The Cost of Intelligence: Proving Machine Learning Inference with Zero-Knowledge

Modulus Labs

January 30, 2023

We thank Zhang Zhenfei (Ethereum Foundation) and Riad Wahby (Carnegie Mellon University) for their diligent guidance and kind mentorship, as well as the Ethereum Foundation for their generous support.

For their reviews and insightful comments, we would like to additionally thank:
Jens Groth, Daniel Lubarov, Tianyi Liu, Max Gillett, Michele Orrú, Yi Sun, Luke Pearson, Nima Vaziri, Jonatan Luther-Bergquist, Remco Bloemen, dcbuilder, Brandon Da Silva, guiltygyoza, and Rand Hindi

To showcase the performance of the various proof systems discussed in this paper, we worked with many of the original authors to optimize performance in our benchmarking. Any remaining mistakes or oversights are entirely our own. If you find any errors, please do not hesitate to contact us. We will make revisions as fast as we can validate them and update the paper accordingly. Please mind the version number.

1 Introduction

1.1 Paper Motivation and High Level Summary

The emergence of blockchain systems has quickly enabled entirely new ways of expressing, transforming, and securing networked data. This is especially true for Ethereum, where growing use-cases in asset tokenization, decentralized finance, and other distributed applications has only compounded in both bandwidth demand and compute complexity.

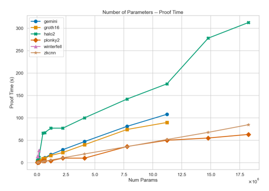
And yet, the same underlying security architecture which has heralded this new age of robust, verifiable computation is also responsible for Ethereum's infamously high utilization costs. In particular, high fees have translated to a total absence of intelligent features for on-chain services: from a lack of recommender systems personalizing the latest dApps to the notable shortage of matching algorithms operating over user embeddings – there simply exists no intelligent processing layer for web3. As Vitalik Buterin explains: "running deep learning inside the EVM is probably not the solution."

Thankfully, so-called scaling solutions have sprung into existence – Zero-Knowledge Proofs (ZK or ZKPs) in particular have become an increasingly dominant technique behind popular rollups. And unlike other systems, ZK-based constructions directly inherit the security standards of the verifying chain. Already, DeFi and NFT services have shifted to the rollup paradigm, enjoying far lower gas fees and an established security standard, inviting the big question: if the ZK-rollup paradigm is poised to solve general compute costs for Ethereum broadly, **can it also bring artificial intelligence inference to the decentralized internet?**

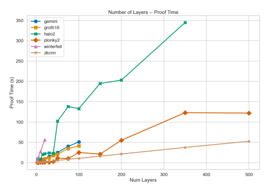
To that end, this technical whitepaper tackles this question by benchmarking two common suites of multi-layer perceptrons (MLPs) across a set of zero-knowledge proof systems. These included architectures that scaled with increasing parameter count (up to 18M parameters and 22B mult-adds), as well as architectures that scaled with an increasing number of layers (up to 500 layers). On the zero-knowledge prover side, we test **Groth16, Gemini, Winterfell (via Cairo VM implementation), Halo2, Plonky2, and zkCNN**. In particular, we showcase comparisons of proof time and memory consumption between the aforementioned proof systems, examining bottlenecks as each system scales with increasingly large and deep MLPs.

In so doing, it becomes clear that with respect to proving time, Plonky2 is by far the most performant system thanks to its use of FRI-based polynomial commitments and the Goldilocks field. In fact, for our largest benchmarked architectures it is *5 times* faster than Halo2, another popular general ZK proof system. This, however, comes at the notable cost of prover memory consumption, where Plonky2 consistently performs worse, at times doubling Halo2's peak RAM usage. With respect to both proving time *and* memory, the GKR-based zkCNN prover appears best suited to tackle large models – even without an optimized implementation.

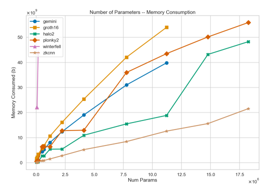
Taken together, we end the paper by examining two *real world* use-cases for verifiable inference that exemplify distinct axes of scale: model sophistication in the case of Worldcoin's iris verification model, and cost-effective proof generation in the case of AI Arena's gameplay verification. These two applications, along with emergent use-cases involving ambitious AI algorithms, motivate our next work in a ZK proving systems tailor-made for highly structured neural network operations — the *Remainder* proof system.



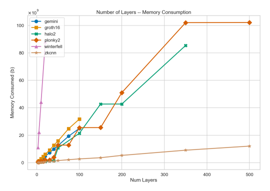
(a) Params Suite Timing



(b) Deep Benchmark Timing



(c) Params Suite Memory



(d) Deep Benchmark Memory

1.2 Paper Outline

We begin with a short outline of the key sections of the paper.

1. **Introduction:** Blockchains today sacrifice efficiency for uncompromising security. This has enabled new paradigms around finance, digital assets, and distributed applications. And yet, because of the sizable compute barrier, there exists no intelligent processing on-chain.

   Thankfully, rollup solutions are already using zero-knowledge proofs to massively scale blockchains without compromising on security, highlighting the driving question of this paper: ***Can Zero-Knowledge Proofs also bring AI algorithms to the blockchain?***

2. **Related Works:** In section 2, we conduct a literature review of the existing landscape of academic work, with a focus on papers concerned with the overlap between zero-knowledge proofs and machine learning inference.

3. **Benchmarking Methodology:** In section 3, we describe our benchmark design and metrics, as well as the benchmarking suite, giving a quick primer on each benchmarked proof system. This includes a discussion of the benchmarked MLP architectures, as well as how and why the tested proof systems were selected, including Groth16, Gemini, Winterfell, Halo2, Plonky2, and zkCNN.

4. **Results:** In section 4, we report concrete results on how the various proof systems perform across proof time and memory consumption, and show comparisons between proof systems.

5. **Future Directions:** In section 5, we tie the prover performance results to several emergent production-level applications, and end by discussing work from peer-reviewed academic publications which may be relevant to supporting a future proving system fine-tuned for verifiable machine learning computation.

In the next two sections, we give an short overview of zk-SNARKs and AI.

1.3 Primer on zk-SNARKs

In this section, we discuss zk-SNARKs and SNARKs. We start by recalling the relevant notion of zero-knowledge proofs.

Zero-Knowledge Proof systems (ZKPs) were introduced in the seminal work of Goldwasser, Micali and Rackoff [GMR89] and allow a prover to convince a verifier that a statement is true without revealing anything other than its validity. A line of work within zero-knowledge proof systems focuses on constructing *succinct* and *efficiently* verifiable proofs under the notion of *computational soundness* and in particular on constructing *zero-knowledge Succinct Non-interactive ARguments of Knowledge* (zk-SNARKs). Specifically, computational soundness guarantees that under cryptographic assumptions a (computationally bounded) cheating prover cannot convince the verifier about the validity of a false statement except with negligible probability. More precisely, the zk-SNARK *knowledge* property is a strengthening of computational soundness. It implies that whenever a (computationally bounded) prover convinces the verifier about a statement, then this prover must possess a valid witness for that same statement which can be *efficiently* extracted from the prover, instead of proving only the existence of such a witness that can be trivial in some cases. In summary, zk-SNARKs allow a prover to generate a very short proof about the validity of a statement without leaking any secret information, which amounts to large savings in communication cost and verification time as required for large scaling. Note that a SNARK defined without the zero-knowledge property is an equally useful primitive with the same benefits, for use cases where privacy is not a concern.

In recent years, emerging technologies such as blockchain and cloud services have used zk-SNARKs to efficiently verify the execution of complicated computations. For example, smart contracts rely on zk-SNARKs to efficiently verify the validity of a transaction, based on a proof that is generated by the prover using a large number of cryptographic operations. This notion of *Verifiable Computation (VC)* (e.g. see [WB15]) allows a weak verifier with limited computing resources to outsource a large computation to a powerful prover, which uses a zk-SNARK to create a short certificate. The certificate then convinces the verifier of the validity of its execution in time much smaller than running the actual computation, while simultaneously maintaining privacy. Note that we refer only to zk-SNARKs with proofs that can be verified by anyone (i.e. public verification) without requiring the verifier to use any secret information for this task (e.g. a secret key shared between the prover and verifier).

1.4 Primer on AI

Artificial Intelligence (AI) has ballooned into many distinct subfields since its conception – from game-playing to search, Bayesian belief networks to genetic algorithms, the term itself now applies to all these and more.

The focus of this whitepaper, however, is on *modeling algorithms* – algorithms which take on a fixed structure yet have indeterminate parameters, learned via feeding large amounts of example data to the model. Traditionally, such systems are comprised of three major components – the training data $\{(x, y)\}$, the model architecture f, and the model parameters θ. Additionally, model production and deployment happens in two phases, i.e., the *training phase* and the *inference phase*.

- During the training phase, the model is fed data from the training set and uses an update algorithm to change its parameters, attempting to minimize a "loss" function inversely correlated with the model's performance on the training set.
- Afterwards, during the inference phase, the model's weights are fixed, and given a new input x the model outputs prediction $\hat{y} = f(x; \theta)$ where θ are the post-trained weights.

In this whitepaper, we focus strictly on the inference phase, i.e., simply computing the model function f over an input x, given pre-trained model weights θ. Note that additionally, we focus solely on MLPs (multi-layer perceptrons [Ros57]) as an example candidate for modeling algorithms, although this is in no way exhaustive – the class of modeling algorithms is indeed quite large, encapsulating both supervised and unsupervised learning techniques, generative models [GPAM$^+$14], classical machine learning (ML) techniques such as decision trees [Qui86], support vector machines (SVMs) [BGV92], etc., and modern deep learning (DL) techniques such as convolutional neural networks [LBBH98], language models [DCLT18], deep reinforcement learning [MKS$^+$13], etc.

2 Related Academic Works

We give an overview of the relevant literature for verifiable inference. While schemes with properties of zk-SNARKs date back to the seminal works of [Kil92, Mic00] proven secure in the Random Oracle Model (ROM) and concretely instantiated using a hash function, the schemes below mainly use zk-SNARKs from the Halo2 paper [BGH19, hal], the Groth16 paper [Gro16], and in some cases combine this with the sumcheck protocol [LFKN92] or the GKR protocol [GKR15].

Roughly speaking, previous work can be categorized as follows:

2.1 Schemes based on Groth16

The schemes below rely on Groth16 and use various implementation of it for their benchmarks.

- ZEN [FQZ+21] constructs a compiler from a PyTorch model to R1CS, which is an intermediate representation (IR) that is supported by a large number of zk-SNARK systems. Moreover, the authors introduce various R1CS optimizations targeted for machine learning operations, such as R1CS-friendly quantization, resulting in very small accuracy losses compared to the model's accuracy without the use of cryptography.
- vCNN [LKKO20] uses a combination of zk-SNARK techniques for different parts of the neural network, resulting in improved overall efficiency. In particular, two different intermediate representations are used: the authors use Quadratic Arithmetic Programs (QAP) [GGPR13, PHGR13] for ReLU and pooling layers, and Quadratic Polynomial Programs (QPP) [KPP+14] for convolution layers. At a high level, the QPP allow them to translate the computation of convolution, which accounts for a large part of the overall computation, into polynomial multiplication – an operation better suited for zk-SNARK systems. Finally, the authors use techniques from the Commit-and-Prove zk-SNARK approach from [CFQ19] to ensure consistency between the proofs generated using the two different representations.
- VeriML [ZWW+21] uses zk-SNARKs to construct a system for verifiable computation with respect to outsourcing the training component of a model rather than inference. Additionally, the paper provides a mechanism for purchasing the trained model, subject to the constraint that the proof of training must be verified first.

2.2 Schemes based on interactive proofs

An interactive proof system [GMR89, BM88] is a protocol which allows a prover to convince a verifier about the validity of a statement given multiple rounds of communication between the two. Just as with conventional non-interactive proofs, completeness and soundness must both hold. Additionally, if the prover and verifier meet certain efficiency requirements, the system is called doubly-efficient. This was introduced in [GKR08, GKR15], henceforth GKR, and later improved in a series of works [CMT12, Tha13, WJB+17, ZGK+18a, XZZ+19]. A GKR scheme can be made zero-knowledge by using a commitment scheme [WTS+18a, ZGK+17], and furthermore can be made non-interactive and into a zk-SNARK using the Fiat-Shamir paradigm [FS87]. The schemes below rely on the GKR protocol and implement it for their benchmarks.

- SafetyNets [GGG17] follows a GKR approach and constructs a doubly-efficient interactive protocol for small depth convolution neural networks. The authors rely on existing interactive protocols for proving matrix multiplication, which is used for both fully connected and convolution layers. Additionally, they replace common activation functions as ReLU and sigmoid with ones much friendlier to GKR-style circuits such as sumpool, and show that networks using such can be trained to comparable accuracy while seeing remarkable speedups in prover time.

2.3 Schemes based on composition, and other related work

- [KMC18] follows a hybrid approach. They rely on sumcheck [LFKN92] techniques to construct proofs about matrix multiplication for fully connected (FC) layers, and the final verifier is comprised of a QAP that implements both the sumcheck verification and other parts of the neural network computation, which are verified using a Groth16 zk-SNARK.
- pvCNN [WWT+22] uses a combination of zk-SNARK techniques to obtain a proof system that works in a collaborative setting where several parties are involved in the process. Anyone can encrypt their input data using a Fully Homomorphic Encryption (FHE) scheme and submit this ciphertext to the public cloud. The authors rely on FHE for privacy of specific parts of the neural network and the input data. Once the FHE part is completed, they use a re-encryption technique which allows a different party holding a different key than the FHE one to continue with the rest of the neural network computation. For proofs that are constructed in these steps and a

few more listed in the paper, the authors use an approach as in [FNP20] for the integrity of the FHE computation, Groth16 with aggregation [GMN21], and a Commit-and-Prove approach as in LegoSNARK [CFQ19] for verifying the final proof.

- [ZFZS20] constructs a proof system for pre-trained Decision Trees and variations, such as random forests and regression, which are used for prediction and accuracy tests. For zero-knowledgeness and succinctness, the authors rely on the Aurora ZKP [BCR+19] and in addition use techniques from Authenticated Data Structures [Tam03].
- [KHSS22] encodes the operations of a MobileNetV2 [SHZ+18] in the Halo2 proof system and measures proof preprocessing and generation time. The work contributes ideas to efficient model quantization within the Plonkish circuit model, and discusses applications of SNARKing inference. For example, proving model accuracy and performing semantic retrieval using a combination of ZK and economic arguments.

While previous papers have focused on constructing verifiable ML systems using various ZKP schemes, this work is the first paper, to the best of our knowledge, which directly compares a variety of ZKP systems benchmarked on the same circuit architectures and performs a detailed analysis on the performance of each.

3 Benchmarking Methodology

We benchmark six ZK proof systems to perform a comparison with respect to proof generation time and prover maximum memory usage. This was conducted on roughly equal terms, using two consistent benchmark suites. Concretely, we create circuits representing MLPs (multi-layer perceptrons) within each proof system and run the prover, measuring specifically the *wall clock time* of creating a proof of inference, as well as the maximum prover *memory consumption* during the proof generation process. Note that we omitt comparing the verifier runtime and generated proof size, as the question of feasibility lies heavily on the prover side – indeed, this is an excellent starting point for future work – and is thus beyond the scope of this whitepaper. Moreover, note that all measurements are performed with respect to proof generation time, and do not take into account pre-processing or witness generation steps.

Though by no means exhaustive nor fully representative of most existing techniques, we chose a set of proof systems that represent a diverse variety of popular arithmetization schemes. To preview: with respect to R1CS-based proof systems, we benchmark Groth16 and Gemini – the former for its small proof sizes, and the latter for its ability to handle extremely large circuits in its "space-efficient" mode and otherwise for its linear time prover in its "time-efficient" mode; with respect to STARK-based systems, we benchmark Winterfell through Cairo/Giza, for its standing as a robust open-source STARK VM-based prover and implementation of an imperative programming language within a proving system; with respect to Plonkish proof systems, we benchmark Halo2 and Plonky2 – the former for its sophisticated developer tooling and flexibility, and the latter for its application of FRI to Plonkish constraints; finally, with respect to GKR-based systems, we benchmark zkCNN, an efficient proof implementation using novel techniques that handle various circuit-unfriendly neural net operations.

We first discuss the benchmark suites (i.e., MLP architectures) tested, including number of parameters, FLOPs, and number of layers, then introduce each proof system tested and our methodology for encoding MLPs within each.

3.1 ML Benchmark Suites

All of our benchmark architectures are MLPs, and consist of four primary operations: **linear weights** – matrix-vector multiplication, **linear biases** – element-wise addition, **ReLU activation function** – element-wise application of the function $max(0, x)$, and **scaling** – element-wise division.

Note that architectures are described in terms of layers, and that each layer sans the final one consists of a linear weight, a linear bias, a ReLU application and a final scaling step. We generate two distinct "suites" of benchmark architectures, described below:

- **Parameter count benchmarks.** These architectures were designed with a constant (40) number of layers each, but with an increasing number of parameters in each model, and were designed to test the scaling of proof systems with respect to raw parameter count and FLOPs. The number of parameters in these architectures ranged from roughly 500k to just over 18M. The number of FLOPs, on the other hand, ranged from 275M to 22B, with the latter eclipsing the number of add-mults for both VGG-16 [SZ14] and ResNet-50 [HZRS15] on an ImageNet-sized [DDS$^+$09] input. Note that although we chose not to test more complex architectures (e.g., convolutional networks) or activation functions (e.g., sigmoid), both the parameter and FLOP count of the models we benchmark are comparable to those of performant convolutional architectures, as cited above.
- **Depth benchmarks.** These architectures were designed to test the scaling of various proof systems with respect to number of model layers. The actual architectures are uniform in layer size (200 x 100 FC layers), and range from 10 layers to 500 layers – see complete description in appendix A. Again, we note that the deeper end of this suite has a comparable number of layers to performant convolutional and even transformer architectures, and is thus roughly representative of prover scaling for such models.

We now give a short description of each proof system, and outline the methodology of applying each system to an MLP below:

3.2 R1CS Proof Systems

We start by reviewing the *Rank-1 Constraint Satisfaction (R1CS)* problem, which underlies many proof systems in terms of arithmetization, including Groth16 and Gemini that we discuss below. The R1CS problem was implicitly defined in [GGPR13] where it was shown to be NP-Complete via a reduction from the Circuit Satisfiability, but it was explicitly defined in [SBV$^+$13, BCG$^+$13, BCR$^+$19] and it allows to convert statements about arithmetic circuits to those about products between matrices and vectors.

The R1CS problem is defined over a finite field $\mathbb{F}$ where we are given matrices $A, B, C \in \mathbb{F}^{N \times N}$ for an appropriate dimension N and ask whether there exists a vector $(1, z)$ such that $Az \circ Bz = Cz$. The matrices A, B, C describe a statement for which a proof is generated and are reusable across many executions, and the witness is contained in the vector $(1, z)$. Roughly speaking, A describes left inputs, B describes right inputs, and C describes outputs to binary gates within the circuit, while z is all of the assigned values. To illustrate the use of R1CS in proof systems, assume an arithmetic circuit C defined over a finite field $\mathbb{F}$ using addition and multiplication gates. The circuit $\mathcal{C}$ takes a public input x and a private witness w and outputs a value y, that is $\mathcal{C}(x, w) = y$. In terms of R1CS, given a tuple $(\mathcal{C}, x, w, y)$ there exists a transformation that produces matrices $A, B, C \in \mathbb{F}^{N \times N}$ of appropriate dimension N and a vector $(1, z)$ that depend exactly on $(\mathcal{C}, x, w, y)$ and satisfy $Az \circ Bz = Cz$ where "$\circ$" denotes element-wise product. In addition, this transformation does not incur any essential overhead when it comes to N and the number of non-zero elements of A, B, C when compared to the size of the circuit $\mathcal{C}$. This allows us to convert a claim for an arithmetic circuit of the form *"given $\mathcal{C}(x, \cdot)$ and y, does the prover know a w such that $\mathcal{C}(x, w) = y$?"* to an equivalent R1CS claim *"given matrices A, B, C, does the prover know a vector $(1, z)$ such that $Az \circ Bz = Cz$?"* which is easier to work with in proof systems. For example, if $\mathcal{C}(x, \cdot)$ is the circuit of a collision-resistant hash function h and y is a value in the range of h, we can create an R1CS statement that can certify the prover's *knowledge* of a w such that $h(w) = y$.

With respect to benchmarks, we use the Circom language [Ide21] to generate R1CS constraints. Specifically, we express our MLPs in Circom and compile the Circom files into R1CS descriptor files and WebAssembly executables, with the latter generating the witness values. Moreover, since R1CS requires gates with fan-in 2, our MLP operations require a slightly different circuit design when compared to the Plonkish and GKR circuits used elsewhere:

- **Linear layer.** For matrix-vector multiplication such as $W \cdot x$, we implement a naive row-wise dot product in Circom, multiplying the i-th element of each matrix row with the i-th element of the input vector. Since all gates are fan-in 2, we use accumulators to roll up the dot product running total: $\mathsf{acc}_{i+1} = w_i \cdot x_i + \mathsf{acc}_i$.
- **ReLU.** Recall that the function is defined as $\mathsf{ReLU}(x) = \mathtt{max}(0, x)$. To implement this, we need a test for negative values, and we use the Circom standard library's helper function that compares a given value to $p/2$ through bit decomposition, where p is the prime modulus of the finite field $\mathbb{F}$ and $p/2$ represents the upper bound of any positive value (i.e. the range $p/2 < x < p$ represents negative values). Given this test, we force the output to be zero if the input was negative, and otherwise leave it be identical to the input value.
- **Fixed-point division.** One of the primary issues in writing machine learning models as arithmetic circuits in a cryptographic setting is the fundamental incompatibility of traditionally floating point numbers within a vanilla ML model versus the need for an arithmetic circuit to be defined within a finite field. To combat this issue, we scale the model weights and inputs by a factor of s each, and the bias values by a factor of s^2 each, and after performing a typical linear layer computation $Wx + b$, we "un-scale" the output by a factor of s, i.e. we divide by s. For the division operation we use bit decomposition, truncating the k least significant bits to effectively divide by $s = 2^k$.

3.2.1 Groth16

Early approaches to nearly-practical SNARK constructions as Pinocchio [GGPR13] take advantage of the implicit reformulation of R1CS problems as quadratic arithmetic programs (QAPs) and pairing-based cryptography, for succinctness. At a high level, a QAP encodes the constraints in an arithmetic circuit, using the following polynomial form

$$\left(\sum_{i=0}^{m} c_i u_i(x)\right)\left(\sum_{i=0}^{m} c_i v_i(x)\right) = \left(\sum_{i=0}^{m} c_i w_i(x)\right) + h(x)t(x) \tag{1}$$

In the above expression, m is the number of wires, including inputs, outputs, and intermediate values. The intuition here is that coefficients c_i encode circuit values, while u_i, v_i, w_i act as selectors, with e.g. $u_i(r_g) = 1$ if and only if the i-th wire is the left input to the g-th gate and 0 otherwise. Additionally, define $t(x) = \prod_g (x - r_g)$ to be the "divisor polynomial" such that $t(x)$'s roots are precisely the gate indices r_g. The Groth16 construction in essence enforces two key points – firstly, that the above equation (1) holds, and secondly, that $t(x)$ is indeed a polynomial divisor of the polynomial

$$\left(\sum_{i=0}^{m} c_i u_i(x)\right)\left(\sum_{i=0}^{m} c_i v_i(x)\right) - \left(\sum_{i=0}^{m} c_i w_i(x)\right) \tag{2}$$

by showing that $h(x)$ is a polynomial of degree at most $n - 2$ (note that $t(x)$ has degree n by definition and $u_i(x), v_i(x), w_i(x)$ each have degree $n - 1$) – this ensures that all gate outputs are consistent with their respective inputs. We use the Groth16 arkworks-rs implementation [ar19] and feed in Circom-generated R1CS instances, after we convert them into a format that can be parsed by Arkworks using the ark-circom library [Kon21].

3.2.2 Gemini

A limiting factor for scaling SNARKs for large computation is the high and in some cases prohibitive memory usage of the prover during proof generation. With a memory consideration in mind, Gemini [BCHO22] is a SNARK proof system that relies on the R1CS representation using a universal setup, and can operate in two different modes by introducing the notion of an *elastic* prover. In the *time-efficient* mode, the prover runs in linear time but uses *linear* space (memory). In the *space-efficient* mode, the prover runs in quasi-linear time but uses *logarithmic* space by processing the R1CS instance in a streaming fashion (similar to [BHR+20]), i.e. in blocks of data rather than working with the entire R1CS instance at once, but this requires making multiple passes over the input data blocks. A similar result was previously achieved in [BHR+21]. Furthermore, using the prover's execution state Gemini can switch between these two modes at any time during the generation of a proof for the same statement, while the proof size and verification time remain the same, i.e. logarithmic, in both modes. Note that for computation where the space requirement is *much less* than the time needed, the prover would ideally run in linear time while using only the required (small) space, and Gemini achieves a meaningful relaxation of this by introducing two modes of operation for the prover.

While there has been a lot of work in constructing linear-time provers (which also require linear-space), for the space-efficient mode in Gemini the authors show that the combination of a polynomial Interactive Oracle Proof (PIOP) with a polynomial commitment scheme [CHM+20, BFS20] can be adapted for streaming access. In summary, the authors construct a space-efficient prover by first applying such an adaptation to the KZG [KZG10] polynomial commitment scheme and second to the dot-product operation between two vectors and also variations of it which are used heavily in the PIOP.

For our benchmarks, we rely on the non-preprocessing and time-efficient version of Gemini to generate proofs of inference that are encoded as R1CS instances.

3.3 STARK Proof Systems

STARK proof systems, following the original design proposed in [BSBHR18], offer an approach to zk-SNARKs which is rather different from that of other widely used constructions. The 'T' in STARK stands for "Transparent", referring to the fact that no "trusted setup" is necessary: there are no special keys for proving or verifying computations. Moreover, no elliptic curve cryptography is necessary in the construction: the only cryptographic primitive needed is a polynomial commitment scheme. This is typically instantiated using Merkle trees, providing potential post-quantum security and concretely fast provers at the expense of somewhat longer proofs than those found by other leading systems.

Like other types of SNARKs, STARKs (as described in [BSBHR18]) model the computation to be verified using the arithmetic of polynomials over a finite field. However, unlike most other SNARKs, STARKs do not use arithmetic circuits. Instead, the "arithmetization" uses a model called AIR (Algebraic Intermediate Representation): in terms of the hardware metaphor, AIR looks more like a state machine/CPU than a circuit. In this representation, the intermediate values of the computation are recorded in a "trace table", a $\mathcal{T} \times \mathcal{R}$ table of elements of a finite field $\mathbb{F}$, where each of the $\mathcal{R}$ columns represents a "register" and each of the $\mathcal{T}$ rows represents a time-step of the computation. Correctness of the computation is enforced by requiring that, for each time-step τ, a set of constraint polynomials $\{C_j\left((x_{\tau,0}, ..., x_{\tau,r-1}), (x_{\tau+1,0}, \ldots, x_{\tau+1,r-1})\right)\}$ vanish when evaluated on the adjacent rows $(x_{\tau,*}), (x_{\tau+1,*})$. In addition, the values of public inputs and outputs are enforced by requiring some number of "boundary constraints".

Note that this definition allows for "non-determinism", i.e. asserting that the values in row $\tau + 1$ are solutions to some polynomial equation without specifying how these solutions are computed. For example, we might have the constraint $C(x_\tau, x_{\tau+1}) = x_{\tau+1}^2 - x_\tau$, expressing that $x_{\tau+1}$ is the square root of x_τ without specifying an algorithm to compute this square root. A principal difficulty of working with STARKs lies in the design of AIR constraints to describe a given computation. For efficiency (both in terms of proof length and computation time), it is crucial that the degree of the constraint polynomials is as low as possible. In general, it is more difficult to design AIR constraints than a circuit modeling a computation - the constraints must be *the same* for each time step.

With the AIR model for the computation set, the STARK system then provides transformations to reduce the proof of knowledge of a trace table satisfying this system of constraints to a number of instances of low-degree testing.

The prover commits to each of the columns, then computes and then commits to "transition quotient" polynomials by formally substituting the polynomials representing the columns into the constraint polynomials and dividing by an appropriate vanishing polynomial. Thus, correctness of the computation is reduced to two Interactive Oracle Proofs (IOPs): first, that the transition quotient polynomials bear the correct relationship with the trace polynomials, and second, that all committed polynomials are actual Reed-Solomon codewords, i.e. that they have low degree.

One of the primary novel contributions of the STARK paper [BSBHR18] is the second IOP. This is known as FRI (Fast Reed-Solomon IOP of proximity), and provides an IOP that a given polynomial oracle is close in Hamming distance to a Reed-Solomon codeword, i.e. that it represents the values of a polynomial with bounded degree. As a primitive, FRI has been incorporated to other proof systems outside the realm of STARKS, such as Plonky2 [Tea]: it offers concretely fast performance and its security is dependent only on the underlying polynomial commitment scheme.

With respect to MLPs, the AIR mode of circuit representation is trickier to directly take advantage of. Additionally, naive approaches such as making the table wide enough to contain an entire layer's parameters – what one would need to compute a matrix multiplication two-row constraint – would lead to a wildly inefficient proof generation process. Instead, we take advantage of the AIR representation's design with respect to zk virtual machines (zkVMs) and implement our MLP operations in Cairo [GPR21], compiling the programs into an AIR representation using the Giza library [Gil22] and proving the generated AIR circuits via Winterfell [KGL$^+$21]. We implement matrix-vector multiplication the naive way in Cairo, and implement both ReLU and division via non-determinism, supplying hints and verifying that the operations were performed correctly. Specifically, we take advantage of Cairo's math library for comparison and integer division, which in turn makes use of range check built-ins for soundness.

3.3.1 Winterfell, Cairo, and Giza

Winterfell [KGL$^+$21] is an open-source STARK prover/verifier library, originally developed by the Novi Financial group at Meta. The design emphasizes support for concurrent proof generation, with an ultimate (as yet not fully realized) goal of enabling distributed proof generation across many machines. It is also designed to be flexible and modular, allowing the user to configure the various parameters occurring in the STARK system to optimize performance. Notably, these options include the choice of finite field, the hash function, and even the underlying implementation of the trace table datatype: for each of these, a number of default implementations are provided, but the code is written in terms of Rust traits which may be implemented at will by the user.

Cairo [GPR21], developed by STARKWare, is a Turing-complete programming language and associated "virtual machine" for generating STARK proofs for arbitrary computation. It provides a "universal AIR" describing an optimized Von Neumann architecture in which the public inputs for the AIR include a "bytecode" description of the computation to be verified. The design of Cairo includes a number of features designed to mitigate the overhead of such a universal format, breaking out of the strict "virtual machine" metaphor. For example, the Cairo language takes advantage of the non-determinism allowed by AIR through its concept of "hints", pieces of code executed outside of the virtual machine to generate an output which can then be passed into the virtual machine. Another valuable feature is the ability for the user to modify the virtual machine's architecture via "built-ins". These consist of additional AIR constraints on certain memory cells verifying some property directly without passing it through the "CPU": for example, one very useful built-in is for range checks, which could otherwise be expensive to implement via non-native arithmetic in the CPU.

Giza [Gil22] is an open-source library which closes the gap between the above two tools: it takes as input the trace generated by running a Cairo program, then uses Winterfell to generate a STARK proof attesting to the correctness of the result. Note that we use Giza's default security settings, with 54 queries, a blowup factor of 4, and a grinding factor of 16, yielding $16 + \log_2\left(\left(\frac{1}{\sqrt{4}}\right)^{54}\right) = 70$ bits of provable security and $16 + \log_2\left(\left(\frac{1}{4}\right)^{54}\right) = 124$ bits of conjectured security.

3.4 Plonkish Proof Systems

"Plonkish" proof systems [GWC19] follow a table-style arithmetization scheme, where inputs, intermediate values, hints, and outputs are laid out within cells. Constraints are applied row-by-row, and can be selectively applied to rows using a constraint selector. Moreover, columns within the table are described via univariate polynomials over a 2^n-th root of unity – concretely, $f_k(\omega^i)$ describes the (k, i)-th entry within the table – and can be sent to the verifier in compressed form via a polynomial commitment. We give a short description of our arithmetization techniques below. Additionally, Plonkish constraints provide a high degree of flexibility, such that there exist many ways to express the same computation via different constraints. Those described below were selected after trial and error to find efficient variants (see appendix A for a detailed description):

- **Linear layer**. Let $W \in \mathbb{F}^{M \times N}$ be the weight matrix, $x \in \mathbb{F}^N$ be the input vector, and $y \in \mathbb{F}^M$ be the output vector such that $Wx = y$. Then our table is laid out such that x is copied horizontally M times and arranged once next to each row of W. Our matrix multiplication gate is then simplified to a dot product constraint, where the ith row of W dotted against x must equal the ith term in y.
- **ReLU function**. Recall that the ReLU non-linearity is defined as $\mathsf{ReLU}(x) = \mathtt{max}(0, x)$. Since this non-linearity is difficult to compute in general using only addition and multiplication gates, we instead provide the bitwise decomposition $b_s, b_{n-1}b_{n-2}...b_0$ of each element x, where b_s is the "sign bit" and $b_{n-1}...b_0$ are the "value bits". We now enforce three constraints – firstly, that each bit is indeed either 0 or 1, and secondly, that the recomposition of the bits is equivalent to the original value, and thirdly, that the recomposition of the unsigned bits (i.e., the positive value) is equivalent to the output. Note that there is also an implicit constraint in our table layout in that we only provide enough decomposition bits to fit positive numbers, preventing overflow.
- **Fixed-point division**. As mentioned above in the R1CS section 3.2, we tackle the fixed-point issue by simply scaling inputs and weights by s, and biases by s^2, un-scaling after each MLP layer. The "un-scaling" division can be computed in a similar fashion, i.e. via bit decomposition, to the ReLU computation described above. We simply first verify the correctness of the sum of the bit decomposition, and then set the output to be the sum of the bit decomposition, ignoring the k least significant bits. This has the effect of dividing the number by 2^k, and truncating the decimal portion.

3.4.1 Halo2

Halo2 [hal] is a Plonk-ish proof system developed by the Zcash team. It features KZG [KZG10] and IPA-based [BBB$^+$18] polynomial commitment schemes, and allows for *custom gates* [GW19], *rotations* and *lookup tables* [GW20]. For benchmarking, we use the PSE fork of Halo2 [Com21] with KZG for smaller proofs and lower verification costs. Halo2 is written in Rust, and features a thorough set of developer tooling, including a mock prover and the ability to define modular gadgets. In particular, we were able to implement ReLU and fixed-point division as described above, but with bits of base 1024 rather than 2; the checks to ensure that bits fell in the range $[0, 1023]$ were then done via lookup. Halo2 additionally boasts a table layouter, which automatically rearranges the constraints/elements within a Plonkish table to reduce the number of rows – we use the two-pass layouter in our benchmarks and see notable speedups, as Halo2 proof times scale directly (rounded up to the nearest power of 2) with the number of rows within the table.

3.4.2 Plonky2

Plonky2 [Tea] is also a Plonkish proof system originally developed by the Mir Protocol. It features a FRI-based (fast Reed-Solomon IOPP, [BSBHR17]) polynomial commitment scheme and uses the much smaller Goldilocks field (prime modulus of $2^{64} - 2^{32} + 1$) to represent circuit data. Notably, elements within the Goldilocks field can be represented in 64 bits, resulting in faster arithmetic operations on standard hardware. Plonky2's frontend is also written in Rust, and features similar constraint and table definition tooling to those of Halo2's, though with fewer features (no rotations, lookup tables, or layouter). Additionally, Plonky2 proof sizes scale polylogarithmically with the table height and linearly with table width, and due to this, recursion tends to be a necessary step to reduce verifier time and proof size. Luckily, Plonky2 offers substantial support for such recursive proving, and in our benchmarks we recurse once, yielding 70kB proofs. Finally, we use Plonky2's default FRI settings and achieve a conjectured 100-bit computational security level with 28 Merkle queries, a blowup factor of 8, and a 16-bit proof-of-work grinding challenge.

3.5 GKR-based Proof Systems

As mentioned earlier in Section 2.2, the GKR approach features a prover convincing a verifier of the output of a *layered* arithmetic circuit – this is similar to the R1CS specification, but with each gate having a pre-defined "layer" i within the circuit and only taking inputs from layer $i + 1$ and feeding outputs to layer i (i.e. gates interact only within adjacent layers).

At a high level, the GKR protocol proceeds by reducing a claim about the output of a particular layer (i.e. specifically, the circuit output in the final layer) to claims about the output of the previous layer via the *sumcheck protocol*. This reduction happens for every layer of the circuit until the claim about the circuit output is reduced to a claim about the circuit input, which the verifier can simply check for itself. For succinctness with respect to the witness, the prover first commits to an MLE (multilinear extension) – a multivariate polynomial representing the input vector – of the inputs and provides an opening proof for the evaluation of the input MLE at the verifier's desired challenge point, using a polynomial commitment scheme. Moreover, this protocol can be made efficiently zero-knowledge (see e.g. [WTS$^+$18a, XZZ$^+$19]).

Next, we describe a few key points with respect to applying GKR to machine learning inference:

- **Interaction:** Each sumcheck round corresponding to verifying some layer claim, requires random challenges from the verifier in an interactive fashion. This can be made non-interactive via the Fiat-Shamir transformation, for which we instantiate the random oracle using the SHA-256 hash function for benchmarking.
- **Non-linearities:** The circuits which GKR is able to operate over are comprised entirely of addition and multiplication gates, similarly to R1CS. Modern approaches (as zkCNN, discussed below) tackle this similarly to the Plonkish approach above with bit decomposition, and supply "hint" decomposition bits to compute ReLU and division.
- **Layered circuits:** GKR traditionally operates over *layered* circuits, implying that model parameters used in later layers must be "passed through" earlier ones. Instead, each layer of the circuit reduces a claim on itself to a claim on both the previous layer *and* the input layer; claims on the input layer are aggregated via random linear combination across all layers and a sumcheck is performed at the end to ascertain the validity of these claims.

3.5.1 zkCNN

zkCNN [LXZ21] introduces an efficient GKR-based protocol for the Fast Fourier transform (FFT) alongside addressing the challenges above. This speed-up is used for convolution layers by relying on the fact that convolution can be expressed as a multiplication of polynomials, which can itself be accelerated using FFT. Moreover, the authors use the polynomial commitment scheme from [WTS$^+$18a] to obtain zero-knowledge for the input, and implement the codebase in C++. For benchmarking purposes, we replace the polynomial commitment scheme with an Arkworks implementation based off of [ZGK$^+$18b], as the Hyrax-based commitment scheme from [WTS$^+$18a] incurs $O(\sqrt{n})$ communication complexity and verifier overhead.

4 Results

4.1 Timing Benchmarks

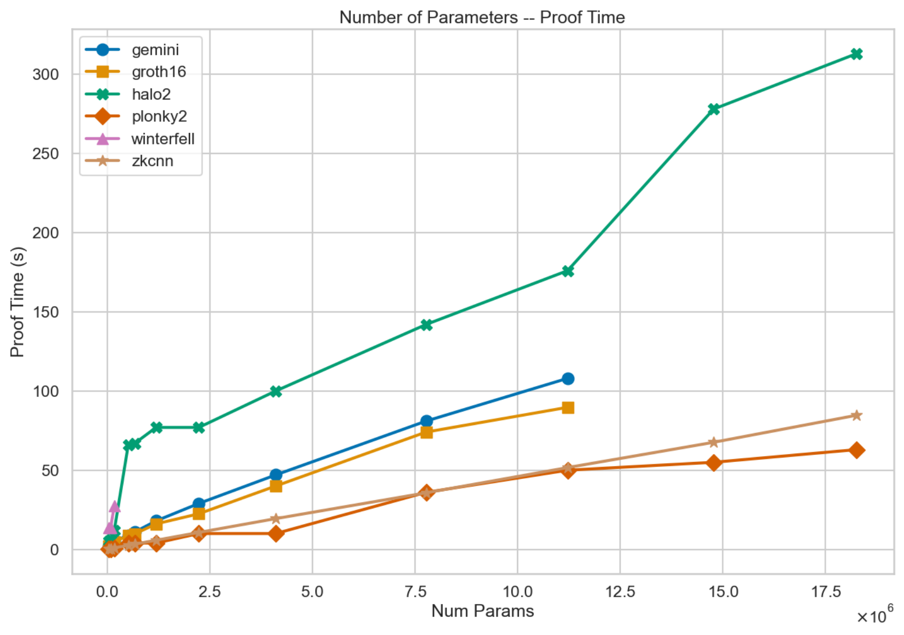


Figure 2: Num Params Suite Timing Tests

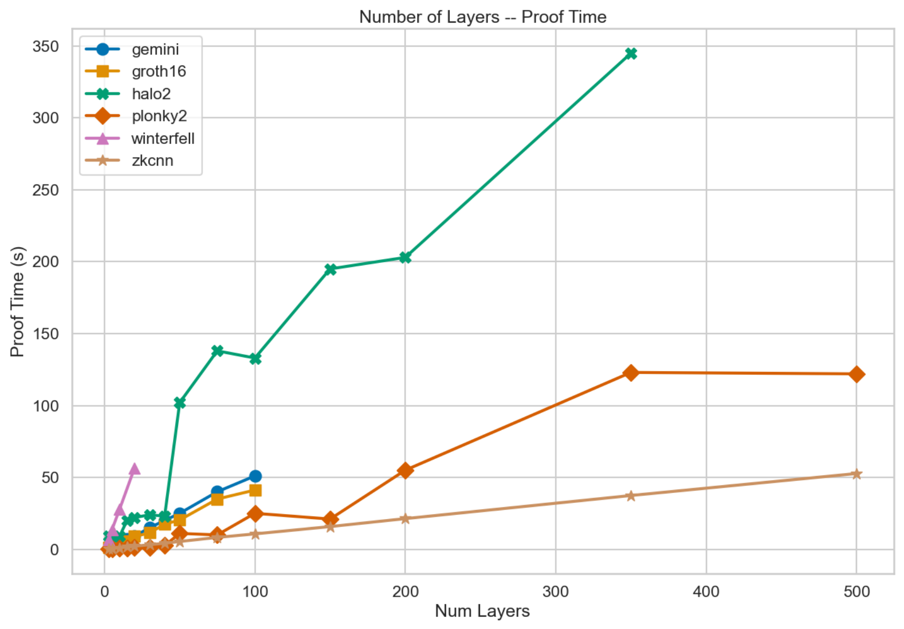


Figure 3: Deep Benchmark Suite Timing Tests

4.2 Memory Benchmarks

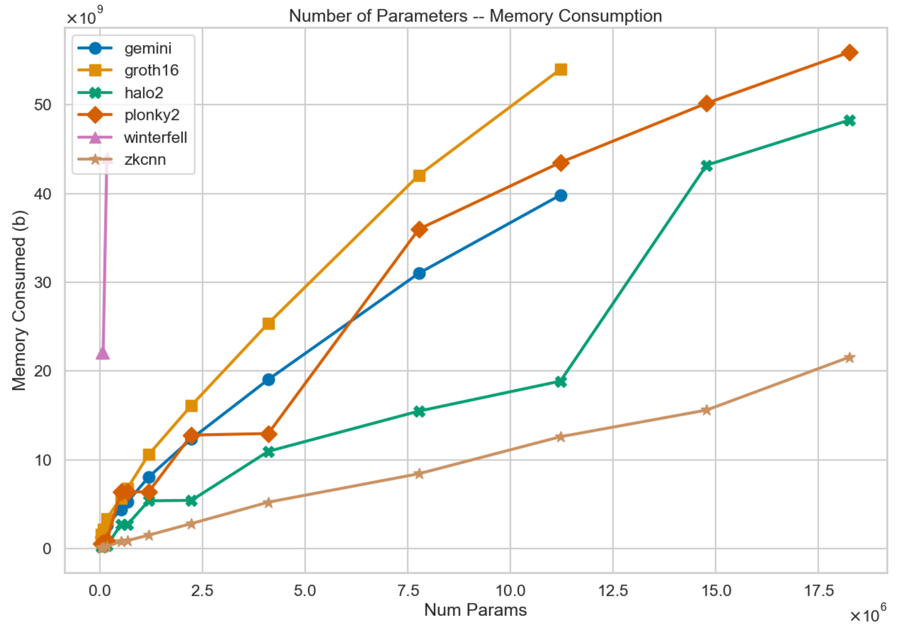


Figure 4: Num Params Suite Memory Tests

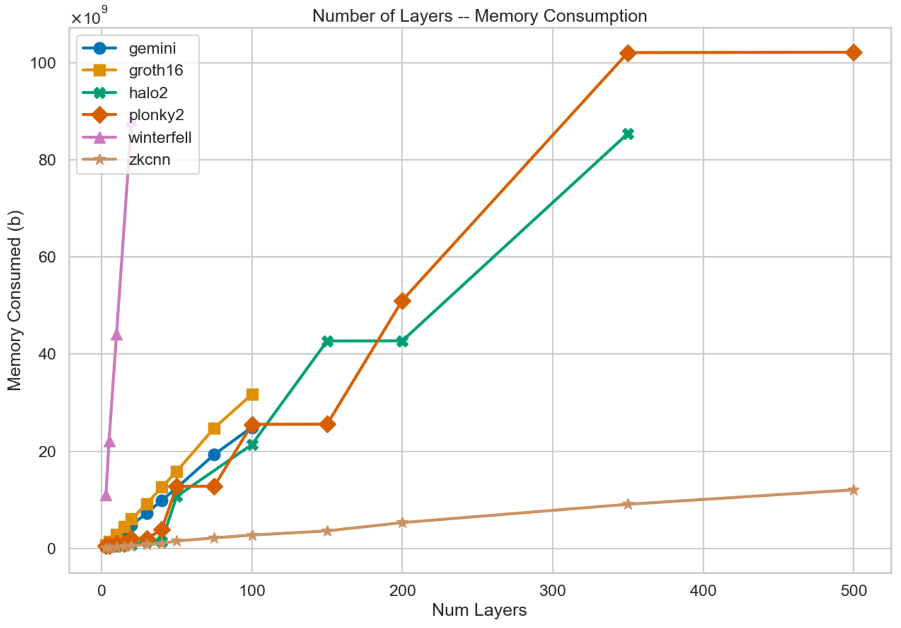


Figure 5: Deep Benchmark Suite Memory Tests

4.3 Analysis

All benchmarks were run on AMD EPYC 7R32 processors (AWS's c4ad.16x large instances) with 128GB of RAM. Multithreading was used wherever possible; exceptions to such are stated below.

4.3.1 Groth16

Groth16's cost model for proof generation (post-trusted setup, QAP circuit generation, and CRS generation) is actually quite simple – the bulk of the prover work comes down to a) converting witnesses, including circuit inputs, outputs, and intermediate values, into QAP coefficient form, and b) computing each of the A, B, and C terms described in [Gro16] using their respective group elements from the trusted setup, as shown below:

Proving Action	**% of Overall Proof Time**
R1CS to QAP witness map, i.e. computing the wire values for the reduced QAP instance (constant and addition gate folding) from R1CS coefficients	$35\% - 40\%$
Computing g^A, where the exponent is $\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta$	$< 5\%$
Computing g^B, where the exponent is $\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta$	$20\% - 25\%$
Computing g^C for $C = \frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+h(x)t(x)}{\delta} + As + rB - rs\delta$	$25\% - 30\%$

Groth16 performed significantly better than expected, largely as a result of the preprocessing nature of the construction, as well as the parallel Arkworks implementation we ran – in particular, Groth16 provers enjoy a per-circuit trusted setup, where e.g. $g^{u_i(x)}$ is already part of the CRS, with $u_i(x)$ denoting the polynomial representing the left inputs of each gate, and need only run MSM operations to compute both A and B, a highly parallelizable operation. Due to the parallelized MSMs, however, Groth16 provers see high memory consumption, and in particular must trade off MSM computation speed with maximum memory usage, assuming parallelism.

4.3.2 Gemini

The Gemini cost model can be broken down into two main parts: the polynomial commitment to the witness $\mathbf{w}$, and the polynomial IOP (PIOP) phase. The PIOP phase amounts for the majority of the prover time and uses four main checks: a Hadamard-product check to verify claims of the form $\mathbf{f} \circ \mathbf{g} = \mathbf{h}$, a twisted scalar-product check to verify claims of the form $\langle \mathbf{f} \circ \mathbf{y}, \mathbf{g} \rangle = u$, a scalar-product check to verify claims of the form $\langle \mathbf{f}, \mathbf{g} \rangle = u$ (special case of the twisted scalar-product check), and a tensor-product check to verify claims of the form $\langle \mathbf{f}, \otimes_j (1, \rho_j) \rangle = u$, where $\mathbb{F}$ is a field with $u, \rho_j \in \mathbb{F}$ and $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{y} \in \mathbb{F}^N$ for appropriately chosen N. Without going into detail, proving such claims accounts for most of the computation the prover does to generate a proof.

In summary, the PIOP executes the next three steps.

First, for a R1CS instance A, B, C and vector $\mathbf{z} = (\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is the secret witness, the Hadamard-product check verifies that $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}$, which is what is required by the the R1CS problem. This Hadamard-product check is not done directly but is instead divided into three claims: $u_A = \langle A\mathbf{z}, \mathbf{y} \circ \otimes_j (1, \rho_j) \rangle$, $u_B = \langle B\mathbf{z}, \otimes_j (1, \rho_j) \rangle$, and $u_C = \langle C\mathbf{z}, \mathbf{y} \rangle$, where $\rho_0, \ldots, \rho_{n-1}, v \in \mathbb{F}$ (with $n = \log N$) and $\mathbf{y} = (1, v, \ldots, v^{N-1})$ are verifier randomness. For this exposition and at a high level, a claim is comprised from the messages in an interactive protocol between the prover and a verifier, and this includes a sumcheck execution.

Second, the above three claims are combined into a scalar-product claim using a verifier challenge $\eta \in \mathbb{F}^{\times}$ as follows:

$$u_A + \eta \cdot u_B + \eta^2 \cdot u_C = \langle \mathbf{z}, \mathbf{a}^* + \eta \cdot \mathbf{b}^* + \eta^2 \cdot \mathbf{c}^* \rangle$$

where $a^* = \mathbf{y}^T \circ \otimes_j (1, \rho_j)^T A$, $\mathbf{b}^* = \otimes_j (1, \rho_j)^T B$ and $\mathbf{c}^* = \mathbf{y}^T C$.

Third, the above scalar-product check is not done directly but is instead divided into two claims, as follows: $u_D = \langle \mathbf{z}, \otimes_j (1, \rho'_j) \rangle$ and $u_E = \langle \mathbf{a}^* + \eta \cdot \mathbf{b}^* + \eta^2 \cdot \mathbf{c}^*, \otimes_j (1, \rho'_j) \rangle$ where $\rho'_0, \ldots, \rho'_{n-1}$ is verifier randomness.

Finally, the claim about u_E can be verified directly by the verifier and the claim about u_D is verified directly between the prover and the verifier using a protocol for the tensor-product check. We summarize the prover execution as described above, in the following table.

Proving Action	**% of Overall Proof Time**
Commitment to witness $\mathbf{w}$	$8\% - 12\%$
Divide the Hadamard-product into three claims	$6\% - 10\%$
Prepare the scalar-product claim	$9\% - 13\%$
Divide the scalar-product check into two claims	$6\% - 10\%$
Prepare the tensor-product claim	$< 0.001\%$
Tensor-product check	$55\% - 65\%$

We see that the commitment to the witness $\mathbf{w}$ takes a reasonable amount of the prover time, and the same holds for the first couple of steps of the PIOP because these mostly generate claims by executing interactive protocols. The bulk of the work is done in the final PIOP step which is the tensor-product check that executes a protocol to verify a claim which incorporates all previous claims. We note that the protocol relies on the KZG polynomial commitment scheme, and uses techniques that reduce multilinear polynomial evaluation claims required by sumcheck, to univariate polynomial ones since univariate polynomials seem to be more popular in practice so far; see Remark 2.2 in [BCHO22] for a discussion about this.

4.3.3 Giza/Winterfell

Although the AIR-FRI STARK proving cost model has a reasonably intuitive breakdown (see table below), we emphasize that this proof method is *not quite* comparable to the other techniques benchmarked in the whitepaper. In particular, as mentioned earlier, Cairo is an imperative language with the notion of a once-writeable and readable stack, registers, and instructions encoded as AIR table constraints, and thus packs significant overhead. To ensure a more equal comparison, the cost of generating and populating the trace table, including generating and committing to auxiliary table columns, are omitted from the final runtime numbers – note that for the same Cairo program, with a fixed number and ordering of allocations, thus producing the same-sized and structured trace table regardless of the input *values*, the auxiliary column is unchanged and can thus be precomputed.

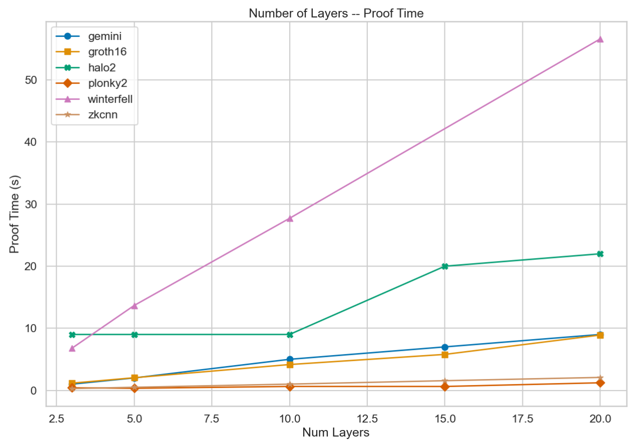


Figure 6: Deep Benchmark Suite Timing Tests (smaller models only, including Giza/Winterfell)

Proving Action	**% of Overall Proof Time**
Execution trace low-degree extension (LDE) polynomial computation and commitment	$45\% - 50\%$
Constraint evaluation over LDE domain	$25\% - 30\%$
Constraint polynomial factoring into lower-degree polynomial components	$6\% - 10\%$
Constraint polynomial evaluation and commitment	$\approx 5\%$
DEEP composition polynomial construction and evaluation	$10\% - 12\%$
FRI folding and query proofs	$\approx 1\%$

We see that the LDE polynomial computation and constraint evaluation tend to be the bottlenecks in proof generation as a result of one particular representation quirk – note that the overhead of expressing MLP operations in Cairo causes a significant (a factor of $\approx 2^4$) blowup in the size of the trace table as compared to the model size. As an example, the largest model of the graph above has around $400000 \leq 2^{19}$ parameters, while the generated trace table is of size 2^{23}, with an LDE evaluation domain of 2^{25} (with a blowup factor of 4). This causes both the LDE computation and the constraint evaluation over the LDE domain to be relatively expensive (though parallelizable), especially compared to the extremely parallelizable FRI folding and query generation steps, and this is even more so when the FRI folding factor is > 2 – indeed, Giza sets this to 8, requiring fewer folding steps and generating slightly larger proofs in the process.

4.3.4 Halo2

The Halo2 cost model depends on a variety of factors, and is highly variable with respect to circuit design (i.e. how custom gates are structured). Its table size is of particular importance, as each advice column is another polynomial to commit to, and each row – rounded up to the nearest power of 2 – increases the degree of the committed polynomials, linearly increasing the cost of commitment/opening. Roughly speaking, however, the prover for Halo2 can be broken down into the following steps:

Proving Action	**% of Overall Proof Time**
Computing and committing to polynomials representing the advice, instance, and lookup columns	$15\% - 20\%$
Computing and committing to polynomials representing permutation arguments (i.e. copy constraints)	$25\% - 30\%$
Computing and committing to the "grand product polynomial" representing all lookup table accesses	$< 1\%$
Computing and committing to the quotient polynomial $h(x)$	$35\% - 40\%$
Evaluating all polynomials at challenge points supplied by the verifier	$< 1\%$
Performing the multi-point KZG opening to evaluate committed polynomials at the verifier's challenge points	$10\% - 15\%$

We see that a majority of the prover time is spent computing and committing to the quotient polynomial $h(x)$ and computing permutation arguments, as well as computing and committing to polynomials representing the advice columns. The permutation argument in particular is fairly expensive here, as we require copy constraints over a significant number of columns – this leads to a linear blowup in the degree of the permutation polynomial, leading to a linear increase in the commit and opening times. Computing and committing to the quotient polynomial $h(x)$ is quite expensive, as although our gates are not particularly high-degree, the numerator of $h(x)$ involves selector polynomials multiplied to gates spanning many columns, all of which must be processed via IFFT to compute the final $h(x)$ in coefficient form. In a similar vein, committing to polynomials representing each of the advice columns is simply linear in the number of advice columns, i.e. roughly linear in the size of the largest matrix spanning the table.

4.3.5 Plonky2

The Plonky2 cost model is similar to Halo2's, albeit sans lookup arguments and with Merkle-FRI commitments to witnesses, permutation polynomials, and the combined constraint quotient polynomial. The prover's time-consuming actions can be roughly broken down as follows:

Proving Action	**% of Overall Proof Time**
Computing and committing to the wire polynomials, i.e. the intermediate computed values	$60\% - 70\%$
Computing and committing to permutation polynomials and permutation argument product polynomials	$5\% - 8\%$
Computing and committing to the vanishing/quotient polynomial	$6\% - 10\%$
Computing opening proofs for the verifier's challenge point (Plonky2 uses a multi-opening version of FRI)	10%

Plonky2's concrete prover time breakdown is quite interesting – its quotient polynomial computation is significantly faster than that of Halo2's, which may be explained by the fact that field operations are computable in a single 64-bit word, resulting in far faster scalar operations and FFT computation. Moreover, it uses much less time for the permutation polynomials' construction and commitment, attributable to the parallelizability of both Merkle tree construction and FRI's folding step for FRI-based polynomial commitments, and is faster with respect to the multi-point opening argument as well – a consequence of FRI opening proofs being quite efficient for the prover (a logarithmic number of Merkle proofs suffices for both membership and low-degree-ness) and the fact that they open all committed polynomials at the same challenge point.

4.3.6 zkCNN

The zkCNN cost model, both with respect to prover runtime and memory consumption, can be broken down into two primary components – the MLE commit/open phase, and the GKR/sumcheck phase. In our benchmarks shown above, the MLE commit/open phase typically comprises about 65 – 75% of the total overhead with respect to prover runtime, with the commit phase contributing around 20 – 25% of the total overhead, and the opening phase contributing between 40 – 50% of the total overhead. We note here that the MLE commit/open phase was tested via a multithreaded implementation (32 threads on 16 cores) while the GKR/sumcheck phase was tested via a single-threaded implementation. Thus the runtime bottleneck lies almost entirely in the MLE commit/open phase – a consequence of the large number of multi-scalar multiplications (MSMs) required for the KZG-style MLE commitment scheme.

With respect to the memory usage, the prover memory consumption during commit/open contributed between 55 – 65% to the total RAM usage, with the remainder comprising mainly of keeping track of bookkeeping tables during the GKR/sumcheck phase.

5 Future Directions

Can today's ZK proof systems bring artificial intelligence to the blockchain? This question has motivated our work through each of the systems we benchmarked in this paper. The answer, of course, is an emphatic yes. In fact, over the course of our work, we successfully constructed inference proofs in six different proof systems, over 200 different times.

A more fitting question, and the one that went on to inspire the title of this paper, is instead: *What is the **cost** of ZK proving machine learning?* After all, the *feasibility* of verifying massive compute directly impacts the scale and potential of value-generating use-cases for this kind of nascent technology. This is why we have chosen to begin the work of constructing a cost model for verifiable machine learning as a whole, starting with this paper. We started with prover runtime and memory, key specifications for any implementation of trustless intelligence. And in reviewing the results of our benchmarks, it is clear that modern proof systems already hold tremendous promise. In fact, techniques as Plonky2 and zkCNN, with respect to prover time and memory consumption, respectively, signal the key improvements needed to bring sophisticated models to the verifiable compute paradigm.

To discover the *real-world* context for these performance results, however, as well as the other dimensions of cost that become relevant in emerging use-cases, we introduce two concrete applications – each a paragon of scale across a radically different dimension. First, Worldcoin's iris verification model, a sophisticated convolutional network with strict precision and memory requirements. And secondly, AI Arena's agents' policy networks, where tens of thousands of inference computations must be cost-effectively proven at a time.

We discuss each of the applications in more detail here, with special attention to the cost metrics relevant to each.

5.1 Worldcoin

Worldcoin is building the world's first "Privacy-Preserving Proof-of-Personhood Protocol." In other words, solving Sybil attacks by tying authentication to a deeply unique biometric characteristic – the iris.

It's a wild idea, and one that has engendered a great amount of excitement and technical innovation towards the goal of preserving privacy and security. After downselecting through different authentication schemes, Worldcoin's approach ultimately involves the use of custom hardware that attains the 1-in-10 billion distinction resolution needed for processing iris data. Afterwards, this high-resolution iris scan is compressed via a convolutional neural network and inserted as an entry into their Semaphore implementation, allowing for membership attestation and useful functions such as voting or once-only actions completed via a zk-SNARK proof of inclusion.

In order to show that sensor data was processed correctly, Worldcoin currently uses a trusted runtime environment with a secure enclave directly on the orb's hardware, verifying the camera-signed iris scan and running the model processing within. For cryptographic-level security guarantees, they'd like to use a ZKP to attest to the correct inference of the neural network, which similarly allows users to have self-custody over their own biometric data. The model itself features on the order of millions of parameters and many tens of layers, a complexity necessary to distinguish between 10 billion different irises. Performance-wise, not only must the network be incredibly precise, but it must also run on local *mobile* hardware with severe compute and memory restrictions to ensure the aforementioned data self-custody. As an example, while proving systems such as Plonky2 on compute-optimized cloud CPUs can generate Worldcoin's proof-of-inference in a matter of minutes, the prover's memory consumption would overwhelm any commercially available mobile hardware – indeed, such proofs require tens of gigabytes of RAM to be successfully generated. This is a direct barrier to Worldcoin's aspirations for self-custody via 32-bit Android devices. Additionally, when Worldcoin's biometric processing network undergoes model updates, re-processing will need to be done entirely locally as well – clearly, efficient runtimes must not come at the cost of ballooning memory consumption.

5.2 AI Arena

AI Arena is a platform fighting game in the style of Super Smash Bros, with a distinctive twist: rather than player-*operated* avatars fighting against one another in real-time, player-*owned* AI models compete and battle autonomously. A particularly exciting example can be found here.

While these models are initially trained through imitation learning on their owners' combat data, they go on to compete against one another in tournament matches, climbing a global leaderboard and earning rewards. The models are linked to corresponding NFTs, with each representing a dynamic, ever-improving algorithm whose fighting style and skill level are directly related to those of their owners.

Though tournaments are currently conducted in a centralized manner, the benefit of enhancing AI Arena's incentive design with a fair and autonomous economy is at once cogent and ambitious. After all, a truly trustless tournament massively rewards player investment, yet also involves the necessary task of verifying a staggering amount of AI computations per game.

Gameplay-wise, matches are run at 60 frames per second and last 3 minutes. This translates to 20,000+ inference results between the two player models each round – a difficult enough task for modern (centralized) hardware, not to even mention the overhead of proving such compute. Unlike Worldcoin's constraint, however, AI Arena's proof work is massively parallelizable – simply, one might partition batches of discrete frames worth of inference proofs across many cores on a compute cluster. Instead, the difficulty comes from the monetary cost of running so many cryptographic operations within proof systems. As an example of the overhead imposed by proof systems, we consider one of AI Arena's policy networks – a relatively small MLP requiring ~ 0.008 seconds to perform a single forward pass with. Proving this same model using zkCNN's approach requires 0.6 seconds, i.e., on the order of a 1000x compute blowup. Translating this into real-world costs, note that the compute-optimized family of AWS instances cost around 15 cents per compute-hour, and that the computation of inferences within a single match, previously:

$$\frac{\$0.15}{\text{hour}} \cdot \frac{\text{hour}}{3600 \text{ seconds}} \cdot \frac{0.008 \text{ seconds}}{\text{frame}} \cdot \frac{21600 \text{ frames}}{\text{match}} = \frac{\$0.072}{\text{match}} \tag{3}$$

now becomes,

$$\frac{\$0.15}{\text{hour}} \cdot \frac{\text{hour}}{3600 \text{ seconds}} \cdot \frac{0.6 \text{ seconds}}{\text{frame}} \cdot \frac{21600 \text{ frames}}{\text{match}} = \frac{\$5.40}{\text{match}} \tag{4}$$

with proof of inference – a problem of scale. As unit economics become increasingly important for on-chain services, service providers must balance the value of decentralized security with the literal cost of proof generation.

5.3 Related Academic Works

Over the past decade, there has been a large number of academic works focusing on improving the concrete efficiency of zk-SNARK proof systems. And while today's zk-proofs are already many orders of magnitude faster than their state-of-the-art counterparts of years prior, it is hard to imagine the demand for compute slowing. This is especially true in the field of Artificial Intelligence, where with each year, larger and more complex models create even more spectacular results. From large language models to generative transformers, the simple trend lies bare: **as AI compute scales, so do results**.

This is why we are interested in the construction of a custom zero-knowledge prover – *REMAINDER* – a proof system purpose-built for efficiently proving large models. This novel system will take advantage of every speed-up available for highly structured computation, while being perfectly positioned to capture parallelism-driven acceleration via custom hardware. It will also be constructed to satiate the constraints and specifications we identified above, both in memory and proving cost, for concrete use-cases.

To that end, below we discuss a number of *peer-reviewed* works published in academic venues, which develop techniques that could lead to efficiency improvements in scaling zk-SNARKs for large machine learning models, alongside recent theoretical developments for SNARKS and other related work.

Peer-reviewed works on SNARKs. There has been a long line of work that improves the (concrete) efficiency of zk-SNARKs in various aspects such as prover time, proof size, verifier time and the cryptographic assumptions used. Moreover other important factors include the size and setup type of the Common Reference String (CRS), i.e. trusted, universal (and updatable) or transparent, and post-quantum security. Early work on efficient SNARKs such as [Gro10, PHGR13] rely on "knowledge assumptions" [Dam92] for security and starting with Groth16 [Gro16] the generic group model was used to prove the security of a proof system with quasilinear prover time, constant size proofs and verification time, but with a CRS that is circuit specific.

Later works, such as Ligero [AHIV17] and Bulletproofs [BCC$^+$16, BBB$^+$18] rely on standard cryptographic assumptions, e.g. Collision-Resistant Hash Functions (CRHF) or the Discrete Logarithm Problem (DLOG), but similar to the above are proven secure in the Random Oracle Model (ROM) and achieve sublinear proof size while improving on the type of setup but at the cost of linear verification time. To improve on proof size and verification time, Sonic [MBKM19], Marlin [CHM$^+$20] (concurrent with Plonk [GWC19]) use a *one-time trusted* preprocessing phase during the CRS setup, and subsequently Fractal [COS20] and Spartan [Set20] achieve this with a transparent setup. Recently, Orion [XZS22] obtained linear prover time with polylog proof size and verification time (a similar theoretical result was obtained in [BCL22]), while previous linear time prover works in [BCG$^+$17] and Brakedown [GLS$^+$21] result in larger proof size and verification time.

Finally, a few works target computation that is more structured, such as Hyrax [WTs$^+$18b], Libra [XZZ$^+$19] and Virgo [ZXZS20] that are aimed for parallel computation (e.g. layered circuits), and Nova [KST22] that is aimed for recursive composition of the same computation (e.g. applies to Verifiable Delay Functions).

We note that there are many influential works that have not explicitly appeared in an academic venue but are actively being deployed in practice such as STARK [BBHR18], Plonk [GWC19] including many follow-up works such as plookup [GW20], Halo2 [BGH19, hal] and many more.

Recent theoretical work on SNARKs. For certain classes of computation (complexity class $\mathcal{P}$), a recent work [WW22] presents a SNARK construction under standard bilinear group assumptions, while another line of work focuses on the sound instantiation of the Fiat-Shamir paradigm in the plain model under standard cryptographic assumptions, e.g. [CCH$^+$19, PS19, HLR21], that lead to new constructions of SNARKs, e.g. [JKKZ21, CJJ21, KVZ21]. For general computation (complexity class $\mathcal{NP}$), the heavy hammer of code obfuscation [SW14, JLS21] can be used to construct SNARKs. Finally, we note that for general computation (complexity class $\mathcal{NP}$) there are barriers [GW11] for constructing SNARKs with optimal parameters in the plain model.

While this is an active field of academic research, these results are theoretical and currently do not compete in terms of concrete efficiency with constructions in the random oracle model that are currently used in practice.

Other related work. There exist other approaches for privacy-preserving machine learning (PPML), but the goal of these works is to efficiently compute large statements about machine learning in an *interactive* manner rather than to construct publicly-verifiable zk-SNARKs.

For completeness, we mention a few of these works that rely on secure Multiparty Computation (MPC) [Yao82, GMW87, BGW88, CCD88], or Fully Homomorphic Encryption (FHE) [Gen09b, Gen09a, BV11, GSW13, BV14] and variants such as Leveled Homomorphic Encryption (LHE) and Additively Homomorphic Encryption (AHE). In the remaining of this section, we use the the term *homomorphic encryption* to refer collectively to FHE, LHE and AHE. Moreover, this line of work frequently requires a specific preprocessing phase before the actual computation begins, and any generated proofs in this setting can be verified only by a party that possesses *a-priori* some secret information, which is known as the *designated* or *private* verifier setting. Note that this remains an active area of academic research.

With respect to MPC, early works [NIW$^+$13, NWI$^+$13] rely on MPC to achieve privacy preserving machine learning regression problems and recommendation systems, respectively. Later works combine arithmetic and boolean operations using AHE, secret-sharing techniques and garbled circuits that are best suited for machine learning algorithms, and rely on a preprocessing (offline) phase. Specifically, [DSZ15, MZ17, JVC18, MLS$^+$20, PSSY21] focus on the (semi-honest) two-party setting where for example two parties can jointly train a model while each maintaining privacy of their data, while [LJLA17] focuses on inference in the same setting. Muse [LMSP21] achieves security for PPML in the setting of where one party (client) could be malicious while the other party (server) is semi-honest. Furthermore, the works of [MR18, WGC19, KPPS21, WTB$^+$21] extend model training and inference to the setting of three and four parties, with a subset of these works achieving active security. Recently, the work in CryptGPU [TKTW21] which builds on top of the CryptTen PPML MPC framework [KVH$^+$21], and the work in Piranha [WWP22] both achieve significant efficiency improvements by relying on GPU acceleration.

With respect to zero-knowledge proofs for inference in the designated verifier setting, Mystique [CK18] uses MPC with preprocessing to generate proofs about the correct execution of large models such as ResNet-50 and ResNet-101.

With respect to homomorphic encryption, CryptoNets [DGBL$^+$16] relies on LHE and [BMMP18] relies on FHE and also constructs an FHE friendly representation of a neural network, to privately make predictions using a pre-trained model in a setting where two parties trust one another with respect to the integrity of the computation but want to maintain privacy of the data and model. To exemplify the main idea behind the FHE approach, consider a patient that submits their FHE encrypted data to a hospital that makes a diagnosis based on a model that is executed under FHE on the patient's data. The hospital then returns the encrypted result to the patient that uses their FHE decryption key to obtain the diagnosis in the clear. Note that using FHE, both the hospital and the patient maintain privacy of the data and model in the process. The use of FHE in this example is equivalent to semi-honest two-party secure computation. While homomorphic encryption can be computationally expensive it has the benefit of requiring a minimal number of messages that two parties have to exchange during the protocol (i.e. low communication complexity), when compared to MPC solutions where generally communication scales with the size of the computation. Thus, homomorphic encryption could also be used only for specific parts of a protocol that would require such properties.

5.4 Conclusion

In this work, we evaluated proving two suites of MLP inference algorithms using a variety of existing zero-knowledge proof systems. This included architectures that scaled with increasing parameter count (up to 18M parameters and 22B mult-adds), as well as architectures that scaled with an increasing number of layers (up to 500 layers).

We compared results across Gemini, Winterfell, Halo2, Plonky2, and zkCNN, and saw that the flexibility of custom gates within the Plonkish approach is extremely powerful. Despite this, however, GKR-based approaches such as zkCNN demonstrate similar proving time performance – even without a parallel prover implementation – while incurring far lower memory costs. This opens the door to future work in specialized ZK proving systems tailor-made for highly structured computation such as deep neural nets and other artificial intelligence algorithms.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[ar19] arkworks rs. groth16. https://github.com/arkworks-rs/groth16, 2019.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 336–365. Springer, Heidelberg, December 2017.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 427–457. Springer, Heidelberg, May / June 2022.

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 275–304. Springer, Heidelberg, May / June 2022.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BHR+20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 168–197. Springer, Heidelberg, November 2020.

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 123–152, Virtual Event, August 2021. Springer, Heidelberg.

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

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Heidelberg, August 2018.

[BSBHR17] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. *Electron. Colloquium Comput. Complex.*, TR17, 2017.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, page 1–12, New York, NY, USA, 2014. Association for Computing Machinery.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: From practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 1082–1090, New York, NY, USA, 2019. Association for Computing Machinery.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[CJJ21] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from lwe. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79, 2021.

[CK18] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1687–1700. ACM Press, October 2018.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *ITCS 2012*, pages 90–112. ACM, January 2012.

[Com21] Electric Coin Company. Halo2. `https://github.com/privacy-scaling-explorations/halo2`, 2021.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, August 1992.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[DGBL$^+$16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 201–210. JMLR.org, 2016.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, February 2015.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Heidelberg, May 2020.

[FQZ$^+$21] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Report 2021/087, 2021. `https://eprint.iacr.org/2021/087`.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGG17] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. 2017.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[Gil22] Max Gillett. Giza. https://github.com/maxgillett/giza, 2022.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), sep 2015.

[GLS$^+$21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021. https://eprint.iacr.org/2021/1043.

[GMN21] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical snark aggregation. Cryptology ePrint Archive, Paper 2021/529, 2021. https://eprint.iacr.org/2021/529.

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

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GPAM$^+$14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo – a turing-complete stark-friendly cpu architecture. Cryptology ePrint Archive, Paper 2021/1063, 2021. https://eprint.iacr.org/2021/1063.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[GW19] Ariel Gabizon and Zachary Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs, 2019.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[hal] Halo2. https://zcash.github.io/halo2. Accessed: 2022-11-07.

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat–shamir via list-recoverable codes (or: Parallel repetition of gmw is not zero-knowledge). In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 750–760, New York, NY, USA, 2021. Association for Computing Machinery.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[Ide21] Iden3. Circom. https://github.com/iden3/circom, 2021.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. Snargs for bounded depth computations and ppad hardness from sub-exponential lwe. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 708–721, New York, NY, USA, 2021. Association for Computing Machinery.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 60–73, New York, NY, USA, 2021. Association for Computing Machinery.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 1651–1668, USA, 2018. USENIX Association.

[KGL+21] Irakliy Khaburzaniya, Franc cois Garillot, Kevin Lewi, Konstantinos Chalkias, and Jasleen Malvai. Winterfell. https://github.com/novifinancial/winterfell, 2021.

[KHSS22] Daniel Kang, Tatsunori B. Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless dnn inference with zero-knowledge proofs. *ArXiv*, abs/2210.08674, 2022.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery.

[KMC18] Julien Keuffer, Refik Molva, and Hervé Chabanne. Efficient proof composition for verifiable computation. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part I*, volume 11098 of *LNCS*, pages 152–171. Springer, Heidelberg, September 2018.

[Kon21] Georgios Konstantopoulos. ark-circom. https://github.com/gakonst/ark-circom, 2021.

[KPP+14] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: Faster verifiable set computations. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 765–780. USENIX Association, August 2014.

[KPPS21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 2651–2668. USENIX Association, August 2021.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Heidelberg, August 2022.

[KVH$^+$21] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 619–631. ACM Press, October / November 2017.

[LKKO20] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network. Cryptology ePrint Archive, Report 2020/584, 2020. https://eprint.iacr.org/2020/584.

[LMSP21] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 2201–2218. USENIX Association, August 2021.

[LXZ21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2968–2985. ACM Press, November 2021.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[MLS$^+$20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2505–2522. USENIX Association, August 2020.

[MR18] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017.

[NIW$^+$13] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 801–812. ACM Press, November 2013.

[NWI$^+$13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE Computer Society Press, May 2013.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019.

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 2165–2182. USENIX Association, August 2021.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Ros57] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[SBV$^+$13] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, page 71–84, New York, NY, USA, 2013. Association for Computing Machinery.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

[SHZ$^+$18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. pages 4510–4520, 06 2018.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[Tam03] Roberto Tamassia. Authenticated data structures. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, pages 2–5, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Tea] Polygon Zero Team. Plonky2. https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf. Accessed: 2022-11-10.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 71–89. Springer, Heidelberg, August 2013.

[TKTW21] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *2021 IEEE Symposium on Security and Privacy*, pages 1021–1038. IEEE Computer Society Press, May 2021.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, jan 2015.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, 2019(3):26–49, July 2019.

[WJB$^+$17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2071–2086, New York, NY, USA, 2017. Association for Computing Machinery.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[WTB$^+$21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proc. Priv. Enhancing Technol.*, 2021(1):188–208, 2021.

[WTS$^+$18a] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zk-snarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018.

[WTs$^+$18b] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zk-SNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[WW22] Brent Waters and David J. Wu. Batch arguments for sfNP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2022.

[WWP22] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 827–844, Boston, MA, August 2022. USENIX Association.

[WWT$^+$22] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. pvcnn: Privacy-preserving and verifiable convolutional neural network testing, 2022.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 299–328. Springer, Heidelberg, August 2022.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Heidelberg, August 2019.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2039–2053. ACM Press, November 2020.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vsql. Cryptology ePrint Archive, Paper 2017/1146, 2017. https://eprint.iacr.org/2017/1146.

[ZGK+18a] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018.

[ZGK+18b] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vram: Faster verifiable ram with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 908–925, 2018.

[ZWW+21] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems*, 32(10):2524–2540, 2021.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.

A Appendix

A.1 Naive EVM-Neural Network Cost

To illustrate the astronomical costs of naively operating artificial intelligence algorithms on-chain, we turn to the Ethereum whitepaper [Woo14]. Wood explains that the cost of running even several hundred thousand FLOPs worth of compute – barely enough for the computation of a tiny neural net – is millions of gas, equivalent to tenths of an ETH, or hundreds of dollars. For now, it seems, we would do best to heed Vitalik Buterin's sage advice: "running deep learning inside the EVM is probably not the solution."

A.2 Plonkish Arithmetization

Below we give our full description for Plonkish arithmetization:

- **Fully-connected layer**. Specifically, we focus on the matrix-vector multiplication subcircuit: let $W \in \mathbb{F}^{M \times N}$ be the weight matrix, $x \in \mathbb{F}^N$ be the input vector, and $y \in \mathbb{F}^M$ be the output vector such that $Wx = y$. Then our table looks as follows (note that f_i is the polynomial representing the values in the ith column when evaluated at roots of unity):

f_1	$\dots$	f_n	f_{n+1}	$\dots$	f_{2n}	f_{2n+1}
w_{11}	$\dots$	w_{1n}	x_1	$\dots$	x_n	y_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
w_{m1}	$\dots$	w_{mn}	x_1	$\dots$	x_n	y_m

Table 1: Matrix-vector table data layout

Our per-row "dot product constraint" looks like the following (note that the second line shows the same constraint with respect to the per-column generator functions):

$$\forall i : y_i - \left(\sum_{j=1}^{n} w_{ij} \cdot x_j \right) = 0 \tag{5}$$

$$\implies f_{2n+1}(u) - \left(\sum_{j=1}^{n} f_j(u) \cdot f_{j+n}(u) \right) = 0 \tag{6}$$

We note that the per-row constraint is very simple – we enforce that y_i, the purported output of the dot product operation, is actually equivalent to the actual dot product $\sum_{j=1}^{n} w_{ij} \cdot x_j$. Although in theory this constraint needs to be enforced for every row of the weight matrix, in practice the final constraint polynomial actually only requires a single extra term (the latter equation), since $f_k(u)$ is a polynomial encoding all the values within the kth column of the table (again, recall that $f_k(\omega^i)$ evaluates to the ith entry in the kth column of the table, where ω is the 2^nth root of unity for tables of height 2^n).

- **ReLU function**. Recall that the ReLU non-linearity is defined as $\mathsf{ReLU}(x) = \mathtt{max}(0, x)$. Since this non-linearity is difficult to compute in general using only addition and multiplication gates, we instead provide the bitwise decomposition $b_s, b_{n-1}b_{n-2}...b_0$ of each element x, where b_s is the "sign bit" and $b_{n-1}...b_0$ are the "value bits". Our table setup is thus as follows:

f_1	f_2	f_3	$\ldots$	f_{n+2}	f_{n+3}
x	b_s	b_0	$\ldots$	b_{n-1}	y

Table 2: ReLU bit decomposition table layout

Our constraints for ReLU are as follows:

$$\forall i \in [0, ..., n-1] : b_i(1 - b_i) = 0 \tag{7}$$

$$\implies \forall i \in [3, ..., n-2] : f_i(u)(1 - f_i(u)) = 0 \tag{8}$$

This forces all elements in the bitwise decomposition to be either 0 or 1. Moreover,

$$b_s\left(x - \sum_{i=0}^{n-1} b_i 2^i\right) + (1 - b_s)\left(x + \sum_{i=0}^{n-1} b_i 2^i\right) = 0 \tag{9}$$

$$\implies f_2(u)\left(f_1(u) - \sum_{i=3}^{n+2} f_i(u) 2^{i-2}\right) + (1 - f_2(u))\left(f_1(u) + \sum_{i=3}^{n+2} f_i(u) \cdot 2^{i-2}\right) = 0 \tag{10}$$

This ensures that the bitwise decomposition was performed correctly (note that $b_s = 1$ if $x \geq 0$ and 0 otherwise). Finally, we constrain y to be $ReLU(x)$:

$$b_s \cdot (x - y) + (1 - b_s) \cdot y = 0 \tag{11}$$

$$\implies f_2(u) \cdot (f_1(u) - f_{n+3}(u)) + (1 - f_2(u)) \cdot f_{n+3}(u) = 0 \tag{12}$$

Note that if $x \geq 0$, i.e. $b_s = 1$, the above constraint becomes $x - y = 0$, i.e. $y = x$. If $x < 0$ on the other hand, i.e. $b_s = 0$, the above constraint becomes $y = 0$, as required.

- **Fixed-point division**. One of the primary issues in writing machine learning models — as arithmetic circuits in a cryptographic setting — is the fundamental incompatibility of traditionally floating point numbers within a vanilla ML model vs. the need for an arithmetic circuit to be defined within a finite field. To combat this issue, we scale the model weights and inputs by a factor of s each, and the biases by a factor of s^2 each, and after performing a typical linear layer computation $Wx + b$, we "un-scale" the output by a factor of s. More concretely, let W, x, b, y be the usual (non-scaled) weight, input, biases, and output for a fully-connected layer. Our computation is as follows:

$$y' = (sW)(sx) + (s^2 b) = s^2(Wx + b) \tag{13}$$

$$y = \frac{y'}{s} = s(Wx + b) \tag{14}$$

Thus the invariant holds that throughout every layer of the neural network, the inputs to a particular layer are always scaled by s, and the outputs are scaled by that same s.

The above operations can be constrained in a fairly straightforward way, with the exception of the division by s. Let $s = 2^l$ be the scale factor, let $s^2 y$ be the scaled output from a layer, let $b_s, b_{n-1}b_{n-2}...b_0$ be the sign bit and bit decomposition of $s^2 y$, and let y' be the "desired" output (in other words, it should be the case that $y' = sy$), where y is the "true" output value. We show the division by s operation as follows:

Our first two constraints for division are exactly the same as the constraints for ReLU above, i.e. we constrain each of $b_0, ..., b_{n-1}$ to be either 0 or 1, and also ensure that the bit decomposition was done correctly. Our last

f_1	f_2	f_3	$\dots$	f_{n+2}	f_{n+3}
s^2y	b_s	b_0	$\dots$	b_{n-1}	y'

Table 3: Division bit decomposition table layout

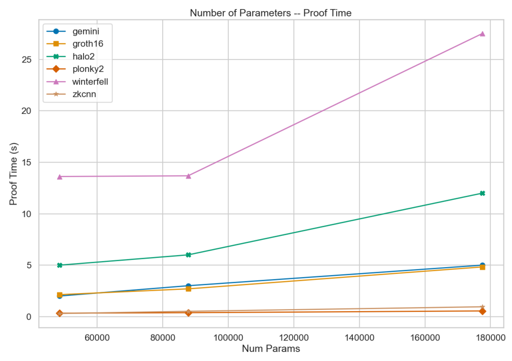
constraint is an equality check to make sure that y' is indeed $\frac{s^2y}{s} = sy$:

$$b_s \cdot \left(\sum_{i=l}^{n-1} b_i 2^{i-l} - y\right) + (1 - b_s) \cdot \left(\sum_{i=l}^{n-1} b_i 2^{i-l} + y\right) = 0 \tag{15}$$

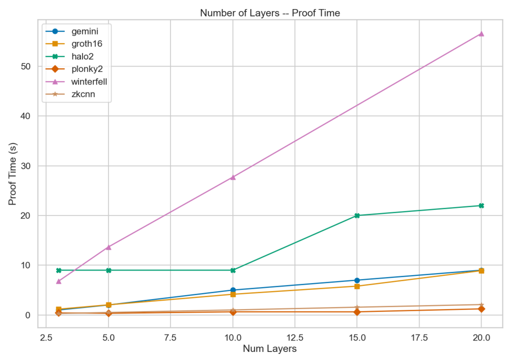
$$\implies f_2(u) \cdot \left(\sum_{i=l}^{n-1} f_{i+3}(u) 2^{i-l} - f_{n+3}(u)\right) + (1 - f_2(u)) \cdot \left(\sum_{i=l}^{n-1} f_{i+3}(u) 2^{i-l} + f_{n+3}(u)\right) = 0 \tag{16}$$

Similarly to ReLU, note that b_s forms a "selector" depending on whether the value is positive or negative; in either case the $\sum_{i=l}^{n-1} b_i 2^{i-l}$ term gives the bitwise recomposition of all but the l *least* significant bits, and scales by 2^{i-l}, effectively dividing (with truncation) by $2^l = s$.

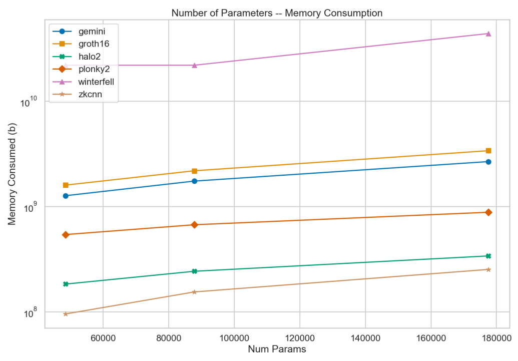
A.3 Benchmark Results (Winterfell/Giza)



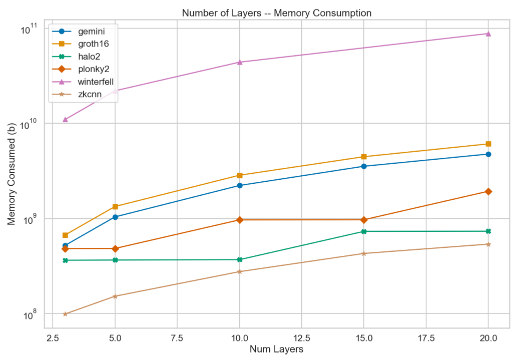
(a) Params Suite Timing



(b) Deep Benchmark Timing



(a) Params Suite Memory (Logarithmic)



(b) Deep Benchmark Memory (Logarithmic)

A.4 Benchmark Architectures

A full description of each suite of benchmarked MLP architectures is as follows:

A.4.1 Parameter count benchmarks

Models described below can be broken down into two subcategories – "small" and "large" parameter count models. The former have input/output sizes of 32 and 100, respectively, as well as just 20 layers, while the latter have input/output sizes of 784 and 1000, respectively, as well as 30 total layers.

Architecture Name	**Subcategory**	**Param Count**	**FLOPs**	**Largest FC Matrix dim**
FC_48564_params	Small	49,473	3.42M	88×100
FC_87771_params	Small	88,988	7.88M	111×122
FC_177522_params	Small	179,249	22.18M	161×177
FC_517529_params	Large	520,892	275.89M	230×1000
FC_676836_params	Large	680,646	349.32M	273×1000
FC_1195804_params	Large	1,200,801	605.53M	389×1000
FC_2236477_params	Large	2,243,260	1.21G	562×1000
FC_4107399_params	Large	4,116,558	2.62G	793×1000
FC_7770136_params	Large	7,782,715	6.33G	1124×1124
FC_11216646_params	Large	11,231,754	10.77G	1369×1369
FC_14773663_params	Large	14,790,999	16.18G	1586×1586
FC_18252933_params	Large	18,272,201	22.17G	1773×1773

A.4.2 Depth benchmarks

Similarly to the above suite, models described below can be broken down into two subcategories – "small" and "large" depth models. The former have input/output sizes of 32 and 100, respectively, while the latter have input/output sizes of 784 and 1000, respectively.

Architecture Name	**Subcategory**	**Param Count**	**FLOPs**	**Number of Layers**
FC_3_layers	Small	43,600	6.38M	3
FC_5_layers	Small	83,900	12.43M	5
FC_10_layers	Small	174,600	25.54M	10
FC_15_layers	Small	285,400	42.68M	15
FC_20_layers	Small	376,100	55.79M	20
FC_30_layers	Small	577,600	86.04M	30
FC_40_layers	Small	779,100	116.29M	40
FC_50_layers	Large	1,146,700	254.05M	50
FC_75_layers	Large	1,750,500	430.69M	75
FC_100_layers	Large	2,154,200	405.30M	100
FC_150_layers	Large	3,161,700	556.55M	150
FC_200_layers	Large	4,169,200	707.80M	200
FC_350_layers	Large	7,191,700	1.16G	350
FC_500_layers	Large	10,214,200	1.62G	500