Electronic Colloquium on Computational Complexity, Report No. 180 (2013)

On Interactivity in Arthur-Merlin Communication and Stream Computation[*]

Amit Chakrabarti[†] Graham Cormode[‡] Andrew McGregor[§] Justin Thaler[¶]
Suresh Venkatasubramanian[||]

Abstract

We introduce *online interactive proofs* (OIP), which are a hierarchy of communication complexity models that involve both randomness and nondeterminism (thus, they belong to the Arthur–Merlin family), but are *online* in the sense that the basic communication flows from Alice to Bob alone. The complexity classes defined by these OIP models form a natural hierarchy based on the number of rounds of interaction between verifier and prover. We give upper and lower bounds that (1) characterize every finite level of the OIP hierarchy in terms of previously-studied communication complexity classes, and (2) separate the first four levels of the hierarchy. These results show marked contrasts and some parallels with the classic Turing Machine theory of interactive proofs.

Our motivation for studying OIP is to address computational complexity questions arising from the growing body of work on data stream computation aided by a powerful but untrusted helper. By carefully defining our complexity classes, we identify implicit assumptions in earlier lower bound proofs. This in turn indicates how we can break the mold of existing protocols, thereby achieving dramatic improvements. In particular, we present two-round stream protocols with logarithmic complexity for several query problems, including the fundamental INDEX problem. This was thought to be impossible based on previous work. We also present a near-optimal, one-round stream protocol for counting triangles in a dynamic graph. Our protocols leverage classic algebraic techniques, including multilinear extensions, sum check, and arithmetization of Boolean formulas.

[*]This work was supported in part by the Simons Institute for the Theory of Computing, and was performed while the authors were visiting the Institute.

[†]Department of Computer Science, Dartmouth College. Supported in part by NSF grant CCF-1217375.

[‡]Department of Computer Science, University of Warwick.

[§]Department of Computer Science, UMass Amherst. Supported in part by NSF grant IIS-1251110.

[¶]Simons Institute for the Theory of Computing, UC Berkeley. Supported by a Research Fellowship from the Simons Institute for the Theory of Computing.

[||]School of Computing, University of Utah. Supported in part by NSF grant IIS-1251049 and the Simons Institute for the Theory of Computing.

ISSN 1433-8092

1 Introduction

In a seminal work from the mid-1980s, Babai, Frankl and Simon [6] introduced and studied communication complexity analogues of the major Turing Machine complexity classes, including **P**, **NP**, $\mathbf{\Sigma}_2$, $\mathbf{\Pi}_2$. They hinted at similar analogues of **MA** and the **AM** hierarchy, a topic taken up in considerable detail by Klauck [25, 26] and Aaronson [1]. These works had largely approached the study of communication complexity classes as an intellectual exercise. Yet, a recent flurry of work on *verifiable* data stream computation has revealed compelling applications for the study of Arthur–Merlin communication.

To understand the idea of verifiable stream computation, let us consider the following scenario inspired by the surging popularity of commercial cloud computing services. A *verifier* (e.g., a retailer using a cloud-based service) lacks the resources to locally process a massive input (say, the set of all its transactions), but can access a powerful but untrusted *prover* (modeling the cloud service provider), who processes the input on the verifier's behalf. The verifier must work within the confines of the restrictive *data streaming* paradigm, using only a small amount of working memory. The prover must both answer queries about the input (say, "how many pairs of blue jeans have I ever sold?"), and prove that the answer is correct.

Several recent works have introduced closely related verifiable stream computation models that are of considerable theoretical interest in their own right. Some, such as the *annotated data streams* of Chakrabarti et al. [10] (subsequently studied in [11, 14, 15, 27]), are non-interactive, requiring the correctness proof to consist of just a single message from the prover to the verifier. Others, such as the *Arthur–Merlin streaming protocols* of Gur and Raz [11, 23] are "barely interactive" in the sense that the prover and the verifier may exchange a constant number of messages. Meanwhile, the *streaming interactive proofs* (SIPs) of Cormode et al. [15, 16] permit "many" rounds of interaction between the prover and the verifier. These works have begun to reveal a rich theory, leveraging algebraic techniques developed in the classical theory of interactive proofs [5, 19, 21, 36] to obtain efficient verification protocols for a variety of problems that require linear space in the standard (sans prover) streaming model.

A data-stream-with-prover algorithm naturally gives rise to a communication protocol in the Arthur–Merlin family: the split of the input between Alice and Bob (who together constitute Arthur) models the space constraint on the algorithm, while Merlin models the prover. The aforementioned works have thus led to new and interesting results for various flavors of Arthur–Merlin communication.

In this work, we provide a number of results belonging to two interrelated threads. In the communication complexity thread, we first identify a new hierarchy of Arthur–Merlin style classes called *online interactive proof* (**OIP**) classes, that are designed to model SIPs. We give various inclusion and separation results that together characterize the lower levels of this hierarchy and, more importantly, demonstrate that the **OIP** classes behave in starkly different ways than the classical complexity classes $\mathbf{IP}_{\mathsf{TM}}$ and $\mathbf{AM}_{\mathsf{TM}}$.[1] In the data stream thread, we give constant-round SIPs with polylogarithmic space and communication costs for several "query problems," including INDEX, range counting, and nearest neighbor search, yielding provably exponential improvements over prior art. (We remind the reader that "rounds" refers to interaction between the verifier and prover after reading the stream *in a single pass*.) We also give a near-optimal annotated data streaming protocol for counting triangles in a dynamic graph.

1.1 Overview of Our Contributions

Communication Complexity. Suppose Alice holds an input $x \in \mathcal{X}$, Bob holds $y \in \mathcal{Y}$, and they wish to compute $f(x,y)$ for some Boolean function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, using random coins and settling for some constant probability of error. Say this costs $\mathrm{R}(f)$ bits of communication. Can Merlin, who knows (x,y), convince them that $f(x,y) = 1$ in such a way that the overall communication is $o(\mathrm{R}(f))$? For several

[1] Throughout this paper, we use the subscript "TM" to denote a Turing-machine-based complexity class, to resolve the notation clash with the analogous communication complexity classes.

interesting functions f the answer is “Yes” and this is the general subject of Arthur–Merlin communication complexity. However, the complexity of f can vary a lot depending on the precise interaction pattern between Alice, Bob, and Merlin.

In the **MA** model, Merlin first broadcasts a “proof” that $f(x,y) = 1$ and then Alice and Bob use a randomized communication protocol to check this proof. A more restrictive communication model that nevertheless allows simulation of an annotated data stream algorithm is *one-way* or *online* **MA**, where messages go only from Merlin to Bob and Alice to Bob, with Bob announcing the output. To investigate SIPs, we introduce a more general communication model $\mathbf{OIP}^{[\mathbf{k}]}$, where the Alice-to-Bob communication remains one-way but Bob and Merlin are permitted $k \geq 1$ rounds of interaction. Essentially all “data-stream-with-prover” protocols developed to date can be simulated in this model with an appropriate value of k.

We submit that the careful study of this new model that we undertake here addresses key theoretical questions about the role of interactivity in the complexity of streaming interactive proofs. For example, our study reveals that the $\mathbf{OIP}^{[\mathbf{k}]}$ hierarchy behaves in dramatically different ways from classical $\mathbf{IP}_{\mathsf{TM}}$ and $\mathbf{MA}_{\mathsf{TM}}$. In particular, the following two fundamental phenomena are observed in classical interactive proofs:

- **Equivalence of private coin and public coin protocols.** Goldwasser and Sipser [22] proved that a k-round private coin interactive proof (à la $\mathbf{IP}_{\mathsf{TM}}$) can be simulated (with a polynomial blowup in complexity) by a $(k+2)$-round public coin one (à la $\mathbf{AM}_{\mathsf{TM}}$). Thus, in the resulting protocol, the verifier can perform his interaction with the prover before even looking at the input!
- **Round reduction.** Babai and Moran [7] showed that a $(k+1)$-round interactive proof can be simulated by a k-round interactive proof with a polynomial blowup in the verifier’s complexity. Thus, a 2-round (verifier-to-prover-and-back) interactive proof is just as powerful as any constant-round one.

We show that *analogous results do not hold in the OIP world*. First, there are $\mathbf{OIP}^{[\mathbf{2}]}$ protocols that cannot be efficiently simulated by any constant-round protocol where Bob interacts with Merlin without looking at his input. Second, there are exponential separations between $\mathbf{OIP}^{[\mathbf{1}]}, \mathbf{OIP}^{[\mathbf{2}]}, \mathbf{OIP}^{[\mathbf{3}]}$, and $\mathbf{OIP}^{[\mathbf{4}]}$.

Data Stream Computation. The technical core of many of our algorithmic results is a certain 2-round SIP that we call the *polynomial evaluation protocol*. Its simplest incarnation is a 2-round SIP for the INDEX problem with $O(\log n \log\log n)$ space and communication costs. This is obtained by adapting a result of Raz [33] about **IP/rpoly** (see Section 3). In particular, this gives a polylog-cost $\mathbf{OIP}^{[\mathbf{2}]}$ protocol for INDEX.

We proceed to give constant-round SIPs with polylogarithmic costs for a number of important *query problems* on data streams, in which the input consists of a streamed data set, followed by a query on the data set specified by a single additional token in the stream. Examples include histogram point queries, nearest neighbor queries, and range count queries. These SIPs permit both the prover and the verifier to process each stream update in logarithmic time, making them suitable for practical use. Our primary technical departure from earlier work on SIPs and related models [15, 16, 20, 23, 27] is that we exploit the verifier’s ability to send messages that *depend on the data stream*. In earlier protocols, verifier messages were independent of the input; our protocols are exponentially more efficient than what can be achieved in that restrictive setting.

Finally, we give a new annotated data stream algorithm for counting triangles in a graph. For n-vertex graphs, we achieve both space usage and proof length in $O(n \log n)$, which is nearly optimal. This affirmatively answers a question of Cormode [13], resolves the **MA** communication complexity of the problem up to a logarithmic factor, and improves over the best previous upper bound of $O(n^{3/2} \log n)$ [10].

1.2 Related Work

Aaronson and Wigderson [2] gave sublinear upper bounds on the online **MA** complexities of DISJOINTNESS and INNER-PRODUCT via a beautiful protocol using algebraic techniques similar to those in the famous *sum-check protocol* of Lund et al. [19]. Their protocol is nearly optimal, as shown by a lower bound of

Klauck [25] that applies more generally to **MA** protocols. The Aaronson–Wigderson protocol has served as the starting point for many annotated data streaming protocols. Klauck [26] performed a careful study of **AM**, **MA**, and its quantum analogue **QMA**. In particular, he gave a promise problem separating **QMA** from **AM**; we shall show that this same problem separates $\mathbf{OIP}^{[\mathbf{3}]}$ from $\mathbf{OIP}^{[\mathbf{4}]}$.

Work on annotated data streams has established optimal protocols for problems including frequency moments and frequent items [10]; linear algebraic problems such as matrix rank [27]; and graph problems like shortest *s*–*t* path [14]. Many of these protocols have subsequently been optimized for streams whose length is much smaller than the universe size [11]. Gur and Raz [23] studied "Arthur–Merlin streams," in which the verifier may send a single random string to the prover at the start of the protocol.

Cormode, Thaler, and Yi [16] showed that, given sufficient additional rounds of interaction, numerous problems can be solved with SIPs using exponentially less space and communication than in the annotated data stream model. Furthermore, several general $\mathbf{IP}_{\mathsf{TM}}$ protocols can be simulated in this model. These include the powerful, general-purpose protocol of Goldwasser, Kalai, and Rothblum [20]. Given any problem in **NC**, the resulting protocol requires only polylogarithmic space and communication while using polylogarithmic rounds of verifier–prover interaction. Refinements and implementations of these protocols [15, 37, 38] have demonstrated scalability and the practicality of this line of work.

2 Preliminaries and Statements of Results

2.1 Data Streaming and Communication Models

In a data stream problem, the input σ is a *stream*, or sequence, of *tokens* from some data universe $\mathcal{U}$. The goal is to compute or approximate some function $g(\sigma)$, keeping space usage sublinear in the two key size parameters: (1) the length of σ, and (2) the size of the universe $|\mathcal{U}|$. Practically speaking, we would also like to process each stream update (token arrival) quickly. All our data stream algorithms will be randomized, and we shall allow them to err with some small constant probability on each input stream. In the *streaming interactive proofs* (SIP) model, after processing σ, the algorithm (called the "verifier") may engage in k rounds of interaction with an oracle (the "prover") who knows σ and whose goal is to lead the verifier to output the correct answer $g(\sigma)$. The verifier, being distrustful, will output "$\perp$" (indicating "abort") if he suspects the prover to be cheating. When $k = 1$, the model corresponds to the *annotated data streams model*.

Communication problems arise naturally out of data stream problems if we suppose Alice holds a prefix of the input stream, and Bob the remaining suffix. The primary goal of such reductions is to obtain space lower bounds on data stream algorithms, so we are free to split the stream at any place we like. For example, many of the data stream problems in this work are *query problems*, where the input consists of a streamed data set, S, followed by a query, q, to S. In this case, it would be natural to split the input by giving S to Alice and q to Bob. Communication problems that will play an important role in this paper include the index problem $\textsc{index} : \{0,1\}^n \times [n] \to \{0,1\}$ where $\textsc{index}(x, j) = x_j$, and the set-intersection and set-disjointness problems $\textsc{inter}, \textsc{disj} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ where $\textsc{inter}(x,y) = \neg\textsc{disj}(x,y) = \bigvee_{i=1}^{n}(x_i \wedge y_i)$.

Communication Complexity Classes. All our communication models provide random coins and allow two-sided error probability up to a constant; when unspecified, this constant defaults to $1/3$. Given a communication model **C**, we denote the corresponding complexity measure of a problem f by $\mathrm{C}(f)$. Following Babai et al. [6], we also denote by **C** the corresponding complexity class, defined as the set of all functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ such that $\mathrm{C}(f) = (\log n)^{O(1)}$, i.e., functions that are "easy" in the model **C**.

We let $\mathbf{R}^{[\mathbf{k},\mathbf{A}]}$ denote the model of randomized communication complexity where Alice and Bob exchange $k \geq 1$ messages in total with Alice sending the first; $\mathbf{R}^{[\mathbf{k},\mathbf{B}]}$ is similar, except that Bob starts. In the **MA** model, the super-player Merlin, who sees all of the input, broadcasts a message at the start, following which Alice and Bob run a (two-way, arbitrary-round) randomized "verification" protocol. The $\mathbf{MA}^{[\mathbf{k},\mathbf{A}]}$

and $\mathbf{MA}^{[\mathbf{k},\mathbf{B}]}$ models are restrictions of **MA** where Merlin speaks only to Bob [2] and the verification protocol following Merlin's single message is restricted to lie in $\mathbf{R}^{[\mathbf{k},\mathbf{A}]}$ and $\mathbf{R}^{[\mathbf{k},\mathbf{B}]}$ respectively.

The **MA** model (indeed, its restriction $\mathbf{MA}^{[\mathbf{1},\mathbf{A}]}$) allows us to simulate annotated data stream protocols in an obvious way: Merlin sends Bob the prover's message, and Alice sends Bob the verifier's memory contents after it has processed her prefix of the stream. Notice that the order of the two messages is not important, modulo one crucial consideration: Alice must have a private channel to Bob and the random coins used to generate the message from Alice to Bob must be *hidden coins*, invisible to Merlin but shared between Alice and Bob (which is why we called them "hidden coins" rather than "private coins").

The models $\mathbf{OMA}^{[\mathbf{k}]}$, $\mathbf{OIP}^{[\mathbf{k}]}$, and $\mathbf{OIP}_{+}^{[\mathbf{k}]}$, for $k \geq 1$, are obtained by extending $\mathbf{MA}^{[\mathbf{1},\mathbf{A}]}$ to simulate k-round SIP protocols. These communication models work as follows. In each case, Alice and Bob first toss some hidden coins. Then, upon receiving the input, two things happen: (1) Merlin and Bob interact for k rounds, with Merlin sending the last message in the interaction, and (2) Alice sends Bob a message, randomized using the hidden coins. After these actions are completed, Bob produces an output in $\{0,1\}$. The differences between the three series of models are as follows.

- In $\mathbf{OMA}^{[\mathbf{k}]}$, (1) happens before (2) and Bob must interact with Merlin before looking at his input. This is directly analogous to $\mathbf{AM}_{\mathsf{TM}}$; see the discussion in Section 1.1.
- In $\mathbf{OIP}^{[\mathbf{k}]}$, (1) happens before (2) and Bob may look at his input before talking to Merlin.
- Finally, $\mathbf{OIP}_{+}^{[\mathbf{k}]}$ is like $\mathbf{OIP}^{[\mathbf{k}]}$ except that (2) happens before (1). Thus, Bob's messages may depend on Alice's actual message to Bob, not just on Bob's input and the hidden coins.

We remark that although $\mathbf{OIP}_{+}^{[\mathbf{k}]}$ appears to be the most natural communication analogue of k-round SIPs, all actual SIPs designed thus far, and all the ones we design in this work, can be simulated in the more restrictive $\mathbf{OIP}^{[\mathbf{k}]}$ model. In fact, earlier SIPs fit in the even more restrictive $\mathbf{OMA}^{[\mathbf{k}]}$ model.

In the **AM** model, the parties first choose a public random string, then Merlin broadcasts a message to Alice and Bob, who then run a deterministic communication protocol to arrive at a Boolean output. Since Merlin can in fact predict the exact transcript that Alice and Bob will generate following his message, we can assume without loss of generality that after Merlin's message, Alice and Bob output one bit each indicating whether or not they accept Merlin's prediction.

Cost and Value of Protocols. Let $\mathcal{P}$ be a protocol in a model **C** involving Merlin. For each input (x,y), $\mathcal{P}$ defines a game between Merlin and Arthur (recall that Alice and Bob together constitute Arthur), wherein Merlin's goal is to make Arthur output 1. We define the *value* $V^{\mathcal{P}}(x,y)$ to be Merlin's probability of winning this game with optimal play. Given a Boolean function f, we say that $\mathcal{P}$ computes f with *soundness error* ε_s and *completeness error* ε_c if, for all x,y we have

$$f(x,y)=0 \Rightarrow V^{\mathcal{P}}(x,y) \leq \varepsilon_s, \quad \text{and} \quad f(x,y)=1 \Rightarrow V^{\mathcal{P}}(x,y) \geq 1-\varepsilon_c. \tag{1}$$

When the above holds with $\varepsilon_c = 0$, we say that $\mathcal{P}$ computes f with perfect completeness.

The *verification cost* of $\mathcal{P}$, denoted $\mathrm{vc}(\mathcal{P})$, is the (worst-case) number of bits sent by Alice plus the number of hidden coin tosses; its *help cost* $\mathrm{hc}(\mathcal{P})$ is the number of bits communicated between Merlin and Bob; its communication cost $\mathrm{cc}(\mathcal{P}) = \mathrm{hc}(\mathcal{P}) + \mathrm{vc}(\mathcal{P})$. For a problem f, we define its complexity $\mathrm{C}(f) = \min\{\mathrm{cc}(\mathcal{Q}) : \mathcal{Q}$ is a **C** protocol that solves f with $\max\{\varepsilon_s, \varepsilon_c\} \leq 1/3\}$.

When analyzing SIPs, we usually have a non-Boolean function g in mind. We say that an SIP computes g with completeness error ε_c and soundness error ε_s if for all inputs x there exists a prover strategy that will cause the verifier to output $g(x)$ with probability at least $1-\varepsilon_c$, and no prover strategy can cause the verifier to output a value outside $\{g(x), \perp\}$ with probability larger than ε_s. The total length of the verifier–prover

[2]Our definition breaks symmetry between Alice and Bob because our eventual goal is to study online protocols.

interaction is the *help cost*. The space used by the data stream algorithm is the *space cost*. The cost of an SIP is then the sum of its help cost and its space cost. When designing SIP protocols we will also discuss the time complexities of the prover and the verifier. To keep things simple, we consider a model in which all arithmetic operations on a finite field of size $n^{O(1)}$ can be executed in unit time.

2.2 Exponentially Improved Constant-Round SIPs

In Section 3, we lead off with an important two-round SIP for the INDEX problem, considered as a query problem in the data stream setting: the input stream consists of n bits $x_1, \dots, x_n$, followed by an integer $j \in [n]$. The goal is to output x_j with high probability. The structure of the interaction in our SIP lets us draw an important conclusion about the corresponding communication problem.

Theorem 2.1. *The data stream problem* INDEX *has a two-round SIP with cost* $O(\log n \log\log n)$*, in which the verifier processes each stream token in* $O(\log n)$ *time and the prover runs in total time* $O(n \log n)$*. For the communication problem* INDEX*, we have* $\mathrm{OIP}^{[2]}(\mathrm{INDEX}) = O(\log n \log\log n)$*. In particular,* $\mathrm{INDEX} \in \mathbf{OIP}^{[2]}$.

We stress that this is a very unexpected result! Previous work gave a one-round SIP with cost $\tilde{O}(n^{1/2})$ [10] and a $(2k-1)$-round SIP with cost $\tilde{O}(n^{1/(k+1)})$ [16]. Meanwhile, Klauck and Prakash [27] gave lower bounds that appeared to have shown tightness of this rounds/cost tradeoff. However, their lower bound made an implicit assumption about the model: in our terminology, they only showed that $\mathrm{OMA}^{[2k-1]}(\mathrm{INDEX}) = \Omega(n^{1/(k+1)})$ for each constant k. With our nuanced understanding of the difference between **OMA** and **OIP**, we can pinpoint the essential feature that makes our protocol exponentially better than previous constant-round ones: the verifier's messages to the prover *must depend on some part of the input*. In the case of our protocol, the verifier's single message depends on the index j.

The protocol behind Theorem 2.1 can be seen as an incarnation of a rather general abstract protocol that we call the *polynomial evaluation protocol*. By instantiating this abstract protocol in other ways, we can obtain very efficient SIPs for other important query problems, including POINTQUERY, RANGECOUNT, and NEARESTNEIGHBOR. In the POINTQUERY problem, the input is a stream of updates to a vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{Z}^n$, followed by a query $j \in [n]$. Initially $\mathbf{x} = \mathbf{0}$, and an update is a tuple $(i, c) \in [n] \times \mathbb{Z}$, which has the effect of adding c to the entry x_i. The goal is to output x_j.

Theorem 2.2. *Suppose the input to* POINTQUERY *is guaranteed to satisfy* $|x_i| \le q$ *at end of the data stream, for all entries of* $\mathbf{x}$*, where the bound* q *is known a priori. Then there is a two-round SIP for* POINTQUERY *with space and help costs in* $O(\log n \log(q + \log n))$.

We obtain similar "logarithmic" cost SIPs for RANGECOUNT and NEARESTNEIGHBOR as well. Full formal statements of these results require the introduction of several parameters, so we defer them to Section 5. Our SIPs for NEARESTNEIGHBOR require an extra round between verifier and prover, ultimately because NEARESTNEIGHBOR is a retrieval problem (the answer is an actual data item) and we spend one round for the prover to claim to the verifier what the answer is. Therefore, these SIPs are three-round SIPs.

2.3 Relations Among Communication Complexity Classes

We prove a number of inclusion and separation results among our "new" communication complexity classes and relate them to previously studied classes. These are summarized in Figure 1.

The figure has several striking features. Every constant-height layer of the **OIP** hierarchy is *equivalent* to a natural communication complexity class that has previously been studied without reference to stream computation. The first four levels of the hierarchy are provably separated (in fact each separation is exponential). Concretely, for $\mathbf{OIP}^{[2]}$, we show that the INDEX problem in Theorem 2.1 cannot be replaced with the (much harder) DISJ problem. At least up to constant height, the **OIP** hierarchy collapses to the fourth

$$\begin{tikzcd}
\mathbf{R}^{[\mathbf{1,A}]} \arrow[d, leftrightarrow] & \mathbf{R}^{[\mathbf{2,B}]} \arrow[d, leftrightarrow] & \mathbf{MA}^{[\mathbf{2,B}]} \arrow[d, leftrightarrow] & \mathbf{AM} \arrow[d, leftrightarrow] & \mathbf{OMA}^{[\mathbf{k}]} \arrow[d, twoheadrightarrow] \\
\mathbf{OIP}^{[\mathbf{1}]} \arrow[r, twoheadrightarrow] \arrow[d, leftrightarrow] & \mathbf{OIP}^{[\mathbf{2}]} \arrow[r, twoheadrightarrow] \arrow[d, twoheadrightarrow] & \mathbf{OIP}^{[\mathbf{3}]} \arrow[r, twoheadrightarrow] & \mathbf{OIP}^{[\mathbf{4}]} \arrow[r, leftrightarrow] & \mathbf{OIP}^{[\mathbf{k}]} \\
\mathbf{OIP}_{+}^{[\mathbf{1}]} & \mathbf{R}^{[\mathbf{3,A}]} \arrow[r] & \mathbf{OIP}_{+}^{[\mathbf{2}]} \arrow[ur] & &
\end{tikzcd}$$

Figure 1: The layout of our communication complexity zoo. An arrow from $\mathbf{C}_1$ to $\mathbf{C}_2$ indicates that $\mathbf{C}_1 \subseteq \mathbf{C}_2$. If the arrow is double-headed, then the inclusion is strict. Within the figure, k is an arbitrary constant larger than 1.

level. In contrast, the $\mathbf{AM}_{\mathsf{TM}}$ hierarchy collapses to the second level [7]. Adding to the contrast with Turing Machine phenomena, the $\mathbf{OMA}^{[\mathbf{k}]}$ classes are exponentially weaker than their $\mathbf{OIP}^{[\mathbf{k}]}$ counterparts (in fact, weaker than even $\mathbf{OIP}^{[\mathbf{2}]}$) whereas their classical counterparts $\mathbf{IP}^{[\mathbf{k}]}_{\mathsf{TM}}$ and $\mathbf{AM}^{[\mathbf{k}]}_{\mathsf{TM}}$ are equivalent [22].

2.4 A Non-Interactive Protocol for Counting Triangles

Finally, we consider the data stream problem TRIANGLES, where the input consists of arrivals and departures of edges (u, v) of an undirected graph on vertex set $[n]$. The goal is to output the number of triangles in the final resulting graph. This requires $\Omega(n^2)$ space in the ordinary data stream model. Chakrabarti et al. [10] gave an annotated data stream algorithm for TRIANGLES with cost $O(n^{3/2} \log n)$; Cormode [13] asked whether this could be improved. Our next result shows that it can; we give a near-optimal upper bound, thereby also resolving the **MA** complexity of TRIANGLES up to a logarithmic factor.

Theorem 2.3. *There is an annotated data stream algorithm for* TRIANGLES *with space and help costs $O(n \log n)$. Every such algorithm requires the product of the space and help costs to be $\Omega(n^2)$.*

Like most prior work on annotated data streams, we use algebraic techniques as in the sum-check protocol of Lund et al. [19]. Yet, we deviate from all earlier annotated data stream protocols, as well as many prominent interactive protocols, in a big way. Roughly speaking, in previous protocols, the verifier's updates to her memory state were "commutative," in the sense that reordering the stream tokens would not change the final state reached by the verifier. However, our new verifier is inherently "non-commutative": her update to her state at time i depends on her actual state at time i. See Section 6.1 for further discussion.

Our protocol does not achieve smooth tradeoffs between space and help costs: we do not know how to reduce the space usage to $o(n \log n)$ without blowing the annotation length up to $\Omega(n^2)$, or vice versa. This is in contrast to prior work on annotated data streams [10, 11, 14, 23], which typically achieved any combination of space and help costs subject to the product of these two costs being above some threshold. We conjecture that achieving such smooth tradeoffs for TRIANGLES is impossible.

3 The Polynomial Evaluation Protocol and Its Applications

We shall present a two-round SIP for an abstract data stream problem called "polynomial evaluation," where the input consists of a multivariate polynomial described implicitly, as a table of values, followed by a point at which the polynomial must be evaluated. Without space constraints, this problem simply amounts to interpolation followed by direct evaluation, but our goal is to obtain a protocol where the verifier uses space roughly logarithmic in the size of the table of values, and is convinced by the prover about the correct answer after a similar amount of communication. For ease of presentation, we shall first consider a special setting that is important in its own right: the INDEX problem.

With very different motivations from ours, Raz [33] gave an interactive proof protocol placing *every* language in $\mathbf{IP}_{\mathsf{TM}}/\mathbf{rpoly}$, the class of languages that have interactive proofs with polynomial-time verifiers that take randomized advice, where the advice is kept secret from the prover. Our SIP for INDEX can be seen as an adaptation of Raz's interactive proof to the streaming setting.

The Setup. Recall that the input in the INDEX problem is a stream of *n data bits* $x_1, \ldots, x_n$, followed by a *query index* $j \in [n]$. Assume WLOG that $n = 2^b$, for some integer b. Identify each integer $z \in [n]$ with a Boolean vector $\mathbf{z} = (z_1, \ldots, z_b) \in \{0,1\}^b$ in some canonical way, such as by using the binary representation of z. We can then view the data bits as a table of values for the Boolean function $g_x : \{0,1\}^b \to \{0,1\}$ given by $g_x(\mathbf{z}) = x_z$, and thus for the multilinear b-variate polynomial $\widetilde{g}_x(Z_1, \ldots, Z_b)$ given by

$$\widetilde{g}_x(Z_1, \ldots, Z_b) = \sum_{\mathbf{z} \in \{0,1\}^b} g_x(\mathbf{z}) \chi_{\mathbf{z}}(Z_1, \ldots, Z_b), \text{ where} \tag{2}$$

$$\chi_{\mathbf{u}}(Z_1, \ldots, Z_b) = \prod_{i=1}^{b} \big((1-u_i)(1-Z_i) + u_i Z_i\big) \tag{3}$$

is the indicator function of the vector $\mathbf{u} = (u_1, \ldots, u_b)$. We shall interpret $\widetilde{g}_x$ as a polynomial in $\mathbb{F}[Z_1, \ldots, Z_b]$ for a fixed "large enough" finite field $\mathbb{F}$. With this interpretation, $\widetilde{g}_x$ is called the multilinear extension of g_x to $\mathbb{F}$. We define a *line* in $\mathbb{F}^b$ to be the range of a nonconstant affine function from $\mathbb{F}$ to $\mathbb{F}^b$. Every line contains exactly $|\mathbb{F}|$ points. Given such a line, ℓ, we define its *canonical representation* to be the degree-1 polynomial $\lambda_\ell(W) \in \mathbb{F}^b[W]$ such that $\lambda_\ell(0)$ and $\lambda_\ell(1)$ are, respectively, the lexicographically first and second points in ℓ. We define the *canonical restriction* of a polynomial $f(Z_1, \ldots, Z_b)$ to ℓ to be the univariate polynomial $f(\lambda_\ell(W)) \in \mathbb{F}[W]$, whose degree is at most the total degree of f.

Proof of Theorem 2.1. Adopting the notations and conventions outlined above, we give a two-round SIP protocol for the data stream problem INDEX, shown in Figure 2.

Input: Stream of data bits $(x_1, \ldots, x_n)$ where $n = 2^b$, followed by index $j \in [n]$.
Goal: Prover to convince Verifier to output the correct value of x_j.
Shared Agreement: Finite field $\mathbb{F}$ with $3b+1 \le |\mathbb{F}| \le 6b+2$; bijective map $u \in [n] \longleftrightarrow \mathbf{u} \in \{0,1\}^b$.

Initialization: Verifier picks $\mathbf{r} \in_R \mathbb{F}^b$ uniformly at random, sets $Q \leftarrow 0$.

Stream Processing: Upon reading x_z, where $z \in [n]$, Verifier updates $Q \leftarrow Q + x_z \chi_{\mathbf{z}}(\mathbf{r})$.

Query Handling: Upon reading the index j, Verifier interacts with Prover as follows:

1. If $\mathbf{j} = \mathbf{r}$, Verifier outputs Q as the answer. Otherwise, he sends Prover ℓ, the unique line in $\mathbb{F}^b$ through $\mathbf{j}$ and $\mathbf{r}$.
2. Prover sends Verifier a polynomial $h(W) \in \mathbb{F}[W]$ of degree at most b, claiming that it is the canonical restriction of the multilinear polynomial $\widetilde{g}_x(Z_1, \ldots, Z_b)$ to the line ℓ. That is, Prover claims that $h(W) \equiv \widetilde{g}_x(\lambda_\ell(W))$.
3. Let $w, t \in \mathbb{F}$ be such that $\lambda_\ell(w) = \mathbf{j}$ and $\lambda_\ell(t) = \mathbf{r}$. Verifier checks that $h(t) = Q$, aborting if not. If the check passes, Verifier outputs $h(w)$ as the answer.

Figure 2: A Two-Round Streaming Interactive Proof (SIP) Protocol for the INDEX Problem

To analyze this protocol, first note that after reading all the data bits, the verifier would have computed $Q = \widetilde{g}_x(\mathbf{r})$, by Eq. (2). Now the protocol is easily seen to have perfect completeness. Since $\widetilde{g}_x(Z_1, \dots, Z_b)$ is multilinear, it follows that $\deg \widetilde{g}_x(\lambda_\ell(W)) \le b$, so the prover can always honestly choose $h(W) = \widetilde{g}_x(\lambda_\ell(W))$. If he does so, then we will indeed have $h(t) = \widetilde{g}_x(\lambda_\ell(t)) = \widetilde{g}_x(\mathbf{r}) = Q$, and the verifier's check will pass. Finally, the verifier will output $h(w) = \widetilde{g}_x(\lambda_\ell(w)) = \widetilde{g}_x(\mathbf{j}) = x_j$, the correct answer to the INDEX instance.

Next, we analyze soundness. If the prover supplies a polynomial $h(W) \not\equiv \widetilde{g}_x(\lambda_\ell(W))$, then, since both polynomials have degree at most b, they agree at at most b points in $\mathbb{F}$. From the prover's perspective after he receives the verifier's message, $\mathbf{r}$ is uniformly distributed in $\ell \setminus \{\mathbf{j}\}$. Thus, $\Pr_{\mathbf{r}}[h(t) = Q] \le b/(|\mathbb{F}| - 1) \le 1/3$.

Now we consider this protocol's costs. The verifier maintains the random point $\mathbf{r} \in \mathbb{F}^b$ and the running sum $Q \in \mathbb{F}$, using $O(b \log |\mathbb{F}|)$ space. He sends the prover ℓ, which is specified by two elements of $\mathbb{F}^b$, and receives a degree-b polynomial in $\mathbb{F}[W]$; both communications use at most $O(b \log |\mathbb{F}|)$ bits. Recalling that $|\mathbb{F}| \le 6b + 2$, we see that both space and communication costs are in $O(b \log b) = O(\log n \log\log n)$.

Finally, we consider the verifier's and prover's runtimes. The honest prover must send the univariate polynomial $\widetilde{g}_x(\lambda_\ell(W))$. Since $\widetilde{g}_x$ has degree at most b, it suffices for the prover to specify the evaluations of $\widetilde{g}_x(\lambda_\ell(W))$ at $b + 1 = O(\log n)$ points. A direct application of Eqs. (2) and (3) shows that each evaluation can be done in $O(n \log n)$ time, resulting in a total runtime of $O(n \log^2 n)$. However, using now-standard memoization techniques (see e.g. [38, Section 5.1]), it is possible for the prover to in fact perform each of these evaluations in just $O(n)$ time, resulting in a total runtime of $O(n \log n)$.

The verifier can clearly run in $O(b) = O(\log n)$ time per stream update, as each stream update x_z only requires the verifier to compute $\chi_{\mathbf{z}}(\mathbf{r})$, and it follows from Eq. (3) that this can be done with $O(b)$ field operations. During the interaction with the prover, the verifier runs in $\operatorname{polylog} n$ time. Indeed, to compute the prescribed message to the prover, the verifier merely needs determine the line ℓ through the points $\mathbf{j}$ and $\mathbf{r}$, which can be done in $O(b) = O(\log n)$ time. To process the message from the prover, the verifier must evaluate the polynomial h sent by the prover at the points t and w; both of these evaluations can be done in $\operatorname{polylog} n$ time.

This proves the data streaming portion of Theorem 2.1. To prove the communication complexity portion, we simply note that the above SIP protocol can be simulated in $\mathbf{OIP}^{[2]}$: the hidden coins shared between Alice and Bob determine $\mathbf{r}$, and Bob can send Merlin ℓ without having to hear from Alice, since ℓ is determined entirely by $\mathbf{r}$ and Bob's input j. □

Generalization to Polynomial Evaluation. The above SIP protocol uses very little of the special structure of the INDEX problem. Let us abstract out its salient features, so as to handle the general problem described at the start of this section. First, note the protocol treats the data set given by $(x_1, \dots, x_n)$ as an implicit description of the polynomial $\widetilde{g}_x$. Second, note that our soundness analysis did not require multilinearity per se, only an upper bound on the total degree of $\widetilde{g}_x$. Finally, note that the specific form of Eqs. (2) and (3) is not crucial either; all we used was that it allows the verifier an easy streaming computation.

Thus, suppose our data stream implicitly describes a v-variate polynomial g of total degree d over a field $\mathbb{F}$, followed by a point $\mathbf{j} \in \mathbb{F}^v$. Suppose this implicit description allows a streaming verifier to evaluate g at a random point $\mathbf{r} \in_R \mathbb{F}^v$ using space S. Then the technique of the protocol in Figure 2 gives a two-round SIP for computing $g(\mathbf{j})$, with the following properties: (1) perfect completeness; (2) soundness error bounded by $d/(|\mathbb{F}| - 1)$; (3) space usage $O(v \log |\mathbb{F}| + S)$; (4) communication cost $O((d + v) \log |\mathbb{F}|)$. We shall refer to this abstract protocol as the *polynomial evaluation protocol.*

Update Streams and Point Queries. We turn to proving Theorem 2.2 about the POINTQUERY problem.

Proof of Theorem 2.2. Let σ be an input stream for POINTQUERY, consisting of updates to the vector $\mathbf{x} \in \mathbb{Z}^n$, followed by a query $j \in [n]$. Assume WLOG that $n = 2^b$ for an integer b, and use a bijection $u \in [n] \longleftrightarrow \mathbf{u} \in \{0, 1\}^b$ as in Theorem 2.1. The vector $\mathbf{x}$ resulting from the updates defines a multilinear polynomial $\widetilde{g}_{\mathbf{x}}(Z_1, \dots, Z_b)$ by Eq. (2), where $g_{\mathbf{x}}(\mathbf{z}) := x_z$. We can treat $\widetilde{g}_{\mathbf{x}}$ as a polynomial over any field we like, but to solve our problem, we need to tell apart the $2q+1$ possible values taken on by the entries of $\mathbf{x}$ (recall that q is an upper bound on $\|\mathbf{x}\|_\infty$ at the end of the stream). For this it suffices to have $\text{char}(\mathbb{F}) \geq 2q+1$.

Applying the polynomial evaluation protocol is now straightforward. The verifier starts with $\mathbf{r} \in_R \mathbb{F}^b$ and $Q = 0$. Upon receiving an update indicating "$x_i \leftarrow x_i + c$," he updates $Q \leftarrow Q + c\chi_{\mathbf{i}}(\mathbf{r})$. The other details are as in Figure 2. The space and communication costs are both in $O(b \log |\mathbb{F}|)$ as before.

To ensure a soundness error of at most $1/3$, we let $|\mathbb{F}| > 3b$ as before. This and the earlier condition on $\text{char}(\mathbb{F})$ can both be satisfied by, e.g., taking $\mathbb{F} = \mathbb{F}_p$, for a prime $p > 3b + 2q$. This translates to cost bounds in $O(\log n \log(q + \log n))$, as claimed. $\square$

Other Applications. In Section 4, we shall use the polynomial evaluation protocol as a key technical tool in some of our theorems characterizing several levels of the **OIP** hierarchy. In Section 5, we build on the polynomial evaluation protocol to obtain efficient constant-round SIPs for the NEARESTNEIGHBOR and RANGECOUNT problems.

4 A Communication Complexity Zoo

We now study our central communication models $\mathbf{OIP}^{[\mathbf{k}]}$ and $\mathbf{OIP}_+^{[\mathbf{k}]}$, and prove the web of relationships given in Figure 1. Our results are of two types: (1) establishing separations or collapses between levels of the **OIP** and $\mathbf{OIP}_+$ hierarchies, as the case may be, and (2) relating these hierarchies to other previously studied communication complexity classes.

We start with an easy observation that is immediate from our definitions.

Observation 4.1. *We have* $\mathbf{OMA}^{[\mathbf{1}]} = \mathbf{OIP}^{[\mathbf{1}]} = \mathbf{OIP}_+^{[\mathbf{1}]} = \mathbf{MA}^{[\mathbf{1,A}]}$.

Throughout this section, f will denote an arbitrary communication problem given by a Boolean function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, and n will parametrize its "instance size" up to a constant factor, i.e., we will have $\log|\mathcal{X}| + \log|\mathcal{Y}| = \Theta(n)$. We shall use big-$O$ and big-Ω notation to hide constants independent of f, $|\mathcal{X}|$ and $|\mathcal{Y}|$. We shall use the term "ordinary protocol" to mean a randomized communication protocol involving Alice and Bob alone (and no Merlin).

4.1 A Characterization of $\mathbf{OIP}^{[\mathbf{2}]}$

The fact that INDEX $\in \mathbf{OIP}^{[\mathbf{2}]}$ (Theorem 2.1) is striking: combined with the well-known lower bound $\mathbf{R}^{[\mathbf{1,A}]}(\text{INDEX}) = \Omega(n)$, it shows that introducing Merlin into the picture while keeping the one-way restriction on the Alice/Bob communication lowers cost exponentially. It is now natural to ask whether $\mathbf{OIP}^{[\mathbf{2}]}$ allows such exponential savings for harder problems, such as DISJ. Our next result—a lower bound on $\mathbf{OIP}^{[\mathbf{2}]}$ complexity—implies that it does not.

Theorem 4.2. *Let* $\mathcal{P}$ *be an* $\mathbf{OIP}^{[\mathbf{2}]}$ *protocol computing* f. *Then* $\text{hc}(\mathcal{P})\,\text{vc}(\mathcal{P}) = \Omega(\text{R}^{[2,B]}(f))$. *In particular,* $\text{OIP}^{[2]}(f) = \Omega\big(\text{R}^{[2,B]}(f)^{1/2}\big)$, *which implies* $\mathbf{OIP}^{[\mathbf{2}]} \subseteq \mathbf{R}^{[\mathbf{2,B}]}$.

Proof. After appropriate parallel repetition, we may assume that the soundness and completeness errors of $\mathcal{P}$ at most $1/12$ each. In general, $\mathcal{P}$ takes the following shape: (1) hidden coins are tossed, generating random string r according to distribution $\mathcal{D}$; (2) Bob sends Merlin a message $m_B = m_B(y,r)$; (3) Merlin responds with a message $m_M = m_M(x,y,m_B)$; (4) Alice sends Bob a message $m_A = m_A(x,r)$; (5) Bob outputs a bit given by a function $\text{out}^P(y, m_M, m_A)$. Let $\mathcal{D}_m$ be $\mathcal{D}$ conditioned on the event $\{m_B = m\}$. With this notational setup, we now describe (in Figure 3) a two-message ordinary protocol $\mathcal{Q}$ that we claim computes f.

1. Bob samples $\bar{r} \sim \mathcal{D}$, computes $\overline{m} = m_B(y, \bar{r})$, then sends Alice i.i.d. samples $r^{(1)}, \ldots, r^{(h)} \sim \mathcal{D}_{\overline{m}}$, where $h = 36(\mathrm{hc}(\mathcal{P}) + 4)$.
2. Alice sends Bob $m_A\big(x, r^{(1)}\big), \ldots, m_A\big(x, r^{(h)}\big)$.
3. Bob outputs 1 iff $\exists m_M : \; |\{i \in [h] : \mathrm{out}^{\mathcal{P}}(y, m_M, m_A(x, r^{(i)})) = 1\}| > h/2$.

Figure 3: The $\mathbf{R}^{[\mathbf{2},\mathbf{B}]}$ protocol $\mathcal{Q}$, which simulates the $\mathbf{OIP}^{[\mathbf{2}]}$ protocol $\mathcal{P}$.

To analyze this protocol, let us first define the *weight* $W_{x,y}(\overline{m})$ of a Bob-message $\overline{m}$ to be the probability that Merlin, playing optimally after receiving $\overline{m}$, convinces Bob to output 1. That is,

$$W_{x,y}(\overline{m}) = \max_{m_M} \Pr_{r \sim \mathcal{D}_{\overline{m}}} \Big[\mathrm{out}^{\mathcal{P}}(y, m_M, m_A(x, r)) = 1 \Big] . \tag{4}$$

Then, with $\overline{m} \sim m_B(y, \mathcal{D})$, the expected weight $\mathbb{E}_{\overline{m}}[W_{x,y}(\overline{m})]$ is at least $11/12$ when $f(x,y) = 1$ and at most $1/12$ when $f(x,y) = 0$.

Correctness on 1-inputs: Fix $(x,y) \in f^{-1}(1)$. We shall proceed assuming that the specific Bob-message $\overline{m}$ chosen in Step 1 of $\mathcal{Q}$ satisfies $W_{x,y}(\overline{m}) > 2/3 = 1 - 4(1/12)$; by Markov's inequality, this fails to happen with probability at most $1/4$. Studying Eq. (4) tell us that there exists a specific Merlin-message m_M^* such that $\Pr_r[\mathrm{out}^{\mathcal{P}}(y, m_M^*, m_A(x, r)) = 1] > 2/3$. Therefore, according to the strategy in Steps 2 and 3, the size of the set $\{i \in [h] : \mathrm{out}^{\mathcal{P}}(y, m_M^*, m_A(x, r^{(i)})) = 1\}$ is a sum of h i.i.d. indicators and exceeds $2h/3$ in expectation. By standard Chernoff bounds (e.g., [29, Theorem 4.4]), the probability that Bob outputs 0 is $2^{-\Omega(h)}$. Thus, overall, the probability that $\mathcal{Q}$ outputs 0 on input (x,y) is at most $1/4 + 2^{-\Omega(h)} < 1/3$.

Correctness on 0-inputs: Fix $(x,y) \in f^{-1}(0)$. We shall proceed assuming that the specific Bob-message $\overline{m}$ chosen in Step 1 of $\mathcal{Q}$ satisfies $W_{x,y}(\overline{m}) < 1/3$; by Markov's inequality, this fails to happen with probability at most $1/4$. For each specific Merlin-message m_M, define

$$\mathrm{size}(m_M) = \Big| \{i \in [h] : \mathrm{out}^{\mathcal{P}}(y, m_M, m_A(x, r^{(i)})) = 1\} \Big| .$$

Then $\mathrm{size}(m_M)$ is a sum of h i.i.d. indicators and has expectation below $h/3$. By standard Chernoff bounds, $\Pr[\mathrm{size}(m_M) > h/2] \leq e^{-h/36}$. By a union bound over all possible Merlin-messages m_M, the probability that Bob outputs 1 is at most $2^{\mathrm{hc}(\mathcal{P})} e^{-h/36} < 2^{-4}$, using our choice of h. Adding in the $1/4$ from our Markov argument earlier, the overall probability that $\mathcal{Q}$ outputs 1 on input (x,y) is at most $1/4 + 2^{-4} < 1/3$.

Communication Cost: By definition of the $\mathbf{OIP}^{[\mathbf{2}]}$ model, we have $|r| \leq \mathrm{vc}(\mathcal{P})$ and $|m_A| \leq \mathrm{vc}(\mathcal{P})$. Thus, each of the two messages in $\mathcal{Q}$ costs at most $h \cdot \mathrm{vc}(\mathcal{P}) = O(\mathrm{hc}(\mathcal{P})\,\mathrm{vc}(\mathcal{P}))$ bits. □

The above proof exploits a key property of $\mathbf{OIP}^{[\mathbf{2}]}$ protocols: Bob can sample from the conditional distribution $\mathcal{D}_{\overline{m}}$. This is possible because $m_B = m_B(y, r)$ is independent of Alice's message m_A, a property not satisfied in the stronger $\mathbf{OIP}_{+}^{[\mathbf{2}]}$ model. This explains why Theorem 4.2 does not apply to $\mathbf{OIP}_{+}^{[\mathbf{2}]}$, and indeed we shall later give an exponential separation between $\mathbf{OIP}^{[\mathbf{2}]}$ and $\mathbf{OIP}_{+}^{[\mathbf{2}]}$ in Corollary 4.17.

Corollary 4.3. *We have* $\Omega(n^{1/2}) \leq \mathrm{OIP}^{[2]}(\textsc{disj}) \leq O(n^{1/2} \log n)$. *In particular,* $\textsc{disj} \notin \mathbf{OIP}^{[\mathbf{2}]}$.

Proof. For the lower bound, we combine Theorem 4.2 with the fact that $\mathbf{R}^{[\mathbf{2},\mathbf{B}]}(\textsc{disj}) \geq \mathrm{R}(\textsc{disj}) = \Omega(n)$, the last step being a celebrated lower bound [24]. The upper bound follows from the Aaronson–Wigderson protocol [2] for $\textsc{disj}$, which is in fact an $\mathbf{MA}^{[\mathbf{1},\mathbf{A}]}$ protocol. □

We have now seen that up to polynomial (specifically, quadratic) blowup, $\mathbf{OIP}^{[\mathbf{2}]}$ is no more powerful than ordinary $\mathbf{R}^{[\mathbf{2},\mathbf{B}]}$. We now show that up to another quadratic blowup this is in fact a characterization.

Theorem 4.4. *For all f, we have* $\mathrm{OIP}^{[2]}(f) = O\big(\mathrm{R}^{[2,B]}(f)^2\big)$. *In particular,* $\mathbf{OIP^{[2]}} \supseteq \mathbf{R^{[2,B]}}$.

Proof. Let $\mathcal{Q}$ be an $\mathbf{R^{[2,B]}}$ protocol for f with cost C and error at most $1/6$. Assume WLOG that $C \geq 5$ and that each of the two messages in $\mathcal{Q}$ is a string in $\{0,1\}^C$. We shall treat Alice's messages as elements of the field $\mathbb{F} = \mathbb{F}_{2^C}$ via an agreed-upon bijection.

We design an $\mathbf{OIP^{[2]}}$ protocol $\mathcal{P}$ for f, based on $\mathcal{Q}$. Given an input (x,y), $\mathcal{P}$ begins by choosing a (hidden) random string r shared between Alice and Bob exactly as $\mathcal{Q}$ would have. From now on, think of x,y,r as fixed. This then fixes a message m_B that Bob would have sent Alice in $\mathcal{Q}$, as well as a function $m_A : \{0,1\}^C \to \mathbb{F}$ specifying Alice's response to each Bob-message. Let $\widetilde{m}_A(Z_1,\ldots,Z_C) \in \mathbb{F}[Z_1,\ldots,Z_C]$ be the multilinear extension of this function m_A. In $\mathcal{P}$, Alice needs to send a message to Bob that allows him to determine $m_A(m_B) = \widetilde{m}_A(m_B)$ with Merlin's help. This is an instance of polynomial evaluation, so we solve it by applying the $\mathbf{OIP^{[2]}}$ polynomial evaluation protocol (PEP) from Section 3.

The polynomial $\widetilde{m}_A$ is C-variate and has total degree C. Therefore, by the discussion in Section 3, PEP has communication cost $O(C \log|\mathbb{F}|) = O(C^2)$, as does $\mathcal{P}$. Next, PEP has perfect completeness, so an honest Merlin can cause $\mathcal{P}$ to output 1 whenever the choice of r would have caused $\mathcal{Q}$ to output 1. Finally, PEP has soundness error at most $C/(|\mathbb{F}|-1) = C/(2^C-1) < 1/6$, so a dishonest Merlin can cause $\mathcal{P}$ to differ in output from $\mathcal{Q}$ with probability at most $1/6$. Using the error bound of $1/6$ on $\mathcal{Q}$, we conclude that $\mathcal{P}$ has completeness error at most $1/6$ and soundness error at most $1/6 + 1/6 = 1/3$. □

Corollary 4.5. *For all f, we have* $\Omega\big(\mathrm{R}^{[2,B]}(f)^{1/2}\big) \leq \mathrm{OIP}^{[2]}(f) \leq O\big(\mathrm{R}^{[2,B]}(f)^2\big)$. *Thus,* $\mathbf{OIP^{[2]}} = \mathbf{R^{[2,B]}}$.

Proof. Combine Theorems 4.2 and 4.4. □

4.2 A Characterization of $\mathbf{OIP^{[3]}}$

We now turn to characterizing $\mathbf{OIP^{[3]}}$. We give a lower bound that builds on the argument in Theorem 4.2. Just as before, we can then derive a lower bound for the specific problem DISJ.

Theorem 4.6. *Let $\mathcal{P}$ be an $\mathbf{OIP^{[3]}}$ protocol computing f. Then there is an $\mathbf{MA^{[2,B]}}$ protocol $\mathcal{Q}$ computing f with* $\mathrm{hc}(\mathcal{Q}) \leq \mathrm{hc}(\mathcal{P})$ *and* $\mathrm{vc}(\mathcal{Q}) = O(\mathrm{hc}(\mathcal{P})\,\mathrm{vc}(\mathcal{P}))$. *In particular,* $\mathrm{OIP}^{[3]}(f) = \Omega\big(\mathrm{MA}^{[2,B]}(f)^{1/2}\big)$, *which implies* $\mathbf{OIP^{[3]}} \subseteq \mathbf{MA^{[2,B]}}$.

Proof. The high-level idea for building $\mathcal{Q}$ is as follows. After Merlin sends his first message to Bob in $\mathcal{P}$, the remainder of $\mathcal{P}$ is an $\mathbf{OIP^{[2]}}$ protocol. Theorem 4.2 shows how to cut Merlin out of this remaining protocol, replacing it with $\mathbf{R^{[2,B]}}$ protocol. After this replacement, the result is an $\mathbf{MA^{[2,B]}}$ protocol.

In more detail, suppose $\mathcal{P}$ has completeness and soundness errors at most $1/12$. Let $\mathcal{P}_m$ denote the $\mathbf{OIP^{[2]}}$ protocol obtained from $\mathcal{P}$ by fixing Merlin's *first* message to m. Let $\mathcal{Q}_m$ be the $\mathbf{R^{[2,B]}}$ protocol simulating $\mathcal{P}_m$ as in Figure 3. Note that $\mathrm{cc}(\mathcal{Q}_m) = O(\mathrm{hc}(\mathcal{P}_m)\,\mathrm{vc}(\mathcal{P}_m))$. Let $\mathcal{Q}$ be the $\mathbf{MA^{[2,B]}}$ protocol where we use $\mathcal{Q}_m$ as Arthur's verification strategy for a message from Merlin. We claim that $\mathcal{Q}$ computes f.

Completeness: Fix $(x,y) \in f^{-1}(1)$. By the completeness of $\mathcal{P}$, there exists some first message m^* from Merlin such that $\mathcal{P}_{m^*}$ outputs 1 with probability at least $11/12$. By the completeness analysis in Theorem 4.2, $\mathcal{Q}_{m^*}$ outputs 1 with probability at least $2/3$. Therefore, if Merlin sends the message m^* in $\mathcal{Q}$, he will cause the output to be 1 with probability at least $2/3$.

Soundness: Fix $(x,y) \in f^{-1}(0)$. By the soundness of $\mathcal{P}$, for every possible first message, m, that Merlin may send, $\mathcal{P}_m$ outputs 1 with probability at most $1/12$. By the soundness analysis in Theorem 4.2, for all m, $\mathcal{Q}_m$ outputs 1 with probability at most $1/3$. Therefore, no matter what Merlin sends as his message in $\mathcal{Q}$, he can cause the output to be 1 with probability at most $1/3$.

Costs: Merlin in $\mathcal{Q}$ sends only part of what Merlin in $\mathcal{P}$ sends; therefore $\mathrm{hc}(\mathcal{Q}) \leq \mathrm{hc}(\mathcal{P})$. Furthermore, $\mathrm{vc}(\mathcal{Q}) \leq \max_m \mathrm{cc}(\mathcal{Q}_m) = \max_m O(\mathrm{hc}(\mathcal{P}_m)\,\mathrm{vc}(\mathcal{P}_m)) = O(\mathrm{hc}(\mathcal{P})\,\mathrm{vc}(\mathcal{P}))$. □

Corollary 4.7. *We have* $\Omega(n^{1/3}) \le \mathrm{OIP}^{[3]}(\textsc{disj}) \le O(n^{1/3}\log n)$*. In particular,* $\textsc{disj} \notin \mathbf{OIP^{[3]}}$.

Proof. Klauck [25] proved that $\mathrm{MA}(\textsc{disj}) = \Omega(n^{1/2})$. Applying Theorem 4.6 to this result gives the non-tight bound $\mathrm{OIP}^{[3]}(\textsc{disj}) = \Omega(n^{1/4})$. But we observe that Klauck's proof shows something stronger: namely, if an **MA** protocol $\mathcal{Q}$ computes $\textsc{disj}$, then $\mathrm{hc}(\mathcal{Q})\,\mathrm{vc}(\mathcal{Q}) = \Omega(n)$. Combining Theorem 4.6 with *this* result, we conclude that if an $\mathbf{OIP^{[3]}}$ protocol $\mathcal{P}$ computes $\textsc{disj}$, then $\mathrm{hc}(\mathcal{P})^2\,\mathrm{vc}(\mathcal{P}) = \Omega(n)$, and therefore $\mathrm{hc}(\mathcal{P}) + \mathrm{vc}(\mathcal{P}) = \Omega(n^{1/3})$.

For the upper bound, we note that Aaronson and Wigderson [2] also gave an online **MAMA** protocol for $\textsc{disj}$ of cost $O(n^{1/3}\log n)$. Every online **MAMA** protocol admits a simulation in $\mathbf{OIP^{[3]}}$. □

As we did for the second level in the **OIP** hierarchy, we give an upper bound that applies to the third level and gives a characterization that is tight up to a quadratic blowup.

Theorem 4.8. *For all f, we have* $\mathrm{OIP}^{[3]}(f) = O\big(\mathrm{MA}^{[2,B]}(f)^2\big)$*. In particular,* $\mathbf{OIP^{[3]}} \supseteq \mathbf{MA^{[2,B]}}$.

Proof sketch. We build on the argument in Theorem 4.4 exactly as the proof of Theorem 4.6 builds on Theorem 4.2. Given an $\mathbf{MA^{[2,B]}}$ protocol $\mathcal{Q}$ of cost C, the verification strategy used by Alice and Bob in $\mathcal{Q}$ is an $\mathbf{R^{[2,B]}}$ protocol of cost C, which we can replace with an $\mathbf{OIP^{[2]}}$ protocol of cost $O(C^2)$, by Theorem 4.4. After this replacement we have an $\mathbf{OIP^{[3]}}$ protocol. The remaining analysis is routine. □

Corollary 4.9. *For all f,* $\Omega\big(\mathrm{MA}^{[2,B]}(f)^{1/2}\big) \le \mathrm{OIP}^{[3]}(f) \le O\big(\mathrm{MA}^{[2,B]}(f)^2\big)$*. Thus,* $\mathbf{OIP^{[3]}} = \mathbf{MA^{[2,B]}}$.

Proof. Combine Theorems 4.6 and 4.8. □

4.3 A Characterization of $\mathbf{OIP^{[4]}}$ and Beyond

The fourth level of the **OIP** hierarchy turns out to have surprising power. It can capture all of **AM**, a model that lies at the frontier of our current understanding of communication complexity classes in the sense that we do not know any nontrivial **AM** lower bounds. Thanks to this surprising power, we can show that all constant-height levels of the **OIP** hierarchy collapse to the fourth level.

Theorem 4.10. *For all f, we have* $\mathrm{OIP}^{[4]}(f) = O(\mathrm{AM}(f)\log \mathrm{AM}(f))$*. In particular,* $\mathbf{OIP^{[4]}} \supseteq \mathbf{AM}$.

Proof. Suppose $\mathrm{AM}(f) = C$. WLOG, there is a protocol $\mathcal{Q}$ for f with the following shape: Bob tosses coins to generate a random string r and sends it to Merlin, who responds with a message m, where $|r| + |m| \le C$. Bob then sends (r,m) to Alice, who responds with a single bit, after which Bob announces the output.

The interaction between Bob and Alice is an $\mathbf{R^{[2,B]}}$ protocol (in fact, it is deterministic) of cost C. Theorem 4.4 shows that it can be replaced with an $\mathbf{OIP^{[2]}}$ protocol of cost $O(C^2)$. Performing this replacement gives us an $\mathbf{OIP^{[4]}}$ protocol for f. The cost bound can be improved to $O(C\log C)$ by revisiting the analysis of the polynomial evaluation protocol used to prove Theorem 4.4 and using the fact that Alice's message in $\mathcal{Q}$ is just a single bit. □

Theorem 4.11. *For each $k > 0$, there exists a constant $c_k > 0$ such that for all f,* $\mathrm{OIP}_+^{[k]}(f) \ge \Omega\big(\mathrm{AM}(f)^{c_k}\big)$*. In particular, for every constant k, we have* $\mathbf{OIP_+^{[k]}} \subseteq \mathbf{AM}$.

Proof. Let $C = \mathrm{OIP}_+^{[k]}(f)$ and let $\mathcal{P}$ be an $\mathbf{OIP_+^{[k]}}$ protocol with cost C that computes f. By definition, $\mathcal{P}$ uses a hidden random string and Merlin learns about this string only indirectly, from Bob's computed messages. We apply the Goldwasser–Sipser set lower bound technique [22] to convert $\mathcal{P}$ into a protocol where all random coins are directly revealed. Specifically, we can convert $\mathcal{P}$ into an $\mathbf{AMAM}\cdots\mathbf{AM}$ protocol $\mathcal{Q}'$, where $k+3$ messages are sent in total: Merlin's messages are broadcast and after his final message Alice sends a message to Bob, who announces the output. We have $\mathrm{cc}(\mathcal{Q}') = O(C^{a_k})$ for some constant $a_k \ge 1$.

We apply Babai and Moran's round elimination techniques [7] to turn $\mathcal{Q}'$ into a standard **AM** protocol $\mathcal{Q}$ of cost at most $O(\mathrm{cc}(\mathcal{Q}')^{b_k})$ for some constant $b_k \ge 1$. The result follows by taking $c_k = 1/(a_k b_k)$. □

We remark that our transformations in the above proof do not preserve online-ness. The final protocol $\mathcal{Q}$ is no longer online, i.e., we cannot require communications to go to Bob alone.

Corollary 4.12. *For all f, we have $\Omega\big(\mathrm{AM}(f)^{c_4}\big) \le \mathrm{OIP}^{[4]}(f) \le O(\mathrm{AM}(f)\log \mathrm{AM}(f))$, where c_4 is the constant from Theorem 4.11. In particular,* $\mathbf{OIP}^{[4]} = \mathbf{AM}$.

Proof. Combine Theorems 4.10 and 4.11, noting that $\mathbf{OIP}^{[4]} \subseteq \mathbf{OIP}_{+}^{[4]}$. □

Remark. Theorems 4.10 and 4.11 together imply that, up to polynomial factors, the $\mathbf{OIP}_{+}^{[2]}$ communication model is no more powerful than $\mathbf{OIP}^{[4]}$. These theorems also establish that all constant-round **OIP** protocols with four or more messages are equivalent in power, up to polynomial factors.

4.4 Exponential Separations in Our Complexity Zoo

Among the first four levels of the **OIP** hierarchy, we can now show that every pair of adjacent levels is exponentially separated. The next three results make this precise. Recall that $\text{INTER} = \neg\text{DISJ}$ is the set intersection problem.

Theorem 4.13. *We have* $\mathrm{OIP}^{[1]}(\text{INDEX}) = \Omega(n^{1/2})$ *whereas* $\mathrm{OIP}^{[2]}(\text{INDEX}) = O(\log n \log\log n)$.

Proof. We combine Observation 4.1 and Theorem 2.1, and then use the known results that $\mathrm{MA}^{[1,A]}(f) = \Omega\big(\mathrm{R}^{[1,A]}(f)^{1/2}\big)$ for all f [10] (see also Theorem 4.18 in Section 4.5), and that $\mathrm{R}^{[1,A]}(\text{INDEX}) = \Omega(n)$ [3]. □

Theorem 4.14. *We have* $\mathrm{OIP}^{[2]}(\text{INTER}) = \Omega(n^{1/2})$ *whereas* $\mathrm{OIP}^{[3]}(\text{INTER}) = O(\log^2 n)$.

Proof. For the lower bound we use $\mathrm{R}^{[2,B]}(\text{INTER}) \ge \mathrm{R}(\text{INTER}) = \mathrm{R}(\text{DISJ}) = \Omega(n)$ and then apply Theorem 4.2.

For the upper bound, we note that INTER has a nondeterministic protocol with cost $O(\log n)$, wherein Alice and Bob guess an element in the intersection of their respective sets and they verify membership. In particular this gives $\mathrm{MA}^{[2,B]}(\text{INTER}) = O(\log n)$; in fact, Bob need not send anything to Alice in the $\mathbf{MA}^{[2,B]}$ protocol. Now apply Theorem 4.8. □

While we do not know of a *total* Boolean function that separates $\mathbf{OIP}^{[3]}$ from $\mathbf{OIP}^{[4]}$, we do know of a *partial* Boolean function whose $\mathbf{OIP}^{[3]}$ communication complexity is exponentially larger than its $\mathbf{OIP}^{[4]}$ communication complexity. Specifically, Klauck [26, Corollary 3] gives a promise problem he calls PAPPMP which has *Quantum* Merlin-Arthur (**QMA**) communication complexity $\Omega(n^{1/6})$ and **AM** communication complexity $O(\log n)$. Since Theorem 4.6 shows that any $\mathbf{OIP}^{[3]}$ protocol can be transformed into an equivalent $\mathbf{MA}^{[2,B]}$ protocol with a quadratic blowup in cost, and $\mathbf{MA}^{[2,B]}$ protocols are simply restricted versions of QMA protocols, Klauck's lower bound on the QMA cost of PAPPMP implies that $\mathrm{OIP}^{[3]}(\text{PAPPMP}) = \Omega(n^{1/12})$.

Meanwhile, Theorem 4.10 shows that any **AM** communication protocol can be transformed into an equivalent $\mathbf{OIP}^{[4]}$ protocol with a logarithmic blowup in costs. Thus, Klauck's upper bound on the **AM** communication complexity of PAPPMP implies that $\mathrm{OIP}^{[4]}(\text{PAPPMP}) = O(\log n \log\log n)$.

Theorem 4.15. *We have* $\mathrm{OIP}^{[3]}(\text{PAPPMP}) = \Omega(n^{1/12})$ *whereas* $\mathrm{OIP}^{[4]}(\text{PAPPMP}) = O(\log n \log\log n)$.

Next, we show that, up to polynomial factors, $\mathbf{OIP}_{+}^{[2]}$ is at least as powerful as $\mathbf{R}^{[3,A]}$, the class of *three-message* randomized communication protocols in which Alice speaks first. This will enable us to exhibit an explicit function f on domain $\{-1,1\}^n \times \{-1,1\}^n$ such that $\mathrm{OIP}^{[2]}(f) = \Omega(\sqrt{n/\log n})$, while $\mathrm{OIP}_{+}^{[2]}(f) = O(\log^2 n)$.

Theorem 4.16. *For all f, we have* $\mathrm{OIP}_+^{[2]}(f) = O\big(\mathrm{R}^{[3,A]}(f)^2\big)$.

Proof. Let $\mathcal{Q}$ be any three-message randomized communication protocol of cost C, with Alice speaking first. We show how to convert $\mathcal{Q}$ into an $\mathbf{OIP}_+^{[2]}$ protocol $\mathcal{P}$ of cost $O(C^2)$.

We think of $\mathcal{Q}$ as consisting of one message $m_A^{(1)}$ from Alice to Bob, followed by a *two-message* communication protocol $\mathcal{Q}'$ in which Bob speaks first. Theorem 4.4 shows how to transform $\mathcal{Q}'$ into an equivalent $\mathbf{OIP}^{[2]}$ protocol $\mathcal{P}'$ of cost $O(C^2)$ (note this $\mathbf{OIP}^{[2]}$ protocol *depends* on $m_A^{(1)}$).

Thus, we obtain an $\mathbf{OIP}_+^{[2]}$ protocol $\mathcal{P}$ as follows. Alice's message to Bob in $\mathcal{P}$ consists of two parts. The first specifies $m_A^{(1)}$, and the second is the message she would have sent to Bob in $\mathcal{P}'$. Bob, who learns $m_A^{(1)}$ from the first part of Alice's message, now knows what $\mathbf{OIP}^{[2]}$ protocol $\mathcal{P}'$ to execute, and simply behaves the same as he would in $\mathcal{P}'$. □

Exponential separations between $\mathbf{R}^{[3,A]}$ and $\mathbf{R}^{[2,B]}$ are known. In particular, consider the k-step (bipartite) pointer jumping function PJ_k, which interprets each of Alice and Bob's inputs as a list of $N = \Theta(n/\log n)$ *pointers*, a pointer being a $(\log N)$-bit integer. Each pointer in a player's list is interpreted as pointing to (i.e., indexing) a pointer in the other player's list. The goal is to follow these pointers, starting at the first pointer in Alice's list, and output the kth pointer encountered. For example, if Alice's input is $x = (00,01,10,00)$ and Bob's input is $y = (01,10,11,00)$, then $\mathrm{PJ}_1(x,y) = 01$, $\mathrm{PJ}_2(x,y) = 01$, $\mathrm{PJ}_3(x,y) = 10$, and so on. To turn PJ_k into a Boolean function BPJ_k, we take the parity of the $(\log N)$-bit output of PJ_k.

Corollary 4.17. *We have* $\mathrm{OIP}^{[2]}(\mathrm{BPJ}_2) = \Omega(\sqrt{n/\log n})$, *while* $\mathrm{OIP}_+^{[2]}(\mathrm{BPJ}_2) = O(\log^2 n)$.

Proof. Nisan and Wigderson [30] showed that $\mathrm{R}^{[k,B]}(\mathrm{BPJ}_k) = \Omega(N/k^2 - k\log N)$. In particular, any two-message randomized communication protocol in which Bob speaks first has cost $\Omega(N)$. Hence, Theorem 4.2 implies that $\mathrm{OIP}^{[2]}(\mathrm{BPJ}_2) = \Omega(\sqrt{n/\log n})$.

To prove the upper bound on $\mathrm{OIP}_+^{[2]}(\mathrm{BPJ}_2)$, note that there is a trivial three-message protocol for PJ_2 (and hence for BPJ_2) of cost $O(\log n)$ in which Alice speaks first. The upper bound then follows from Theorem 4.16. □

4.5 An Exponential Separation Between $\mathbf{OIP}^{[2]}$ and $\mathbf{OMA}^{[k]}$

In this section, we establish that for any function f, $\mathrm{OMA}^{[2k]}(f) = \Omega\big(\mathrm{R}^{[1,A]}(f)^{1/(k+1)}\big)$. An essentially identical lower bound was proven by Klauck and Prakash for a closely related (though not identical) communication model; we provide details for completeness, and in the process identify the crucial details of the communication model that enable the lower bound to hold.

Theorem 4.18. *For any function f and constant k,* $\mathrm{OMA}^{[2k]}(f) = \Omega\big(\mathrm{R}^{[1,A]}(f)^{1/(k+1)}\big)$.

Proof. We begin by proving the result for the case $k = 1$, showing how to transform any $\mathbf{OMA}^{[2]}$ protocol $\mathcal{P}$ into an $\mathbf{R}^{[1,A]}$ protocol $\mathcal{Q}$ of cost $O(\mathrm{hc}(\mathcal{P})\,\mathrm{vc}(\mathcal{P}))$. This transformation is almost identical to the one of Theorem 4.2, with one crucial change.

Analogously to the proof of Theorem 4.2, $\mathcal{P}$ takes the following shape: (1) hidden coins are tossed, generating random string r according to distribution $\mathcal{D}$; (2) Bob sends Merlin a message $m_B = m_B(r)$; (3) Merlin responds with a message $m_M = m_M(x,y,m_B)$; (4) Alice sends Bob a message $m_A = m_A(x,r)$; (5) Bob outputs a bit given by a function $\mathrm{out}^P(y,m_M,m_A)$.

The key difference between our current setting and the setting of Theorem 4.2 is that in our current setting, m_B is a function only of r, and not of Bob's input y. The proof of Theorem 4.2 (see Figure 3) described a standard protocol $\mathcal{Q}$ in which Bob sends to Alice i.i.d. samples $r^{(1)},\ldots,r^{(h)} \sim (\mathcal{D} \mid m_B = \overline{m})$, where $h = 36(\mathrm{hc}(\mathcal{P})+4)$. In our case, Alice can choose these i.i.d. samples herself, because m_B does not depend on Bob's input y. We therefore obtain an $\mathbf{R}^{[\mathbf{1,A}]}$ protocol $\mathcal{Q}$ of cost $O(\mathrm{hc}(\mathcal{P})\,\mathrm{vc}(\mathcal{P}))$, instead of an $\mathbf{R}^{[\mathbf{2,B}]}$ protocol as in Theorem 4.2.

The general case proceeds by induction on k. We view an $\mathbf{OMA}^{[\mathbf{2k}]}$ protocol $\mathcal{P}$ as an $\mathbf{OMA}^{[\mathbf{2}]}$ protocol $\mathcal{P}_1$ followed by an $\mathbf{OMA}^{[\mathbf{2k-2}]}$ protocol $\mathcal{P}_2$. Inductively, we can replace $\mathcal{P}_2$ with a $\mathbf{R}^{[\mathbf{1,A}]}$ protocol $\mathcal{Q}_2$ of cost $O\left(\mathrm{hc}(\mathcal{P}_2)^{k-1}\,\mathrm{vc}(\mathcal{P}_2)\right) \le O\left(\mathrm{hc}(\mathcal{P})^{k-1}\,\mathrm{vc}(\mathcal{P})\right)$. By concatenating $\mathcal{P}_1$ and $\mathcal{Q}_2$, we obtain an $\mathbf{OMA}^{[\mathbf{2}]}$ protocol $\mathcal{P}_3$ for f with $\mathrm{vc}(\mathcal{P}_3) = O\left(\mathrm{hc}(\mathcal{P})^{k-1}\,\mathrm{vc}(\mathcal{P})\right)$ and $\mathrm{hc}(\mathcal{P}_3) \le \mathrm{hc}(\mathcal{P})$. By our argument in the case $k=1$, we can transform $\mathcal{P}_3$ into an $\mathbf{R}^{[\mathbf{1,A}]}$ protocol $\mathcal{Q}$ of cost $O\left(\mathrm{hc}(\mathcal{P})^{k}\,\mathrm{vc}(\mathcal{P})\right)$.

In particular, if $C = \max\{\mathrm{hc}(\mathcal{P}), \mathrm{vc}(\mathcal{P})\}$, then the cost of $\mathcal{Q}$ is $O(C^{k+1})$. This immediately implies that $\mathrm{OMA}^{[2k]}(f) = \Omega\left(\mathrm{R}^{[1,A]}(f)^{1/(k+1)}\right)$, completing the proof. □

The main property of the $\mathbf{OMA}^{[\mathbf{k}]}$ communication model exploited in our proof of Theorem 4.18 is the following: in any $\mathbf{OMA}^{[\mathbf{k}]}$ protocol $\mathcal{P}$, for all $i \le k$, Alice can determine Bob's ith message to Merlin in $\mathcal{P}$ on her own. In particular, the same lower bound would apply to any variant of online Arthur-Merlin communication models in which Bob's messages to Merlin must be independent of his input y. This is the intuitive reason why the $\mathbf{OIP}^{[\mathbf{2}]}$ model is exponentially more powerful than the $\mathbf{OMA}^{[\mathbf{k}]}$ model for any constant k: in the $\mathbf{OIP}^{[\mathbf{2}]}$ model, Bob's message to Merlin may depend on his input y, while this is not allowed in the $\mathbf{OMA}^{[\mathbf{k}]}$ model.

Combining Theorem 4.18 with the logarithmic upper bound of Theorem 2.1 on the cost of an $\mathbf{OIP}^{[\mathbf{2}]}$ protocol for INDEX, we obtain an exponential separation between $\mathbf{OIP}^{[\mathbf{2}]}$ and $\mathbf{OMA}^{[\mathbf{k}]}$ for any constant $k > 0$.

Corollary 4.19. *For any constant $k > 0$,* $\mathbf{OIP}^{[\mathbf{2}]} \not\subseteq \mathbf{OMA}^{[\mathbf{k}]}$.

Discussion. Recall that in the Turing Machine world, the $\mathbf{AM}_{\mathsf{TM}}$ and $\mathbf{IP}_{\mathsf{TM}}$ hierarchies are equivalent. Corollary 4.19 shows that this equivalence fails badly in the **OIP** world.

It is instructive to observe why standard transformations [22] from private coin to public coin protocols in the Turing Machine world fail to apply in our context. In a sentence, it is because these transformations do not preserve online-ness. This is a subtle point that appears to have been missed in prior work [27].

In more detail, the Goldwasser–Sipser transformation from private coin to public coin protocols [22] has at its core a *set lower bound protocol*, wherein the prover has to convince the verifier that a certain set S is "large". To this end, the verifier sends the prover a hash function h chosen at random from a pairwise independent family, plus a random value z in the codomain of h. The prover tries to respond with an element $w \in S$ such that $h(w) = z$, and separately convinces the verifier that w indeed lies in S. If S is "large", then such a w is likely to exist, whereas if S is "small", then such a w is unlikely (or at least less likely) to exist.

Eliding many further details, Goldwasser and Sipser generically turn an $\mathbf{IP}_{\mathsf{TM}}$ protocol $\mathcal{P}$ for a language $\mathcal{L}$ into an $\mathbf{AM}_{\mathsf{TM}}$ protocol $\mathcal{Q}$ for $\mathcal{L}$ roughly as follows. They apply the set lower bound protocol to the set consisting of all settings of the private random string that would have caused the verifier in $\mathcal{P}$ to accept the input. Since $\mathcal{P}$ is a valid interactive proof for $\mathcal{L}$, this set is "large" if and only if the input is in $\mathcal{L}$.

The natural application of this technique to the standard **AM** communication model transforms a "hidden coin" **AM** communication protocol $\mathcal{P}$ into a public coin one by having Alice and Bob send Merlin a hash function h and a value w in the range of h. Merlin responds by *broadcasting* to Alice and Bob an r such that $h(r) = w$, and Alice and Bob check that they would have accepted in $\mathcal{P}$ given hidden random coins r. It is crucial that Merlin's communication is a broadcast, and as a result, this approach does not apply to our **OIP** communication models in which Merlin is not allowed to talk to Alice. Of course, if Alice does not get to know r, then it leaves Alice and Bob with no way of checking that they would have accepted in the original protocol $\mathcal{P}$ given hidden random string r. This difficulty is fundamental, as evidenced by our exponential separation between $\mathbf{OIP}^{[\mathbf{2}]}$ and $\mathbf{OMA}^{[\mathbf{k}]}$ for any constant $k > 0$.

5 Constant-Round SIPs for NEARESTNEIGHBOR and RANGECOUNT

Thus far, we have applied the polynomial evaluation protocol of Section 3 to the basic problems INDEX and POINTQUERY, and used it to obtain several structural complexity results (Theorems 4.4, 4.8 and 4.10) for our communication complexity classes. In this section, we shall see that the polynomial evaluation protocol is also a powerful algorithmic tool. We shall build upon it to obtain protocols of polylogarithmic cost for much more complicated problems, namely, important geometric problems such as NEARESTNEIGHBOR and RANGECOUNT.

5.1 Nearest Neighbor

Consider a "premetric" space[3] $(\mathcal{X}, D)$ given by a finite ground set $\mathcal{X}$ and distance function $D : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ satisfying $D(\mathbf{x},\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{X}$. Let $B_D(\mathbf{z}, r) = \{\mathbf{x} \in \mathcal{X} : D(\mathbf{x},\mathbf{z}) \le r\}$ denote the corresponding ball of radius $r \in \mathbb{R}^+$ centered at $\mathbf{z} \in \mathcal{X}$. In the NEARESTNEIGHBOR problem, the input consists of a stream $\langle \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \rangle$ of m points from $\mathcal{X}$, constituting the data set, followed by a query point $\mathbf{z} \in \mathcal{X}$. The goal is to output $\mathbf{x}^\star = \arg\min_{\mathbf{x}^{(i)}} D(\mathbf{x}^{(i)}, \mathbf{z})$, the nearest neighbor of $\mathbf{z}$ in the data set. We shall give highly efficient SIPs for this problem that handle rather general distance functions D. To keep our statements of bounds simple, we shall impose the following structure on $(\mathcal{X}, D)$.

- We assume that $\mathcal{X} = [n]^d$. We think of d as the dimensionality of the data, and $[n]^d$ as a very fine "grid" over the ambient space of possible points.
- For all $\mathbf{x}, \mathbf{y} \in [n]^d$, $D(\mathbf{x},\mathbf{y}) \le 1$ is an integer multiple of a small parameter $\varepsilon \ge 1/n^d$.

Overall, this amounts to assuming that our data set has polynomial *spread*: the ratio between the maximum and minimum distance. We proceed to give two SIPs for NEARESTNEIGHBOR. Our basic SIP has cost roughly logarithmic in the stream length and the spread (and therefore linear in d but only logarithmic in n). After we present it, we shall critique it and then give a more sophisticated SIP to handle its faults.

Theorem 5.1. *Under the above assumptions on the premetric space $(\mathcal{X}, D)$, the* NEARESTNEIGHBOR *problem has a three-round SIP with cost $O(d \log n \log(m + \log(d \log n)))$.*

Proof. Let $\mathcal{B} = \{B_D(\mathbf{x}, j\varepsilon) : \mathbf{x} \in \mathcal{X}, j \in \mathbb{Z}, 0 \le j \le 1/\varepsilon\}$ be the set of all balls of all radii between 0 and 1 (quantized at granularity ε). By our assumptions on the structure of $(\mathcal{X}, D)$, we have $|\mathcal{B}| \le n^d/\varepsilon \le n^{2d}$. The input stream $\langle \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \rangle$ defines a *derived stream*, consisting of updates to a vector $\mathbf{v}$ indexed by the elements of $\mathcal{B}$. We shall denote by $v[\boldsymbol{\beta}]$ the entry of $\mathbf{v}$ indexed by $\boldsymbol{\beta} \in \mathcal{B}$. The derived stream is defined as follows: the token $\mathbf{x}^{(i)}$ increments $v[\boldsymbol{\beta}]$ for every ball $\boldsymbol{\beta}$ that contains $\mathbf{x}^{(i)}$. The verifier runs the POINTQUERY protocol of Theorem 2.2 on this derived stream.

The verifier learns the query point $\mathbf{z}$ at the end of the stream. The prover then supplies a point $\mathbf{y}$ claimed to be a valid nearest neighbor (note that there may be more than one valid answer). To check this claim, it is sufficient for the verifier to check two properties: (1) that $\mathbf{y}$ did appear in the stream, and (2) that the stream contained no point closer to $\mathbf{z}$ than $\mathbf{y}$. The first property holds iff $v[B_D(\mathbf{y}, 0)] \neq 0$. The second property holds iff $v[B_D(\mathbf{z}, D(\mathbf{y},\mathbf{z}) - \varepsilon)] = 0$. Clearly, these two properties can be checked by two point queries over the derived stream.

Following the protocol of Theorem 2.2, the two point queries (executed in parallel) involve two more rounds between the verifier and the prover, for an overall three-round SIP. Since the entries of $\mathbf{v}$ never exceed m, each POINTQUERY protocol requires space and help costs $O(d \log n \log(m + \log(d \log n)))$. □

[3] This very general setting, which includes metric spaces as special cases, captures several important distance functions such as the Bregman divergences from information theory and machine learning that satisfy neither symmetry nor the triangle inequality.

While the protocol of Theorem 5.1 achieves very small space and help costs, the prover's and verifier's runtimes could be as high as $\Omega(n^d)$, because processing a single stream token $\mathbf{x}^{(i)}$ may require both parties to enumerate all balls containing $\mathbf{x}^{(i)}$. Ultimately, this inefficiency is because the protocol assumes hardly anything about the nature of the distance function D and, as a result, does not get to exploit any structural information about the balls in $\mathcal{B}$.

To rectify this, we shall make the entirely reasonable assumption that the distance function D is "efficiently computable" in the rather mild sense that membership in a ball generated by D can be decided by a short (say, polynomial-length) formula. Accordingly, we shall express our bounds in terms of a parameter that captures this notion of efficient computation.

Definition 5.2. Suppose the distance function D on $\mathcal{X}$ satisfies the assumptions for Theorem 5.1. Let $\Phi_D : \mathcal{B} \times \mathcal{X} \to \{0,1\}$ be the ball membership function for D, i.e., $\Phi_D(B_D(\mathbf{z},r),\mathbf{x}) = 1 \iff \mathbf{x} \in B_D(\mathbf{z},r)$. Think of Φ_D as a Boolean function of $(3d\log n)$-bit inputs. We define the *formula size complexity* of D, denoted $\operatorname{fsize}(D)$, to be the length of the shortest de Morgan formula for Φ_D.

Since addition and multiplication of b-bit integers can both be computed by Boolean circuits in depth $\log b$ (see, e.g., [31, 39]), they can be computed by Boolean formulae of size $\operatorname{poly}(b)$. It follows that for many natural distance functions D, including the Euclidean, Hamming, ℓ_1, and ℓ_∞ metrics (and in fact ℓ_p for all suitably "small" positive p), we have $\operatorname{fsize}(D) = \operatorname{poly}(d, \log n)$.

Theorem 5.3. *Suppose the premetric space $(\mathcal{X}, D)$ satisfies the assumptions made for Theorem 5.1. Then* NearestNeighbor *on $(\mathcal{X}, D)$ has a three-round SIP, whose space and help costs are both at most* $O(\operatorname{fsize}(D)\log(m + \operatorname{fsize}(D)))$*, in which the prover and the verifier each run in time* $\operatorname{poly}(m, \operatorname{fsize}(D))$*. In particular, if* $\operatorname{fsize}(D) = \operatorname{poly}(d, \log n)$*, as is the case for many natural distance functions D, then the space and help costs are both* $\operatorname{poly}(d, \log m, \log n)$ *and the runtimes are* $\operatorname{poly}(d, m, \log n)$.

Before describing the protocol in detail, let us explain the high level idea that allows us to avoid the high runtimes of the previous protocol. Essentially, the SIP of Theorem 5.1 ran our polynomial evaluation protocol on a *multilinear* extension of the vector $\mathbf{v}$ defined by the derived stream. That SIP took $\mathbf{v}$ to be a completely arbitrary table of values. As a result, the verifier's computation—evaluating the multilinear extension at a random point—became costly. The honest prover incurred similar costs. A closer examination of the nature of $\mathbf{v}$ reveals that if D is a "reasonable" distance function, then $\mathbf{v}$ itself has plenty of structure. In particular, an appropriate *higher degree* extension of $\mathbf{v}$ can in fact be evaluated much more efficiently (by both the verifier and the prover) than the above multilinear extension. Details follow.

Proof. Put $b = d\log n$ and $S = \operatorname{fsize}(D)$. According to Definition 5.2, the function Φ_D is computed by a length-S formula that takes a $2b$-bit input $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{2b})$ describing a ball in $\mathcal{B}$ and a b-bit input $\mathbf{x} = (x_1, \ldots, x_b)$ describing a point in $\mathcal{X}$. With each gate G of this formula we associate a polynomial $\widetilde{G}$ in the variables $W_1, \ldots, W_{2b}, X_1, \ldots, X_b$, as follows:

$$\begin{aligned}
G = \beta_i &\implies \widetilde{G} = W_i\,, \\
G = x_i &\implies \widetilde{G} = X_i\,, \\
G = \neg G_1 &\implies \widetilde{G} = -\widetilde{G}_1\,, \\
G = G_1 \wedge G_2 &\implies \widetilde{G} = \widetilde{G}_1\widetilde{G}_2\,, \\
G = G_1 \vee G_2 &\implies \widetilde{G} = 1 - (1-\widetilde{G}_1)(1-\widetilde{G}_2)\,.
\end{aligned}$$

Let $\widetilde{\Phi}_D(W_1, \ldots, W_{2b}, X_1, \ldots, X_b)$ denote the polynomial thereby associated with the output gate; this polynomial is the standard arithmetization [36] of the formula. We will interpret $\widetilde{\Phi}_D$ as a polynomial in

$\mathbb{F}[W_1,\ldots,X_1,\ldots]$ for a "large enough" finite field $\mathbb{F}$. By construction, $\widetilde{\Phi}_D$ has total degree at most S and agrees with Φ_D on every Boolean input. Define the polynomial $\Psi(W_1,\ldots,W_{2b}) = \sum_{i=1}^m \widetilde{\Phi}_D(W_1,\ldots,W_{2b},\mathbf{x}^{(i)})$.

Observe that the vector $\mathbf{v}$ defined by the derived stream in the proof of Theorem 5.1 behaves as follows:

$$v[\boldsymbol{\beta}] = \sum_{i=1}^{m} \Phi_D(\boldsymbol{\beta},\mathbf{x}^{(i)}) = \sum_{i=1}^{m} \widetilde{\Phi}_D(\boldsymbol{\beta},\mathbf{x}^{(i)}) = \Psi(\boldsymbol{\beta}). \tag{5}$$

Thus, Ψ is a degree-S extension of $\mathbf{v}$ to $\mathbb{F}$. The input stream defines Ψ implicitly, and the verifier can easily evaluate $\Psi(\mathbf{r})$ for random $\mathbf{r} \in_R \mathbb{F}^{2b}$. So we can invoke the polynomial evaluation protocol (twice, in parallel) and answer the NEARESTNEIGHBOR query just as in Theorem 5.1. For full clarity, we spell out the resulting SIP below. The term "canonical representation" and notation λ_ℓ are as in Section 3 and Figure 2.

Input: Stream of points $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}$ from $\mathcal{X}$ defining a data set, followed by query $\mathbf{z} \in \mathcal{X}$.
Goal: Prover to convince Verifier to output a nearest neighbor of $\mathbf{z}$ w.r.t. distance function D.
Shared Agreement: Finite field $\mathbb{F}$ of prime order with $6S+2m \le |\mathbb{F}| \le 12S+4m$, where $S = \text{fsize}(D)$.

Initialization: Verifier picks $\mathbf{r}^{(1)},\mathbf{r}^{(2)} \in_R \mathbb{F}^{2b}$ independently and uniformly, sets $Q_1 \leftarrow 0$ and $Q_2 \leftarrow 0$.

Stream Processing: Upon reading $\mathbf{x} \in \mathcal{X}$, Verifier updates $Q_i \leftarrow Q_i + \widetilde{\Phi}_D(\mathbf{r}^{(i)},\mathbf{x})$ for $i \in \{1,2\}$.

Query Handling: Upon reading query $\mathbf{z}$, Verifier interacts with Prover as follows:

1. Prover sends Verifier a point $\mathbf{y} \in \mathcal{X}$, claiming that it is a nearest neighbor of $\mathbf{z}$ in the data set.
2. Verifier identifies balls $\boldsymbol{\beta}^{(1)} = B_D(\mathbf{y},0)$ and $\boldsymbol{\beta}^{(2)} = B_D(\mathbf{z},D(\mathbf{y},\mathbf{z})-\varepsilon)$. For $i \in \{1,2\}$, if $\boldsymbol{\beta}^{(i)} = \mathbf{r}^{(i)}$, Verifier sets $A_i \leftarrow Q_i$ and skips Steps 3 and 4 for this i; otherwise he sends Prover $\ell^{(i)}$, the unique line in $\mathbb{F}^{2b}$ through $\boldsymbol{\beta}^{(i)}$ and $\mathbf{r}^{(i)}$.
3. For $i \in \{1,2\}$, Prover sends Verifier a polynomial $h_i(V) \in \mathbb{F}[V]$ of degree at most S, claiming that it is the canonical restriction of $\Psi(W_1,\ldots,W_{2b})$ to the line $\ell^{(i)}$. That is, Prover claims that $h_i(V) \equiv \Psi(\lambda_{\ell^{(i)}}(V))$, where $\lambda_\ell(V)$ denotes the canonical representation of the line ℓ in F^{2b}.
4. For $i \in \{1,2\}$, let $v_i,t_i \in \mathbb{F}$ be such that $\lambda_{\ell^{(i)}}(v_i) = \boldsymbol{\beta}^{(i)}$ and $\lambda_{\ell^{(i)}}(t_i) = \mathbf{r}^{(i)}$. Verifier checks that $h_i(t_i) = Q_i$, aborting if not. Otherwise, he sets $A_i \leftarrow h_i(v_i)$.
5. If $A_1 \neq 0$ and $A_2 = 0$, then Verifier outputs $\mathbf{y}$ as the answer. Otherwise he aborts.

Figure 4: A Three-Round SIP for the NEARESTNEIGHBOR Problem

This protocol's correctness can be analyzed using the same ideas as in the proofs of Theorems 2.1 and 5.1. It has perfect completeness. If a dishonest prover supplies an incorrect polynomial for either $h_1(V)$ or $h_2(V)$, the verifier will fail to notice with probability at most $S/(|\mathbb{F}|-1) \le 1/6$, leading to a soundness error of at most $1/6+1/6 = 1/3$. Entries of $\mathbf{v}$ always lie between 0 and m and char$(\mathbb{F}) = |\mathbb{F}| > m$, since $\mathbb{F}$ is of prime order, therefore there are no "wrap around" problems in Eq. (5).

Turning to the protocol's costs, the verifier needs $O(S)$ space to evaluate $\widetilde{\Phi}_D$ and $O(b\log|\mathbb{F}|)$ space to maintain $\mathbf{r}^{(1)},\mathbf{r}^{(2)},Q_1$, and Q_2. The prover needs to communicate two degree-S polynomials, which costs $O(S\log|\mathbb{F}|)$. Under the reasonable assumption that the optimal formula for Φ_D depends on all its input variables, we have $S \ge 3b$, which yields a bound of $O(S\log(m+S))$ on the space and help costs. The claim about the runtimes is straightforward from the protocol's description. □

We remark that our above theorem made the tacit *uniformity* assumption that a formula for Φ_D of length fsize(D) could be constructed in space $O(\text{fsize}(D))$ and time $O(\text{poly}(\text{fsize}(D)))$.

5.2 RANGECOUNT Queries

Let $\mathcal{U}$ be any data universe and $\mathcal{R} \subseteq 2^{\mathcal{U}}$ a set of *ranges*. In the RANGECOUNT problem, the data stream $\sigma = \langle \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}, R^* \rangle$ specifies a sequence of universe elements $\mathbf{x}^{(i)} \in \mathcal{U}$, followed by a *query* or *target* range $R^* \in \mathcal{R}$. The goal is to output $|\{i : \mathbf{x}^{(i)} \in R^*\}|$, i.e., the number of elements in the target range that appeared in the stream.

We easily obtain a *two-message* streaming interactive proof for the RANGECOUNT problem in which both the help and space cost are bounded by $O(\log|\mathcal{R}|\log(|\mathcal{R}|m))$. The verifier simply runs a POINTQUERY on the derived stream σ' defined to have data universe $\mathcal{R}$. σ' is obtained from σ as follows: on each stream update $\mathbf{x}^{(i)} \in \mathcal{U}$, the verifier inserts into σ' one copy of each range $R \in \mathcal{R}$ such that $\mathbf{x}^{(i)} \in R$. The range count problem is equivalent to a POINTQUERY on σ', with the target item being R^*, and we obtain the following theorem.

Theorem 5.4. *There is a two-message SIP with $O(\log|\mathcal{R}|\log(|\mathcal{R}| \cdot m))$ cost for* RANGECOUNT.

In particular, for spaces of bounded *shatter dimension* ρ, $\log|\mathcal{R}| = \rho \log m = O(\log m)$. The above protocol also implies a *three-message* SIP for the problem of *linear classification*, a core problem in machine learning. Just like the protocol for NEARESTNEIGHBOR invokes a two-message protocol for INDEX, a SIP for linear classification (find a hyperplane that separates red and blue points) verifies that the proposed hyperplane is empty of red points on one side and blue points on the other using the above two-message RANGECOUNT protocol.

The prover and verifier in the protocol of Theorem 5.4 may require time $\Omega(|\mathcal{R}|)$ per stream update. This could be prohibitively large. However, we can obtain savings analogous to Theorem 5.3 if we make a mild "efficient computability" assumption on our ranges. Specifically, suppose there exists a (poly(S)-time uniform) de Morgan formula Φ of length S that takes as input a binary string representing a point $\mathbf{x}^{(i)} \in \mathcal{U}$, as well as the label of a range $R \in \mathcal{R}$ and outputs a bit that is 1 if and only if $\mathbf{x}^{(i)} \in R$. We then obtain the following more practical SIP.

Theorem 5.5. *Suppose membership in ranges from $\mathcal{R}$ can be decided by de Morgan formulas of length S as above. Then there is a two-round SIP for* RANGECOUNT *on $\mathcal{R}$, with space and help costs at most* poly(S)*, in which both the prover and the verifier run in time* poly(m, S).

6 A Non-Interactive Protocol for Counting Triangles

Consider a data stream σ consisting of a sequence of undirected edges $\langle e_1, \ldots, e_m \rangle$, where each edge $e_i \in [n] \times [n]$. σ defines an undirected graph G in the natural manner. In the TRIANGLES problem, the goal is to determine the number of unordered triples of distinct vertices (u, v, z) such that edges (u, v), (v, z), and (z, u) all appear in G. Computing the number of triangles is a well-studied problem [4] and there has been considerable interest in designing algorithms in a variety of models including the data stream model [8, 32], MapReduce [35], and the quantum query model [28]. One motivation is the study of social networks where important statistics such as the clustering coefficient and transitivity coefficient are based on the number of triangles. Understanding the complexity of counting triangles captures the ability of a model to perform a non-trivial correlation within large graphs. Chakrabarti et al. [10] gave two annotated data streaming protocols for this problem. The first protocol had help cost $O(n^2 \log n)$, and space cost $O(\log n)$. The second protocol achieved help cost $O(x \log n)$ and space cost $O(y \log n)$ for any x, y such that $x \cdot y \geq n^3$. In particular, by setting $x = y = n^{3/2}$, the second protocol of Chakrabarti et al. ensured that both help cost and space cost equaled $O\left(n^{3/2} \log n\right)$. Cormode [13] asked whether it is possible to achieve an annotated data streaming protocol in which both the help cost and space cost are $\tilde{O}(n)$. We answer this question in the affirmative.

Theorem 6.1 (Restatement of Theorem 2.3)**.** *There is an annotated data stream algorithm for* TRIANGLES *with space and help costs $O(n \log n)$. Every such algorithm requires the product of the space and help costs to be $\Omega(n^2)$.*

Proof. The lower bound was proved in [10, Theorem 7.1]. Details of the upper bound follow.

Let G_i denote the graph defined by the first i stream updates $\langle e_1, \dots, e_i \rangle$, and let $E_i : [n] \times [n] \to \{0,1\}$ denote the function that outputs 1 on input (u,v) if and only if $e_j = (u,v)$ for some $j < i$. On edge update $e_i = (u_i, v_i)$, notice that the number of triangles that e_i *completes* in E_i is precisely $\sum_{z \in [n]} E_i(u_i,z)E_i(v_i,z)$. Thus, at the end of the stream, the total number of triangles in the graph $G = G_m$ is precisely

$$\sum_{i \le m} \sum_{z \in [n]} E_i(u_i,z)E_i(v_i,z). \tag{6}$$

We will use sum-check techniques to compute this quantity. To this end, let $\mathbb{F}$ denote a field of prime order $6n^3 \le |\mathbb{F}| \le 12n^3$, and let $\widetilde{E}_i(X,Y)$ denote the unique polynomial over $\mathbb{F}$ of degree at most n in each variable X, Y such that $\widetilde{E}_i(u,v) = E_i(u,v)$ for all $(u,v) \in [n] \times [n]$.

Then Quantity (6) equals

$$\sum_{i \le m} \sum_{z \in [n]} E_i(u_i,z)E_i(v_i,z) = \sum_{i \le m} \sum_{z \in [n]} \widetilde{E}_i(u_i,z)\widetilde{E}_i(v_i,z) = \sum_{z \in [n]} \sum_{i \le m} \widetilde{E}_i(u_i,z)\widetilde{E}_i(v_i,z). \tag{7}$$

In turn, the right hand side of Equation (7) can be written as $\sum_{z \in [n]} g(z)$, where g denotes the *univariate* polynomial defined via:

$$g(Z) = \sum_{i \le m} \widetilde{E}_i(u_i,Z)\widetilde{E}_i(v_i,Z). \tag{8}$$

Notice $g(Z)$ is a univariate polynomial of degree at most $2n$. Our annotated data streaming protocol proceeds as follows.

Prover's computation. At the end of the stream, the prover is required to send a univariate polynomial $s(Z)$ of degree at most $2n$, where $s(Z)$ is claimed to equal $g(Z)$. Notice that since $s(Z)$ has degree at most $2n$, $s(Z)$ can be specified by sending its values on all inputs in $\{0, \dots, 2n\}$ – this requires help cost $O(n \log |\mathbb{F}|) = O(n \log n)$.

Verifier's computation. At the start of the stream, the verifier picks a random field element $r \in \mathbb{F}$, and keeps the value of r secret from the prover. We will show below that the verifier can evaluate $g(r)$ with a single streaming pass over the input, using space $O(n \log n)$. The verifier checks whether $s(r) = g(r)$. If this check fails, the verifier halts and rejects. If the check passes, the verifier outputs $\sum_{z \in [n]} s(z)$ as the correct answer.

We now explain how the verifier can evaluate $g(r)$ with a single streaming pass over the input. The high-level idea is as follows. Equation (8) expresses $g(r)$ as a sum of m terms, where the ith term equals $\widetilde{E}_i(u_i,r)\widetilde{E}_i(v_i,r)$. For each $u \in [n]$, we will show how the verifier can incrementally maintain the quantity $\widetilde{E}_i(u,r)$ at all times i. The verifier will maintain all n of these quantities, resulting in a total space cost of $O(n \log |\mathbb{F}|) = O(n \log n)$. With these quantities in hand, it is straightforward for the verifier to incrementally maintain the sum $\sum_{j \le i} \widetilde{E}_i(u_i,r)\widetilde{E}_i(v_i,r)$ at all times i: upon the ith stream update, the verifier simply adds $\widetilde{E}_i(u_i,r) \cdot \widetilde{E}_i(v_i,r)$ to the running sum.

To maintain the quantity $\widetilde{E}_i(u,r)$, we begin by writing the bivariate polynomial $\widetilde{E}_i(X,Y)$ in a convenient form. Given a pair $(u,v) \in [n] \times [n]$, let $\widetilde{\delta}_{(u,v)}$ denote the following (Lagrange) polynomial:

$$\widetilde{\delta}_{(u,v)}(X,Y) = \left(\frac{\prod_{1 \le u' \le n : u' \ne u} (X - u')}{\prod_{1 \le u' \le n : u' \ne u} (u - u')} \right) \left(\frac{\prod_{1 \le v' \le n : v' \ne v} (Y - v')}{\prod_{1 \le v' \le n : v' \ne v} (v - v')} \right). \tag{9}$$

Notice that $\widetilde{\delta}_{(u,v)}$ evaluates to 1 on input (u,v), and evaluates to 0 on all other inputs $(x,y) \in [n] \times [n]$. Thus, we may write $\widetilde{E}_i(X,Y) = \sum_{j \leq i} \widetilde{\delta}_{(u_j,v_j)}(X,Y)$. In particular, for each node $u \in [n]$, $\widetilde{E}_i(u,r) = \widetilde{E}_{i-1}(u,r) + \widetilde{\delta}_{(u_i,v_i)}(u,r) + \widetilde{\delta}_{(v_i,u_i)}(u,r)$. Thus, the verifier can incrementally maintain the quantity $\widetilde{E}_i(u,r)$ in a streaming manner using space $O(\log|\mathbb{F}|)$: while processing the ith stream update, the verifier simply adds $\widetilde{\delta}_{(u_i,v_i)}(u,r) + \widetilde{\delta}_{(v_i,u_i)}(u,r)$ to the running sum tracking $\widetilde{E}_i(u,r)$.

Completeness. It is evident that if the prover sends the true polynomial $g(Z)$, then the verifier's check will pass.

Soundness. If the prover sends a polynomial $s(Z) \neq g(Z)$, then with probability at least $1 - 2n/|\mathbb{F}| \geq 1 - 1/3n$ over the verifier's random choice of $r \in \mathbb{F}$, it will hold that $s(r) \neq g(r)$. Hence, with probability at least $1 - 1/3n \geq 2/3$, the verifier's check will fail and the verifier will reject. □

Several remarks regarding Theorem 6.1 are in order.

- **Verifier Time.** The verifier in the protocol of Theorem 6.1 can process each stream update in constant time as follows. On stream update $e_i = (u_i, v_i)$, the verifier must add $\widetilde{\delta}_{(u_i,v_i)}(u,r) + \widetilde{\delta}_{(v_i,u_i)}(u,r)$ to each of the $E_i(u,r)$ values. However, using Equation (9), it is straightforward to check that $\widetilde{\delta}_{(u_i,v_i)}(u,r) = 0$ for all $u \neq u_i$, so the verifier need only update two quantities a time i: $E_i(u_i,r)$ and $E_i(v_i,r)$. We explain how both of these updates can be computed in constant time.

  It can be seen from Equation (9) that

$$\widetilde{\delta}_{(u_i,v_i)}(u_i,r) = \frac{\prod_{1 \leq v' \leq n : v' \neq v_i}(r - v')}{\prod_{1 \leq v' \leq n : v' \neq v_i}(v_i - v')} \tag{10}$$

  The right hand side of Equation (10) can be computed in $O(1)$ time if the verifier maintains a pre-computed lookup table consisting of $O(n)$ field elements. Specifically, for each $v \in [n]$, the verifier needs to maintain the quantities $\prod_{1 \leq v' \leq n : v' \neq v}(r - v')$ and $\left(\prod_{1 \leq v' \leq n : v' \neq v_i}(v_i - v')\right)^{-1}$. All $O(n)$ of these quantities can be computed in pre-processing in total time $O(n \log n)$, where the $\log n$ term is due to the time required to compute a multiplicative inverse in the field $\mathbb{F}$.

  Finally, the verifier can process the proof itself in time $O(n)$. Indeed, recall that the proof consists of the values $s(x)$ for $x \in \{0, \ldots, 2n\}$, and the verifier simply needs to compute $\sum_{1 \leq x \leq n} s(x)$ as well as $s(r)$. The first quantity can trivially be computed in time $O(n)$, and the second can be computed in time $O(n)$ as well using standard techniques (see, e.g., [15]).

- **Prover Time.** The honest prover in the protocol of Theorem 6.1 can be implemented to run in time $O(m \cdot n)$. Indeed, the honest prover needs to evaluate $g(x)$ for $O(n)$ points $x \in \mathbb{F}$, and we have explained above how $g(x)$ can be computed in $O(m)$ time (in fact, in $O(1)$ time per stream update). Note that this time is comparable to the cost of a naive triangle counting algorithm that, for each edge and node combination, tests whether the two edges incident on the edge and node exist in the graph.

- **More general streaming models.** The protocol of Theorem 6.1 can be trivially modified to handle streams with deletions, as well as weighted and directed graphs. In the case of weighted graphs, the weight assigned to a triangle (u,v,z) is the *product* of weights of the constituent edges (u,v), (v,z), and (z,u).

- **Extensions.** Let H be a graph on k vertices. It is possible to extend the protocol underlying Theorem 6.1 to count the number of occurrences of H as a subgraph of G. The protocol requires $k-2$ rounds, and its help and space costs are $O(kn \log n)$ and $O(n \log n)$ respectively. We omit the details for brevity.

- **MA communication.** Theorem 6.1 implies that the (online) MA communication complexity of counting triangles is $O(n \log n)$. This essentially matches an $\Omega(n)$ lower bound on the (even non-online) MA communication complexity of the problem, proved by Chakrabarti et al. [10] via a standard reduction to SET-DISJOINTNESS, and answers a question of Cormode [13].

6.1 Comparison to Prior Work

As is typical in the literature on interactive proofs, the verifier in our TRIANGLES protocol evaluates $g(r)$ for a random point $r \in \mathbb{F}$, where g is a polynomial derived from the input stream σ. This quantity serves as a "secret" that the verifier can use to catch a dishonest prover. However, our protocol qualitatively departs from prior work in that for our definition of g, $g(r)$ is not a *linear* sketch of the input. For purposes of this discussion, we define a linear sketch as any summary of the form $\mathbf{v} \in \mathbb{F}^w$ for some $w > 0$ which can be computed as $\mathbf{v} = S\mathbf{f}(\sigma)$. Here, $S \in \mathbb{F}^{w \times n}$ is a "sketch matrix" and $\mathbf{f}(\sigma)$ denotes the *frequency-vector* of the stream, i.e., the ith element of $\mathbf{f}(\sigma)$ is the number of times item i appears in σ. The vast majority of protocols in the interactive proofs literature only require the verifier to compute a linear sketch of the input [2, 10, 15, 16, 19, 20, 23, 36]. Typically, this linear sketch consists of one or more evaluations of a *low-degree extension* of the frequency vector $\mathbf{f}$ itself.

In contrast, in our TRIANGLES protocol, we view the quantity $\sum_{j \le i} \widetilde{E}_j(u_j, r) \cdot \widetilde{E}_j(v_j, r)$ as the verifier's sketch at time i. While $\widetilde{E}_j(u, r)$ is a linear sketch of the input for each j and node $u \in [n]$ (in fact, this is what enables the verifier to compute $\widetilde{E}_j(u, r)$ for each $u \in [n]$ in a streaming manner), $\sum_{j \le i} \widetilde{E}_j(u_j, r) \cdot \widetilde{E}_j(v_j, r)$ is not. A consequence is that, in our TRIANGLES protocol, the verifier's update to her state at time j depends on her state at time j (specifically, her update depends on the stored values $\widetilde{E}_j(u_j, r)$ and $\widetilde{E}_j(v_j, r)$). This contrasts with linear sketches, as in a linear sketch, each stream update (i, δ) contributes the value Se_i to the sketch independently of all other stream updates, where e_i is the vector consisting of a 1 in the ith coordinate and zeros elsewhere.

Refereed vs. Unrefereed Communication. Another way to formalize the way in which our counting triangles protocol differs from prior work is the following. Given a data stream σ and an annotated data streaming protocol $\mathcal{P}$, fix the verifier's random coins r within $\mathcal{P}$, and let $\mathbf{v}(\sigma)$ denote the summary of the stream that the verifier must compute in order to execute the protocol $\mathcal{P}$ with random coins r. Consider a three-party *refereed communication* model in which the first player (Alice) holds the first half of a data stream, the second player (Bob) holds the second half, and the third player (Reginald) is a referee. The goal is to compute $\mathbf{v}(x \circ y)$, where $x \circ y$ denotes the "concatenated" data stream. Alice and Bob each send a message to Reginald, who must output $\mathbf{v}$. The cost of the protocol is the sum of Alice and Bob's messages.

To our knowledge, in all existing annotated data streaming protocols $\mathcal{P}$, the verifier's summary $\mathbf{v}(x \circ y)$ can be computed by a refereed communication protocol of cost proportional to the space cost of $\mathcal{P}$. This is because $\mathbf{v}$ is a linear sketch in existing protocols: letting S be the sketch matrix, Alice can send the referee $S\mathbf{f}(x)$, Bob can send the referee $S\mathbf{f}(y)$, and the referee can compute $S\mathbf{f}(x \circ y) = S(\mathbf{f}(x) + \mathbf{f}(y)) = S\mathbf{f}(x) + S\mathbf{f}(y)$.

In contrast, the verifier's summary in our counting triangles protocol cannot obviously be computed by a refereed communication protocol of cost proportional to $O(n \log n)$, the space cost of our protocol. The key point is that, in our protocol, the verifier's updates when processing the second half of the data depend on the first half of the data stream. Hence, in the refereed communication setting, Bob is unable to simulate the verifier's computation on his input y, as he needs information about x to do so.

7 Conclusion

In this paper, we advanced the study of constant-round interactive protocols for verifying outsourced streaming computations. We introduced new communication models that closely capture constant-round streaming interactive proofs, and we showed that constant-round SIPs behave very differently from classical interactive proofs in several fundamental ways. In particular, we demonstrated that in the streaming setting, "secret" coins are exponentially more powerful than public coins, and generic round elimination is impossible. Our main algorithmic contributions were to give constant-round streaming interactive proofs that are exponentially more efficient than prior work for a large class of problems, as well as giving an essentially optimal annotated data streaming protocol for counting triangles.

Many questions remain for future work, but here we highlight just one: proving superlogarithmic lower bounds on $\mathbf{OIP}_{+}^{[2]}$ communication complexity. Klauck [26] has identified the problem of proving superlogarithmic lower bounds on **AM** communication complexity as an important "first step" toward resolving the $\mathbf{\Pi_2} \neq \mathbf{\Sigma_2}$ problem in two-party communication complexity, one of the most important problems left open by Babai et al. [6]. As we have shown, standard techniques easily establish that $\mathbf{OIP}_{+}^{[2]}$ is a subset of **AM**, but have been unable to prove any superlogarithmic lower bounds against $\mathbf{OIP}_{+}^{[2]}$ protocols. Proving $\mathbf{OIP}_{+}^{[2]}$ lower bounds therefore represents an important (and potentially tractable) "zeroth step" toward resolving $\mathbf{\Pi_2} \neq \mathbf{\Sigma_2}$.

References

[1] S. Aaronson. QMA/qpoly $\subseteq$ PSPACE/poly: De-Merlinizing quantum protocols. In *CCC*, pages 261–273, 2006.

[2] S. Aaronson and A. Wigderson. Algebrization: a new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1:1, pages 1–54, 2009. Preliminary version appeared in *STOC* 2008.

[3] F. Ablayev. Lower Bounds for One-way Probabilistic Communication Complexity and Their Application to Space Complexity. *Theoretical Computer Science*, 175:2, pages 139–159, 1996.

[4] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[5] L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

[6] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity. In *FOCS*, pages 337–347, 1986.

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

[8] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[9] Z. Bar-Yossef, T. S. Jayram, R.Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[10] A. Chakrabarti, G. Cormode, A. McGregor, and J. Thaler. Annotations in data streams. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:22, 2012. A preliminary version of this paper by A. Chakrabarti, G. Cormode, and A. McGregor appeared in *ICALP* 2009.

[11] A. Chakrabarti, G. Cormode, N. Goyal, and J. Thaler. Annotations for sparse data streams. In *SODA*, 2013.

[12] A. Condon. The complexity of space bounded interactive proof systems. In *Complexity Theory: Current Research*, S. Homer, U. Schöning and K. Ambos-Spies (Eds.), Cambridge University Press, pages 147–190, 1993.

[13] List of Open Problems in Sublinear Algorithms: Problem 47. `http://sublinear.info/47`

[14] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65:2, pages 409–442, 2013. A preliminary version of this paper appeared in *ESA*, 2010.

[15] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.

[16] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011.

[17] A. Das Sarma, R.J. Lipton, and D. Nanongkai. Best-order streaming model. *Theor. Comput. Sci.*, 412:23, pages 2544–2555 2011.

[18] R. J. Lipton. Efficient Checking of Computations. *STACS*, pages 207–215, 1990.

[19] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[20] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput*, 18(1):186–208, 1989. Preliminary version in *STOC* 1985.

[22] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Randomness and Computation*. JAI Press, 1987. Extended Abstract in *STOC*, 1986.

[23] T. Gur and R. Raz. Arthur-Merlin streaming complexity. Electronic Colloquium on Computational Complexity (ECCC). Available online at `http://eccc.hpi-web.de/report/2013/020/`, 2013.

[24] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[25] H. Klauck. Rectangle size bounds and threshold covers in communication complexity. In *CCC*, pages 118–134, 2003.

[26] H. Klauck. On Arthur Merlin games in communication complexity. In *CCC*, pages 189–199, 2011.

[27] H. Klauck, and V. Prakash. Streaming computations with a loquacious prover. In ITCS, pages 305–320, 2013.

[28] T. Lee, F. Magniez, and M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *SODA*, pages 1486–1502, 2013.

[29] M. Mitzenmacher and E. Upfal. Probability and computing - randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

[30] N. Nisan and A. Widgerson. Rounds in communication complexity revisited. In *STOC*, pages 419-42, 1991.

[31] Y. Ofman. On the algorithmic complexity of discrete functions. Doklady Akademii Nauk SSSR 145 (1): 48-51, 1962. English Translation in Soviet Physics Doklady 7: 589-591, 1963.

[32] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. In *VLDB*,Counting and sampling triangles from a graph stream. 2013.

[33] R. Raz. Quantum information and the PCP theorem. *Algorithmica* 55(3): 462–489, 2009. Preliminary version in *FOCS*, 2005.

[34] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[35] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

[36] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[37] J. Thaler. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO (2)*, pages 71–89, 2013.

[38] V. Vu, S. T. V. Setty, A. J. Blumberg, M. Walfish. A Hybrid Architecture for Interactive Verifiable Computation. In *IEEE Symposium on Security and Privacy*, pages 223–237, 2013.

[39] C. S. Wallace. A Suggestion for a Fast Multiplier. *IEEE Transaction on Electronic Computers*, vol.EC-13 (1):14-17, 1964.