Efficient Arguments and Proofs for Batch Arithmetic Circuit Satisfiability

Jieyi Long

Theta Labs, Inc.
jieyi@thetalabs.org

Abstract. In this paper, we provide a systematic treatment for the batch arithmetic circuit satisfiability and evaluation problem. Building on the core idea which treats circuit inputs/outputs as a low-degree polynomials, we explore various interactive argument and proof schemes that can produce succinct proofs with short verification time. In particular, for the batch satisfiability problem, we provide a construction of succinct interactive argument of knowledge for generic log-space uniform circuits based on the bilinear pairing and common reference string assumption. Our argument has size in $O(poly(\lambda) \cdot (|\mathbf{w}| + d \log |C|))$, where λ is the security parameter, $|\mathbf{w}|$ is the size of the witness, and d and $|C|$ are the depth and size of the circuit, respectively. Note that the argument size is independent of the batch size. To the best of our knowledge, asymptotically it is the smallest among all known batch argument schemes that allow public verification. The batch satisfiablity problem simplifies to a batch evaluation problem when the circuit only takes in public inputs (i.e., no witness). For the evaluation problem, we construct statistically sound interactive proofs for various special yet highly important types of circuits, including linear circuits, and circuits representing sum of polynomials. Our proposed protocols are able to achieve proof sizes independent of the batch size. We also describe protocols optimized specifically for batch FFT and batch matrix multiplication which achieve desirable properties, including lower prover time and better composability. We believe these protocols are of interest in their own right and can be used as primitives in more complex applications.

Keywords: arithmetic circuit satisfiability, batch arguments, batch proofs

1 Introduction

As cutting-edge technologies such as blockchain, cloud computing, and deep learning continue to advance, verifiable computation is becoming increasingly vital in today's technology landscape. Verifiable computation enables a weak *verifier* to delegate complex computing tasks to a powerful, yet untrusted *prover*. Along with the computation results, the prover also provides a proof for the verifier to validate that the computation was performed correctly.

In many verifiable computation scenarios, the prover is required to generate a single proof for multiple computation tasks that use the same circuit, but

with different public input data, as well as witnesses that are not known to the verifier. This is commonly referred to as the *batch circuit satisfiability* problem. A specific case of this scenario, where the witness is absent, is referred to as the *batch circuit evaluation* problem.

Batch proof has many potential applications. Many outsourced computations are data parallel, meaning the same computation are applied independently to many pieces of data [61]. Amazon Elastic MapReduce is a prominent example. Another useful application of batch proof is deep learning model inference services. These services often want to assure their users that the inferences made were done using the appropriate machine learning models [49]. To accomplish this, the inference service provider can periodically post a proof which guarantees that all inferences made in a specific period were conducted correctly. The intersection of verifiable computation and blockchain systems is driving increasing demand for batch proof. For example, blockchain scaling solutions such as zkRollup [18] has recently been garnering significant attention both from academia and industry. These solutions could benefit tremendously from software systems that can produce a batch proof for all the transactions inside a block. A highly desirable property of batch proofs is that the length of the proof does not increase with the size of the batch. This is particularly crucial for storage-sensitive verification environments, such as on-chain verification using smart contracts, as it allows for more efficient use of the costly storage space.

1.1 Summary of Contributions

In this paper, we provide a systematic treatment for both the batch circuit satisfiability and evaluation problem. Building on the core idea which treats the circuit inputs and outputs as low-degree polynomials interpolating the inputs and outputs across the instances, we explore various efficient argument and proof schemes that can produce succinct proofs. We list our main contributions below:

- **Succinct argument of knowledge for batch circuit satisfiability**. For the batch satisfiability problem, we provide a construction of computationally sound succinct interactive argument of knowledge for generic log-space uniform circuits under the bilinear pairing and common reference string assumption. To the best of our knowledge, among all the known schemes that allow public verification, asymptotically our work achieves the smallest argument length in the literature, which is $O(poly(\lambda) \cdot (|\mathbf{w}| + d \log |C|))$, independent of the batch size. Here λ is the security parameter, d is the depth of the circuit, and $|C|$ and $|\mathbf{w}|$ are the size of the circuit and witness, respectively. If all the instances in the batch share the same witness (but have different public inputs), the argument size can be further reduced to $O(poly(\lambda) \cdot (d \log |C| + \log |\mathbf{w}|))$. Using the Fiat-Shamir transformation, the interactive argument can be made non-interactive under the random oracle model.
- **Succinct proofs for batch circuit evaluation**. The batch satisfiablity problem degenerates to a batch evaluation problem when the circuit only takes in public inputs (i.e., no witness). For the evaluation problem, we

construct statistically sound interactive proofs for various special yet highly import types of circuits, including linear circuits, and circuits representing sum of polynomials. Our protocols are able to achieve a proof size independent of the batch size. As far as we are aware of, this is the first batch proof proposal in the literature that achieves this for these special types of circuits.

- **New primitives**. Our third contribution is to describe optimized interactive protocols for batch FFT and batch matrix multiplication which have certain desirable properties, including lower prover time, better composability, etc. We believe these protocols are of interest in their own right and can be used as primitives in more complex applications.

In Table 1, we compare various aspects of our batch argument protocol with the prior art. Please refer to Section 1.2 for more details.

Table 1. A comparison of different batch argument schemes, where m is the batch size, λ is the security parameter, d is the depth of the circuit, and $|C|$ and $|\mathbf{w}|$ are the size of the circuit and witness, respectively.

	Argument Size	**Verifier Time**	**Computation Model**	**Assumptions**
Bootle et al. [13]	$O(\sqrt{m})$	$O(m\|C\|)$	Low-degree circuit	Discrete log
Brakerski et al. [15]	$O(poly(\lambda) \cdot \|\mathbf{w}\|)$	$O(poly(\lambda) \cdot (m\|\mathbf{io}\| + \|\mathbf{w}\|))$	3SAT	Private verification
Choudhuri et al. [23]	$\tilde{O}(\lambda(\|C\| + \sqrt{m\|C\|}))$	$O(poly(\lambda) \cdot (m\|\mathbf{io}\| + \|C\|))$	C-SAT	SE-LHC, CIH, CRS
Devadas et al. [26]	$O(\|\mathbf{w}\| + poly(\lambda, \log m))$	$O(poly(\lambda, \|\mathbf{io}\|, \|\mathbf{w}\|, m))$	Boolean circuit	LWE
Garg et al. [33]	$O(poly(\lambda, \log m, \log \|C\|))$	$O(poly(\lambda, m, \|\mathbf{io}\|) + poly(\lambda, \log m, log\|C\|))$	Boolean circuit	IO, OWF, CRS
Waters et al. [65]	$O(poly(\lambda, \|C\|))$	$O(poly(\lambda, m, \|\mathbf{io}\|) + poly(\lambda, \|C\|))$	Boolean circuit	Bilinear group, CRS
Our Work	$O(poly(\lambda) \cdot (\|\mathbf{w}\| + d \log \|C\|))$	$O(poly(\lambda) \cdot (m\|\mathbf{io}\| + \|\mathbf{w}\| + d \log \|C\|))$	Uniform circuit	Pairing, RO, CRS

1.2 Related Work

Argument and proof systems have a long and rich history in cryptography research [35, 45, 52]. In recent years, driven by real world use cases like smart contract and blockchain applications [18], the so-called succinct non-interactive arguments of knowledge (SNARK) systems received particular attention. Tremendous progress have been made over the years on various aspects of the SNARK systems [5, 6, 8, 14, 22, 30, 36, 37, 44, 51, 55, 59, 64, 67, 70]. With such an arsenal of SNARK protocols, batch arguments can be trivially constructed by applying any of them to individual instances in the batch and then concatenate the proofs together. However, with this method, the overall proof size would grow linearly with the batch size.

As an alternative approach, one may consider treating the entire batch as a giant "super circuit" and apply the known proof or argument constructions to it. Along this line, Thaler [61] proposed an interactive protocol for proving the results of data parallel computation, in which the same computation are applied independently to many pieces of data. The technique can be adapted for

proving batch circuit evaluation. The communication complexity of the protocol is $O(d \cdot \log(m|C|))$ field elements, where d and $|C|$ are the depth and size of the circuit, and m is the batch size. While this approach is highly promising, the proof size still increases with batch size. Williams [66] explored another approach to fold multiple instances into a single instance, and came up with an non-interactive proof protocol for batch evaluation. However, its communication cost is $O(m|\mathbf{io}|D)$, which scales linearly with the batch size. Moreover, both Thaler's and Williams techniques only apply to circuit evaluations, not circuit satisfiability problems.

A closely related field of research is SNARK aggregation. Different from batch proof, where the proof is generated by processing the instances directly, SNARK aggregation works by first generating the SNARK proof for each individual instances, and then aggregate the proofs together. SnarkPack [31] is a recent proposal for aggregating Groth16 zkSNARKs [36], which is able to aggregate m Groth16 zkSNARKs into a single proof with $O(\log m)$ length and verifier time. The size of the aggregated proof is sublinear in the batch size, but still grows as the batch size becomes larger.

Incremental verifiable computation (IVC) [46,53,62] and proof-carrying data (PCD) [17] are two research directions relevant to batch proof. These techniques aim to prove the correctness of an ongoing computation in a way that a verifier can efficiently verify the correct execution of any prefix of the computation. While this technique is relevant to batch verification, it requires that the output of an instance is identical to the input of the next instance. Thus, the existing schemes cannot be applied for batch verification directly. Potentially, we can modify these schemes to tree-like recursions where the leaves are the batch instances and the internal nodes recursively aggregates their children nodes. However, the computational and communication complexity of this approach need to be further analyzed.

On a separate track, Reingold et al. investigated the settings where a prover wants to convince a verifier the correctness of m **NP** statements [56,57]. A later work extends the techniques to achieve zero-knowledge [42]. One of their main result is that for a special complexity class **UP** (i.e., **NP** statements that have a unique witness), there exists a statistically sound interactive proof protocol that uses a constant number of rounds with communication cost $O(m^{\delta} \cdot poly(|\mathbf{w}|))$, where $\delta > 0$ is an arbitrarily small constant, and $|\mathbf{w}|$ is the size of the witness of a single instance. Note that even though δ can be made arbitrarily small, the communication cost still depends on the batch size.

Instead of focusing on statistically sound prove systems, there is a line of work considering computationally sound argument systems for batch **NP** statements, which is also known as "BARGs" [13,15,23,26,33,65]. A BARG is an argument system for a batch of m **NP** statements where the size of the proof and the verification time grow sublinearly with the batch size. Brakerski et al. achieved 2-message non-adaptive delegation protocols for batch NP verification that requires $O(|\mathbf{w}| \cdot poly(\lambda))$ bits of communication, and $O(poly(\lambda) \cdot (m|\mathbf{io}| + |\mathbf{w}|))$ verifier time [15]. The communication cost is independent of the batch size.

However, the verification requires a secret key, which limits its applications [15]. Bootle et al. [13] proposed a polynomial commitment protocol. Based on that, they constructed an efficient zero-knowledge BARG scheme for arithmetic circuits representing low degree polynomials. The communication cost of this protocol is proportional to $O(\sqrt{m})$, the square root of the batch size. Choudhuri et al. consider BARGs from standard cryptographic assumptions such as SE-LHC and CIH for TC^0 circuits [23]. Assuming these standard assumptions, they constructed a BARG for C-SAT in the CRS mode with non-adaptive soundness. The argument size is $\tilde{O}(\lambda(|C|+\sqrt{m|C|}))$, where λ is the security parameter. Waters et al. investigated the same problem but under the bilinear group assumptions [65]. Their construction follows he "commit-and-prove" strategy, and produces arguments with $O(poly(\lambda, |C|))$ size, and $O(poly(\lambda, m, |\mathbf{io}|)+poly(\lambda, |C|))$ verification complexity. Devadas et al. proposed rate-1 BARGs built on top of the LWE assumption which achieves argument size of $O(|\mathbf{w}| + poly(\lambda, \log m))$ [26]. Garg et al. discussed fully succinct zero-knowledge BARGs from indistinguishability obfuscation under the CRS model where the argument size can be made to be $O(poly(\lambda, \log m, \log |C|))$ which scales sublinearly with both m and C [33].

All of these BARGs constructions aside from Brakerski et al.'s and Waters et al.'s construction produce arguments whose size grows along with the batch size. Compared to these prior works, our proposed protocol focuses on the batch evaluation and satisfiability problem of *log-space uniform circuits*, which is a subset of class **NP**, but still captures a large set of computation problems which are highly important both in theory and practice. By leveraging the structure of the log-space uniform circuits, our protocol achieves $O(poly(\lambda) \cdot (d \log |C| + |\mathbf{w}|))$ argument size, which does not depend on the batch size, and is asymptotically better than $O(poly(\lambda, |C|))$ achieved by Waters et al.'s construction. Our scheme also allows public verification, unlike Brakerski et al.'s approach which relies on a secret verification key. For a special case where all the instances in the batch shares the same witness, the argument size of our protocol can be further reduced to $O(poly(\lambda) \cdot (d \log |C| + \log |\mathbf{w}|))$. Hence, our argument scheme achieves the best known asymptotic argument size for the batch satisfiability problem for log-space uniform circuits.

1.3 Roadmap

The remainder of this paper is organized as follows. Section 2 lays down the technical foundation necessary for constructing the proposed protocols. Then, we formally define the batch circuit satisfiability and evaluation problem in Section 3, and provide an example to motivate the proposed protocols. In Section 4, we present a computationally sound argument scheme for generic log-space uniform circuits which can produce highly compact arguments even for large batches. For the batch evaluation problem, Section 5 presents our statistically sound interactive proofs for special circuits. In particular, we discuss special versions of protocols optimized for batch FFT evaluation and batch matrix production verification.

2 Preliminaries

2.1 Arithmetic Circuits

An arithmetic circuit C over field $\mathbb{F}$ is a directed *acyclic* graph whose nodes are labelled by $+$ or $*$, computing field element addition and multiplication respectively, for the values on the incoming wires. In this paper, we focus on the so-called *log-space uniform circuit* which has a succinct implicit description that can be efficiently represented in logarithmic space [60, 67]. This means that there is a logarithmic-space algorithm that takes as input the label of a gate g in C, and is capable of determining all relevant information about that gate, including the labels of g's neighboring gates, as well as the type of gate (e.g., addition or multiplication) [60]. Various proofs and argument schemes in the literature, including libSTARK [4], zkVSQL [70], Hyrax [64], and Libra [67] also focus on this type of circuits. Throughout the rest of this paper, the term "circuit" refers to a log-space uniform circuit unless stated otherwise. Most computations can be mapped to low-depth log-space uniform circuits [67]. Examples include matrix multiplication, image scaling and Merkle hash tree. Moreover, as shown in [4, 60], asymptotically all random memory access (RAM) programs can be validated using log-space uniform circuits with log-depth in program running time through RAM-to-circuit reduction, which illustrates the expressive power of such circuits.

It can be proved that any log-space uniform arithmetic circuit can be transformed into a *layered circuit*, while increasing the circuit size by a factor of at most the circuit depth [64]. A circuit is layered if it can be partitioned into subsets called layers, such that every wire in the circuit is between adjacent layers. We call the number of layers the depth of the circuit, which is denoted by d. We also assume all the gates in the circuit have two inputs. Following the literature convention, we label the input layer as the d^{th} layer and the output layer as the 0^{th} layer. Correspondingly, we denote the input vector as $\mathbf{v}_d$, and the output vector as $\mathbf{v}_0$. In addition to the public inputs, the circuit may also take in a witness vector that is only known to the prover. We use $\mathbf{w}$ to denote the witness vector. Fig. 1 depicts two small layered circuits where the left one only has public inputs, while the right one takes in both the public inputs and the witness.

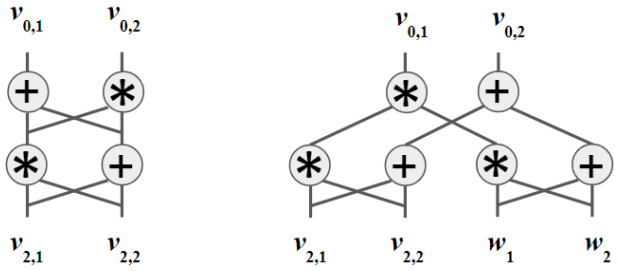

Fig. 1. Two layered circuit examples, where the left one only takes in public input vector $\mathbf{v}_2$, while the right one takes in both the public input vector $\mathbf{v}_2$ and witness $\mathbf{w}$.

Based on the above definition, for the example circuit on the left, the input and output vector are denoted as $\mathbf{v}_2$ and $\mathbf{v}_0$, respectively. Correspondingly the two input variables are labeled as $v_{2,1}$ and $v_{2,2}$. The first subscript indicates they are on the 2^{nd} layer. The second subscript is the index of the input variable in the input vector. Similarly the two output variables are labeled as $v_{0,1}$ and $v_{0,2}$.

Note that since the circuit only contains addition and multiplication gates, each output can be expressed as a multivariate polynomial in terms of the input variables. For example, for the left circuit in Fig. 1, the two outputs can be expressed $v_{0,1} = v_{2,1}v_{2,2} + v_{2,1} + v_{2,2}$ and $v_{0,2} = v_{2,1}^2 v_{2,2} + v_{2,1}v_{2,2}^2$, respectively. We use letter D to denote the *total degree* of the circuit, which is the maximum total degree across all the outputs expressed as polynomial in terms of input variables. In this circuit the degree of output $v_{0,1}$ is 2, and that of $v_{0,2}$ is 3. Hence, the total degree of the circuit is 3.

2.2 Succinct Noninteractive Argument Systems

Argument systems assume *computationally bounded* provers [16,60]. In more detail, an argument system for a function f is an interactive proof for f in which the soundness condition is only required to hold against prover strategies that run in polynomial time. This notion of soundness is called *computational soundness*. Arguments are allowed to use cryptographic primitives, as we assume that a polynomial time prover is unable to break the primitives. Incorporating cryptography achieves additional useful properties that are otherwise unattainable, such as public verifiablity, succinctness, and non-interactivity.

In particular, *succinctness* means that the argument is short, at least sublinear of the size of the witness. *Non-interactivity* means that the argument is static, consisting of a single message from the prover to the verifier. An interactive proof or argument can be turned into a non-interactive argument by appliying the Fiat-Shamir transformation [7, 21, 27, 49, 67]. There are other desirable properties of an argument, for example, *argument of knowledge* which roughly means that not only that a statement is valid, but also that the prover knows a witness to the veracity of the statement. To be more precise, if the prover can convince the verifier to accept the statement with non-negligible probability, then it is possible to efficiently extract the witness from the prover. Argument systems that satisfy all of or even a subset of these properties have a myriad of applications in various fields both in theory and practice.

2.3 Interactive Proof Protocols

Different than argument systems, proof systems assume a stronger *computationally unbounded* prover. An interactive proof is an interactive protocol where a prover tries to convince a verifier a statement by communicating with the verifier through a series of rounds. In each round, the verifier is allowed to ask questions based on the prover's previous answers [60]. Interactive protocols have two key properties, namely completeness and soundness. Roughly speaking, completeness means that for a valid instance, the verifier is convinced with probability 1.

Soundness means that there exists a constant ϵ, such that the prover can convince the verifier to accept an invalid instance with probability at most ϵ. The constant ϵ is called the soundness error. Note that in this definition, the soundness should hold even against computationally unbounded provers that could allocate enormous computational resources in attempt to to trick a verifier into accepting an incorrect answer. This soundness notion is referred to as *statistical soundness* or *information-theoretic soundness*. Please refer to [49, 60] for more formal definitions.

Sumcheck Protocol. The Sumcheck protocol plays a crucial role in the interactive proof literature. The Sumcheck problem computes the sum of a multivariate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ over all binary inputs, i.e., $\sum_{b_1,b_2,..,b_n \in \{0,1\}} f(b_1, b_2, .., b_n)$. Since there are $N = 2^n$ possible input combinations, computing the sum directly takes $O(N)$ time. Lund et al. [50] devised an ingenious *Sumcheck* protocol which allows a verifier to delegate the computation to a powerful prover, who can compute the sum and convince the verifier through an interactive protocol that the summation was computed correctly. It can be proved that the *Sumcheck* protocol runs in $O(\log N)$ rounds with proof size $O(\log N)$, and is complete and sound with soundness error $\epsilon = O(\log N/|\mathbb{F}|)$ [49, 50, 60].

GKR Protocol. The GKR protocol was proposed in the seminal work by Goldwasser et al. [34] as an interactive protocol for proving the correctness of layered circuit evaluation. The GKR protocol uses the Sumcheck protocol as a core building block. In the very beginning, the prover and verifier agree on a layered circuit C composed of addition and multiplication gates with two inputs. In the first message, the prover sends the claimed output of the circuit to the verifier. The verifer cannot verify this claim by herself. However, she can work with the prover to reduce this claim to the a claim about the previous layer. Then, the protocol processes the circuit layer by layer until the input layer where the verify has sufficient information to verify the final claim. In more detail, in the i^{th} iteration, the GKR protocol invokes the Sumcheck protocol to reduce a claim about the *multilinear extension* (see Section 2.4) of the gate values at layer i to that of the gate values at layer $i + 1$. Finally, at the input layer, the verifier has all the data needed to derive the multilinear extension of the input vector on her own. Therefore, the verifer can check the final claim herself, which in turn proves or disproves the correctness of the circuit evaluation.

2.4 Polynomial Interpolation

Lagrange interpolation. Given a set of m distinct field elements $\{t_1, t_2, ..., t_m\}$, the *Lagrange basis* for polynomials with degree $\leq m - 1$ is a set of polynomials $\{\delta_{t_1}(t), \delta_{t_2}(t), ..., \delta_{t_m}(t)\}$ each of degree $m - 1$, such that

$$\delta_{t_i}(t) = \begin{cases} 1, & t = t_i \\ 0, & \forall\ t \neq t_i, 1 \leq i \leq m \end{cases}$$

We can express $\delta_{t_i}(t)$ explicitly as follows:

$$\delta_{t_i}(t) = \prod_{1 \le j \le m,\ j \ne i} \frac{t - t_j}{t_i - t_j}, \text{ for } i = 1, 2, ...m \tag{1}$$

Then, given m points $\{(t_1, y_1)(t_2, y_2), ..., (t_m, y_m)\}$, the unique single variable polynomial with degree $\le m-1$ passing through these points can be written as:

$$L(t) = \sum_{i=1}^{m} y_i \delta_i(t) \tag{2}$$

In many applications, the evaluations points $\{t_i\}$ are simply integers, for example, $t_1 = 1, t_2 = 2, .., t_m = m$. Another popular choice is to let $t_i = \omega^{i-1}$ where ω is the m^{th} root of the unity of field $\mathbb{F}$. This choice can sometimes simplify calculations.

Multilinear extension. A multilinear function is a function of multiple variables that is linear separately in each variable. Given a function $f : \mathbb{F}^n \to \mathbb{F}$ and its evaluations over Boolean hypercube $\{0,1\}^n$, its multilinear extension $\tilde{f}(t)$ is a multilinear function which agrees with $f(t)$ for any $t \in \{0,1\}^n$ [60]. We can derive the explicit expression of the multilinear expression for $f(\cdot)$ using the identity function define below.

Denote the b^{th} bit of the binary representation of $t \in \{0, 1, 2, .., 2^n - 1\}$ by t_b, the *identity function* is defined as below:

$$\tilde{eq}(x, t) = \prod_{b=1}^{n} (t_b \cdot x_b + (1 - t_b) \cdot (1 - x_b)) \tag{3}$$

A key property of the identity function is that $\tilde{eq}(x, t) = 1$ if $x = t$, and $\tilde{eq}(x, t) = 0$ otherwise. Using the property, we can derive the explicit expression of $\tilde{f}(\cdot)$ as $\tilde{f}(x_1, x_2, .., x_n) = \sum_{t \in \{0,1\}^n} \tilde{eq}(x, t) \cdot f(t)$, where x_b are the b^{th} bits of the binary representation of x.

2.5 Polynomial Evaluation

Single variable polynomials have two equivalent forms of representation, the coefficient form which uses the monomial basis, and the point-value form which uses the Lagrange basis.

Single point evaluation. Given a single variable polynomial $L(t)$, and a evaluation point $r \in \mathbb{F}$, our goal is to calculate $L(r)$.

- **Coefficient form.** Horner's method applies to the coefficient form $L(t) = a_0 + a_1 t + a_2 t^2 + ... + a_m t^m$. The trick is to rewrite the polynomial as $L(t) = a_0 + t(a_1 + t(a_2 + t(a_3 + ... + t(a_{m-1} + t a_m)...)))$. This allows us to compute $L(r)$ from the innermost parenthesis with $O(m)$ field multiplications and additions.

- **Point-value form.** The *Barycentric evaluation* method [9] applies to the point-value form $L(t) = \sum_{i=1}^{m} y_i \delta_i(t)$. Let us use $M(t)$ to denote the product $\prod_{1 \leq j \leq m} (t - t_j)$. Let $d_i = \prod_{1 \leq j \leq m, j \neq i} (t_i - t_j)$. We can rewrite $L(t)$ as $L(t) = M(t) \sum_{i=1}^{m} \frac{y_i}{d_i(t-t_i)}$. In particular, if we let $t_i = \omega^{i-1}$ where ω is the m^{th} root of unity, the expression simplifies to $L(t) = \frac{t^m - 1}{m} \sum_{i=1}^{m} \frac{y_i \omega^i}{x - \omega^i}$. Thus, given the point-value form $\{(1, y_1), (\omega, y_2), .., (\omega^{i-1}, y_i), .., (\omega^{m-1}, y_m)\}$, we can calculate $L(t)$ with $O(m)$ field multiplication and additions.

Multi-point evaluations. Given a single variable polynomial $L(t)$, the multi-point evaluation problem ask for the values of the polynomial at a batch of points $\{r_i \in \mathbb{F} \mid i = 1, 2, .., m\}$. Similar to single point evaluation, here we look at both the coefficient and point-value form.

- **Coefficient form.** Given $L(t) = a_0 + a_1 t + a_2 t^2 + ... + a_m t^m$, our goal is to evaluate it at N different points $t_1, t_2, ..., t_N$ where N could be much larger than m. It is well-known that this can be solved using the FFT algorithm if the evaluation points are $\{1, \omega, .., \omega^i, .., \omega^{N-1}\}$. The time complexity is $O(N \log N)$.
- **Point-value form.** Given the point-value form of the polynomial, we can use IFFT to transform it to the coefficient form, and then apply FFT to evaluate it on $\{1, \omega, .., \omega^i, .., \omega^{N-1}\}$. The overall time complexity is $O(N \log N)$.

2.6 Bilinear Groups

Given security parameter 1^λ, there exists bilinear group generators that can produce bilinear parameters $BP = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ [32, 51]. Here $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generators $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerative bilinear map. That is, $e(g^a, h^b) = e(g, h)^{ab}$ $\forall a, b \in \mathbb{F}_p$ and $e(g, h)$ generates $\mathbb{G}_T$.

2.7 Polynomial Commitment Schemes

At a high level, a polynomial commitment scheme allows a prover to compute a commitment to a polynomial. The commitment can later be "opened" at any evaluation point. To be more specific, to prove the committed polynomial $L(t)$ evaluates to y_r at point r, the prover can present the claimed value of $L(r)$ and an associated opening proof π_r to the verifer. The verifier can then checks the claimed value and proof against the commitment c, and decide if the claimed value is correct. More formally, a polynomial commitment scheme can be defined as [22, 43, 51, 59, 64, 69]:

Definition 1. *A polynomial commitment scheme is a tuple of three protocols* $PC = (Commit, Open, Verify)$*, such that*

- $c \leftarrow Commit(BP, crs, N, L(x))$ *takes as input the bilinear parameters* BP*, a common reference string* crs*, and a polynomial* $L(x)$ *whose degree is at most* N*, and produces a commitment* c*.*

- $(L(r), \pi_r) \leftarrow Open(BP, crs, N, c, r, L(x))$ *takes as input the same parameters as the commit algorithm, as well as a commitment* c *and a point in the field* r*. It returns the evaluation* $L(r)$ *and a proof of its correctness* π_r*.*
- $accept \leftarrow Verify(BP, crs, N, c, r, y_r, \pi_r)$ *takes as input the bilinear parameters, the common reference string, the maximum degree, a commitment, a field element* r*, and the claimed* $y_r = L(r)$*, and the corresponding proof* π_r*. It outputs a Boolean bit indicating acceptance or rejection.*

Kate et al. [43] introduced the concept of polynomial commitment and provided the first construction for univariable polynomials based on pairing assumptions. This constructions is later extended to multivariate polynomials [54,69–71] and/or based on different assumptions [12,13,29,48]. Note that the Kate commitment requires a trusted setup to produce a common reference string crs induced by a secret trapdoor τ. The trusted setup can be conducted through a multi-party computation process to minimize the security risks. Such a process allows multiple parties to collectively generate the crs and yet no single party knows τ as long as there is an honest party participated in the process [25].

3 Batch Circuit Satisfiability and Evaluation Problems

This section defines the main problems we will address in this paper, namely, the batch arithmetic circuit satisfiability and evaluation problems. We first formally define these two problems. Next, we use an example to motivate the main techniques for the argument and proof systems proposed later in the paper.

Common Notations. In the remainder of this paper, we will use m to denote the batch size. A batch of size m consists of m "instances", i.e., m pieces of data. $|\cdot|$ means the size/length of an object. For example, $|C|$ refers to the size of the circuit, i.e. the total number of gates in the circuit. $|\mathbf{io}|$ represents the total length of the input and output vector. $|\mathbf{w}|$ denotes the length of the witness vector. d represents the depth of the circuit. We number the output layer as the 0^{th} layer and the input layer as the d^{th} layer. Correspondingly, the output vector of the circuit is denoted as $\mathbf{v}_0$, and the input vector as $\mathbf{v}_d$. D represents the maximum total degree of the circuit. For a function defined over the instance indices (e.g., the input/output polynomials in Section 3.3), we will use the "hat" symbol over the function to represent its low-degree polynomial extension, e.g. $\hat{\mathbf{v}}_{d,i}(t)$. For a function defined over a Boolean hypercube, we will use the "tilde" symbol over the function to represents its multilinear extension, e.g. $\tilde{\mathbf{w}}(r)$.

3.1 Batch Circuit Satisfiability Problem

Definition 2. *Consider a probabilistic polynomial time (PPT) prover* $\mathcal{P}$ *and a verifer* $\mathcal{V}$ *with a generation phase* $G(1^\lambda)$ *which produces public parameters* pp *given security parameter* λ*. Both* $\mathcal{P}$ *and* $\mathcal{V}$ *are given an arithmetic circuit* C *and a batch of input/output vectors* $B = \{(\mathbf{v}_d^{(t)}, \mathbf{v}_0^{(t)}) \mid t = 1, 2, .., m\}$*. In addition, the prover also has access to a batch of witness vectors* $W = \{\mathbf{w}^{(t)} \mid t =$

$1, 2, .., m\}$. *They exchange a sequence of messages* π*, and then* $\mathcal{V}$ *outputs a Boolean bit* $accept(\mathcal{P}(W), \mathcal{V}, B, \pi, pp)$*. We call* π *an* ***interactive batch argument of knowledge*** *for* C *if the following holds:*

- ***Completeness.*** *For every* $pp \leftarrow \mathcal{G}(1^\lambda)$ *and every batch with* $C(\mathbf{v}_d^{(t)}, \mathbf{w}^{(t)}) = \mathbf{v}_0^{(t)}$ *for* $t = 1, 2, ...m$*, it holds that* $\Pr[accept(\mathcal{P}(W), \mathcal{V}, B, \pi, pp) = true] = 1$.
- ***Knowledge-Soundness.*** *For any PPT prover* $\mathcal{P}^*$*, there exists a PPT extractor* $\mathcal{E}$ *such that given the access to the entire executing process and randomness of* $\mathcal{P}^*$*,* $\mathcal{E}$ *can extract a batch of witness vectors* $W = \{\mathbf{w}^{(t)} \mid t = 1, 2, .., m\}$*. With* $pp \leftarrow \mathcal{G}(1^\lambda)$*,* $\pi^* \leftarrow \mathcal{P}^*(B, pp)$ *and* $W \leftarrow \mathcal{E}^{\mathcal{P}^*}(pp, B, \pi^*)$*, it holds that* $\Pr[accept(\mathcal{P}^*, \mathcal{V}, B, \pi^*, pp) = true] < negl(\lambda)$ *if there exists an instance* $1 \leq t \leq m$ *such that* $C(\mathbf{v}_d^{(t)}, \mathbf{w}^{(t)}) \neq \mathbf{v}_0^{(t)}$.

Definition 3. ***Efficient Interactive Batch Argument****. We say that an interactive batch argument scheme is prover-efficient if the prover online running time is in* $O(poly(\lambda, |C|, m))$*. It is verifier-efficient if the verifier time is in* $O(m \cdot |\mathbf{io}| + poly(\lambda, |\mathbf{w}|, \log |C|, \log m))$*. It is said to have a succinct argument if the argument size is in* $O(poly(\lambda, |\mathbf{w}|, \log |C|, \log m))$.

Remark 4. We note that in the efficient verifier time definition $O(m \cdot |\mathbf{io}| + poly(|\mathbf{w}|, \log |C|, \log m))$, the term $O(m \cdot |\mathbf{io}|)$ is inevitable since the verifiable at least needs to read the input and output vectors of all the instances in the batch. This definition essentially requires that the *additional* verifier time is in $O(poly(\lambda, |\mathbf{w}|, \log |C|, \log m))$.

3.2 Batch Circuit Evaluation Problem

The batch circuit evaluation problem is similar to satisfiability, but it is for circuits without witness and assumes a computationally unbounded prover.

Definition 5. *Consider a computationally unbounded prover* $\mathcal{P}$ *and a verifer* $\mathcal{V}$*. Both* $\mathcal{P}$ *and* $\mathcal{V}$ *are given an arithmetic circuit* C *and a batch of input/output vectors* $B = \{(\mathbf{v}_d^{(t)}, \mathbf{v}_0^{(t)}); \mid t = 1, 2, .., m\}$*. They exchange a sequence of messages* π*, and then* $\mathcal{V}$ *outputs a single bit* $accept(\mathcal{P}, \mathcal{V}, B)$*. We call* π *an* ***interactive batch proof*** *for* C *with soundness error* ϵ *if the following holds:*

- ***Completeness.*** *For every batch such that* $C(\mathbf{v}_d^{(t)}) = \mathbf{v}_0^{(t)}$ *for* $t = 1, 2, ...m$*, it holds that* $\Pr[accept(\mathcal{P}, \mathcal{V}, B) = true] = 1$.
- ϵ***-Soundness.*** *There exists a constant* $\epsilon < 1/2$*, such that for any batch* B *where there is one* t *such that* $C(\mathbf{v}_d^{(t)}) \neq \mathbf{v}_0^{(t)}$ *and any* P^**, it holds that* $\Pr[accept(\mathcal{P}^*, \mathcal{V}, B) = true] < \epsilon$.

Definition 6. ***Efficient Interactive Batch Proof****. We say that an interactive batch proof scheme is prover-efficient if the prover online running time is in* $O(poly(|C|, m))$*. It is verifier-efficient if the verifier time is in* $O(m \cdot |\mathbf{io}| + poly(\log |C|, \log m))$*. It is said to have a succinct proof if the proof size is in* $O(poly(\log |C|, \log m))$.

3.3 A Motivating Example

Our goal is to design efficient interactive argument and proof protocols for batch circuit satisfiability and evaluation. As a motivating example, we use a simple arithmetic circuit on the left side of Fig. 1 to illustrate the basic ideas. The small circuit has two layers, each with an addition and a multiplication gate for prime field $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$ where p is a large prime. Fig. 2 shows the evaluation of this circuit for a batch with four input vectors $\{(2,4),(5,7),(10,10),(17,13)\}$.

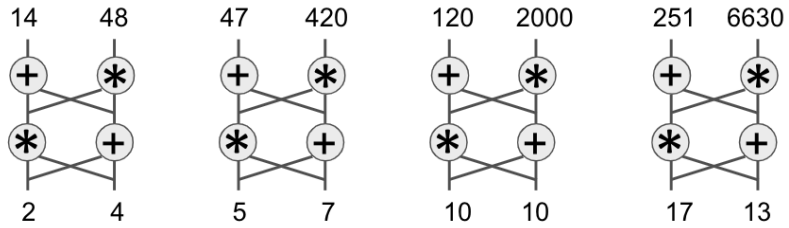

Fig. 2. An example circuit and its batch evaluation.

Let us use $t = 1, 2, .., m$ to denote the "instance index," i.e., the index of the instance in the batch, where m is the size of the batch. For instance, in the above example $m = 4$, and input vector $(10, 10)$ has instance index $t = 3$.

We can view $v_{2,1}(t)$ and $v_{2,2}(t)$ as mappings from instance index t to the input values, i.e, $v_{2,1}(1) = 2$, $v_{2,1}(2) = 5$, $v_{2,1}(3) = 10$, $v_{2,1}(4) = 17$ and $v_{2,2}(1) = 4$, $v_{2,2}(2) = 7$, $v_{2,2}(3) = 10$, $v_{2,2}(4) = 13$. Using Lagrange interpolation, the prover can derive the *input polynomials*, i.e. the low degree extensions of these mappings, $\hat{v}_{2,1}(t) = t^2 + 1$ and $\hat{v}_{2,2}(t) = 3t + 1$. The prover can feed these polynomials as inputs into the circuit, compute the polynomials for all the intermediate gates, and eventually obtain the *output polynomials* $\hat{v}_{0,1}(t) = 3t^3 + 2t^2 + 6t + 3$ and $\hat{v}_{0,2}(t) = 3t^5 + 10t^4 + 12t^3 + 12t^2 + 9t + 2$. Please refer to Fig. 3 for more details.

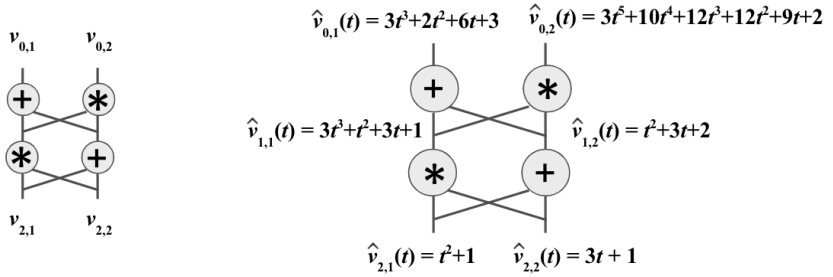

Fig. 3. Derive the output polynomials from the input polynomials.

Next, the prover sends these two output polynomials $\hat{v}_{0,1}(t)$ and $\hat{v}_{0,2}(t)$ to the verifier. The verifer first needs to evaluate these polynomials at $t = 1, 2, .., m$ to see if all the evaluation results match with the claimed outputs of the t^{th} circuit instance. if any inconsistency is found, the verifier should reject the batch.

Otherwise, the verifier samples a random field element r, and send it back to the prover. After that, the prover evaluates the output polynomials at r to obtain outputs $(\hat{v}_{0,1}(r), \hat{v}_{0,2}(r))$. The verifier calculates inputs $(\hat{v}_{2,1}(r), \hat{v}_{2,2}(r))$ on her own. Here we note that since the verifier has access to the input vectors, she can derive the input polynomials by herself. After these preparation steps, the prover and verifier can execute the GKR protocol to verify whether the circuit indeed outputs $(\hat{v}_{0,1}(r), \hat{v}_{0,2}(r))$ for inputs $(\hat{v}_{2,1}(r), \hat{v}_{2,2}(r))$.

To get a sense why this protocol works, we argue that for completeness, it is straightforward to see that evaluating the circuit with inputs $(\hat{v}_{2,1}(r), \hat{v}_{2,2}(r))$ must result in outputs $(\hat{v}_{0,1}(r), \hat{v}_{0,2}(r))$ for any $r \in \mathbb{F}$ if the output polynomials sent from the prover are correct. For soundness, the intuition is that due to the Schwartz-Zippel Lemma [58, 72], if any output of the claimed batch evaluation is incorrect, only with a negligible probability that the circuit outputs $(\hat{v}_{0,1}(r), \hat{v}_{0,2}(r))$ for inputs $(\hat{v}_{2,1}(r), \hat{v}_{2,2}(r))$ for a random r.

This protocol essentially "reduces" the entire batch to a single circuit induced by random value r. The verifier only needs to engage with the prover to verify the evaluation of this circuit instead of the entire batch, which significantly decreases the verifer time complexity. Fig. 4 illustrates the intuition behind the batch to single circuit reduction.

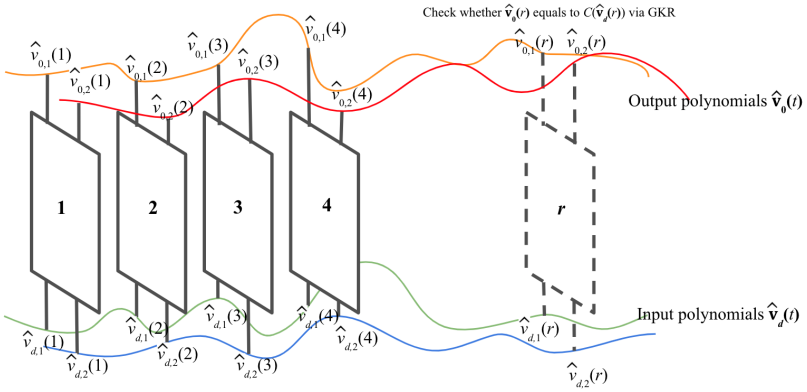

Fig. 4. Intuition of the batch to single circuit reduction. In this example, both the output polynomials $\hat{\mathbf{v}}_0(t) = (\hat{v}_{0,1}(t), \hat{v}_{0,2}(t))$ and input polynomials $\hat{\mathbf{v}}_d(t) = (\hat{v}_{d,1}(t), \hat{v}_{d,2}(t))$ are vectors of polynomials of size 2. $\hat{\mathbf{v}}_d(t)$ can be obtained through Lagrange interpolation. To derive $\hat{\mathbf{v}}_0(t)$, the prover feeds $\hat{\mathbf{v}}_d(t)$ into the circuit C, and conducts polynomial addition and multiplication layer by layer until reaching the outputs. To check if the batch evaluation is correct, the verifier picks a random field element r, and engages with the prover to check whether $\hat{\mathbf{v}}_0(r) = C(\hat{\mathbf{v}}_d(r))$ via the GKR protocol.

However, to turn these ideas into efficient interactive argument or proof systems, there are several issues to be addressed:

- **Argument size**: The degree of the output polynomials grows quickly with the batch size. Sending these polynomials directly to the verifier could lead to high communication cost and argument/proof size.

- **Prover efficiency**: For the prover, directly calculating the polynomial form of all the intermediate gates could be costly even with optimized polynomial multiplication algorithm utilizing FFT.
- **Circuit satisfiability**: The above protocol only applies to the batch circuit evaluation problem. How to handle circuits with witnesses?

In the remainder of this paper, we will discuss how we can address these hurdles. The next couple sections will dive into the details.

4 Succinct Arguments for Generic Circuit Satisfiability

In this section, we will present an argument system for generic log-space uniform circuits. At a high level, the construction of our argument system follows the "commit-and-prove" strategy [38,39,65]. Extending the core ideas in our motivating example, we first designed an efficient compiler which can reduce the entire batch to a single instance via an interactive argument protocol. The several shortcomings mentioned above in motivating example are addressed using polynomial commitments. Next, we can apply the GKR protocol to prove the correctness of this reduced instance. Finally, using the Fiat-Shiamir transformation, we can render the entire argument system non-interactive.

4.1 Batch to Single Instance Compiler

The compiler works by first transforming the circuit under consideration into one that has a single output. Then, the prover derives the low degree output polynomial for this output. However, instead of sending the output polynomial directly to the verifier, the prover sends its polynomial commitment. This addresses the communication cost bottleneck. The witnesses are handled similarly with polynomial commitments. Next, the verifier samples a random value $r \in \mathbb{F}$ and send it to the prover. The prover sends back the evaluations of the output polynomial and witness polynomials at r, which effectively reduces the batch to a single instance derived from r. Below we first describe the key building blocks, and then present the construction of the compiler.

Circuit Transformation. Given circuit evaluation $C(\mathbf{v}_d, \mathbf{w}) = \mathbf{v}_0$, we can rewrite it as $C(\mathbf{v}_d, \mathbf{w}) - \mathbf{v}_0 = \mathbf{0}$, where $\mathbf{0}$ is a zero vector of size $|\mathbf{v}_0|$. We can construct a circuit $C'(\mathbf{v}_d, \mathbf{w}, \mathbf{v}_0, \mathbf{s})$ which represents the following formula:

$$b = (C(\mathbf{v}_d, \mathbf{w}) - \mathbf{v}_0) \cdot \mathbf{s} \tag{4}$$

Here vector $\mathbf{s} = (1, s, s^2, .., s^{|\mathbf{v}_0|-1})$, where $s \in \mathbb{F}$ is a randomly sampled field element. This transformed circuit C' has a single output $b \in \mathbb{F}$. We claim that $C(\mathbf{v}_d, \mathbf{w}) = \mathbf{v}_0$ holds with high probability if $b = 0$:

Lemma 7. *If $s \in \mathbb{F}$ is uniform-randomly sampled and $b = 0$, then $C(\mathbf{v}_d, \mathbf{w}) = \mathbf{v}_0$ holds with probability at least $1 - |\mathbf{v}_0|/|\mathbb{F}|$.*

Algorithm 1 Deriving the output polynomial of the transformed circuit C'

Participant: Prover $\mathcal{P}$

Step 1. Input processing
- Evaluate the input polynomials at $\{1, \omega, .., \omega^i, .., \omega^{mD-1}\}$ to obtain the point-value form of the each input polynomial of the transformed circuit C'
- Let us denote the point-value form of the j^{th} input polynomial of circuit C' by $\{(\omega^i, \hat{v}_{d,j}(\omega^i) \mid i = 0, .., mD-1\}$

Step 2. Derive the output polynomial $\hat{b}(t)$
for layer k **in** $\{d-1, ...0\}$ **do**
  for each gate g in layer k **do**
    denote the index of the output wire of g in layer k by out_g
    denote the indices of the two input wires of g in layer $k+1$ by $in1$ and $in2$
    if g is a multiplication gate **then**
      $out_g = \{(\omega^i, \hat{v}_{k+1,in1}(\omega^i) \cdot \hat{v}_{k+1,in2}(\omega^i) \mid i = 0, .., mD-1\}$
    else ▷ gate g is an addition gate in this case
      $out_g = \{(\omega^i, \hat{v}_{k+1,in1}(\omega^i) + \hat{v}_{k+1,in2}(\omega^i) \mid i = 0, .., mD-1\}$
    end if
  end for
end for ▷ the output layer (i.e., when $k = 0$) has a single gate, and out_g is $\hat{b}(t)$

Moreover, the transformation only increases the size and depth of circuit C by a constant factor:

Lemma 8. *The size and the depth of transformed circuit C' is in the same order as C, i.e. $O(|C'|) = O(|C|)$, and $O(d') = O(d)$. Moreover, the total degree of C' remains to be D.*

The proofs for the above two lemmas are provided in Appendix A. The transformation benefits both prover time and argument size. Since the original circuit has multiple outputs, without the transformation, the prover needs to generate the polynomial commitment of all the outputs, which leads to higher prover time and communication overhead. The transformation turns the original outputs into inputs. This way, the verifer can conduct the interpolation by herself, and evaluate the interpolated input polynomials at point r on her own.

Deriving the Output Polynomial. As in the motivating example, the prover needs to derive the output polynomial $\hat{b}(t)$ for the transformed circuit C'. To to that, the prover can first perform the Lagrange interpolation for all the inputs of C' across all instances (see Section 2.4). Then the prover works its way towards the output layer by layer to calculate the polynomials for each intermediate gate's output as described in Algorithm 1. Note that to reduce the prover's time complexity, the algorithm conducts the derivation using the point-value form for the polynomials. To accomplish this, the prover first evaluates each of the input polynomials at $\{1, \omega, .., \omega^i, .., \omega^{mD-1}\}$ using the multi-point evaluation technique described in Section 2.5. Then, she can compute the point-value polynomials of

the intermediate gates layer by layer until reaching the output gate and obtaining $\hat{b}(t)$ in the end. The time complexity of Algorithm 1 is given by Lemma 9, whose proof is provided in the Appendix A:

Lemma 9. *The overall time complexity for derivingt the output polynomial* $\hat{b}(t)$ *is* $O(poly(\lambda) \cdot mD(|C| + \log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|)))$.

In a typical case where $\log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|) < |C|$, the time complexity simplifies to $O(poly(\lambda) \cdot D \cdot m|C|)$, which is just a factor D away from $O(poly(\lambda) \cdot m|C|)$, the time required for the prover to evaluate the batch. Also, it is worth noting that the above described process is easy to parallelize. This is because aside from the initial multi-point evaluations, the prover essentially just needs to evaluate the circuit on a larger batch consisting of mD instances. These evaluations can trivially be conducted in parallel on multiple machines.

Commitment and Multiproof for the Output Polynomial. The Kate polynomial commitment has a nice feature that it can produce a proof with constant size for evaluations at multiple points. Let us use $\hat{b}(t)$ to denote the output polynomial of C' obtained using the algorithm described in the previous subsection. The commitment for multi-point evaluation is still a single elliptic curve point $c = g^{\hat{b}(\tau)}$, where g is a group generator and τ is a trapdoor [11, 43].

In the context of our batch satisfiability problem, recall that with overwhelming probability, the entire batch is satisfiable if the output polynomial $\hat{b}(t) = 0$ for all $t = 1, \omega, .., \omega^{m-1}$. Suppose in addition to this, we also would like to prove that $\hat{b}(t)$ evaluates to z_r at point $r \notin \{1, \omega, .., \omega^{m-1}\}$. Such a claim is equivalent to $\hat{b}(t) - z_r = (t - r) \cdot \prod_{i=1}^{m} (t - \omega^i) \cdot q(t)$. Thus, the opening proof π_r sent to the verifier is just a single elliptic curve point, i.e., $\pi_r = g^{q(\tau)}$. Upon receiving the proof, the verifier just need to check if the following equation holds [11, 43]:

$$e(c - g^{z_r}, h) = e(\pi_r, h^{(\tau - r) \cdot \prod_{i=1}^{m} (\tau - \omega^i)}) \tag{5}$$

Notice that $h^{(\tau-r)\cdot\prod_{i=1}^{m}(\tau-\omega^i)} = h^{\tau\prod_{i=1}^{m}(\tau-\omega^i)} \cdot (h^{\prod_{i=1}^{m}(\tau-\omega^i)})^{-r}$, where both $h^{\tau\prod_{i=1}^{m}(\tau-\omega^i)}$ and $h^{\prod_{i=1}^{m}(\tau-\omega^i)}$ can be pre-computed during the preprocessing phase. Therefore, the online time complexity of the verifier is $O(\log|\mathbb{F}|)$ group multiplication and two pairing computation, which is independent of m.

In summary, we can design an interactive protocol which proves $\hat{b}(t) = 0$ for $t = 1, 2, .., m$, and $\hat{b}(r) = z_r$ with high probability: The prover sends the commitment $c = g^{\hat{b}(\tau)}$ to the verifier, and the verifier sends back a random $r \in \mathbb{F}$. Then, the prover sends back opening proof $g^{q(\tau)}$. Finally, the verifier checks if Equation 5 holds. The communication cost is $O(\lambda)$ and the verifier time complexity is $O(poly(\log|\mathbb{F}|)) = O(poly(\lambda))$.

Commitments and Proofs for the Witness Polynomials. Denoting the k^{th} element of the witness vector $\mathbf{w}$ by w_k, this element has different values across the m instances in the batch. Thus we can treat it as the evaluation of a function at $\{\omega^t \mid t = 1, 2, .., m\}$. Denoting its low-degree polynomial extension of this

Algorithm 2 Compiler Preprocessing

Participants: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Preprocessing (run once per circuit)
- $\mathcal{P}$ and $\mathcal{V}$ both transform $\mathbf{v}_0 = C(\mathbf{v}_d, \mathbf{w})$ to $b = C'(\mathbf{v}_d, \mathbf{w}, \mathbf{v}_0, \mathbf{s})$
- $\mathcal{V}$ computes $h^{\tau \prod_{i=1}^{m} (\tau - \omega^i)}$ and $h^{\prod_{i=1}^{m} (\tau - \omega^i)}$

function by $\hat{w}_k(t)$, the low-degree extensions of the entire witness vector can be written as $\hat{\mathbf{w}}(t) = (\hat{w}_1(t), \hat{w}_2(t).., \hat{w}_{|\mathbf{w}|}(t))$. The prover can use the Kate commitment scheme to commit to $\hat{\mathbf{w}}(t)$ and create the commitments $\mathbf{k} = (k_1, k_2, .., k_{|\mathbf{w}|})$. Given $r \in \mathbb{F}$, the prover can generate proof $\rho_r = (\rho_{1,r}, \rho_{2,r}, .., \rho_{|\mathbf{w}|,r})$ for the vector evaluation $\hat{\mathbf{w}}(r)$.

The Compiler Protocol. The batch to single instance compiler protocol is outlined in Algorithm 2 and Algorithm 3 using the above building blocks.

Algorithm 2 describes the preprocessing step, which only needs to be run once per circuit, and the running time cost can be amortized later as we generate proofs for multiple batches. It performs the circuit transformation *without* setting the values of vector $\mathbf{s}$, i.e., it only transforms the circuit topology but does not set the values for inputs $\mathbf{s}$.

Algorithm 3 details the interactive argument protocol which reduces a given batch to a single instance on the transformed circuit C'. The prover first commits to the witness polynomials and sends the commitments $\mathbf{k} = (k_1, k_2, .., k_{|\mathbf{w}|})$ to the verifier.

Next, the verifier sends a random field element $s \in \mathbb{F}$ to the prover, so that both parties can set vector $\mathbf{s}$ to $(1, s, s^2, .., s^{|\mathbf{v}_0|-1})$. With $\mathbf{s}$, the prover can derive the output polynomial $\hat{b}(t)$, generate commitment c, and send c to the verifer.

Then, the verifier sends the prover another random challenge $r \in \mathbb{F}$. Then, the prover opens the commitments at the prescribed random point and send the polynomial evaluations and the corresponding proofs to the verifier. The verifier checks the evaluation results against the commitment and the proof. For the output polynomial, the verifier needs to check if $\hat{b}(t) = 0$ for all $t = 1, 2, .., m$, in addition to whether $\hat{b}(r) = b_r$. This is accomplished via checking the multiproof through Equation 5. If any of the checks fails, the verifier simply terminates and outputs false.

Finally, as the original input and output vectors of the instance are available to both the prover and verifier, they separately use the Barycentric evaluation method to evaluate $\hat{\mathbf{v}}_d(t)$, $\hat{\mathbf{v}}_0(t)$ at r. Note that both $\hat{\mathbf{v}}_d(t)$ and $\hat{\mathbf{v}}_0(t)$ are vectors of polynomials. Their evaluations at r give $\mathbf{v}_{d,r}$ and $\mathbf{v}_{0,r}$, both of which are vectors of field elements. This evaluation yields a single instance $C'(\mathbf{v}_{d,r}, \mathbf{w}_r, \mathbf{v}_{0,r}, \mathbf{s})$ induced by random field elements s and r, which is expected to evaluate to b_r.

Algorithm 3 Batch to Single Instance Compiler

Participants: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Step 1. Commit to the witness polynomials

- $\mathcal{P}$ derives the witness polynomials $\hat{\mathbf{w}}(t) = (\hat{w}_1(t), \hat{w}_2(t).., \hat{w}_{|\mathbf{w}|}(t))$ and generates Kate commitments $\mathbf{k} = (k_1, k_2, .., k_{|\mathbf{w}|})$
- $\mathcal{P}$ sends $\mathbf{k}$ to $\mathcal{V}$ ▷ sending $|\mathbf{w}|$ elliptic curve points

Step 2. Commit to the output polynomials

- $\mathcal{V}$ chooses a random element $s \in \mathbb{F}$, and send s to $\mathcal{P}$ ▷ sending one field element
- Both $\mathcal{P}$ and $\mathcal{V}$ set vector $\mathbf{s}$ to $(1, s, s^2, .., s^{|\mathbf{v}_0|-1})$
- $\mathcal{P}$ derives the output polynomial $\hat{b}(t)$ and generates commitment c
- $\mathcal{P}$ sends c to $\mathcal{V}$ ▷ sending 1 elliptic curve points

Step 3. Evaluate the output and witness polynomials at r

- $\mathcal{V}$ chooses a random element $r \in \mathbb{F}$, and send r to $\mathcal{P}$ ▷ sending 1 field element
- $\mathcal{P}$ opens $(b_r = \hat{b}(r), \pi_r = g^{q(\tau)})$ where $q(t) = \frac{\hat{b}(t) - b_r}{(t-r)\cdot\prod_{i=1}^{m}(t-\omega^i)}$
- $\mathcal{P}$ sends (b_r, π_r) to $\mathcal{V}$ ▷ sending 1 field element and 1 elliptic curve point
- $\mathcal{V}$ checks b_r against the commitment and proof (c, π_r) by verifying whether this pairing equation holds $e(c - g^{b_r}, h) = e(\pi_r, h^{(\tau-r)\cdot\prod_{i=1}^{m}(\tau-\omega^i)})$
- $\mathcal{V}$ terminates with **False** if the above verification fails
- $\mathcal{P}$ opens $(w_{i,r} = \hat{w}_i(r), \rho_{i,r})$ for $i = 1, 2, .., |\mathbf{w}|$, where $\rho_{i,r}$ is the Kate opening proof for $\hat{w}_i(t)$ at r
- $\mathcal{P}$ sends the evaluations $\mathbf{w}_r = (w_{1,r}, w_{2,r}, .., w_{|\mathbf{w}|,r})$ and the corresponding proofs $\rho_r = (\rho_{1,r}, \rho_{2,r}, .., \rho_{|\mathbf{w}|,r})$ to $\mathcal{V}$ ▷ sending $|\mathbf{w}|$ field elements and $|\mathbf{w}|$ elliptic curve points
- $\mathcal{V}$ checks $w_{i,r}$ against the commitment and proof $(k_i, \rho_{i,r})$ for $i = 1, 2, .., |\mathbf{w}|$
- $\mathcal{V}$ terminates with **False** if any of the above verifications fails

Step 4. Evaluate the input polynomials of C' at r

- Both $\mathcal{P}$ and $\mathcal{V}$ use the Barycentric evaluation method to evaluate the input polynomials $\hat{\mathbf{v}}_d(t)$ and $\hat{\mathbf{v}}_0(t)$ at r to obtain $\mathbf{v}_{d,r} = \hat{\mathbf{v}}_d(r)$ and $\mathbf{v}_{0,r} = \hat{\mathbf{v}}_0(r)$
- Now $\mathcal{P}$ and $\mathcal{V}$ have jointly created a single instance $C'(\mathbf{v}_{d,r}, \mathbf{w}_r, \mathbf{v}_{0,r}, \mathbf{s})$ induced by random field elements r and s, which is expected to evaluate to b_r

4.2 Interactive Argument Protocol for Circuit Satisfiability

With the batch to single instance compiler, the overall batch interactive argument protocol is straightforward to construct. Since both the prover and verifier have the inputs and the claimed output of C' available locally, they can just run the GKR protocol to check whether the reduced instance is evaluated correctly or not. The overall protocol is outlined in Algorithm 4.

Batch with Shared Witness In the above we describe a compiler for the generic case where each instance in the batch can have a unique witness. For a batch with a shared witness vector $\mathbf{w}$ across all the instances, we can optimize the communication cost even further. This is an important use case, e.g. to prove

Algorithm 4 Interactive Argument for Batch Circuit Satisfiability

Participants: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Preprocessing (run once per circuit)
- $\mathcal{P}$ and $\mathcal{V}$ run Algorithm 2 to transform circuit C to C'
- $\mathcal{V}$ pre-process C' in preparation for running the GKR protocol [60, 61]

Upon receiving a batch B
- $\mathcal{P}$ and $\mathcal{V}$ run Algorithm 3 to reduce the batch to a single instance. If $\mathcal{V}$ terminates with **False**, $\mathcal{V}$ rejects the batch B
- $\mathcal{P}$ and $\mathcal{V}$ run the GKR protocol on the reduced single instance. If the single instance is invalid, $\mathcal{V}$ rejects the batch B. Otherwise, $\mathcal{V}$ accepts the Batch B

that the prover knows a secret machine learning model that can generate certain prediction accuracy across a set of inputs.

For this special case, instead of deriving the witness polynomials and sending their commitments, the prover can just send the commitment of $\tilde{\mathbf{w}}(\cdot)$ (i.e. the multilinear extension of $\mathbf{w}$) to the verifier. This reduces the communication cost of this step from $O(|\mathbf{w}|)$ to $O(\log |\mathbf{w}|)$.

The last step of the GKR protocol requires the verifier to evaluate the multilinear extension of the input of the reduced single instance at a random point r'. Without loss of generality, let us assume $|\mathbf{v}_d| = |\mathbf{v}_0| = |\mathbf{w}|$. Denoting the multilinear extension of $\mathbf{v}_{d,r}$, $\mathbf{v}_{0,r}$ by $\tilde{\mathbf{v}}_{d,r}$ and $\tilde{\mathbf{v}}_{0,r}$, we can write the multilinear extension of $\mathbf{v}'_d$, the inputs of C' as [59, 60]

$$\tilde{\mathbf{v}'}_d = \alpha_0\alpha_1 \cdot \tilde{\mathbf{s}} + \alpha_0(1-\alpha_1) \cdot \tilde{\mathbf{v}}_{d,r} + (1-\alpha_0)\alpha_1 \cdot \tilde{\mathbf{v}}_{0,r} + (1-\alpha_0)(1-\alpha_1) \cdot \tilde{\mathbf{w}} \quad (6)$$

Since both the prover and verifier have access to $\mathbf{s}$, $\mathbf{v}_{0,r}$, and $\mathbf{v}_{d,r}$, they can derive $\tilde{\mathbf{s}}$, $\tilde{\mathbf{v}}_{0,r}$, and $\tilde{\mathbf{v}}_{d,r}$ on their own. In addition, the prover sends $\tilde{\mathbf{w}}(r')$ and the corresponding opening proof to the verifier, with which the verifier can evaluate Formula 6 at r'.

Non-Interactivity. It it well-known that we can turn interactive public-coin arguments into non-interactive arguments using the Fiat-Shamir transform [27]. The transform works by replacing the verifier challenges with hashes of the transcript up to that point. We note that the GKR protocol employed by Algorithm 4 is not a constant-round protocol. However, recent results [7, 20, 21, 49, 67] show that applying Fiat-Shamir only incurs a polynomial soundness loss in the number of rounds in GKR.

4.3 Correctness and Complexity of the Protocol

The theorem below demonstrates the completeness, knowledge-soundness, and the efficiency of the proposed interactive argument for batch circuit satisfiability protocol. The proof is provided in Appendix A.

Theorem 10. *There exists an interactive argument of knowledge protocol for the batch circuit satisfiability problem which achieves the following properties, assuming bilinear paring and the common reference string model:*

- ***Completeness***. *For every batch B such that $C(\mathbf{v}_d^{(t)}, \mathbf{w}^{(t)}) = \mathbf{v}_0^{(t)}$ for $t = 1, 2, ...m$, it holds that $\Pr[accept(\mathcal{P}, \mathcal{V}, B) = true] = 1$.*
- ***Knowledge-Soundness***. *The interactive argument protocol is knowledge-sound with soundness error $O(\frac{m(|\mathbf{io}|+|\mathbf{w}|+D)+d\log|C|}{2^\lambda})$ which is in $O(negl(\lambda))$.*
- ***Efficiency***. *The pre-processing time for both the prover and verifier are $O(poly(\lambda)\cdot|C|)$. The prover and verifier online time complexity are $O(poly(\lambda)\cdot m(D|C|+D\log(mD)\cdot(|\mathbf{io}|+|\mathbf{w}|)+|\mathbf{w}|\log m))$, and $O(poly(\lambda)\cdot(m|\mathbf{io}|+|\mathbf{w}|+d\log|C|))$, respectively. The argument size is $O(poly(\lambda)\cdot(|\mathbf{w}|+d\log|C|))$. This meets our definition of efficient interactive batch argument.*

Remark 11. Regarding the efficiency of the protocol, there are several interesting facts worth noting:

First, it is worth pointing out that the argument size not just meets our definition of succinctness in Definition 3, but it achieves an even more desirable property that the argument size is independent of the batch size. Moreover, for the special case where the same witness is shared among all instances, the prover just needs to send the multilinear commitment, evaluation result, and opening proof of $\tilde{\mathbf{w}}$ to the verifier. This can reduce the argument size to $O(poly(\lambda)\cdot(d\log|C|+\log|\mathbf{w}|))$ without increasing the soundness error.

Second, since the witness size is at most the size of circuit, we have $|\mathbf{w}| \leq |C|$. Thus, in a typical case where $\log(mD) < |C|$, the prover time complexity can be bounded by $O(poly(\lambda)\cdot m|C|\cdot(D+\log m))$ which is a factor of $(D+\log m)$ away from $O(poly(\lambda)\cdot m|C|)$, the time needed for pure computation (i.e. evaluating the circuit on the batch without generating the proof). In many potential applications, the batch size m is at most a few thousand. Thus, in practice, the prover time is just a constant factor larger than the computation time.

Third, we note that the verifier time complexity we achieve is only $O(poly(\lambda)\cdot(|\mathbf{w}|+d\log|C|))$ more than the lower bound $O(poly(\lambda)\cdot m\cdot|\mathbf{io}|)$. This additional verifier time remains a constant as the batch size increases.

Finally, compared to BARG schemes that commits to the values of all intermediate gates across all instances [65], our protocol only requires the prover to commit to the witnesses and the single output of the transformed circuit C'. This significantly reduces the prover run time and the argument size.

5 Proof Systems for Special Circuit Batch Evaluation

While the previous section handles the batch satisfiability problem for generic circuits, the argument protocols discussed there rely on certain cryptographic assumptions such as bilinear pairing and common reference string. In this section, we instead focus on the batch evaluation problem which is a simplified version of the batch satisfiability problem. In particular, we delve into the batch evaluation problem for two important special types of circuits: linear circuits, and

circuits representing sum of high degree polynomials. For both of these two types of circuits, we propose *statistically sound* interactive proof protocols where the communication cost is independent of the batch size. Moreover, these interactive proofs are all *public-coin* protocols. Thus, we can apply the Fiat-Shamir transformation [7, 21, 27, 49, 67] to turn them into non-interactive arguments under the random oracle model [3].

In addition, we adapt our interactive proof protocols to address two problems with significant practical relevance, namely, *batch FFT* and *batch matrix multiplication* verification. Utilizing the special structure of these two problems, the protocols can be further optimized to exhibit desirable properties such as reduced prover time complexity and enhanced composability. We believe that these protocols have their own importance and have the potential to be used as subroutines in more complex applications.

5.1 Batch Evaluation Proof for Linear Circuits

Intuitively speaking, a linear arithmetic circuit performs a linear transformation for the input vector. Below we give the formal definition.

Definition 12. *Given input vector $\mathbf{v}_d \in \mathbb{F}^n$, an arithmetic circuit $L(\mathbf{v}_d)$ is called a linear arithmetic circuit if the following two properties hold:*

1. *For any field element $\alpha \in \mathbb{F}$, $L(\alpha \mathbf{v}_d) = \alpha L(\mathbf{v}_d)$, and*
2. *For any $\mathbf{u}_d \in \mathbb{F}^n$ and $\mathbf{v}_d \in \mathbb{F}^n$, $L(\mathbf{u}_d + \mathbf{v}_d) = L(\mathbf{u}_d) + L(\mathbf{v}_d)$.*

We would omit the proof here, but it is straightforward to see that the total degree D of a linear arithmetic circuit is 1. Another way to characterize linear circuits is that for any multiplication gate in the circuit, at most one of the inputs can be non-constant values across the batch.

At the first sight, the power of linear arithmetic circuits seems limited. However, linear arithmetic circuits are quite common and important in practice. For example, the circuit for Fast Fourier Transform (FFT) is a linear circuit. In the context of batch evaluations, linear arithmetic circuits plays even bigger roles. For example, in many of the cutting edge deep learning models (e.g. ResNet [40], Transformer [63], Diffusion models [41]), between the non-linear transformation layers, there are many linear layers which essentially perform multiplication between a parameter matrix and an variable vector or matrix. Note that after the deep learning model is trained, for model inference the parameter matrices can be considered as constant matrices (when the model is also available to the verifier). Hence, for batch model inference with different inputs, these layers can be viewed as linear arithmetic circuits.

Interactive Proof. The interactive proof exploits the linear property of the computation. Following the ideas in the motivating example, first we derives the low degree extension of the input vectors using the Lagrange interpolation method. Treating the inputs of the instances $\{\mathbf{v}_d^{(1)}, \mathbf{v}_d^{(2)}, .., \mathbf{v}_d^{(m)}\}$ as a $(m -$

Algorithm 5 Interactive Proof Protocol for Batch Linear Circuit Evaluation

Participants: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Interative proof

- $\mathcal{P}$ and $\mathcal{V}$ calculate $\hat{\mathbf{v}}_d(r)$ and $\hat{\mathbf{v}}_0(r)$ separately using Barycentric evaluation
- $\mathcal{P}$ and $\mathcal{V}$ run the GKR protocol to prove that $\hat{\mathbf{v}}_0(r) = L(\hat{\mathbf{v}}_d(r))$

1)-degree vector-valued polynomial function evaluated at the instance indices $\{1, 2, ..., m\}$, this polynomial can be written explicitly as $\mathbf{v}_d(t) = \sum_{i=1}^{m} \delta_i(t)\mathbf{v}_d^{(i)}$, where $\delta_i(t)$ is the Lagrange basis as defined in Section 2.4. Note that for a fixed value of t, $\delta_i(t)$ is simply a constant. Using the two properties in Definition 12, it is straightforward to derive $\hat{\mathbf{v}}_0(t)$, the single variable polynomial extension of the output vector:

$$\hat{\mathbf{v}}_0(t) = L(\hat{\mathbf{v}}_d(t)) = L(\sum_{i=1}^{m} \delta_i(t)\mathbf{v}_d^{(i)}) = \sum_{i=1}^{m} \delta_i(t)L(\mathbf{v}_d^{(i)}) = \sum_{i=1}^{m} \delta_i(t)\mathbf{v}_0^{(i)} \quad (7)$$

The last step is because that $\mathbf{v}_0^{(i)}$ is simply the output vector of the i^{th} instance in the batch, and so $L(\mathbf{v}_d^{(i)}) = \mathbf{v}_0^{(i)}$ for $i = 1, 2, .., m$. This above equality holds for any t. Thus, the verifier can sample a public random value r, and calculate a "random input vector" $\hat{\mathbf{v}}_d(r)$ using the Barycentric evaluation method described in Section 2.5. The verifier also can computes the expected output vector $\hat{\mathbf{v}}_0(r)$ using the Barycentric evaluation formula from $\{\mathbf{v}_0^{(i)} \mid i = 1, 2, .., m\}$. Hence, the prover and the verifier can invoke the GKR protocol to check whether input vector $\hat{\mathbf{v}}_d(r)$ evaluates to $\hat{\mathbf{v}}_0(r)$. The entire protocol is outlined in Algorithm 5.

Correctness and Complexity of the Protocol. The correctness and complexity of the above protocol are formally stated in the following theorem, assuming addition and multiplication of two field elements both take $O(1)$ time. The proof for this theorem can be found in Appendix B.

Theorem 13. *The protocol in Algorithm 5 is complete and sound with soundness error* $\epsilon = O(\frac{(m|\mathbf{io}|+d\log|C|}{|\mathbb{F}|})$. *The prover and verifier time complexity are* $O(m|\mathbf{io}|+|C|)$ *and* $O(m|\mathbf{io}|+d\log|C|)$, *respectively. The proof size is* $O(d\log|C|)$.

Applications in Batch FFT Verification. The Fast Fourier Transformation is a widely-used algorithm with numerous practical applications, including in digital recording, sampling, additive synthesis, and pitch correction software. As a result, batch verification of FFT calculations is of significant importance in these fields.

The FFT algorithm can be viewed as an optimized implementation for matrix vector multiplication $FFT(v) = \mathbf{F}_l\mathbf{v}$, where $\mathbf{F}_l$ is a so-called Vandermonde matrix as shown below:

$$\mathbf{F}_l = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^1 & \omega^2 & \dots & \omega^{l-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(l-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{l-1} & \omega^{2(l-1)} & \dots & \omega^{(l-1)^2} \end{pmatrix} \tag{8}$$

In the matrix above, ω is the l^{th} root of the unity of finite field $\mathbb{F}$. Thus, for all l-sized vectors $\mathbf{v}$, $\mathbf{F}_l$ is a constant matrix. As a consequence, $FFT(v) = \mathbf{F}_l\mathbf{v}$ is a linear arithmetic circuit. This means we can apply either the non-interactive or interactive proof we developed to prove the correctness of FFT computation for a batch of l-sized vectors.

Theorem 14. *There exists an interactive batch FFT verification protocol that achieves the optimal prover and verifier running time, both of which are $O(ml)$.*

The proof for the above theorem can be found in Appendix B. It is worth pointing out that computing the FFT transformation for the entire batch takes $O(ml \log l)$ time. Hence, the time to generate the batch proof is actually lower than the computation itself.

5.2 Batch Proof for Sums of Higher Degree Polynomials

The techniques presented in the previous section only apply to linear circuits. For batch verification of non-linear circuits (i.e. total degree $D \geq 2$), the non-interactive and interactive protocols proposed in [61] and [66] work in general. However, we note that for circuits with special structures, the batch verification can be made more efficient and has other useful properties.

In this subsection, we consider a form of circuits which computes the sum of multiple multivariate polynomials, where the polynomials can have total degrees $D \geq 2$. Such a circuit has the characteristic that there is an output gate(s) summing up the outputs of all gates in the previous layer. The matrix multiplication circuit is an example which we will discuss in detail in the subsection. Another important category of application is the widely used MapReduce paradigm for distributed computing. The *map* step applies the same logic to many different pieces of data. And then in the *reduce* step, the results of the *map* step with the same key are aggregated. In many use cases, for example, the canonical "word counting" example, the *reduce* step simply adds up the *map* outputs. Such a map-reduce operation can be modeled using a circuit computing sum of multiple multivariate polynomials. We formulate the sum of polynomials verification problem as follows:

Definition 15. *Suppose $g(\mathbf{z})$ is a polynomial with total degree D, where $\mathbf{z} \in \mathbb{F}^n$. Formula $\sum_{k=1}^{l} g(\mathbf{z}_k)$ calculates the sum of this polynomial over a set of input vectors $Z = \{\mathbf{z}_k \in \mathbb{F}^n \mid k = 1, 2, .., l\}$. Given $y \in \mathbb{F}$, the sum of polynomials verification problem checks whether $y = \sum_{k=1}^{l} g(\mathbf{z}_k)$ holds.*

More concretely, in the MapReduce "word counting" example, $g(\cdot)$ implements the *map* operation with $\mathbf{z}_k$ being the k^{th} part of a document Z. The summation $\sum_{k=1}^{l} g(\mathbf{z}_k)$ corresponds to the *reduce* operation, which sums up the counts of each part to obtain the total word count of the document. Below we define the sum of polynomials *batch* verification problem, which captures the problem of verifying the correctness of invoking MapReduce on a batch of documents:

Definition 16. *Given a batch of inputs* $\{Z^{(1)}, Z^{(2)}, .., Z^{(t)}, .., Z^{(m)}\}$*, where* $Z^{(t)} = \{\mathbf{z}_k^{(t)} \in \mathbb{F}^n \mid k = 1, 2, .., l\}$*, and a batch of outputs* $\{y^{(1)}, y^{(2)}, .., y^{(t)}, .., y^{(m)}\}$*, the sum of polynomials batch verification problem checks if the following holds for all* $t = 1, 2, .., m$*:*

$$y^{(t)} = \sum_{k=1}^{l} g(\mathbf{z}_k^{(t)}) = \sum_{k \in \{0,1\}^{\log l}} g(\mathbf{z}^{(t)}(k)) \tag{9}$$

Note that Equation 9 redefines the index k so the summation is over a Boolean hypercube $\{0,1\}^{\log l}$, which makes it "Sumcheck-friendly". It also views $\mathbf{z}^{(t)}(k)$ as a function of k. Equation 9 is valid if the following holds, where $\tilde{\mathbf{z}}^{(t)}(k)$ is the multilinear extension of $\mathbf{z}^{(t)}(k)$.

$$y_t - \sum_{k \in \{0,1\}^{\log l}} g(\tilde{\mathbf{z}}^{(t)}(k)) = 0 \tag{10}$$

This is because the summation is over Boolean hypercube $k \in \{0,1\}^{\log l}$. As long as $\tilde{\mathbf{z}}^{(t)}(k)$ agrees with $\mathbf{z}^{(t)}(k)$ everywhere over the Boolean hypercube, the Summation result does not change.

Now, to verify Equality 10 for the entire batch in one shot, we need merge the entire batch into a single instance. To do this, we leverage a prior idea [2,10,19]. Consider:

$$Q(x) = l \cdot \sum_{t=1}^{m} (y^{(t)} - \sum_{k \in \{0,1\}^{\log l}} g(\tilde{\mathbf{z}}^{(t)}(k))) \cdot \tilde{eq}(x,t) \tag{11}$$

where $\tilde{eq}(x,t)$ is defined in Equation 3. We can claim that Equation 9 holds for all $t = 1, 2, .., m$ with high probabiliy if $Q(x)$ is a zero-polynomial, i.e. it evaluates to 0 for any $x \in \mathbb{F}$. Next, we rearrange the right hand side of Formula 11 as follows:

$$\begin{aligned} Q(x) &= \sum_{t=1}^{m} \sum_{k \in \{0,1\}^{\log l}} (y^{(t)} - l \cdot g(\tilde{\mathbf{z}}^{(t)}(k))) \cdot \tilde{eq}(x,t) \\ &= \sum_{k \in \{0,1\}^{\log l}} \sum_{t=1}^{m} (y^{(t)} - l \cdot g(\tilde{\mathbf{z}}^{(t)}(k))) \cdot \tilde{eq}(x,t) \\ &= \sum_{k \in \{0,1\}^{\log l}} f_x(k) \end{aligned}$$

where $f_x(k) = \sum_{t=1}^{m} (y^{(t)} - l \cdot g(\tilde{\mathbf{z}}^{(t)}(k)))$. Note that the second to the last step swaps the two summation signs. This can be derived using elementary algebra and we omit the details here. Now to prove that $Q(x)$ is a zero-polynomial, the verifier can just sample a random value $x_r \in \mathbb{F}$, send it to the prover, and run the Sumcheck protocol to prove that $Q(x_r) = 0$. In the last round of the Sumcheck protocol, the verifier has to compute $f_{x_r}(\cdot)$ at a random point k_r. To do this, the verifier needs to first evaluate $\tilde{\mathbf{z}}^{(t)}(k_r)$, the multilinear extension of the input at k_r. Since the verifier has all the input data locally, she can derive the multilinear extension and perform the evaluation.

Remark 17. Since Equation 11 has a summation form, one might attempt to run the Sumcheck protocol on Boolean hypercube $t \in \{0,1\}^{\log m}$. However, to do this the verifier need to derive the low-degree extension (*instead of* the multiliear extension) for the outputs $y^{(t)}$. This is because to execute the Sumcheck on $t \in \{0,1\}^{\log m}$, we need to view $g(\mathbf{z}^{(t)}(k))$ as a function of t. Since $g(\cdot)$ has degree $D > 1$, $y^{(t)}$ as a function of t must also have total degree D. Then, to derive the low-degree extension of $y^{(t)}$, at least $D \cdot m$ output values are needed. As the verifier has access to only m output values $\{y^{(1)}, y^{(2)}, .., y^{(m)}\}$, she cannot derive the low-degree extension herself, unless the prover send those to the verifier. However this increases the communication cost significantly. The "summation sign swap" trick eliminates t, and thus avoids this problem.

Theorem 18. *There exists an interactive protocol for the sum of polynomials batch verification problem which is complete and sound with soundness error* $\epsilon = O(\log ml/|\mathbb{F}|)$. *Moreover, the prover and verifier time complexity and* $O(lmT)$ *and* $O(D \log l + mT)$, *respectively, where* T *denotes the cost of one evaluation of* $g(\cdot)$. *The proof size is* $O(D \log l)$.

Remark 19. Please refer to Appendix B for the proof of the theorem. The above protocol for the sum of polynomial batch verification problem achieves a proof length independent of the batch size m. However, the verifier time increases linearly with m, although it is still smaller than executing the batch computation itself. In the next subsection, we will see that for an important special case, namely the batch matrix multiplication problem, the verifier time can be further optimized and reach its lower bound.

Applications in Batch Matrix Multiplication. As an application of the above framework, this subsection delves into a highly important and frequently employed special circuit, namely matrix multiplications whose total degree $D = 2$. The problem we consider in this sections is defined as follows.

Definition 20. *Given* m *triples of* $l \times l$ *matrices* $\{\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t \mid t = 1, 2, ..., m\}$, *the batch matrix multiplication verification problem checks if* $\mathbf{A}_t\mathbf{B}_t = \mathbf{C}_t$ *holds for all* $t = 1, 2, ..., m$.

Different than batch FFT verification where the FFT matrix is constant for all instances, here the two matrices $\mathbf{A}_t$ and $\mathbf{B}_t$ could both vary across the

batch. This makes the batch verification protocol more challenging to construct. Inspired by the fast single-instance matrix product verification protocol in [61], we let $\mathbf{A}_t(i,j)$ denote functions from $\{0,1\}^{\log l} \times \{0,1\}^{\log l} \to \mathbb{F}$ that map input (i,j) to the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}_t$. We similarly define functions $\mathbf{B}_t(i,j)$, and $\mathbf{C}_t(i,j)$. Also, let $\tilde{\mathbf{A}}_t$, $\tilde{\mathbf{B}}_t$, and $\tilde{\mathbf{C}}_t$ denote their multilinear extensions. We have the following lemma. The proof is provided in Appendix B.

Lemma 21. *For any $(i,j) \in \mathbb{F}^{\log l} \times \mathbb{F}^{\log l}$ and $t \in \{1,2,..,m\}$, it holds that*

$$\tilde{F}_t(i,j) = \sum_{k \in \{0,1\}^{\log l}} (\tilde{\mathbf{C}}_t(i,j) - l \cdot \tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j)) = 0 \tag{12}$$

Equation 12 indicates that we can apply the Sumcheck protocol to prove the correctness for any given instance. However, our goal is to verify the entire batch in a single shot. Similar to the linear circuits, to achieve this, we need to somehow reduce the batch into a single instance. To do this, we instead leverage the multilinear extension [2, 10, 19]. Consider:

$$Q(x,i,j) = \sum_{t \in \{0,1\}^{\log m}} \tilde{eq}(x,t) \cdot \tilde{F}_t(i,j) \tag{13}$$

Observe that $Q(x,i,j)$ is a multilinear polynomial such that $Q(x,i,j) = \tilde{F}_x(i,j)$ for any $x \in \{0,1\}^{\log m}$. Thus, $Q(x,i,j)$ is a zero-polynomial if and only if $\tilde{F}_t(i,j)$ evaluates to 0 at all points in the $2l$-dimensional Boolean hypercube (and hence *if and only if* all the m matrix multiplications are computed correctly). Therefore, to check if matrix multiplication results are correct for all instances in the batch, it is suffice to randomly sample $(x_r, i_r, j_r) \in \mathbb{F}^{\log m} \times \mathbb{F}^{\log l} \times \mathbb{F}^{\log l}$ and check if $Q(x_r, i_r, j_r) = 0$. This introduces a small soundness error, which can be quantified by the following lemma. The proof can be found in Appendix B:

Lemma 22. $\Pr[Q(x_r,i_r,j_r) = 0 \mid \exists i,j \in \{0,1\}^{\log l} \; s.t. \; \tilde{F}_t(i,j) \neq 0] \leq \frac{\log m + 2\log l}{|\mathbb{F}|}$

Next, to develop an interactive proof for $Q(x_r, i_r, j_r) = 0$, we expand Equation 13 and rearrange its terms using the "summation sign swap" trick like we did in the previous section:

$$\begin{aligned}
Q(x,i,j) &= \sum_{t \in \{0,1\}^{\log m}} \tilde{eq}(x,t) \cdot \sum_{k \in \{0,1\}^{\log l}} (\tilde{\mathbf{C}}_t(i,j) - l \cdot \tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j)) \\
&= \sum_{t \in \{0,1\}^{\log m}} \sum_{k \in \{0,1\}^{\log l}} \tilde{eq}(x,t) \cdot (\tilde{\mathbf{C}}_t(i,j) - l \cdot \tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j)) \\
&= \sum_{k \in \{0,1\}^{\log l}} \sum_{t \in \{0,1\}^{\log m}} \tilde{eq}(x,t) \cdot (\tilde{\mathbf{C}}_t(i,j) - l \cdot \tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j)) \\
&= \sum_{k \in \{0,1\}^{\log l}} f_{x,i,j}(k)
\end{aligned} \tag{14}$$

where $f_{x,i,j}(k) = \sum_{t\in\{0,1\}^{\log m}} \tilde{eq}(x,t)\cdot(\tilde{\mathbf{C}}_t(i,j) - l\cdot\tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j))$.

Now that we have expressed $Q(x_r,i_r,j_r) = \sum_{k\in\{0,1\}^{\log l}} f_{x_r,i_r,j_r}(k)$ which is a sum over a Boolean hypercube, we can employ the Sumcheck protocol on polynomial $f_{x_r,i_r,j_r}(k)$ to prove that $Q(x_r,i_r,j_r)=0$. In the final round of the Sumcheck protocol, the verifier is required to compute $f_{x_r,i_r,j_r}(k_r)$ for a random point $k_r \in \mathbb{F}^{\log l}$. The verifier can do this by evaluating $\tilde{eq}(x_r,t)$, $\tilde{\mathbf{C}}_t(i_r,j_r)$, $\tilde{\mathbf{A}}_t(i_r,k_r)$, and $\tilde{\mathbf{B}}_t(k_r,j_r)$ with a single streaming pass over the input [61] for $t=1,2,...,m$, and sum up the results.

Correctness and complexity of the protocol. We have the following results regarding the batch matrix multiplication problem. Please refer to Appendix B for the proof to the theorem.

Theorem 23. *There exists an interactive protocol for the batch matrix multiplication verification problem which is both complete and sound with soundness error $\epsilon = (\log m + 3\log l)/|\mathbb{F}|$. Moreover, the prover and verifier time complexity are both $O(m(l^2+\log m))$ with proof size being $O(\log l)$. For the typical case where $\log m < l^2$, the prover time and verifer time become $O(ml^2)$, both achieving their lower bound.*

Remark 24. We would like to point out that generating the proof has a lower time complexity than that of computing the products of m pairs of $l\times l$ matrices. Asymptotically, the fastest known algorithm to multiply two $l\times l$ matrices runs in time roughly $O(l^{2.37286})$ [1, 47]. Repeating this algorithm for m matrix pairs takes $O(m\cdot l^{2.37286})$ time, which is more expensive than the prover time. Thus, for large matrices, generating the batch proof only adds a small overhead.

Comparison with other batch verification approaches. We consider a few alternative approaches for batch matrix product verification:

The first approach is to repeat fast single-instance matrix product verification protocol [61] separately for all instances in the batch. This yields time complexity $O(ml^2)$ for both the prover and verifier, which is the same as ours for practical cases. However, the communication cost of this approach is $O(m\log l)$, which is worse than $O(\log l)$ in our protocol.

The second approach is to express multiplication of two matrices as a circuit, and apply the data parallel computation verification in [61] to process the batch. The time complexity of the prover is $O(m\cdot S + S\log S)$, where S is the size of the circuit for multiplying two matrices. As pointed out in Section 5.5.1 in [61], such a circuit could have $O(l^3)$ gates, which renders the prover complexity to be $O(m\cdot l^3 + l^3\log l^3) = O((m+\log l)\cdot l^3)$, which is worse than our approach.

As a third approach, the verifier can directly apply Freivalds' algorithm [28] to check whether $\mathbf{C}_t = \mathbf{A}_t\mathbf{B}_t$ for the m instances one-by-one. And advantage for this approach is that it is fully non-interactive, and hence the proof size is zero and no extra work from the prover is required. One downside is that Freivalds' algorithm requires $O(l+\log m)$ verifier memory, while our approach only needs $O(\log l + \log m)$ space by using single streaming pass over the input

when evaluating $f_{x_r,i_r,j_r}(k_r)$ [61]. Another major benefit of our protocol is that it has a variant which can be used as a primitive for verifying more complex computation. We will provide more details below.

Protocol variant as a subroutine for complex computations. Below we describe a variation of the protocol which inherits the advantage of the MAT-MULT protocol proposed in [61]. It adds slightly more communication cost compared to the above, but render the protocol more utilities, such as being used as a proving subroutine for more complex computations. A use case is for verifying batch matrix powerings $D_t = A_t^{2^c}$ for $t = 1, 2, .., m$. Matrix powerings have many important applications, such as calculating the diameter of graphs in batches.

$D_t = A_t^{2^c}$ can be computed by repeated squaring matrix A_t c times. Applying Freivalds' algorithm to this problem would require $O(cl^2)$ communication for each instance in the batch, as analyzed in [61]. This is because after each squaring, the prover must send the intermediate matrix to the verifier. Otherwise, the verifier would not have the data necessary to apply Freivalds' algorithm. Therefore for the entire batch, the verification communication cost is $O(cml^2)$.

To reduce the communication cost, we note that for the matrix powering problem, $f_{x_r,i_r,j_r}(k)$ can be written as:

$$f_{x_r,i_r,j_r}(k) = \sum_{t \in \{0,1\}^{\log m}} \tilde{eq}(x_r, t) \cdot (\tilde{\mathbf{A}}_t^{2^s}(i_r, j_r) - l \cdot \tilde{\mathbf{A}}_t^{2^{s-1}}(i_r, k)\tilde{\mathbf{A}}_t^{2^{s-1}}(k, j_r))$$

We can invoke the Sumcheck protocol for $s = c$ first. Notice that the verifier has $\tilde{\mathbf{A}}_t^{2^c}(i_r, j_r)$ available to her locally, since it is the result of matrix powering. If the prover sends her $\{\tilde{\mathbf{A}}_t^{2^{c-1}}(i_r, k_r), \tilde{\mathbf{A}}_t^{2^{c-1}}(k_r, j_r) \mid t = 1, 2, .., m\}$. The verifier can calculate $f_{x_r,i_r,j_r}(k)$ on her own for the last step in the Sumcheck protocol. Next, the verifier can the random linear combination trick to generate $\{\tilde{\mathbf{A}}_t^{2^{c-1}}(i_{r'}, j_{r'})\}$ from $\{\tilde{\mathbf{A}}_t^{2^{c-1}}(i_r, k_r), \tilde{\mathbf{A}}_t^{2^{c-1}}(k_r, j_r)\}$. To complete the verification for matrix powering, we repeat this process until $s = 1$. Note that this protocol does not change the prover complexity. It increases the communication complexity to $O(m \log l)$ for each Sumcheck. Thus, the overall communication complexity is $O(cm \log l)$, which is much better than that of the approach with Freivalds' algorithm.

6 Conclusions

In this paper, we systematically study the batch arithmetic circuit satisfiability and evaluation problem. We start with the observation that the circuit inputs/outputs can be represented as low-degree polynomials in terms of the instance index. Built on this idea, we construct various argument and proof protocols that can produce succinct proofs with short verification time for log-space uniform circuits. Future work includes turning the proposed argument schemes into zero-knowledge protocols, and further reducing the prover running time.

References

1. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Marx, D. (ed.) 32nd SODA. pp. 522–539. ACM-SIAM (Jan 2021)
2. Babai, L., Fortnow, L., Lund, C.: Nondeterministic exponential time has two-prover interactive protocols. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science. pp. 16–25 vol.1 (1990)
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993)
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer, Heidelberg (Aug 2019)
6. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (Aug 2013)
7. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (Oct / Nov 2016)
8. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (Aug 2014)
9. Berrut, J.P., Trefethen, L.N.: Barycentric lagrange interpolation. SIAM Review 46(3), 501–517 (2004), https://doi.org/10.1137/S0036144502417715
10. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. Cryptology ePrint Archive, Paper 2014/846 (2014), https://eprint.iacr.org/2014/846, https://eprint.iacr.org/2014/846
11. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 649–680. Springer, Heidelberg, Virtual Event (Aug 2021)
12. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 336–365. Springer, Heidelberg (Dec 2017)
13. Bootle, J., Groth, J.: Efficient batch zero-knowledge arguments for low degree polynomials. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 561–588. Springer, Heidelberg (Mar 2018)
14. Bootle, J., Lyubashevsky, V., Nguyen, N.K., Seiler, G.: A non-PCP approach to succinct quantum-safe zero-knowledge. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 441–469. Springer, Heidelberg (Aug 2020)
15. Brakerski, Z., Holmgren, J., Kalai, Y.T.: Non-interactive delegation and batch NP verification from standard computational assumptions. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC. pp. 474–482. ACM Press (Jun 2017)

16. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (oct 1988), https://doi.org/10.1016/0022-0000(88)90005-0
17. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 1–18. Springer, Heidelberg (Nov 2020)
18. Buterin, V.: An incomplete guide to rollups https://vitalik.ca/general/2021/01/05/rollup.html
19. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2075–2092. ACM Press (Nov 2019)
20. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D.: Fiat-Shamir from simpler assumptions. Cryptology ePrint Archive, Report 2018/1004 (2018), https://eprint.iacr.org/2018/1004
21. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Paper 2022/1355 (2022), https://eprint.iacr.org/2022/1355, https://eprint.iacr.org/2022/1355
22. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Heidelberg (May 2020)
23. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 394–423. Springer, Heidelberg, Virtual Event (Aug 2021)
24. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Goldwasser, S. (ed.) ITCS 2012. pp. 90–112. ACM (Jan 2012)
25. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (May 2000)
26. Devadas, L., Goyal, R., Kalai, Y., Vaikuntanathan, V.: Rate-1 non-interactive arguments for batch-np and applications. In: 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS). pp. 1057–1068 (2022)
27. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
28. Freivalds, R.: Fast probabilistic algorithms. In: Bečvář, J. (ed.) Mathematical Foundations of Computer Science 1979. pp. 57–69. Springer Berlin Heidelberg, Berlin, Heidelberg (1979)
29. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (Aug 1997)
30. Gabizon, A., Williamson, Z.J., Ciobotaru, O.M.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. 2019, 953 (2019)
31. Gailly, N., Maller, M., Nitulescu, A.: Snarkpack: Practical snark aggregation. In: Eyal, I., Garay, J. (eds.) Financial Cryptography and Data Security. pp. 203–229. Springer International Publishing, Cham (2022)

32. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008), https://www.sciencedirect.com/science/article/pii/S0166218X08000449, applications of Algebra to Cryptography
33. Garg, R., Sheridan, K., Waters, B., Wu, D.J.: Fully succinct batch arguments for np from indistinguishability obfuscation. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography. pp. 526–555. Springer Nature Switzerland, Cham (2022)
34. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. J. ACM 62(4) (sep 2015), https://doi.org/10.1145/2699436
35. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC. pp. 291–304. ACM Press (May 1985)
36. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016)
37. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Heidelberg (Aug 2018)
38. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 339–358. Springer, Heidelberg (May / Jun 2006)
39. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008)
40. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
41. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)
42. Kaslasi, I., Rothblum, G.N., Rothblum, R.D., Sealfon, A., Vasudevan, P.N.: Batch verification for statistical zero knowledge proofs. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 139–167. Springer, Heidelberg (Nov 2020)
43. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (Dec 2010)
44. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018)
45. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992)
46. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 359–388. Springer, Heidelberg (Aug 2022)
47. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. p. 296–303. ISSAC '14, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2608628.2608664

48. Libert, B., Ramanna, S.C., Yung, M.: Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) ICALP 2016. LIPIcs, vol. 55, pp. 30:1–30:14. Schloss Dagstuhl (Jul 2016)
49. Liu, T., Xie, X., Zhang, Y.: zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2968–2985. ACM Press (Nov 2021)
50. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM 39(4), 859–868 (oct 1992), https://doi.org/10.1145/146585.146605
51. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019)
52. Micali, S.: Computationally sound proofs. SIAM J. Comput. 30(4), 1253–1298 (oct 2000), https://doi.org/10.1137/S0097539795284959
53. Naor, M., Paneth, O., Rothblum, G.N.: Incrementally verifiable computation via incremental PCPs. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 552–576. Springer, Heidelberg (Dec 2019)
54. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (Mar 2013)
55. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press (May 2013)
56. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 49–62. ACM Press (Jun 2016)
57. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Efficient batch verification for up. In: Proceedings of the 33rd Computational Complexity Conference. CCC '18, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU (2018)
58. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM 27(4), 701–717 (oct 1980), https://doi.org/10.1145/322217.322225
59. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Heidelberg (Aug 2020)
60. Thaler, J.: Proofs, arguments, and zero-knowledge https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
61. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 71–89. Springer, Heidelberg (Aug 2013)
62. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (Mar 2008)
63. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 6000–6010. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)

64. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943. IEEE Computer Society Press (May 2018)
65. Waters, B., Wu, D.J.: Batch arguments for sfNP and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 433–463. Springer, Heidelberg (Aug 2022)
66. Williams, R.R.: Strong eth breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. CCC '16, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU (2016)
67. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 733–764. Springer, Heidelberg (Aug 2019)
68. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy. pp. 859–876. IEEE Computer Society Press (May 2020)
69. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy. pp. 863–880. IEEE Computer Society Press (May 2017)
70. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A zero-knowledge version of vsql. Cryptology ePrint Archive, Paper 2017/1146 (2017), `https://eprint.iacr.org/2017/1146`, `https://eprint.iacr.org/2017/1146`
71. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster verifiable RAM with program-independent preprocessing. In: 2018 IEEE Symposium on Security and Privacy. pp. 908–925. IEEE Computer Society Press (May 2018)
72. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. pp. 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg (1979)

A Proofs for Batch Circuit Satisfiability

In this section, we prove the proofs for several lemmas and theorem presented in Section 4.

Below we prove Lemma 7:

Proof. This follows directly from the Schwartz-Zippel Lemma [58,72]. Note that $(C(\mathbf{v}_d, \mathbf{w}) - \mathbf{v}_0) \cdot \mathbf{s}$ is a degree-$|\mathbf{v}_0|$ polynomial in terms of s defined over $\mathbb{F}$. Since s is uniform-randomly sampled, the probability that $(C(\mathbf{v}_d, \mathbf{w}) - \mathbf{v}_0) \cdot \mathbf{s}$ is not a zero-polynomial and yet $b = 0$ is $|\mathbf{v}_0|/|\mathbb{F}|$.

Below we prove Lemma 8:

Proof. We prove that the circuit transformation increases the size and depth of the circuit, but only up to a constant factor. First, the transformation adds the i^{th} output of C with $-\mathbf{v}_{0,i}$ and them multiply the sum with s^{i-1}. Next, the products are summed together. The first step essentially increases the circuit

size by $2|\mathbf{v}_0|$ and the depth by 2. For the second step, as discussed in [24, 61], the summation of $|\mathbf{v}_0|$ terms can be modeled with a *single* addition gate with multiple inputs. This adds one more gate and increases the depth by one.

Hence, $|C'| = |C| + 2|\mathbf{v}_0| + 1$. Since $|\mathbf{v}_0| \leq |C|$, we have $O(|C'|) = O(|C|)$. As for the circuit depth, according to the above analysis, overall the depth increases by just a constant after the transformation. Hence, $O(d') = O(d)$. Moreover, we note that vector $\mathbf{s}$ remains a constant across all the instances. Hence, the newly added multiplication gates do not change the total degree. Therefore, the total degree of C' is still D.

Before proving Lemma 9, we need the following two lemmas:

Lemma 25. *$\hat{\mathbf{v}}_d(t)$, $\hat{\mathbf{v}}_0(t)$, and $\hat{\mathbf{w}}(t)$ are vectors of polynomials. The degree of any element of these vectors (i.e. a polynomial) is at most $m-1$.*

Proof. This is because each element of these vectors (i.e. a polynomial) is obtained by interpolation across m instances. Note that in the transformed circuit C', the original circuit's output vector $\hat{\mathbf{v}}_0$ has been converted into inputs. Hence, we can use Lagrange interpolation to determine $\hat{\mathbf{v}}_0(t)$ for C'.

Lemma 26. *The degree of polynomial $\hat{b}(t)$ is at most mD.*

Proof. This is because for a multiplication gate, the degree of its output polynomial is the sum of the degrees of its two input polynomials. Moreover, Lemma 25 states that the degree of each input polynomial of the circuit C' is at most $m-1$. Thus, the degree of the output polynomial of the entire circuit is the bounded by the total degree of the circuit multiplied by $m-1$, which is $(m-1)D \leq mD$.

Below we prove Lemma 9:

Proof. Based on Lemma 26, the degree of the output polynomial for circuit C' is bounded by mD. Thus, we need to evaluate each input polynomial at at most mD points. To accomplish this, the prover first evaluates each of the input polynomials at $\{1, \omega, .., \omega^i, .., \omega^{mD-1}\}$ using the multi-point evaluation technique described in Section 2.5. This process has time complexity $O(poly(\lambda) \cdot mD \log(mD))$ for each input. Thus, the overall time complexity for deriving the point-value form for the inputs of C' is bounded by $O(poly(\lambda) \cdot mD \log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|))$.

Next, to derive the polynomial for the output $\hat{b}(t)$, we perform either point-value form multiplication or addition for each intermediate gate, which takes at most $O(poly(\lambda) \cdot mD)$ time for each gate. Hence, the complexity for deriving the polynomials layer by layer is $O(poly(\lambda) \cdot mD|C|)$. The overall time complexity, is therefore $O(poly(\lambda) \cdot mD(|C| + \log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|)))$.

Below we prove Theorem 10:

Proof. We provide the proof sketch which shows that the protocol for batch circuit satisfiability outlined in Algorithm 4 meets the above requirements.

Completeness. We will omit the details here, but it is straightforward to see that following the protocols described in Algorithm 2, 3, and 4, given the completeness of the Kate commitment and GKR protocol, for any batch B such that $C(\mathbf{v}_d^{(t)}, \mathbf{w}^{(t)}) = \mathbf{v}_0^{(t)}$ for $t = 1, 2, ...m$, the verifier accepts it with probability 1.
Knowledge-Soundness. Here we prove that assuming the knowledge-soundness of the Kate polynomial commitment scheme, the protocol outlined in Algorithm 4 is knowledge-sound. Suppose a (potentially malicious) PPT prover $\mathcal{P}^*$ is able to convince the verifier to accept an invalid batch B^*, i.e. there exists an instance $1 \leq t \leq m$ such that $C(\mathbf{v}_d^{(t)}, \mathbf{w}^{(t)}) \neq \mathbf{v}_0^{(t)}$. We just need to show that this happens with a negligible probability $negl(\lambda)$.

To prove this, we note that the extractability of the polynomial commitment scheme implies that $\mathcal{E}$ can efficiently extract a pre-image $b(t)$ of the commitment c sent by $\mathcal{P}^*$. Similarly, $\mathcal{E}$ can efficiently extract the pre-images $w_1(t)$, $w_2(t)$, .., $w_{|\mathbf{w}|}(t)$ of commitments k_1, k_2, .., $k_{|\mathbf{w}|}$ sent by $\mathcal{P}^*$. $\mathcal{E}$ then evaluates these polynomials $b(t)$, $w_1(t)$, $w_2(t)$, .., $w_{|\mathbf{w}|}(t)$ at $t = 1, 2, .., m$ to construct the witness vector $\mathbf{w}^{(t)}$ for each of the m instances for circuit C'.

Now, suppose at least one of the witness vectors are not correct, but the verifier still accepts the batch. This can happen either because of 1) degree-m polynomials $w_1(t)$, $w_2(t)$, .., $w_{|\mathbf{w}|}(t)$ are incorrect but happens to evaluate correctly at $t = r$, or 2) the circuit transformation from C to C' turns an unsatisfiable instance into a satisfiable one, or 3) the degree-mD output polynomial $b(t)$ is incorrect but evaluates correctly at $t = 1, 2, .., m, r$, or 4) due to the soundness error introduced by the GKR protocol.

By the Schwartz-Zippel Lemma, case 1) has a probability of $O(m|\mathbf{w}|/|\mathbb{F}|)$. Here it is worth pointing out that the protocol requires the prover to commit to the witness polynomials before the verifier sending the random challenge s to the prover. This is critical otherwise the prover can potentially come up with witnesses which are invalid for C but satisfy C'. By Lemma 7, the probability that case 2) happens is $O(m|\mathbf{v_0}|/|\mathbb{F}|) = O(m|\mathbf{io}|/|\mathbb{F}|)$. By the Schwartz-Zippel Lemma, case 3) has a probability of $O(mD/|\mathbb{F}|)$. Finally, the soundness error of the GKR protocol is $O(d \log |C|/|\mathbb{F}|)$ [67]. Hence, by the union bound, the probability that the verifier accepts the invalid batch B^* is at most $O(\frac{m(|\mathbf{io}|+|\mathbf{w}|+D)+d \log |C|}{|\mathbb{F}|}) = O(\frac{m(|\mathbf{io}|+|\mathbf{w}|+D)+d \log |C|}{2^\lambda})$, which is in $O(negl(\lambda))$.
Efficiency. We analyze the prover and verifier time complexity, as well as the argument size below.

Prover time complexity: In the pre-processing phase, the prover only needs to transform C to C'. By Lemma 8, the circuit transformation increases the circuit size and depth by at most a constant factor. Hence the transformation has time complexity $O(poly(\lambda) \cdot |C|)$. For online processing, in Step 1 of Algorithm 3, the prover needs to derive the witness polynomials of C' and commit to them, which has a time complexity of $O(poly(\lambda) \cdot m \log m|\mathbf{w}|)$. In Step 2, the prover needs to sets vector $|\mathbf{s}|$, which takes $O(|\mathbf{io}|)$ time. In addition, the prover derives $\hat{b}(t)$, the output polynomial of C' and commit to it. Deriving $\hat{b}(t)$ takes $O(poly(\lambda) \cdot mD(|C| + \log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|)))$ time as shown by Lemma 9. Generating the polynomial commitment for $\hat{b}(t)$ takes $O(poly(\lambda) \cdot mD \log(mD))$, since the degree

of $\hat{b}(t)$ is bounded by mD as shown by Lemma 26. In Step 3, the prover opens the output and witness polynomial commitments at r, which takes $O(poly(\lambda) \cdot mD\log(mD))$ and $O(poly(\lambda) \cdot m\log m|\mathbf{w}|)$ time, respectively. In Step 4, the prover evaluates the input polynomials, which takes $O(poly(\lambda) \cdot m|\mathbf{io}|)$ time. Finally, in Algorithm 4 the prover and verifier runs the GKR protocol, which takes $O(poly(\lambda) \cdot |C|)$ time on the prover side [67]. Note that $|\mathbf{io}| \leq |C|$ and $|\mathbf{w}| \leq |C|$. Summing up the run time of each step yield overall prover time complexity of $O(poly(\lambda) \cdot m(D|C| + D\log(mD) \cdot (|\mathbf{io}| + |\mathbf{w}|) + |\mathbf{w}|\log m))$.

Verifier time complexity: In the pre-processing phase, the verifier needs to transform C to C', and process C' in preparation for running the GKR protocol later. Both steps take $O(poly(\lambda) \cdot |C|)$ time [61], and hence the overall pre-processing time complexity for the verifier is $O(poly(\lambda) \cdot |C|)$. For online processing, in Step 1 of Algorithm 3, the verifier receives and process commitments $\mathbf{k}$ in $O(poly(\lambda) \cdot |\mathbf{w}|)$ time. Step 2 takes the verifier $O(poly(\lambda) \cdot |\mathbf{io}|)$ time. In Step 3, the verifier needs to verify the opening proofs for both the output polynomial and the witness polynomials, which takes $O(poly(\log|\mathbb{F}|)) = O(poly(\lambda))$ and $O(poly(\lambda) \cdot |\mathbf{w}|)$ time, respectively. In Step 4, the verifier evaluates the input polynomials, which takes $O(poly(\lambda) \cdot m|\mathbf{io}|)$ time. Finally, in Algorithm 4 the prover and verifier runs the GKR protocol, which takes $O(poly(\lambda) \cdot d\log|C|)$ time on the verifier side [61,67]. Summing up the run time of all steps, the overall verifier time complexity is $O(poly(\lambda) \cdot (m|\mathbf{io}| + |\mathbf{w}| + d\log|C|))$.

Argument size: The communication costs for each step of Algorithm 3 are summarized in the inline comments. Summing those up, the total communication cost of Algorithm 3 is $O(poly(\lambda) \cdot |\mathbf{w}|)$. The communication cost of the GKR protocol on the reduced instance is $O(poly(\lambda) \cdot d\log|C|)$. Therefore, the argument size, i.e. the overall communication cost is $O(poly(\lambda) \cdot (|\mathbf{w}| + d\log|C|))$.

B Proofs for Batch Circuit Evaluation

In this section, we prove the proofs for several lemmas and theorem presented in Section 5.

Below we prove Theorem 13:

Proof. The proof for completeness is straightforward so we omit the details here. For soundness, by the Schwartz-Zippel Lemma, reducing the input and output polynomials to a single point evaluation introduces a soundness error of $O(m|\mathbf{io}|/|\mathbb{F}|)$, since all the input and output polynomials have degree $m-1$. The soundness error of the GKR protocol is $O(d\log|C|/|\mathbb{F}|)$ [67]. Hence, by the union bound, the overall soundness error is at most $\epsilon = O(\frac{(m|\mathbf{io}|+d\log|C|}{|\mathbb{F}|})$.

For protocol efficiency analysis, we note that calculating $\hat{\mathbf{v}}_d(r)$ and $\hat{\mathbf{v}}_0(r)$ takes $O(m|\mathbf{io}|)$ using the Barycentric evaluation method without communication. For the GKR protocol, the prover complexity can be made to be $O(|C|)$, and the verifier complexity is $O(d\log|C|)$, while the proof size is $O(d\log|C|)$ [60,67]. Hence the overall prover time, verifier time, and proof size are $O(m|\mathbf{io}| + |C|)$, $O(m|\mathbf{io}| + d\log|C|)$, and $O(d\log|C|)$, respectively.

Below we prove Theorem 14:

Proof. The FFT computation can be implemented with an arithmetic circuit with $O(l \log l)$ gates with $O(l)$ inputs/outputs [68]. Hence, directly applying the protocol in Algorithm 5 to batch FFT evaluation results in $O(ml + l \log l)$ prover time complexity. In a recent work [49], Liu et al. proposed a new Sumcheck protocol which achieves $O(l)$ prover time complexity for a single FFT instance verification. Since the last step of our batch interactive proof is proving the correctness of a single instance derived from random value r, We can simply borrow this technique, which reduces the overall prover time complexity to $O(ml)$. Note that the lower bound for the prover time is $O(ml)$, since the prover has to at least read all the m input vectors each having length l. Using a similar line of reasoning, we can show that the verifer time is achieve its lower bound $O(ml)$. Thus, this version of the protocol is optimal in terms of both prover and verifier time complexity.

Below we prove Theorem 18:

Proof. The completeness proof is straightforward and omitted here. For soundness, the Sumcheck protocol introduces soundness error $O(\log l/|\mathbb{F}|)$. Moreover since $Q(x)$ is a multilinear function of the bits of x with total degree $\log m$, $Q(x_r) = 0$ guarantees that $Q(x)$ is a zero-polynomial up to soundness error $O(\log m/|\mathbb{F}|)$. Hence, by union bound the overall soundness error is $O(\log ml/|\mathbb{F}|)$.

For efficiency analysis, first note that $f_x(k) = \sum_{t=1}^{m} (y^{(t)} - l \cdot g(\tilde{\mathbf{z}}^{(t)}(k)))$ is a multivariable polynomial has degree at most D for the bits of k. This is because $\tilde{\mathbf{z}}^{(t)}(k)$ is multilinear with $\log l$ variables and $g(\cdot)$ has degree D. Hence, in each round of the Sumcheck protocol, the communication cost is $O(D)$. The Sumcheck protocol runs in $O(\log l)$ rounds. Hence, the total proof size is $O(D \log l)$. The verifier time is $O(D \log l + mT)$, and the prover time is $O(lmT)$ [60].

Below we prove Lemma 21:

Proof. From Lemma 5 in [61], for any $t \in \{1, 2, .., m\}$, we have

$$\tilde{\mathbf{C}}_t(i,j) - \sum_{k \in \{0,1\}^{\log l}} \tilde{\mathbf{A}}_t(i,k)\tilde{\mathbf{B}}_t(k,j) = 0$$

Simply multiplying the equation by $l \in \mathbb{F}$, and separating $l \cdot \tilde{\mathbf{C}}_t(i,j)$ into l terms and then putting them into the summation, we would arrive at Equation 12.

Below we prove Lemma 22:

Proof. If there exists $i, j \in \{0,1\}^{\log l}$ such that $\tilde{F}_t(i,j) \neq 0$, then $Q(x,i,j)$ is not a zero polynomial. Since the total degree of $Q(x,i,j)$ is at most $\log m + 2 \log l$ in terms of the bits of x, i, and j, by the Schwartz-Zippel Lemma, $Q(x,i,j) = 0$ for at most $(\log m + 2 \log l)/|\mathbb{F}|$ fraction of (x,i,j) points in the domain of $Q(\cdot)$.

Below we prove Theorem 23:

Proof. Again the proof for completeness is trivial and omitted here. For soundness, Lemma 22 states that $Q(x_r, i_r, j_r) = 0$ guarantees $Q(x, i, j)$ is a zero-polynomial with soundness error at most $(\log m + 2\log l)/|\mathbb{F}|$. The Sumcheck protocol also introduces a soundness error of $O(\log l/|\mathbb{F}|)$. Hence, by the union bound the overall soundness error can be bounded by $\epsilon = (\log m + 3\log l)/|\mathbb{F}|$.

Below we analyze the proof size and the time complexity.

First, notice that $f_{x_r,i_r,j_r}(k)$ is a multivariate quadratic polynomial (i.e. $D = 2$) in terms $\{k_b\}$, the bits of the binary representation of k. This means that in each round of the Sumcheck protocol, a constant number of field elements are sent between the prover and the verifier. Hence, the overall proof size is $O(\log l)$, which is independent of the batch size.

Second, the prover time can be made into $O(ml^2)$ across all rounds of the Sumcheck protocol adopting the techniques in [61] to quickly evaluate $\tilde{\mathbf{A}}_t(i_r, k_r)$ and $\tilde{\mathbf{B}}_t(k_r, j_r)$ at all the necessary points. Moreover, note that $\tilde{eq}(x_r, t)$ and $\tilde{\mathbf{C}}_t(i_r, j_r)$ are independent of k_r. Thus, the prover can simply calculate their values before executing the Sumcheck protocol, which takes $O(m \log m)$ and $O(ml^2)$, respectively. Therefore, the overall prover time complexity is $O(m(l^2 + \log m))$. This simplifies to $O(ml^2)$ when $l^2 > \log m$, which is usually the case.

Third, the verifier needs to compute $f_{x_r,i_r,j_r}(k_r)$ in the final round of the Sumcheck protocol. Similar to the above, this can be done in $O(m(l^2 + \log m))$ time,The verifier also needs to process $O(\log l)$ constant size messages during the Sumcheck protocol as discussed above, which takes $O(\log l)$ time. Hence, the overall verifier time complexity is $O(m(l^2 + \log m) + \log l) = O(m(l^2 + \log m))$, which degenerates to $O(ml^2)$ for the majority of the cases.

Finally, we note that $O(ml^2)$ is the lower bound for both the prover and verifier time, since they both at least have to read the matrices themselves, which takes $O(ml^2)$ time. Thus, both the prover and verifier time complexy reaches their lower bound for the typical case where $\log m < l^2$.