)}} = \vec{y}' = \vec{z} \cdot Y'$ (otherwise the verifier rejects). This additional test maintains perfect completeness, because C is a linear code. For soundness, we consider the case where every other row $i \neq i^*$ of Y' contains the correct values of the i-th row of X. W.h.p. row i^* will have a non-zero coefficient in the linear combination. When this is the case, we get that $(\vec{x}, J^{-(0)}, \vec{y}')$ is also at absolute distance $d_{i^*} = d$ from $TVAL$. Thus the absolute Hamming distance between the "real" $\vec{x}$, and $\vec{x}'$ sent by the cheating prover, is at least d (since $(\vec{x}', J^{-(0)}, \vec{y}') \in TVAL$).

To guarantee soundness, the verifier now checks a few randomly chosen coordinates of $\vec{x}'$, to see that it is close to the "real" $\vec{x}$. If the prover is cheating, then the (absolute) Hamming distance between $\vec{x}$ and $\vec{x}'$ is at least d, so it suffices to check $O(k/d) = O(k^{1/2})$ coordinates of $\vec{x}'$. Each

[10] For each $j \in J$, we view it as a pair $(j[0], j[1]) \in \ell \times \ell$, and $J[1]$ contains the coordinate $j[1]$.

coordinate in $\vec{x}$ is a linear combination of k coordinates in X, and so the total query complexity is $O(k^{3/2})$. The communication complexity is $\tilde{O}(k^{3/2})$ for sending Y' (sending $\vec{z}$ and $\vec{x}'$ takes only $O(k)$ bits of communication).

When The Prover Cheats on Many Rows. More generally, the prover's cheating might be spread over many (or all) rows. We show that there always exists a set I of rows, such that $\forall i \in I, d_i = \tilde{\Omega}(d/|I|)$. We saw above, that when the cheating is all on one row ($|I|$=1), then w.h.p. a random linear combination that contains this row specifies an instance that is at distance $\Omega(d)$ from *TVAL* on C. In fact, this is but an example of a more general (and more powerful) phenomenon; We show that, with high probability, a random linear combination $\vec{z}$ of the rows, which contains one or more of the rows in the set I, specifies an instance of distance $\tilde{\Omega}(d/|I|)$. This result follows from a new lemma about distances between linear subspaces, which may be of independent interest (see Lemma 1.6 in the introduction).

With this new insight in mind, suppose that the verifier knows $|I|$ (but doesn't know which rows are in I). The verifier takes a random linear combination of $O(k/|I|)$ rows. W.h.p. the linear combination will include at least one row of I, and by the above it will specify an instance of distance $\tilde{\Omega}(d/|I|)$. There is a tradeoff here: the larger I is, the smaller the distance of the resulting instance, but *the "locality" of $\vec{x}$ is reduced*: each bit of $\vec{x}$ depends on only $O(k/|I|)$ bits of the original input X. When the cheating prover sends $\vec{x}'$, we now know that $\Delta_{absolute}(\vec{x}, \vec{x}') = \tilde{\Omega}(d/|I|)$. After receiving $\vec{x}'$, the verifier can test $\tilde{O}(k/(d/|I|)) = \tilde{O}(|I| \cdot k/d)$ random coordinates, and w.h.p the prover's cheating will be caught. Testing each coordinate requires $O(k/|I|)$ queries to the input X, and so the total query complexity remains $\tilde{O}(k^2/d) = \tilde{O}(k^{3/2})$, regardless of $|I|$.

The above assumed that the verifier knows $|I|$. In reality, the verifier doesn't know what $|I|$ is, but it can (approximately) cover all possibilities by running $\log k$ instantiations of the protocol, and trying all possible powers of 2. This blows up the query and communication complexities by a $\log k$ factor, giving a complete and sound sublinear protocol for *TVAL* on Tensor codes C^2. The query and communication complexities are $\tilde{O}(k^{3/2}) = \tilde{O}(n^{3/4})$.

Composing Tensor Reductions. More generally, the above protocol for *TVAL* can be used to provide a reduction from testing *TVAL* on a linear tensor code $C_1 \otimes C_2$, to testing ($\log k$ instances of) *TVAL* on the C_2. We thus call it the "Tensor Reduction" protocol. (see Sections 3.2 and 3.3 for more details).

Starting with an instance of *TVAL* for a high-power Tensor code C^m, we can use $C^m = C \otimes C^{m-1}$, and repeatedly compose the Tensor Reduction Protocol, reducing the tensor power more and more. This improves the communication complexity (but increases the number of rounds). A theorem statement for the Interactive Proof of Proximity for *TVAL* (and the parameter trade-off) is in Section 3.3. The full protocol is in Figure 2.

Overview: Distance between Linear Subspaces. One main technical tool in the Interactive Proof of Proximity for *TVAL* is a new lemma about the distance between linear subspaces, which may be of independent interest. See Lemma 1.6 in the introduction for a formal statement. The lemma shows that, for any two subspaces $S, T \subseteq F^k$, if there exists even a single point $s \in S$ that is far from *every* point in T, then w.h.p. over the choice of a random point in S, it is also far from every point in T. We use the lemma to show that, in the Tensor Reduction Protocol described above, w.h.p. the distance from *TVAL* of a random linear combination of instances specified by a

set I of rows (rows of X and of Y') is roughly equal to the maximum distance from $TVAL$ of any row in I (up to a constant factor loss).

The proof is below. We note that there is some loss between the distance ε of s from T and the distance guarantee $\varepsilon/2$ for a random point in S. Proving a tighter guarantee (or showing that $\varepsilon/2$ is tight) remains an interesting open question (but would not substantially affect our results).

Proof of Lemma 1.6. We can pick a uniformly random point in S by the following process: pick a random point $\vec{r} \in_R S$, if $\vec{r} = \vec{s}$ then output $\vec{r}$, otherwise, output one of the $|F| - 1$ points on the line between $\vec{r}$ and $\vec{s}$, excluding $\vec{s}$, uniformly at random.

With this sampling procedure in mind, we will show below that for any line L passing through $\vec{s}$, there can be at most one point on L that is at distance less than $\varepsilon/2$ from T. Excluding $\vec{s}$, there are $(|F| - 1)$ points on each such line, and we conclude (because the above sampling procedure produces a uniformly random output) that the fraction of points in S that are at distance less than $\varepsilon/2$ from T is at most $(|F| - 1)^{-1}$.

It remains to show that for any line L passing through $\vec{s}$, there can be at most one point at distance less than $\varepsilon/2$ from T. Assume for contradiction that there are two points, $\vec{u}, \vec{v} \in L$ that are at distance less than $\varepsilon/2$ from T. This means that there are points $\vec{t_u}, \vec{t_w} \in T$ s.t. $\Delta(\vec{u}, \vec{t_u}), \Delta(\vec{w}, \vec{t_w}) < \varepsilon/2$. Since $\vec{u}, \vec{w}, \vec{s}$ are on the same line L, there exists $a \in F$ s.t.

$$\vec{s} = \vec{u} + a \cdot (\vec{w} - \vec{u})$$

Now consider the point

$$\vec{t_s} = \vec{t_u} + a \cdot (\vec{t_w} - \vec{t_u})$$

we get that $\vec{t_s} \in T$, because $\vec{t_u}, \vec{t_w} \in T$ and T is a linear subspace.

For any coordinate $i \in \{1, \ldots, n\}$ s.t. both $\vec{u}[i] = \vec{t_u}[i]$ and $\vec{w}[i] = \vec{t_w}[i]$, we have that $\vec{s}[i] = \vec{t_s}[i]$. Since the fraction of coordinates on which $\vec{u}$ and $\vec{t_u}$ disagree is smaller than $\varepsilon/2$, and the fraction of coordinates on which $\vec{w}$ and $\vec{t_w}$ disagree is less than $\varepsilon/2$, we get that $\vec{s}$ and $\vec{t_s}$ disagree on less than an ε fraction of the coordinates. This is a contradiction, because $\vec{s}$ is at distance ε from T (and $\vec{t_s} \in T$). □

Organization of this Section. Definitions and background about codes and Tensor codes are in Section 3.1. The Tensor Reduction Protocol and its analysis are in Section 3.2. In Section 3.3, we give the Interactive Proof of Proximity for $TVAL$. Finally, in Section 3.4, we show that $PVAL$ is complete for $\mathcal{IPP}$ and prove Theorem 1.1.

3.1 Definitions and Background

Error-Correcting Codes. Let F be a finite field, and $k, \ell \in \mathbb{N}$. A linear code is a linear function $C : F^k \to F^\ell$, where k is the code's message length. For $\vec{x} \in F^k$, its encoding is $C(x)$, a codeword of the code C. The distance of a code is the minimal Hamming distance between two codewords. Since C is linear, there exists a matrix $G \in F^{\ell \times \ell}$ such that $C(\vec{x}) = F \cdot \vec{x}$.

Tensor Codes and $TVAL$. We proceed with background on Tensor Codes, following the overview in [Mei10] (with some changes in notation). See e.g. [MS78] for further details.

Definition 3.4 (Tensor Code)**.** For a finite field F, let $C_1 : F^{k_1} \to F^{\ell_1}$ and $C_2 : F^{k_2} \to F^{\ell_2}$ be two linear codes. The tensor code $(C_1 \otimes C_2) : F^{k_1 \cdot k_2} \to F^{\ell_1 \cdot \ell_2}$ is also a linear code. It encodes a message $\vec{x} \in F^{k_1 \cdot k_2}$ as follows: view $\vec{x}$ as a $k_1 \times k_2$ matrix X. Encode each row of X using C_2, and then encode each column in the resulting $k_1 \times \ell_2$ matrix using C_1. The resulting matrix in $F^{\ell_1 \times \ell_2}$ is the encoding of X by the code $C_1 \otimes C_2$ (this encoding may be viewed as either a matrix, or a vector in $F^{\ell_1 \cdot \ell_2}$).

We note that a matrix $Y \in F^{\ell_1 \times \ell_2}$ is a codeword of $C_1 \otimes C_2$ if and only if all the rows of Y are codewords of C_2, and all columns are codewords of C_1.

Definition 3.5 (Tensor Power Code)**.** For a finite field F and $m \in \mathbb{N}$, let $C : F^k \to F^\ell$ be a linear code. We define C^m, the m-th tensor of C as follows: for $m = 1$, we take $C^1 = C$. For $m > 1$, we recursively define $C^m = C \otimes C^{m-1}$. We get $C^m : F^{k^m} \to F^{\ell^m}$, and note that C^m is also a linear code. We call C^m the *m-th tensor power* of C, and refer to m as the *tensor-power* of C^m.

Definition 3.6 (*TVAL*)**.** *TVAL* is a language on pairs $(X, (C, m, J, \vec{v}))$, where X is the explicit input, and all other inputs are implicit. X specifies a message for the tensor code $C^m : F^n \to F^N$. $J \subseteq [N]$ is a set of codeword coordinates, and $\vec{v} \in F^{|}J|$ is a set of values. The input is in *TVAL* if and only if $C^m(X)_{|J} = \vec{v}$.

We say that $(X, J, \vec{v})$ is an instance of *TVAL* on C^m, if $(X, (C, m, J, \vec{v}) \in TVAL$. We say that X is ε-far from *TVAL* on C^m, if X is ε-far from $TVAL(C, m, J, \vec{v})$. For $j \in J$, we refer to "the j-th element of $\vec{v}$": The value (in F) that $\vec{v}$ specifies for $C^m(X)[j]$.

Finally, we also refer to a variant of *TVAL*, whose inputs are pairs $(X, (C_1, C_2, J, \vec{v}))$, where C_1, C_2 are linear codes, and the tensor code $C_1 \otimes C_2$ takes the place of C^m above.

Multi-Dimensional Arrays. When working with Tensor powers, messages are in F^{k^m} and codewords are in F^{ℓ^m}. We thus consider m-dimensional arrays, and use $F^{k \times k \ldots k}$ to denote the set of m-dimensional arrays over F, of size k in each dimension (where m is clear from the context). For an array $X \in F^{k \times k \times \ldots \times k}$, and an index $i \in [k] \times [k] \times \ldots \times [k]$, we use $X[i]$ to refer to the i-th cell in the array X. Finally, for a vector $\vec{z} \in F^k$, we use $\vec{z} \cdot X$ to refer to vector-array multiplication, where we view X as a matrix in $F^{k \times k^{m-1}}$ and

$$\vec{z} \cdot X = \sum_{i \in [k]} \vec{z}[i] \cdot X[i, *] \in F^{k^{m-1}}$$

PVAL **and the Reed-Muller Code.** We also define the Reed-Muller code used in *PVAL* (or rather a variant of Reed-Muller codes known as the low-degree extension). We remark that *PVAL* (see Definition 3.1) is a specific case of *TVAL*, where C^m is the Reed-Muller code.

Definition 3.7 (Reed-Muller/Low-Degree Extension Code)**.** For a finite field F, a number of variables $m \in \mathbb{N}$, and a degree bound $k \in \mathbb{N}$, where $k < |F|$. We refer to an m-variate degree-k *Reed-Muller code* over F as follows. We note that this is somewhat non-standard, and alternatively this is known as the *low-degree extension* code.

The code is $C_{RM} : F^{k^m} \to F^{|F|^m}$. Fix a subset H of F of size k. A message $X \in F^{k^m}$ specifies a (unique) m-variate polynomial p_X of degree $k-1$ in each variable, which takes the values specified by X on H^m. I.e., each coordinate in X is associated with a vector in H^m in the natural way, and specifies the value of p_X at that coordinate. The encoding of X is the evaluation of p_X at *all* coordinates in F^m.

We note that C_{RM} is a linear code, and is in fact the m-th tensor of a linear code C (the Reed-Solomon code, which we do not define here).

3.2 Tensor Reduction Protocol

We proceed with a full specification of the Tensor Reduction Protocol. An overview was given above. We then state and prove Theorem 3.8, which states the completeness and soundness properties of the protocol.

Tensor Reduction Protocol on shared input $(C_1, C_2, J, \vec{v}, \kappa)$ and prover input X

The input specifies an instance X of *TVAL* on $(C_1 \otimes C_2, J, \vec{v})$, where $C_1 : F^{k_1} \to F^{\ell_1}$ and $C_2 : F^{k_2} \to F^{\ell_2}$ are linear codes. The shared input $(C_1, C_2, J, \vec{v}, \kappa)$ is given (explicitly) to the verifier and the prover. The message X is given only to the prover, and never accessed by the verifier.

Treat X as a $k_1 \times k_2$ matrix, and each $j \in J$ as $j = (j_1, j_2) \in [k_1] \times [k_2]$. $\mathcal{P}$ and $\mathcal{V}$ take $J_2 = \{j_2 : (j_1, j_2) \in J\}$ to be the set of columns occurring in J.

The protocol proceeds in two rounds:

1. $\mathcal{P} \to \mathcal{V}$: for each row $j_1 \in [k_1]$ of X, send its encoding (by C_2) restricted to coordinates J_2. Let Y be the $k_1 \times |J|$ matrix of these restricted row encodings

   $\mathcal{V}$: receive $Y' \in F^{k_1 \times |J|}$, reject if $\exists (j_1, j_2) \in J :\ C_1(Y'[*, j_2])[j_1] \neq j$-th element of $\vec{v}$
2. $\mathcal{V} \to \mathcal{P}$: for each $a \in [\log(k_1/\kappa) + 1]$, send a uniformly random vector $\vec{z}_a \in F^{k_1}$ of Hamming weight $2^a \cdot \kappa$

The output is $(\log(k_1/\kappa) + 1)$ tuples $\{(a, \vec{z}_a, J_2, \vec{z}_a \cdot Y')\}_{a \in [\log(k_1/\kappa)+1]}$

Figure 1: Tensor-Reduction Protocol

Theorem 3.8. *The Tensor-Reduction Protocol of Figure 1, on shared input $(C_1, C_2, J, \vec{v}, \kappa)$ and prover input X, has communication complexity $O(|J| \cdot k_1 \cdot \log k_1 \cdot \log |F|)$ and one round of communication. The honest prover runs in time $\mathrm{poly}(\ell_1 \cdot \ell_2, \log |F|)$, and the verifier runs in time $\mathrm{poly}(|J|, \ell_1, \log |F|)$.*

The protocol's output is $(\log(k_1/\kappa) + 1)$ tuples $\{(a, \vec{z}_a, J_2, \vec{z}_a \cdot Y')\}_{a \in [\log(k_1/\kappa)+1]}$. For each tuple, we take $X_a = \vec{z}_a \cdot X$, $\vec{v}_a = \vec{z}_a \cdot Y'$, and the instance $W_a = (X_a, J_2, \vec{v}_a)$ to be an instance for TVAL on C_2. The output satisfies completeness, soundness, and bounded locality:

- *Perfect Completeness: If $(X, J, \vec{v})$ is a YES instance for TVAL on $C_1 \otimes C_2$, and the prover honestly follows the protocol, then the verifier does not reject, and for every $a \in [\log(k_1/\kappa)+1]$ simultaneously, W_a is a YES instance of TVAL on C_2.*
- *Soundness: If $(X, J, \vec{v})$ is ε-far from TVAL on $C_1 \otimes C_2$, then for any cheating prover $\mathcal{P}'$, with all but $((|F| - 1)^{-1} + e^{-\kappa/(4 \log k_1)})$ probability over $\mathcal{V}$'s coins, either $\mathcal{V}$ rejects, or there* exists *some $a^* \in [\log(k_1/\kappa) + 1]$ s.t. W_{a^*} is $(\varepsilon \cdot 2^{a^*}/4)$-far from TVAL on C_2.*
- *Bounded Locality: For each $a \in [\log(k_1/\kappa) + 1]$, in the a-th instance W_a, each coordinate of X_a is a linear combination of $\tau_a = 2^a \cdot \kappa$ coordinates of X.*

Proof. The communication complexity and running times of $\mathcal{P}$ and $\mathcal{V}$ follow by construction, as does the bounded locality property.

Perfect Completeness. If $(X, J, \vec{v})$ is a YES instance of $TVAL$ on $C_1 \otimes C_2$, then we know that $(C_1 \otimes C_2)(X)_{|J} = \vec{v}$. Since $C_1 \otimes C_2$ is a tensor code, for each $j = (j_1, j_2) \in J$:

$$(C_1 \otimes C_2)(X)[j] = C_1\Big(C_2(X[0,*])[j_2], \ldots, C_2(X[k_1 - 1, *])[j_2]\Big)[j_1]$$

and we conclude that in Step 1, when $\mathcal{P}$ sends the matrix Y, where the i-th row of Y is $C_2(X[i,*])_{|J_2}$, it will indeed be the case that for all $j \in J$:

$$C_1(Y[*, j_2])[j_1] = (C_1 \otimes C_2)(X)[j] = j\text{-th element of } \vec{v}$$

and so the verifier will not reject. Moreover, for each $a \in [\log(k_1/\kappa) + 1]$ and for any linear combination $\vec{z}_a$ chosen by the verifier, by linearity of the code C_2, we have:

$$C_2(X_a)_{|J_2} = C_2(\vec{z}_a \cdot X)_{|J_2} = \sum_{i \in [k_1]} \vec{z}_a[i] \cdot C_2(X[i,*])_{|J_2} = \vec{z}_a \cdot Y = \vec{v}_a$$

and so $(X_a, J_2, \vec{v}_a)$ is a YES instance of $TVAL$ on C_2.

Soundness. Suppose that $(X, J, \vec{v})$ is ε-far from $TVAL$ on $C_1 \otimes C_2$. We use $d = \varepsilon \cdot k_1 \cdot k_2$ to denote the absolute Hamming distance. We begin by showing that (unless $\mathcal{V}$ rejects), there must exist some "large enough" set I of input rows, s.t. for each row $i \in I$, the input values in that row $X[i,*]$ are "far enough" (in Hamming distance) from every X'_i whose encoding is consistent with $Y'[i,*]$ (in the coordinates in J_2).

Towards this, for each row $i \in [k]$ of the input X, take d_i to be the absolute Hamming distance of $X[i,*]$ from the closest $X'_i \in F^{k_2}$ for which $C_2(X'_i)_{|J_2} = Y'[i,*]$.

Claim 3.9. *If $\mathcal{V}$ does not reject in Step 1, then there exists an integer $b \in \{0, \ldots, \log k_1\}$, and a subset $I \subseteq [k_1]$, s.t. $\forall i \in I,\ d_i \geq d/2^{b+1}$ and $|I| \geq 2^b/4 \log k_1$.*

Proof. If in Step 1, $\mathcal{V}$ does not reject, then for each $j = (j_1, j_2) \in J$, the encoding (by C_1) of $Y'[*, j_2]$, in coordinate j_1, matches the value claimed in $\vec{v}$. In particular, for any X' whose row-encodings are all consistent with the matrix Y' sent by $\mathcal{P}'$, it must also be the case that the encoding of X' in its entirety (by $C - 1 \otimes C_2$) is consistent with the values in $\vec{v}$. Any such X' must be at distance at least d from X, and thus $\sum_{i \in [k_1]} d_i \geq d$.

Now suppose for contradiction that for all $b \in \{0, \ldots, \log k\}$ the number of rows i s.t. $d_i \in [d/2^{b+1}, d/2^b)$ is less than $2^b/4 \log k$. Then we have:

$$\begin{aligned} \sum_{i=1}^{k_1} d_i &< \left(\sum_{b=0}^{\log k - 1} (d/2^b) \cdot (2^b/4 \log k_1) \right) + \Big((d/2k_1) \cdot k_1 \Big) \qquad (1) \\ &= (\log k_1 + 1) \cdot (d/4 \log k_1) + (d/2) \\ &= d \end{aligned}$$

where the two summands in the right-hand side of inequality (1) are the contributions of rows for which $d_i \geq d/2k_1$ (left summand), and the remaining rows (right summand, note that there are at most k such rows). This is a contradiction because $\sum_{i \in [k_1]} d_i \geq d$ (see above). □

In what follows, we assume that $\mathcal{V}$ does not reject in Step 1. In this case, Claim 3.9 shows that there must be a "large enough" set of rows that are "far enough" from being consistent with Y. We now show that w.h.p. over $\mathcal{V}$'s choice of non-zero coordinates for the vectors $\{\vec{z}_a\}_{a=\in[\log(k_1/\kappa)+1]}$ in Step 2(i), for at least one value of a, the set of non-zero coordinates will include at least one such "far enough" row.

Claim 3.10. *In Step 2 of the protocol, for $a \in [\log(k_1/\kappa) + 1]$, let I_a be the set of non-zero coordinates in $\vec{z}_a$ (this set is of size $2^a \cdot \kappa$).*

Take b as guaranteed by Claim 3.9 and $a^* = \min(\log(k_1/\kappa), \log k_1 - b)$. With all but $e^{-\kappa/4 \log k_1}$ probability over $\mathcal{V}$'s choice of I_{a^*}, there exists $i^* \in I_{a^*}$ s.t. $d_{i^*} \geq d \cdot 2^{a^*}/2k_1$.

Proof. The claim follows from a "birthday paradox": There are k_1 rows in total. We know (by Claim 3.9) that for some $b \in \{0, \ldots, \log k_1\}$ there is a set I of at least $2^b/4 \log k_1$ rows s.t. for all $i \in I$ it holds that $d_i \geq d/2^{b+1}$. When we pick $\min(k_1, \kappa \cdot k_1/2^b)$ random rows to include in I_{a^*}, with all but $(1 - \frac{|I|}{k_1})^{\kappa \cdot k_1/2^b} \leq e^{-\kappa/4 \log k_1}$ probability, at least one of the rows will be in I (we take the minimum between k_1 and $\kappa \cdot k_1/2^b$ because there are only k_1 rows, and if we pick all of them to be in I_{a^*} then we also picked all rows in I). Taking $a^* = \min(\log(k_1/\kappa), \log k_1 - b)$ guarantees that I_{a^*} of size $\kappa \cdot 2^{a^*}$ is large enough. □

Claim 3.11. *Take a^* as guaranteed by Claim 3.10. With all but $((|F|-1)^{-1} + e^{-\kappa/4 \log k_1})$ probability over $\mathcal{V}$'s choice of $\vec{z}_{a^*}$, it holds that $X_{a^*} = \vec{z}_{a^*} \cdot X$ is at absolute Hamming distance at least $d \cdot 2^{a^*}/4k_1$ from every $X' \in F^{k_2}$ for which $C_2(X')_{|J_2} = \vec{v}_{a^*} (= z_{a^*} \cdot Y)$*

Proof. Take T to be the linear subspace of messages in F^{k_2}, whose encodings under C_2 are 0 on J_2, i.e. $T = \{\vec{t} \in F^{k_2} : C_2(\vec{t})_{|J_2} = \vec{0}\}$. For each row $i \in \{1, \ldots, k_1\}$, take A_i to be the set of "shift" vectors, s.t. adding any of them to the i-th row of the input X makes it consistent with the i-th row of Y'. I.e. $A_i = \{\vec{s} \in F^{k_2} : C_2(X[i, *] + \vec{s})_{|J_2} = Y'[i, *]\}$. By definition, d_i is Hamming weight of the minimal-weight vector in A_i. Observe that A_i is an affine shift of T: for any $\vec{s} \in A_i$, we have $A_i = T + \vec{s}$. For each row i, let $\vec{s}_i$ be some arbitrary vector in A_i.

We can think of $\vec{z}_{a^*}$ as begin sampled by a two-stage process: First, we pick a set I_{a^*} of $2^{a^*} \cdot \kappa$ coordinates, and set all other coordinates to 0. By Claim 3.10, with all but $e^{-\kappa/4 \log k_1}$ probability over the choice of I_{a^*}, there exists $i^* \in I_{a^*}$ s.t. $d_{i^*} \geq 2^{a^*} \cdot d/2k_1$. We assume for the remainder of the proof that this is the case (and later take a Union Bound with the $e^{-\kappa/4 \log k_1}$ probability of failure).

In the second stage of choosing $\vec{z}_a$, the coordinates in I_{a^*} are given uniformly random values in F. This yields a random linear combination $X_{a^*} = \vec{z}_{a^*} \cdot X$ of the rows of X, and a linear combination $\vec{v}_{a^*} = \vec{z}_{a^*} \cdot Y'$ of the rows of Y'. We want to lower bound the Hamming distance between X_{a^*} and every X' s.t. $C_2(X')_{|J_2} = \vec{v}_{a^*}$.

Once $\vec{z}_{a^*}$ is fixed, let A be the set of "shift" vectors that make X_{a^*} consistent with $\vec{v}_{a^*}$. Our goal is lower bounding the minimal weight of a vector in A. As above, A is an affine shift of T: For any shift vector $\vec{s} \in F^{k_2}$ s.t. $C_2(X_{a^*} + \vec{s})_{|J_2} = \vec{v}_{a^*}$, we have $A = T + \vec{s}$. The minimal weight of a vector in A equals the Hamming distance of any $\vec{s} \in A$ from T.

Now examine the vector $\vec{s}_{a^*} = \sum_{i \in [k]} \vec{z}_{a^*}[i] \cdot \vec{s}_i$. We get that $\vec{s}_{a^*}$ is a uniformly random vector in the linear space spanned by $\{\vec{s}_i\}_{i \in I_{a^*}}$. Recall from above that w.h.p. we have $i^* \in I_{a^*}$ s.t. d_{i^*}, the Hamming distance between $\vec{s}_{i^*}$ and T, is at least $d \cdot 2^{a^*}/2k_1$. By Lemma 1.6, we conclude that with all but $(|F| - 1)^{-1}$ probability over the choice of coefficients in $\vec{z}_{a^*}$, the distance between $\vec{s}_{a^*}$

and T is at least $d_{i^*}/2 \geq d \cdot 2^{a^*}/4k_1$. Since $\vec{s}_{a^*} \in A$, we conclude that the minimal weight of any vector in A is at least $2^{a^*} \cdot d/4k_1$. □

Claim 3.11 shows that w.h.p. the *absolute* Hamming distance of X_{a^*} from any input consistent with the prover's claims is at least $d^* = (d \cdot 2^{a^*}/4k_1)$. X_{a^*} is of length k_2, and so the *fractional* Hamming distance is:

$$\varepsilon^* = d^*/k_2 \geq (d \cdot 2^{a^*}/4k_1)/k_2 = (\varepsilon \cdot k_1 \cdot k_2) \cdot 2^{a^*}/(4k_1 \cdot k_2) = \varepsilon \cdot 2^{a^*}/4$$

□

3.3 Interactive Proof of Proximity for *TVAL*

In this section we give an Interactive Proof of Proximity for *TVAL*.

Theorem 3.12. *Take $\varepsilon \in (0,1)$ and $n, k, m, r \in \mathbb{N}$, where $n = k^m$, a finite field F, and a linear code $C : F^k \to F^\ell$, such that $10r \leq |F| \leq (1/\varepsilon)$, $r \leq k$, and $k^r \leq (1/\varepsilon)$.*

For these parameters, the Protocol of Figure 2 is an Interactive Proof of ε-Proximity for TVAL on C^m. The protocol has perfect completeness, soundness $1/2$, query complexity $(1/\varepsilon)^{1+o(1)}$, communication complexity $(k^{m-r} + |J| \cdot k) \cdot (1/\varepsilon)^{o(1)}$, and the number of messages is $(2r+1)$. The honest prover runtime is $\mathrm{poly}(n)$, and the verifier runtime is $((1/\varepsilon) + n/k^r + |J| \cdot k) \cdot n^{o(1)}$.

Corollary 3.2 (stated in the overview above) is obtained from Theorem 3.12 by setting $k = \sqrt{1/\varepsilon}$, $m = \log_k n$, $r = 1$ and $|F| = \mathrm{polylog}(n)$. Corollary 3.3 (stated in the overview above) is obtained from Theorem 3.12 by setting $k = \log n$, $m = \log_k n$, $r = (\log(1/\varepsilon)/\log k)$ and $|F| = \mathrm{polylog}(n)$.

Recall from the overview that we obtain Theorem 3.12 using sequential composition of the Tensor Reduction Protocol. We begin with a high-level outline of the protocol. The Interactive Proof of Proximity is in Figure 2, and the proof of Theorem 3.12 follows.

Protocol Outline. The input is an instance $(X, J, \vec{v})$ for $TVAL$ on C^m. The verifier has explicit access to $(J, \vec{v})$, but only implicit access to the message X (via an oracle). We treat $X \in F^{k^m}$ as an m-dimensional array in $F^{k \times k \times \cdots \times k}$. The protocol proceeds in r phases $0, 1, \ldots, r-1$ (we treat r as a parameter). In each phase the code's tensor-power (initially m) shrinks by 1. Namely, in each phase s, we start with a collection of phase-s instances for $TVAL$ on C^{m-s}, and end up with a (larger) collection of phase-$(s+1)$ instances for $TVAL$ on C^{m-s-1}. This is accomplished by running the Tensor Reduction Protocol on each of the phase-s instances (in parallel), with $C_1 = C$ and $C_2 = C^{m-s-1}$ (so $C^{m-s} = C_1 \otimes C_2$). Taking κ to be a security parameter (fixed below), the output from each run of the Tensor Reduction Protocol specifies a collection of $(\log(k/\kappa)+1)$ instances for $TVAL$ on C^{m-s-1}. We add all of these to the set of phase-$(s+1)$ instances, and proceed to phase $s+1$. After r phases, the verifier runs an additional test on each of the phase-r instances (these are instances for $TVAL$ on C^{m-r}), and accepts only if all of these instances pass this additional test.

In the Tensor Reduction Protocol, on input $(X, J, \vec{v})$, the verifier does not access the "message" part X of the input. Similarly, we also want to avoid having the verifier access the message part of instances, because we want the verifier's running time to be sublinear in $|X|$. Thus, each instance is specified by a tuple, and the set of phase-s instances is specified by a set $\mathcal{U}_s$ of tuples. Each tuple in $\mathcal{U}_s$ is of the form $((\vec{z}_1, \ldots, \vec{z}_s), (a_1, \ldots, a_s), J_s, \vec{v}_s)$, where $\vec{z}_1, \ldots, \vec{z}_s \in F^k$, and $a_1, \ldots, a_s \in$

$[\log(k/\kappa)+1]$. Each such tuple specifies an instance $(X_s, J_s, \vec{v}_s)$ for *TVAL* on C^{m-s}, where the message $X_s \in F^{k^{m-s}}$ is given by:

$$X_s = \vec{z}_s \cdot (\vec{z}_{s-1} \cdot \ldots \cdot (\vec{z}_1 \cdot X)) \tag{2}$$

We begin in phase 0, with a tuple-set $\mathcal{U}_0 = \{(\lambda, \lambda, J, \vec{v})\}$, where λ is the empty string, and so $\mathcal{U}_0$ specifies only the original instance $(X, J, \vec{v})$.

In phase s, each of the Tensor Reduction Protocol's (parallel) runs generates a collection of outputs $\{(a, \vec{z}_{s+1,a}, J_{s+1}, \vec{v}_{s+1,a})\}_{a\in[\log(k/\kappa)+1]}$. For each $a \in [\log(k/\kappa)+1]$, we add to $\mathcal{U}_{s+1}$ the tuple $((\vec{z}_1, \ldots, \vec{z}_s, \vec{z}_{s+1,a}), (a_1, \ldots, a_s, a), J_{s+1}, \vec{v}_{s+1,a})$. The vector $(a_1, \ldots, a_s, a)$ provides an "index", keeping track of each tuple's position within the $(\log(k/\kappa)+1)$ outputs generated by the Tensor Reduction protocols in phases $0, \ldots, s$.

Throughout these r phases, the messages X_s are never accessed by the verifier (implicitly or explicitly). X is only accessed in the end, when $\mathcal{V}$ runs its additional test on each instance in $\mathcal{U}_s$.

Interactive Proof of ε-Proximity for *TVAL* on C^m

The implicit input is $X \in F^{k^m}$, the explicit input is $(F, C, m, J, \vec{v})$ and a round parameter $r \in \mathbb{N}$. Take $n = |X| = k^m$ to be the input message length, and set a security parameter $\kappa = 8 \log r \cdot \log k$.

1. Set $\mathcal{U}_0 \leftarrow \{(\lambda, \lambda, J, \vec{v})\}$, where λ is the empty string. For $s \in \{1, \ldots, r\}$, $\mathcal{U}_s \leftarrow \Phi$.
2. Proceed in phases $s \leftarrow 0, \ldots, r-1$:

   For each $((\vec{z}_1, \ldots, \vec{z}_s), (a_1, \ldots, a_s), J, \vec{v})$ in $\mathcal{U}_s$, *in parallel*, $\mathcal{P}$ and $\mathcal{V}$ run the Tensor-Reduction Protocol of Figure 1 with $C_1 = C$ and $C_2 = C^{m-s-1}$. Taking $X_s = \vec{z}_s(\cdot \ldots \cdot (\vec{z}_1 \cdot X))$, the instance is $(X_s, J, \vec{v})$.

   The output of each run is a collection of tuples $\{(a, \vec{z}_{s+1,a}, J_{s+1}, \vec{v}_{s+1,a})\}_{a\in[\log(k/\kappa)+1]}$. For each $a \in [\log(k/\kappa)+1]$, add $((\vec{z}_1, \ldots, \vec{z}_s, \vec{z}_{s+1,a}), (a_1, \ldots, a_s, a), J_{s+1}, \vec{v}_{s+1,a})$ to $\mathcal{U}_{s+1}$.
3. For each $((\vec{z}_1, \ldots, \vec{z}_r), (a_1, \ldots, a_r), J_r, \vec{v}_r) \in \mathcal{U}_r$, do the following in parallel:
   (a) $\mathcal{P} \to \mathcal{V}$: send $X_r = \vec{z}_r \cdot (\ldots \cdot (\vec{z}_1 \cdot X))$
   (b) $\mathcal{V}$: receive X'_r. Verify that $C^{m-r}(X'_r)_{|J_r} = \vec{v}_r$, else reject immediately
   (c) $\mathcal{V}$: set $\varepsilon_r = \varepsilon \cdot \Pi_{s=1}^{r}(2^{a_s}/4)$. Pick $(10/\varepsilon_r)$ uniformly random coordinates in X'_r. For each coordinate j that was picked, verify that $X'_r[j] = (\vec{z}_r \cdot (\ldots \cdot (\vec{z}_1 \cdot X)))[j]$ by querying the appropriate coordinates in the original input message X. If any of these checks fail, then reject immediately.
4. If $\mathcal{V}$ did not reject so far, then $\mathcal{V}$ accepts.

Figure 2: Interactive Proof of Proximity for *TVAL*

Proof of Theorem 3.12. There are a total of r phases, with 2 messages in each Phase (for running the Tensor Reduction Protocol). There is then a single additional round of communication from $\mathcal{P}$ to $\mathcal{V}$ in Step 3. The total number of messages is $2r+1$. The running time of $\mathcal{P}$ is $\text{poly}(n)$ by construction. We proceed to prove completeness and soundness, and then analyze the query and communication complexity.

Completeness. By completeness of the Tensor Reduction Protocol (Theorem 3.8), if all instances specified by $\mathcal{U}_s$ are YES instances of *TVAL* on C^{m-r}, then all instances specified by $\mathcal{U}_{r+1}$ are also YES instances of *TVAL* on C^{m-r-1}. Thus, when $(X, J, \vec{v})$ (the instance specified in $\mathcal{U}_0$) is a YES instance, we get that all instances specified by $\mathcal{U}_r$ are YES instances. In Step 3 of the protocol, for each tuple $((\vec{z}_1, \ldots, \vec{z}_r), \vec{a}, J_r, \vec{v}_r) \in \mathcal{U}_r$, the prover $\mathcal{P}$ sends $X_r = \vec{z}_r \cdot (\ldots \cdot (\vec{z}_1 \cdot X))$, so that $C^{m-r}(X_r)_{J_r} = \vec{v}_r$, and the verifier will accept.

Soundness. After running all r phases, we obtain a collection $\mathcal{U}_r$ of instances for *TVAL* on C^{m-r}. Each tuple $U_r = ((\vec{z}_1 \ldots \vec{z}_r), (a_1, \ldots, a_r), J_r, \vec{v}_r) \in \mathcal{U}_r$ is derived from a sequence of tuples in $(\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_{r-1})$. In phase $s \in [r]$, we run the Tensor Reduction Protocol on an "ancestor" U_s of U_r, and derive from it a set of $(\log k/\kappa + 1)$ tuples. Let U_{s+1} be the ancestor of U_r derived from U_s. We know that U_{s+1} is the a_{s+1}-th output generated by the Tensor Reduction Protocol in its run on the input specified by U_s.

If the initial input $(X, J, \vec{v})$ is ε-far from *TVAL* on C^m, then by soundness of the Tensor Reduction protocol (Theorem 3.8), and a Union Bound over the r phases, we conclude that with all but $r \cdot ((|F| - 1)^{-1} + e^{-\kappa/4 \log k})$ probability over $\mathcal{V}$'s coins, there exists a tuple $U_r^* = ((\vec{z}_1^* \ldots \vec{z}_r^*), (a_1^*, \ldots, a_r^*), J_r^*, \vec{v}_r^*) \in \mathcal{U}_r$ s.t. the instance $(X_r^*, J_r^*, \vec{v}_r^*)$ specified by U_r^* is ε_r^*-far from *TVAL* on C^{m-r}, where:

$$\varepsilon_r^* \geq \varepsilon \cdot \Pi_{s=1}^r (2^{a_s^*}/4)$$

For the tuple U_r^*, the message $X_r^{*\prime}$ sent by the prover in Step 3 must be at least ε_r^*-far from X_r^* (because $(X_r^{*\prime}, J_r^*, \vec{v}_r^*)$ is a YES instance of *TVAL* on C^{m-r}). In Step 3c of the protocol, $\mathcal{V}$ picks $(10/\varepsilon_r^*)$ uniformly random coordinates of $X_r^{*\prime}$, and compares $X_r^{*\prime}$ and X_r^* on these coordinates. If any inconsistency is found, $\mathcal{V}$ rejects immediately, and so $\mathcal{V}$ will reject with probability (much) greater than 9/10.

By a Union Bound, the total probability of failure (the soundness error) is at most:

$$\begin{aligned} & r \cdot (1/(|F| - 1) + e^{-\kappa/4 \log k}) + 1/10 \leq \\ & r \cdot (1/(10r - 1) + e^{-2 \log r}) + 1/10 < \\ & 1/2 \end{aligned}$$

Query Complexity. The only place where $\mathcal{V}$ queries the input X is in Step 3c. For each tuple $U_r = ((\vec{z}_1, \ldots, \vec{z}_r), (a_1, \ldots, a_r), J_r, \vec{v}_r) \in \mathcal{U}_r$, the verifier computes:

$$\varepsilon_r = \varepsilon \cdot \Pi_{s=1}^r (2^{a_s}/4)$$

$\mathcal{V}$ then tests $(10/\varepsilon_r)$ coordinates in $X_r{}'$. Testing each such coordinate means computing X_r in that coordinate, which requires making τ_r queries to X. By the bounded locality property of the Tensor Reduction Protocol (Theorem 3.8):

$$\tau_r \leq \Pi_{s=1}^r (2^{a_s} \cdot \kappa)$$

The query complexity for testing the instance specified by U_r equals:

$$(10/\varepsilon_r) \cdot \tau_r \leq (10/\varepsilon) \cdot (4 \cdot \kappa)^r = O(1/\varepsilon) \cdot (4 \cdot \kappa)^r$$

The total number of instances in $\mathcal{U}_r$ is $(\log(k/\kappa) + 1)^r$, giving total query complexity:

$$\begin{aligned} \mathit{QueryC} &\leq O(1/\varepsilon) \cdot (4 \cdot \kappa)^r \cdot (\log(k/\kappa) + 1)^r \\ &\leq O(1/\varepsilon) \cdot (4 \cdot 8 \log k \cdot \log r)^r \cdot (\log k + 1)^r \\ &= O(1/\varepsilon)^{1+o(1)} \end{aligned}$$

where above we used the fact that $k^r = O(1/\varepsilon)$ and $r \leq k$, and thus $(\log k \cdot \log r)^r = (1/\varepsilon)^{o(1)}$

Communication Complexity. The communication complexity in each of the r phases is proportional to the number of instances, multiplied by the communication complexity of the Tensor Reduction Protocol. By Theorem 3.8 the communication complexity in all r phases is at most:

$$\sum_{s\in[r]} (\log(k/\kappa)+1)^s \cdot O(|J| \cdot k \log k \cdot \log |F|) = (\log(k/\kappa)+1)^r \cdot O(|J| \cdot k \cdot \log k \cdot \log |F|)$$

The communication complexity in Step 3 is the number of instances, $(\log(k/\kappa)+1)^r$, multiplied by the length of each message $k^{m-r} \cdot \log |F|$, and so the total communication complexity is at most:

$$\begin{aligned} CommC &= (\log(k/\kappa)+1)^r \cdot O(k^{m-r} + |J| \cdot k \log k) \cdot \log |F| \\ &= O(\log k)^r \cdot O(n/k^r + |J| \cdot k \log k) \cdot \log |F| \\ &= (k^{m-r} + |J| \cdot k \log k) \cdot \log |F| \cdot (1/\varepsilon)^{o(1)} \\ &= (k^{m-r} + |J| \cdot k) \cdot (1/\varepsilon)^{o(1)} \end{aligned}$$

Where above we used $\log |F| \leq O(\log(1/\varepsilon))$, and thus $k^r \leq 1/\varepsilon$, so $(\log k)^r = (1/\varepsilon)^{o(1)}$. □

3.4 From Low Depth to *PVAL*

In this section we show that *PVAL* is complete for constructing IPPs for languages that can be computed in low depth. From this result, together with the Interactive Proof of Proximity for *PVAL* (Theorem 3.12), we deduce Interactive Proof of Proximity for every language L that can be computed in low depth, and prove Theorem 1.1.

Recall from the overview, that we reduce from L to *PVAL* using (parallel repetitions of) the GKR interactive proof. We begin by stating the main result of [GKR08] (see also the full exposition in [Rot09]). In Theorem 3.14, we prove the reduction from proving proximity to low depth languages to *PVAL*. Finally, we provide a proof of Theorem 1.1.

Theorem 3.13 (Interactive Proofs "for Muggles" [GKR08, Rot09])**.** *For any language L computable by log-space uniform boolean circuits of depth $D = D(n)$, size $S = S(n)$, and fan-in 2, for any $k \geq D$ and $m = log_k S$ (so that $k^m \geq S$), the following holds. Take $\ell = O(k \cdot D \cdot \log S)$, and a finite field F of size ℓ. Let the code $C : F^k \to F^\ell$ be a Reed-Solomon code with degree k, and $C^m : F^{k^m} \to F^{\ell^m}$ its m-th Tensor power, an m-variate Reed-Muller code with individual degree bound k (see Definition 3.7).*

There exists an interactive protocol between a prover and a verifier, where the prover gets input X, and the verifier only gets the input $(n = |X|, 1^D)$. The output of the protocol is a coordinate $j \in F^m$, and a value $v \in F$, such that:

- *Completeness. If $X \in L$ and the prover honestly follows the protocol, then $C^m(X)[j] = v$.*
- *Soundness. If $X \notin L$, then for any (unbounded) cheating prover, with probability at least $1/2$ over the verifier's coins, $C^m(X)[j] \neq v$.*

The prover runs in time $\mathrm{poly}(S)$*, and the verifier runs in time* $\mathrm{poly}(D)$ *(without accessing X). The communication complexity is* $\mathrm{poly}(D)$*, and the number of rounds is $O(D \cdot \log S)$.*

Theorem 3.14 (*PVAL* is "Complete" for Constructing IPPs)**.** *For any distance bound $\varepsilon = \varepsilon(n)$, and any language L computable by log-space uniform boolean circuits of depth $D = D(n)$, size $S = S(n)$, and fan-in 2, any $k \geq D$ and $m = \log_k S$ (so that $k^m \geq S$), the following holds. Take $\ell = O(k \cdot D \cdot \log S)$, and a finite field F of size ℓ. Let $C : F^k \to F^\ell$ be a Reed-Solomon code with degree k, and $C^m : F^{k^m} \to F^{\ell^m}$ its m-th tensor power, an m-variate Reed-Muller code with individual degree bound k (see Definition 3.7).*

There exists an interactive protocol between a prover and a verifier, where the prover gets input X and the verifier only gets the input $(n, 1^D)$. The output of the protocol is a coordinate set $J \subseteq F^m$ of size $t = 4\varepsilon \cdot n \cdot \log n$, and a value vector $\vec{v} \in F^{|J|}$, such that:

- *Completeness. If $X \in L$ and the prover honestly follows the protocol, then $X \in PVAL(F, k, m, J, \vec{v})$.*
- *Soundness. If X is ε-far from L, then for any cheating prover, with all but $1/2$ probability over the verifier's coins, X is ε-far from $PVAL(F, k, m, J, \vec{v})$.*

The prover runs in time $\text{poly}(S)$, and the verifier runs in time $\varepsilon \cdot n \cdot \text{poly}(D)$ (without accessing X). The communication complexity is $\varepsilon \cdot n \cdot \text{poly}(D)$, and the number of rounds is $O(D \cdot \log S)$.

Proof. As described in the introduction, we run the GKR interactive proof $t = 2\varepsilon \cdot n \cdot (\log n + \log |F|) \leq 4\varepsilon \cdot n \cdot \log n$ times in parallel. This yields t pairs of the form $(j, v) \in F^m \times F$, which we aggregate into a size-t collection $J \subseteq F^m$ of coordinates, and a vector $\vec{v} \in F^t$ of claimed values for those coordinates. The communication and round complexities, as well as the running times of the honest prover and the verifier, all follow immediately from the construction.

To prove completeness, observe that by completeness of the GKR protocol (Theorem 3.13), when $X \in L$ and the prover honestly follows the protocol, $C^m(X)$ and $\vec{v}$ agree on every coordinate $j \in J$.

To prove soundness, by soundness of the GKR Protocol (Theorem 3.13), for any $X' \notin L$, for any cheating prover strategy (which may or may not depend on X'), with all but $2^{-t} < 1/(2n \cdot |F|)^{\varepsilon \cdot n}$ probability over the verifier's coin tosses, there exists $j \in J$ s.t. $C^m(X)$ and $\vec{v}$ do not agree on the j-th coordinate. Now, for X that is ε-far from L, there are $(n \cdot |F|)^{\varepsilon \cdot n}$ inputs X' at distance ε or less from X, and all of these inputs are not in L (because X is ε-far). Taking a union bound, we conclude that with probability at least $1/2$ over the verifier's coin tosses, there is no X' at distance ε or less from X for which $C^m(X')$ and $\vec{v}$ agree on all the coordinates in J. We conclude that X is ε-far from $PVAL(F, k, m, J, \vec{v})$. □

Proof of Theorem 1.1. Theorem 1.1 (stated in the introduction) follows by combining Theorems 3.14 and 3.12. For any language L as specified in the theorem statement, and any input X, by running the protocol reduction to *PVAL* (Theorem 3.14), we obtain $(J, \vec{v})$ s.t. if $X \in L$ then $X \in PVAL(J, \vec{v})$, and if X is ε-far from L, then w.h.p. X is ε-far from $PVAL(J, \vec{v})$. We then run the Interactive Proof of Proximity for *PVAL* (Theorem 3.12). If $X \in PVAL(J, \vec{v})$, then the verifier will accept, and if X is ε-far, then w.h.p. the verifier will reject. The complexity parameters follow by construction. □

4 Lower Bounds

In this section, we prove lower bounds on the tradeoff between query complexity, communication complexity, and round complexity for interactive proofs of proximity:

Theorem 4.1. *There is a language L in logspace-uniform $\mathcal{NC}^1$ such that in any interactive proof of $(1/8)$-proximity for L with prover-to-verifier communication $c(n)$, verifier query complexity $q(n)$, and $r(n)$ verifier messages (where the verifier messages are uniformly random strings, and verifier has no additional randomization), we have*

$$\min\{r(n), c(n), q(n)\} = n^{\Omega(1/r(n))}.$$

In particular, it is impossible to have $r(n) = O(1)$ and $c(n) = q(n) = n^{o(1)}$.

Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.

We begin by proving a lower bound for an "$\mathcal{AM}$ analogue of $\mathcal{IPP}$" (which we denote $\mathcal{AM}$-$\mathcal{IPP}$), where on input length n, the verifier sends its random coins $v \xleftarrow{R} \{0,1\}^{t(n)}$ to the prover, the prover replies with a message $p \in \{0,1\}^{c(n)}$, and then the verifier makes $q(n)$ queries to the input oracle $x \in \{0,1\}^n$ (depending on n, v and p) and then accepts or rejects (based on v, p, and the oracle responses).

Our lower bound is based on the observation that the maximum acceptance probability of the verifier in such a protocol can be computed by a "small" CNF formula.

Lemma 4.2. *Let V be a verifier in an $\mathcal{AM}$-$\mathcal{IPP}$ protocol where on inputs of length n, the prover's message has length $c(n)$ and the verifier makes $q(n)$ queries to its input oracle $x \in \{0,1\}^n$. Let $\mathrm{Acc}_V(x)$ denote the maximum, over all prover strategies P^*, that the verifier V accepts on input x. Then for every input length n, there is a distribution $\mathcal{D}$ on CNF formulas with at most $2^{c(n)+q(n)}$ clauses of width at most $q(n)$ such that*

$$\mathrm{Acc}_V(x) = \mathop{\mathrm{E}}_{\varphi \xleftarrow{R} \mathcal{D}} [\varphi(x)].$$

Proof. For each setting of the verifier's coin tosses v and prover message p, the verifier V's acceptance predicate $A_{v,p}(x)$ is given by a decision tree of height $q(n)$. Specifically, based on v and p, the verifier queries one input variable x_{i_1}, and based on the result, it queries another input variable x_{i_2}, and so on. Any decision tree of height $q(n)$ can also be represented as a CNF formula with at most $2^{q(n)}$ clauses of width $q(n)$ (one clause for each accepting leaf in the decision tree). Now, fixing only the verifier's message v, the formula $\varphi_v(x) = \bigvee_{p \in \{0,1\}^{c(n)}} A_{v,p}(x)$ tells us whether there exists a prover message p that will make the verifier accept. So $\mathrm{Acc}_V(x) = \mathrm{E}_{v \xleftarrow{R} \{0,1\}^{t(n)}} [\varphi_v(x)]$. Note that φ_v is a CNF formula with at most $2^{c(n)} \cdot 2^{q(n)}$ clauses of width at most $q(n)$. □

Given that the verifier's acceptance probability is computed by (distributions over) CNF formulas, we can use a pseudorandom generator for CNF formulas to obtain a language for which it is hard to prove proximity.[11]

Lemma 4.3. *Let V be a verifier in an interactive protocol, and $\mathrm{Acc}_V(x)$ be the maximum probability, over all prover strategies P^*, that the V accepts on input x. Suppose $G_n : \{0,1\}^{n/8} \to \{0,1\}^n$ be a generator that fools Acc_V in the sense that*

$$\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] - \mathrm{E}[\mathrm{Acc}_V(U_n)] < 1/8.$$

Then V cannot be a verifier in an interactive proof of $n/8$-proximity for $\mathrm{Image}(G_n)$.

[11] This has the flavor of results in proof complexity, where for certain proof systems it is hard to certify that a given string is outside the image of an appropriate PRG [ABSRW04].

Proof. Assume V were an interactive proof of $n/8$-proximity for $\mathrm{Image}(G_n)$. By completeness, we have

$$\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] \geq 2/3, \tag{3}$$

By soundness, we have

$$\begin{aligned} \mathrm{E}[\mathrm{Acc}_V(U_n)] &\leq 1/3 + \Pr[U_n \text{ is } n/8\text{-close to } \mathrm{Image}(G_n)] \\ &= 1/3 + \frac{1}{2^n} \cdot |\mathrm{Image}(G_n)| \cdot V_n(n/8), \end{aligned}$$

where $V_n(n/8)$ is the volume of the Hamming Ball of radius $n/8$ in $\{0,1\}^n$. We have $|\mathrm{Image}(G_n)| \leq 2^{n/8}$, and $|B(0^n, n/8)| \leq 2^{H(1/8)n+o(n)} \leq 2^{.6n}$ for suffficiently large n (where H is the binary entropy function). Thus,

$$\mathrm{E}[\mathrm{Acc}_V(U_n)] \leq 1/3 + 2^{-\Omega(n)}. \tag{4}$$

Equations (3) and (4) contradict our assumption about G_n fooling Acc_V. □

We can now use known explicit pseudorandom generators that fool CNF formulas, such as [Nis91, Baz09], to obtain languages that have no low-complexity $\mathcal{AM}$-$\mathcal{IPP}$ protocol.

Lemma 4.4. *For every n, there is an explicit generator $G_n : \{0,1\}^{n/8} \to \{0,1\}^n$ such that*

1. *For every CNF formula φ on n variables with at most $m(n)$ clauses, for $m(n) = 2^{-\Omega(\sqrt{n/\log n})}$, we have:*
$$|\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] - \mathrm{E}[\mathrm{Acc}_V(U_n)]| < 2^{-\Omega(\sqrt{n/\log n})}$$
2. *G_n is an $\mathbb{F}_2$-linear map, and the $n \times n/8$ matrix for evaluating G_n can be constructed in logspace-uniform $\mathcal{NC}$.*

Proof. Bazzi [Baz09] proved that m-term CNF formulas cannot distinguish a k-wise independent distribution on $\{0,1\}^n$ from U_n except with probability $m \cdot 2^{-\Omega(\sqrt{k})}$. (See [Raz09] for a simpler proof.) With a uniformly random seed of length $n/8$, we can generate an $\Omega(n/\log n)$-wise independent distribution on $\{0,1\}^n$ using the standard construction based on polynomial evaluation: the seed y of G_n specifies a polynomial p_y of degree $\Omega(n/\log n)$ over the field $\mathbb{F}_q$ of size $q = 2^{\lceil \log_2 n \rceil}$, and the i'th bit of $G_n(y)$ is the the first bit of the evaluation of p_y at the i'th element of $\mathbb{F}_q$. It can be verified that this mapping G_n is an $\mathbb{F}_2$-linear map that can be constructed in logspace-uniform $\mathcal{NC}$. Constructing the matrix to check membership in $\mathrm{Image}(G_n)$ can be done by linear algebra, also in logspace-uniform $\mathcal{NC}$. □

Putting all the above together we deduce:

Theorem 4.5. *There is a language L in logspace-uniform $\mathcal{NC}$ that has no $\mathcal{AM}$-$\mathcal{IPP}$ proof of $(1/8)$-proximity in which the prover's message has length $o(\sqrt{n/\log n})$ and the verifier's query complexity is $o(\sqrt{n/\log n})$. Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.*

Proof. We take $L = \bigcup_n \mathrm{Image}(G_n)$ where G_n is as in Lemma 4.4. Assume that (P, V) were an $\mathcal{AM}$-$\mathcal{IPP}$ of $(1/8)$-proximity in which the prover's message has length $c(n) = o(\sqrt{n/\log n})$ and the verifier's query complexity is $q(n) = o(\sqrt{n/\log n})$. By Lemma 4.2, $\mathrm{Acc}_V(x)$ equals the expectation of a distribution on CNF formulas with $2^{o(\sqrt{n/\log n})}$ clauses evaluated on x. Thus, by the pseudorandomness property of G_n, we have $|\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] - \mathrm{E}[\mathrm{Acc}_V(U_n)]| < 2^{-\Omega(\sqrt{n/\log n})}$. By Lemma 4.3, (P, V) cannot be a proof of $(1/8)$-proximity for $L_n = \mathrm{Image}(G_n)$. □

Now, to handle protocols with more rounds and private coins, we use classic transformations to convert the protocols to $\mathcal{AM}$ protocols, keeping track of the effect of the transformations on the communication complexity and the query complexity. The first transformation collapses any r-round public-coin interactive proof to an $\mathcal{AM}$ proof system, with a complexity blow-up that is exponential in r. This is a more quantitative version of the $\mathcal{AM}$ Collapse Theorem of Babai and Moran [BM88], as worked out in [GVW02].

Lemma 4.6. *If a language L has a public-coin interactive proof of $\varepsilon(n)$-proximity with prover-to-verifier communication complexity $c(n)$, verifier query complexity $q(n)$, and $r(n)$ verifier messages (where the verifier messages are uniformly random strings, and verifier has no additional randomization), then L has an $\mathcal{AM}$-$\mathcal{IPP}$ of $\varepsilon(n)$-proximity with prover-to-verifier communication complexity $c(n) \cdot t(n)$ and verifier query complexity $q(n) \cdot t(n)$, for $t(n) = r(n)^{O(r(n))} \cdot c(n)^{r(n)-1}$.*

Combining this lemma with Theorem 4.5, we obtain the following lower bound for multi-round public-coin protocols.

Theorem 4.7. *There is a language L in logspace-uniform $\mathcal{NC}$ such that in any public-coin interactive proof of $(1/8)$-proximity for L with prover-to-verifier communication $c(n)$, verifier query complexity $q(n)$, and $r(n)$ verifier messages (where the verifier messages are uniformly random strings, and verifier has no additional randomization), we have*

$$r(n)^{O(r(n))} \cdot \left(c(n)^{r(n)} + q(n) \cdot c(n)^{r(n)-1}\right) = \Omega(\sqrt{n/\log n}).$$

Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.

To handle private-coin protocols, we use the Goldwasser–Sipser transformation [GS86] to convert them to public-coin interactive proofs.

Lemma 4.8. *If a language L has a (private-coin) interactive proof of $\varepsilon(n)$-proximity with communication complexity $c(n) \geq \log n$, verifier query complexity $q(n)$, and $r(n)$ verifier messages, then L has a (public-coin) interactive proof of $\varepsilon(n)$-proximity with communication complexity $c(n) \cdot t(n)$, verifier query complexity $q(n) \cdot t(n)$, and $r(n) + 2$ verifier messages, where $t(n) = O(r(n) \cdot \log(c(n) \cdot r(n)))$.*

Proof. Let (P, V) be the private-coin interactive proof for L. We may assume without loss of generality that the verifier uses at most $(c(n) + O(1))$ coin tosses; otherwise, a random subsample of $O(2^{c(n)} + n)$ random tapes for the verifier will yield a new verifier that remains complete and sound, but only makes $(\max\{c(n), \log n\} + O(1))$ coin tosses. The modified verifier preserves completeness and soundness by a union bound over all $2^{2^{c(n)}}$ deterministic prover strategies and 2^n inputs of length n (the modified verifier is not uniform or computationally efficient, but the results in this section only refer to the verifier's query complexity and not its computational complexity or uniformity.)

The Goldwasser–Sipser transformation begins by performing $t(n) = O(r(n) \cdot \log(c(n) \cdot r(n)))$ parallel repetitions to obtain a new private-coin interactive proof (P', V') where the ratio of the completeness and soundness probabilities at least $(c'(n) \cdot \mathrm{poly}(r(n)))^{r(n)}$, where $c'(n) = t(n) \cdot c(n)$ is the new communication complexity.

Then a public-coin protocol (P'', V'') is constructed that simulates an execution of (P', V'). The messages in (P'', V'') consist of a transcript of messages for one execution of (P', V'), one

sequence of coin tosses for the verifier V', $r(n)+1$ pairwise independent hash functions whose inputs and outputs have length $O(c'(n))$, and $r(n)$ positive integers of magnitude $O(c'(n))$. It can be verified that all of these amount to $O(t(n)\cdot c(n))$ bits of communication. V'' needs to run V' on the simulated transcript, and thus needs to make $t(n)\cdot q(n)$ queries to the input oracle.

The reason the parallel repetitions are needed is that for each round, the simulation of (P'', V'') loses a factor of $O(c'(n)\cdot \mathrm{poly}(r(n)))$ in the ratio between completeness and soundness, as compared to (P', V'). (A factor of $c'(n)$ comes from bucketing the messages of V' according to the approximate number of coin tosses that lead to them, and a factor of $\mathrm{poly}(r(n))$ is to ensure that the set-size lower bound protocol used in each round has completeness and soundness error smaller than $1/r(n)$.) □

Combining Lemma 4.8 with Theorem 4.7, we get the following theorem (which is a more refined version of Theorem 4.1):

Theorem 4.9. *There is a language L in logspace-uniform $\mathcal{NC}$ such that in any interactive proof of $(1/8)$-proximity for L with prover-to-verifier communication $c(n) \geq \log n$, verifier query complexity $q(n)$, and $r(n)$ verifier messages (where the verifier messages are uniformly random strings, and verifier has no additional randomization), we have*

$$r(n)^{O(r(n))} \cdot (c(n)+q(n)) \cdot \left((c(n)\log(c(n))^{r(n)}\right) = \Omega(\sqrt{n/\log n}).$$

In particular, if $r(n) = O(1)$, we cannot have $c(n)$ and $q(n)$ both be $n^{o(1)}$.

Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.

5 Interactive Proofs of Proximity for Specific Language

5.1 Bipartiteness on Well-Mixing Graphs

The Bounded Degree Model. The bounded degree model is as follows (see the excellent introduction of Goldreich [Gol10a] for further discussion and motivation). The input is an undirected N-vertex graph $G = ([N], E)$, of (constant) maximum degree D. The graph is represented as a function $g : [N] \times \{1, \ldots, D\} \to ([N] \cup \perp)$, which on input (v, i) outputs $u \in [N]$ if u is the i-th neighbor of v (or $\perp$ if v has fewer than i neighbors). Since the degree is bounded by D, the number of (undirected) edges is at most $D \cdot N/2$. The distance between two graphs the number of edges on whose presence or absence they disagree.

Main Result. Our main result is an Interactive Proof of Proximity in the bounded degree model.

Theorem 5.1. *For any specified $\varepsilon = \varepsilon(n)$, the protocol of Figure 3 is an Interactive Proof of ε-Proximity in the bounded degree model as follows. If the input is a bipartite graph, then the verifier will accept. If the input graph is both ε-far from being bipartite* and well-mixing*, then the verifier will reject with probability at least $1/2$.*

The query and communication complexities are $\mathrm{poly}((1/\varepsilon), \log N)$, the round complexity is 1 (2 messages). The honest prover runs in time $\mathrm{poly}(N)$, and the honest verifier runs in time $\mathrm{poly}((1/\varepsilon), \log N)$. The verifier uses secret coins.

By "well mixing", we mean that a random walk of length $L = O(\log n)$, which begins in any arbitrary vertex s in the graph, ends at each vertex v with probability no less than $1/2N$ (the random walk performs a self loop with probability at least $1/2$ in each step). The length L can also be treated as a parameter (affecting the query complexity). This is a protocol for a promise problem: YES instances are bipartite, NO instances are well mixing and far from bipartite.

We note that Goldreich and Ron [GR02] showed a $\Omega(\sqrt{n})$ query complexity lower bound for any property tester for this promise problem (with constant-fraction distance). Thus, this Interactive Proof of Proximity demonstrates that a limited amount of interaction with a prover can provide dramatic benefits: in this case, it yields an exponential improvement in query complexity. An interesting open problem is avoiding the need for a promise, i.e. an IPP with completeness and soundness for all graphs. In the property testing setting (i.e. without a prover), [GR99] showed a $\tilde{O}(\sqrt{N})$ property tester for graph bipartiteness that works for all graphs (without needing the graph to be well-mixing for soundness).

Protocol Overview. The protocol builds on techniques and ideas put forward in the property tester of [GR99]. The basic idea is for the verifier to initiate a random walk of length L at some (random) vertex s, reaching another vertex v. In each intermediate step, at a vertex u, the random walk walks to each neighbor of u with probability $1/2D$, and with the remaining probability (at least $1/2$) it performs a self-loop. The verifier can send s and v to the prover, and asks the prover to determine whether the number of "real" (non self-loop) steps was odd or even. This requires only $O(\log n)$ queries to the graph.

If the graph is bipartite, then when s and v are on the same side of the partition, the honest prover knows that the number of "real" steps was even. When s and v are on different sides, the number of "real" steps must have been odd. Thus, the protocol has perfect completeness. On the other hand, we show that if the graph is far from bipartite and well-mixing, then for every vertex s, the distributions of vertices reached by paths of odd and even number of "real steps" are somewhat close: they have statistical distance bounded away from 1. The statistical distance is $1 - \Theta(\varepsilon)$. Thus, a cheating prover will be caught with probability $\Theta(\varepsilon)$. We can repeat the protocol (in parallel) to boost soundness.

We remark that this protocol is reminiscent of the interactive proof for Graph Non-Isomorphism [GMW91].

Bipartiteness Protocol on input $G = (N, g)$ with parameter ε

Set $T = \Theta(1/\varepsilon)$ and $L = \Theta(\log n)$

1. $\mathcal{V}$: for $i \leftarrow 1, \ldots T$: pick a uniformly random $s_i \in [N]$, and perform a random walk of length L on G, starting at s_i and ending at v_i: in each step, walk to each neighbors with probability $1/2D$, and with the remaining probability stay at the current vertex ("self loop step")
   Let $b_i \in \{0, 1\}$ be the parity of the number of real (non "self-loop") steps in the i-th walk
   $\mathcal{V} \to \mathcal{P}$: $((s_1, v_1), \ldots, (s_T, v_T))$
2. $\mathcal{P} \to \mathcal{V}$: for each $i \in \{1, \ldots, T\}$, send the parity $p_i \in \{0, 1\}$ of the length of the shortest path between s_i and v_i
3. $\mathcal{V}$: receive $(p'_1, \ldots, p'_T)$, accept if and only if $\forall i \in \{1, \ldots, T\}, b_i = p'_i$ (otherwise reject).

Figure 3: Bipartiteness Protocol

Proof of Theorem 5.1. Completeness, as well as the query, communication, and round complexity, all follow by construction.

For soundness on well-mixing graphs, at a high level the proof follows the analysis of Goldreich and Ron [GR99], though we will focus on statistical distance between the random walks of odd and even parity, whereas they focused on collision probability. Recall that for well-mixing graphs, for any nodes $s, v \in [N]$, the probability that a random walk of length L reaches node v is at least say $1/2N$. For example, this is the case for an expander graph, as $L = \Omega(\log n)$.

For a random walk of length L originating at s, we say that the random walk has parity $b \in \{0,1\}$ if the parity of the real (non self-loop) steps is b. We consider two distributions $\mathcal{A}^0(s), \mathcal{A}^1(s)$, where $\mathcal{A}^b(s)$ is the distribution, over graph vertices, obtained by taking a random walk of length L and parity b.

Lemma 5.2 below shows that if (for any s) the distributions $\mathcal{A}^0(s), \mathcal{A}^1(s)$ are distinguishable, i.e. their statistical distance is bounded away from 1, then the graph must be close to bipartite.

We are left with the case that (for all $s \in [N]$) the statistical distance between $\mathcal{A}^0(s)$ and $\mathcal{A}^1(s)$ is no more than $1 - \varepsilon/(64D)$. In this case, observe that each (s_i, v_i) pair sent by $\mathcal{V}$ to the prover is drawn uniformly at random from $A^{b_i}(s_i)$, where b_i is also uniformly random in $\{0,1\}$. In this case, for each $i \in \{1, \ldots, T\}$, the probability that the prover correctly sends back $p'_i = b_i$ is at most $\varepsilon/(64D)$. Repeating the experiment $T = \Omega(D/\varepsilon)$ times, the probability that the prover correctly guesses $(b_1, \ldots, b_T)$ is less than $1/2$. Soundness follows.

Lemma 5.2. *If there exists $s \in [N]$ such that $\Delta(\mathcal{A}^0(s), \mathcal{A}^1(s)) > 1 - \frac{\varepsilon}{64D}$, then G is at distance at most ε from a bipartite graph.*

Proof. Suppose there exists an s as claimed in the lemma. For $v \in V$ and $b \in \{0,1\}$, we define p_v^b to be the probability that a random walk of length L and parity b terminates at v (recall that the path's parity is the parity of the number of real, non self-loop, steps). We define a partition (S^0, S^1) of V as follows: $S^0 = \{v : p_v^0 \geq p_v^1\}$ and $S^1 = (V \setminus S^0)$. Let E^b be the set of edges violating the partition whose endpoints are both within S^b. Without loss of generality, suppose that $|E^0| \geq |E^1|$. We show below that $|E^0| \leq \varepsilon \cdot N/2$, and the lemma follows.

Claim 5.3. *For $v \in S^0$:*

$$p_v^1 \geq \frac{1}{16D} \cdot \sum_{u \in (\Gamma(v) \cap S_0)} p_u^0$$

Proof. For any vertex u, the probability that an $(L-1)$-step walk of parity 0 terminates at u, is at least $1/8$ of the probability that an L-step walk of parity 0 terminates at u. This is shown in the claim inside the proof of Lemma 4.4 in [GR99]. The intuition is w.h.p. an L-step walk will make many self loop steps, and when this is the case, removing any of these self-loops leads to an $(L-1)$-step walk of the same parity. The combined probability of all these $(L-1)$-step walks is not much smaller than the combined probability of the L-step walks.

Examining L step walks that end at v, observe that if in the first $(L-1)$ steps, the walk had parity 0 and ended at $u \in \Gamma(v)$, then with probability $(1/2D)$ over the L-th step, it will reach v in its next step (and terminate). The claim follows. □

From Claim 5.3, we conclude that:

$$\begin{aligned}\Delta(\mathcal{A}^0(s),\mathcal{A}^1(s)) &= \sum_{v\in S_0}(p_v^0-p_v^1)\\ &\le \sum_{v\in S_0}\left(p_v^0-\frac{1}{16D}\cdot\sum_{u\in(\Gamma(v)\cap S_0)}p_u^0\right)\\ &\le \left(\sum_{v\in S_0}p_v^0\right)-\frac{|E^0|}{16D}\cdot\frac{1}{2N}\\ &\le 1-\frac{|E^0|}{32D\cdot N}\end{aligned}$$

Thus if $\Delta(\mathcal{A}^0(s),\mathcal{A}^1(s)) > 1-\varepsilon/(64D)$, then it must be the case that $|E^0| < (\varepsilon\cdot N/2)$. □

□

5.2 Hamming Weight

In this section we give an Interactive Proof of Proximity for verifying a vector's Hamming Weight. Our main result, stated in Corollary 5.7 below, shows how to verify the Hamming weight of an n-bit vector, to within (fractional) additive Hamming distance ε, using $\tilde{O}(1/\varepsilon)$ queries and communication. We note that without the help of a prover it is well-known by sampling lower bounds that $\Omega(1/\varepsilon)^2$ queries are necessary. Moreover, even in the IPP setting $\Omega(1/\varepsilon)$ queries are necessary (see Remark 1.2 in the introduction), and so the query complexity is nearly optimal. We begin by defining the problem, then proceed with precise statements of our results and a protocol overview. The full protocol and proofs follow.

Definition 5.4 (*Hamming.*)**.** For input length n, the language *Hamming* is on pairs (X,w), where $X\in\{0,1\}^n$ is an implicit input, and $w\in\{0,1,\ldots,n\}$ is an explicit input. (X,w) is a YES instances of *Hamming* if and only if $\|X\|=w$.

Note that an Interactive Proof of ε-Proximity to *Hamming* also gives a protocol that lets a sublinear time verifier compute approximate majority up to an ε additive error.

5.2.1 Hamming Dimension Reduction

Theorem 5.5. *The Hamming Weight Dimension Reduction Protocol of Figure 4, on shared input (n,k,ℓ,ε), where $n=k\cdot\ell$, $w\in\{0,\ldots,n\}$, and $\varepsilon\in(0,1)$, prover input $X\in\{0,1\}^n$, and security parameter κ has a single message (from $\mathcal{P}$ to $\mathcal{V}$) of length $k\cdot\log\ell$. The prover runs in time $\tilde{O}(n)$, and the verifier runs in time $k\cdot\kappa\cdot\text{polylog}n$.*

The protocol's output is a collection $\mathcal{U}$ of $2k\cdot\kappa$ tuples, where each $U\in\mathcal{U}$ is of the form $U=\{(\ell,X_U,w_U,\varepsilon_U)\}$, X_U is a row of X, $w_U\in[\ell]$, and $\varepsilon_U\in(0,1)$. The protocol satisfies completeness and soundness as follows:

- *Completeness: If $(X,w)\in$ Hamming, then for all $U\in\mathcal{U}$, $(X_U,w_U)\in$ Hamming.*
- *Soundness: If (X,w) is ε-far from Hamming, then with all but $\exp(-\Omega(secp))$ probability over $\mathcal{V}$'s coins, there exists an instance $U^*\in\mathcal{U}$, s.t. (X_{U^*},w_{U^*}) is ε_{U^*}-far from Hamming.*

Hamming Dimension Reduction Protocol on input (n, k, ℓ, X, w) **with parameters** ε, κ

1. $\mathcal{P} \to \mathcal{V}$: for each row $j \in [k]$ of X, send its Hamming weight w_j
   $\mathcal{V}$: receive $\{w'_j\}_{j\in[k]}$. If $w \neq \sum_{j\in[k]} w'_j$, then reject
2. $\mathcal{U} \leftarrow \Phi$. For $a \leftarrow 0, \ldots, \log k$
   $\mathcal{V}$ chooses a set I_a of $2^a \cdot \kappa$ uniformly random rows in $[k]$. For each $i \in I_a$, it adds to $\mathcal{U}$ the instance $(\ell, X[j, *], w'_j, \varepsilon \cdot 2^a)$
3. $\mathcal{V}$'s output is the collection $\mathcal{U}$ of $O(k \cdot \kappa)$ instances

Figure 4: Hamming Weight Protocol

5.2.2 A One-Message IPP for *Hamming*

Proof. Take $k = n^{2/3}, \ell = n^{1/3}$. The verifier uses the Dimension Reduction Protocol of Figure 4 to obtain a set $\mathcal{U}$ of instances. For each $U \in \mathcal{U}$, where $U = (\ell, X_U,)$

□

5.2.3 A Multi-Round IPP for *Hamming*

We now state our main result, a multi-round protocol for *Hamming*, and also a simpler one-round protocol. We then proceed with an overview of the one-round case, and the ideas for extending to a multi-round protocol with improved query and communication complexity.

Theorem 5.6 (Interactive Proof of Proximity for *Hamming*(n))**.** *The protocol of Theorem 5.6 is an Interactive Proof of d-Proximity for Hamming. It has completeness and soundness error* $1/3$*. The query complexity is* $\tilde{O}(n/d)$*, the communication complexity is* $\tilde{O}(n/d)$*, and the number of messages is* $\log d$*.*

Corollary 5.7 (1-round Proof of Proximity for *Hamming*)**.** *There exists an Interactive Proof of d-Proximity for Hamming. It has completeness and soundness error* $1/3$*. The query complexity is* $\tilde{O}(n/d^{2/3})$*, the communication complexity is* $\tilde{O}(n/d^{2/3})$*, and the number of rounds is 1 (2 messages).*

1-round Protocol Overview. The input to the protocol is $X \in \{0,1\}^n$ and a claimed weight w. For this intuitive overview, we take $w = n/2$ and we test for distance $d = \sqrt{n}$ from *Hamming*. Without the help of a prover, by sampling lower bounds we know that this requires $\Omega(n)$ queries. Here we sketch a sublinear query and communication protocol, and the ideas behind the full protocol which uses only $\tilde{O}(\sqrt{n})$ queries (or, more generally, $\tilde{O}(n/d)$ queries).

We treat X as a matrix in $\{0,1\}^{k\times\ell}$, where we set k, ℓ below. The initial idea is for the prover to send the verifier the Hamming weight w_i of each row i of X. The verifier verifies that $\sum_{i\in[n]} w_i = w$ (and rejects otherwise). The difficulty now, similarly to the protocol for *TVAL* in Section 3, is that the prover might concentrate its cheating on just one row, but the verifier can't check all row claims.

Here, however, there is a different idea that directly gives sublinear query complexity. The verifier can send to the prover a random permutation $\pi : [n] \to [n]$, and ask for the row-sums of $\pi(X)$ (viewing $\pi(X)$ as a $k \times \ell$ matrix).[12] Because the input was permuted, the (real) row weights

[12]Note that the communication complexity from the verifier to the prover for sending uniformly random permutation

$\{w_i\}_{i\in[k]}$ of $\pi(X)$ should be close: for each row, its expected weight is $\|X\|/k$ (the expectation is taken over the permutation π). Moreover, since π is a uniformly random permutation, w_i should be concentrated around its expectation; certainly they should be within an additive $\tilde{O}(\sqrt{\ell})$ of the expectation.

The verifier receives some (possibly cheating) claimed sums $\{w'_i\}_{i\in[k]}$, and checks that they are all within $\tilde{O}(\sqrt{\ell})$ of w/k (the expected weight of each row if X's Hamming weight is indeed w). If not, then the verifier rejects immediately. For completeness, if the prover is honest then w.h.p. the honest permuted row-sums should all be close to their expectation, and the verifier will not reject (but note that this test does not have perfect completeness). For soundness, if $\|X\|$ is far from w and the test passes, then the verifier knows that the prover's cheating cannot be concentrated in a single row. In fact, the prover must be cheating in $\tilde{\Omega}(d/\sqrt{\ell})$ rows. The verifier can pick $\tilde{O}(k/(d/\sqrt{\ell})) = \tilde{O}(k\cdot\sqrt{\ell}/d)$ random rows and check their sums itself (by reading ℓ input bits per row). This requires only $\tilde{O}(k\cdot\ell^{3/2}/d) = \tilde{O}(n^{3/2}/(d\cdot\sqrt{k}))$ queries. The communication complexity from the prover to the verifier for sending the row-sums is $k\cdot\log n$. For our choice of $d=\sqrt{n}$, we can plug in $k=n^{2/3}$ and $\ell=n^{1/3}$, and we get sublinear query and communication complexities $\tilde{O}(n^{2/3})$. This used only two messages (1 round of communication).

Multi-Round Protocol. We improve the above query-communication tradeoff using more interaction. The initial idea is to take a random subset of half the rows, and recurse on them. This is a step in the right direction: observe that the *expected* distance of the new instance from *Hamming*, is (exactly) 1/2 the distance of the original instance. Suppose that the verifier could repeatedly shrink the instance size by a factor of 2 without losing much more than that factor in the instance's distance. Then the verifier would repeat this shrinking step $\log d$ times, get to an instance of size (n/d) at distance greater than 0, and check that instance by reading all of its coordinates. Indeed, this will be our strategy. Unfortunately, a subtle problem arises, because when we shrink by picking 1/2 of the rows, the distance of the new instance might not be concentrated around its expectation. If, in any iteration, the new distance is much smaller than it should be, then the prover is "home free" and soundness fails.

To see this issue, take $d=2\sqrt{n}$ and $k=\ell=\sqrt{n}$. Suppose that the prover cheats by $n^{1/4}$ on every single row: over-estimating the weight of $(k/2+n^{1/4})$ rows, and under-estimating the weight of $(k/2-n^{1/4})$ rows. The row sums are still all within $O(n^{1/4})$ of each other, and indeed this type of behavior cannot be detected just by looking at the row sums. Now, when the verifier picks half of the rows at random, the *expected* distance of the new instance shrinks by 1/2 to $\sqrt{n}$ (as we wanted). However, the *variance* of the new distance is quite high: $\Omega(n)$. With constant probability, the distance of the new instance shrinks by more that 1/2, and the prover "wins". We note, however, that for this to happen the cheating prover *must* cheat on a large fraction of the rows (as in the example above). The verifier can detect this by checking the sums of a few rows (at only a mild cost in query complexity). This additional test guarantees that the set of cheating rows is not too large, which in turn bounds the variance of the new instance's distance—ensuring that it is concentrated around its expectation.

Proof of Theorem 5.6. The protocol is composed $r=\log(d/10)$ times sequentially. In each execution, the query complexity in Step 3b is at most $\text{poly}(\kappa)\cdot\ell = (n\cdot\text{polylog}(n)/d)$. In the base case, the

π is large: this requires$\Omega(n\log n)$ bits. However, we can use instead a de-randomized permutation, which requires only $\text{polylog}(n)$ bits to send (see details below).

Hamming Weight Protocol on input (X, w) with parameters d, r, κ

Set parameters $r = \log(d/10)$ and $\kappa = \text{polylog}(n)$. Take $\ell = d/(\kappa^2 \cdot \sqrt{\log n})$, and $k = n/\ell$. View X as a $k \times \ell$ matrix.

1. Permute. $\mathcal{V} \to \mathcal{P}$: send an $\log n$-wise independent function $\tau : [\ell] \to [k]$
   define the permutation $\pi : [k] \times [\ell] \to [k] \times [\ell]$, by $\pi(X)[i, j] = X[(i + \tau(j) \mod k), j]$
2. Sum Rows. $\mathcal{P} \to \mathcal{V}$: for each row $i \in [k]$, send $w_i = \|\pi(X)[i, *]\|$
3. Check Rows. $\mathcal{V}$: receive some $(w'_0, \ldots, w'_{k-1})$
   (a) verify that $\forall i \in [k], |w'_i - w/k| \leq \sqrt{\ell \cdot \kappa}/2$ (otherwise reject)
   (b) if $k > \ell$, then choose a uniformly random set $I \subseteq [k]$ of $(\kappa \cdot k/\ell)$ rows. Verify that $\forall i \in I, \|\pi(X)[i, *]\| = w'_i$ (otherwise reject)
4. Recurse. If $r > 1$, then:
   $\mathcal{V} \to \mathcal{P}$: pick uniformly at random $(b_0, \ldots, b_{k-1}) \in \{0, 1\}^k$
   $\mathcal{V}$ and $\mathcal{P}$ recursively run the protocol on input (X', w'), where X' is X restricted to rows i for which $b_i = 1$, and $w' = \sum_{i \in [k]} b_i \cdot w'_i$. The parameters for the recursive run are $d' = (d/2) \cdot (1 - 2/\kappa)$, $r' = r - 1$, and $\kappa' = \kappa$.
5. Base Case. If $r = 1$, $\mathcal{V}$: verify that $\|X\| = w$: if so, then accept, otherwise reject

Figure 5: Hamming Weight Protocol

instance size is (w.h.p.) at most $O(n \cdot \kappa/d)$. Thus the total query complexity is $(n \cdot \text{polylog}(n)/d)$. The communication complexity in Step 1 is $O(\log n \cdot (\log k + \log \ell))$ for sending the $\log n$-wise independent function τ, and in Step 2 it is $k \cdot \log n$ bits for sending the row sums. Thus, the total communication complexity is $O(k \cdot r \cdot \log n) = (n \cdot \text{polylog}(n)/d)$. The total number of messages exchanged is $2r$ (note that the Verifier's messages in Step 4, and then in Step 1 of the recursive execution, can be combined).

Completeness. If $(X, w) \in \textit{Hamming}$ and the prover honestly follows the protocol, then the only test that might make the verifier reject is checking that the row sums are all close to w/k in Step 3a. We will show that w.h.p. the (honest) row sums are all close to $\|X\|/k$ in each execution of the protocol. Completeness of the protocol follows.

Claim 5.8. *For each row $i \in [k]$, let w_i be its weight in $\pi(X)$. With all but $\text{negl}(n)$ probability over the choice of π, for every $i \in [k]$ simultaneously: $|w_i - (\|X\|/k)| \leq \sqrt{\ell \cdot \kappa}/2$*

Proof. Consider a protocol execution on input (X, w). For each row i and column j, examine the random variable $\pi(X)[i, j]$, where the randomness is over the $\log n$-wise independent function $\tau : [\ell] \to [k]$ chosen by the verifier. The weight w_i of the i-th row is the sum of the ℓ random variables $(\pi(X)[i, 0], \ldots, \pi(X)[i, \ell - 1])$.

Take $u_j = \|X[*, j]\|$ to be the Hamming Weight of X's j-th column. The expectation of $\pi(X)[i, j]$ is $E_{\tau(j)}[\pi(X)[i, j]] = u_j/k$. By linearity of expectation, for each row i we get that $E_\tau[w_i] = \|X\|/k$. Moreover, since τ is chosen from a $O(\log n)$-wise independent family of functions, we get that for each row i, the random variables $\{\pi(X)[i, j]\}_{j \in [\ell]}$ are $\log n$-wise independent. By

tail inequalities for the sum of $O(\log n)$-wise independent random variables (e.g. [CG89]) we get:

$$\Pr_{\tau}\left[|w_i - (\|X\|/k)| \geq \sqrt{\ell \cdot \kappa}/2\right] \leq (4\log n/\kappa)^{\log n} = negl(n)$$

The claim follows by taking a union bound over the k rows. □

Soundness. In each activation of the protocol on instance (X, w), we obtain a new instance (X', w'), and the instance size shrinks by (roughly) a half. The main concern for soundness is whether the distance d of the original instance might shrink by more than a half. We show that this is not the case (except with small probability):

Lemma 5.9. *If (X, w) is d-far from Hamming, then in a single protocol execution, before recursing, with all but $negl(n)$ probability over $\mathcal{V}$'s coins, either $\mathcal{V}$ rejects or the new instance (X', w') is at distance at least $(d/2) \cdot (1 - 2/\kappa)$ from Hamming.*

The proof is below. Using Lemma 5.9, we can recurse r times to obtain an instance of size $O(n/2^r)$. W.h.p. the resulting instance has distance $\Omega(d/2^r)$ (we use here the fact that $\kappa > r$). Since $r = \log(d/10) < \log d$, the resulting instance will not be in *Hamming*, and $\mathcal{V}$ will reject when it reaches the base case. Thus, soundness follows from the proof of Lemma 5.9.

Proof of Lemma 5.9. For each row i, let w_i denote its real weight in $\pi(X)$, and w'_i the (potentially cheating) weight as claimer by the prover. Let $d_i = (w'_i - w_i)$ be the difference between the true and claimed Hamming weights of row i (notice that this can be positive or negative). In Step 3a, the verifier checks that$\forall i \in [k], |w'_i - w/k| \leq \sqrt{\ell \cdot \kappa}$. Moreover, by Claim 5.8, we know that w.h.p the "true" weights are also concentrated around their expectation: with all but $negl(n)$ probability, for each row i we have that $|w_i - \|X\|/k| \leq \sqrt{\ell \cdot \kappa}$. By a triangle inequality, we conclude that with all but $negl(n)$ probability, for all $i \in [k]$:

$$\begin{aligned} |d_i| &= |w'_i - w_i| \\ &\leq |w'_i - w/k| + |w/k - \|X\|/k| + |\|X\|/k - w_i| \\ &\leq 2\sqrt{\ell \cdot \kappa} + d/k \end{aligned}$$

We have a bound on the magnitude of cheating in each row, but we also need a bound on the number m of cheating rows (see the discussion above in the overview). To upper bound the number of cheating rows, recall that in Step 3b, for $k > \ell$, the verifier checks $(\kappa \cdot k/\ell)$ rows for cheating. If the number of rows on which there is cheating is greater than ℓ, then (by a birthday paradox) with all $\exp(-\Omega(\kappa))$ probability, the the verifier will "hit' a cheating row and reject in this second test. We conclude that (if the verifier doesn't reject in the test above) w.h.p. the number of cheating rows is bounded $m \leq \min(k, \ell)$.

We now analyze the distance d' of the new instance (X', w'). We have that $d' = \sum_{i \in [k]} b_i \cdot d_i$. The expectation of d' is $\sum_{i \in [k]} d_i/2 \geq d/2$. The magnitude of each summand $b_i \cdot d_i$ is bounded by the magnitude of d_i (see above). Also, the number of non-zero summands (on which there is any cheating) is $m \leq \min(k, \ell)$. By Hoeffding's Inequality:

$$\Pr_{(b_0,\ldots,b_{k-1})}[d/2 - d' > d/\kappa] \quad < \quad e^{-2\frac{(d/\kappa)^2}{m \cdot (2\sqrt{\ell \cdot \kappa} + d/k)^2}}$$

To bound this probability, we separate into two cases. First, when $2\sqrt{\ell \cdot \kappa} \leq d/k$, we use $k \geq m$ and get:

$$\begin{aligned} e^{-2\frac{(d/\kappa)^2}{m\cdot(2\sqrt{\ell\cdot\kappa}+d/k)^2}} &\leq e^{-2\frac{(d/\kappa)^2}{k\cdot(2d/k)^2}} \\ &= e^{-k^2/2\kappa^2} \\ &= \exp(-\Omega(\kappa)) \end{aligned}$$

where the last equality follows because we set $k = n/\ell = (n/(d/(\kappa^2 \cdot \sqrt{\log n}))) > \kappa^2$.

In the second case, when $2\sqrt{\ell \cdot \kappa} > d/k$, we use $\ell \geq m$ and get:

$$\begin{aligned} e^{-2\frac{(d/\kappa)^2}{m\cdot(2\sqrt{\ell\cdot\kappa}+d/k)^2}} &\leq e^{-2\frac{(d/\kappa)^2}{\ell\cdot(4\sqrt{\ell\cdot\kappa})^2}} \\ &= e^{-\frac{d^2}{8\ell^2\cdot\kappa^3}} \\ &= \exp(-\Omega(\kappa)) \end{aligned}$$

where the last equality follows because we set $\ell = d/(\kappa^2 \cdot \sqrt{\log n})$.

We conclude that (in both cases), with all but $negl(n)$ probability, $(d' - d/2) \leq d/\kappa$.

□

□

References

[ABSRW04] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudorandom generators in propositional proof complexity. *SIAM J. Comput.*, 34(1):67–88, 2004.

[Baz09] Louay M. J. Bazzi. Polylogarithmic independence can fool dnf formulas. *SIAM J. Comput.*, 38(6):2220–2272, 2009.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 113–131, 1988.

[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.

[CG89] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *J. Complexity*, 5(1):96–106, 1989.

[CL88] Anne Condon and Richard E. Ladner. Probabilistic game automata. *Journal of Computer and System Sciences*, 36(3):452–489, 1988.

[CL89] Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 462–467, 1989.

[CMT10] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. In *ESA (1)*, pages 231–242, 2010.

[Con91] Anne Condon. Space-bounded probabilistic game automata. *Journal of the ACM*, 38(2):472–494, 1991.

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

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the pcp theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[DS92a] Cynthia Dwork and Larry J. Stockmeyer. Finite state verifiers i: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.

[DS92b] Cynthia Dwork and Larry J. Stockmeyer. Finite state verifiers ii: Zero knowledge. *Journal of the ACM*, 39(4):829–858, 1992.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004.

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[GKNR12] Oded Goldreich, Michael Krivelevich, Ilan Newman, and Eyal Rozenberg. Hierarchy theorems for property testing. *Computational Complexity*, 21(1):129–192, 2012.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[Gol10a] Oded Goldreich. Introduction to testing graph properties. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:82, 2010.

[Gol10b] Oded Goldreich, editor. *Property Testing*, volume 6390 of *LNCS*. Springer, 2010.

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, pages 59–68, 1986.

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

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

[Mei10] Or Meir. Ip = pspace using error correcting codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:137, 2010.

[Mic94] Silvio Micali. Cs proofs (extended abstract). In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 436–453, 1994.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. Northholland Publishing Company, 2nd edition, 1978.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[Raz09] Alexander A. Razborov. A simple proof of bazzi's theorem. *TOCT*, 1(1), 2009.

[Ron09] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[Rot09] Guy N. Rothblum. *Delegating computation Reliably: Paradigms and Constructions*. PhD thesis, MIT, 2009.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.