

USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Efficiently Provable Approximations for Non-Polynomial Functions

Sriram Sridhar, *University of California, Berkeley;* Shravan Srinivasan, *Lagrange Labs;*
Dimitrios Papadopoulos, *Hong Kong University of Science and Technology;*
Charalampos Papamanthou, *Lagrange Labs and Yale University*

https://www.usenix.org/conference/usenixsecurity26/presentation/sridhar

This paper is included in the Proceedings of the
35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the
35th USENIX Security Symposium
is sponsored by



جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Efficiently Provable Approximations for Non-Polynomial Functions

Sriram Sridhar*
UC Berkeley

Shravan Srinivasan
Lagrange Labs

Dimitrios Papadopoulos
HKUST

Charalampos Papamanthou
Lagrange Labs
Yale University

Abstract

Zero-Knowledge Proofs (ZKPs) are now widely used to verify the correctness of various types of computations. However, despite phenomenal advancements, current ZKPs are inefficient for applications that need accurate evaluation of non-polynomial functions over floating-point numbers, such as machine learning, decentralized finance, scientific computing, and geolocation. Current state-of-the-art approaches typically emulate floating-point numbers using fixed-point representations (via quantization), and handle *non-polynomial* functions using lookup tables, piece-wise or low-degree polynomial approximations, which lead to sub-optimal performance and/or loss in accuracy or generality, limiting their potential for adoption in practice.

In this work, we present a general framework for approximating a large class of non-polynomial functions using Gauss-Legendre quadrature, which supports efficient ZKPs of correct computation. We show that our approach can scale to decrease the error up to the inherent limits imposed by quantization, without increasing the multiplicative circuit depth beyond a small constant (≤ 4). This is a strong deviation from prior approximation techniques, where decreasing the error leads to increased multiplicative depth – the main factor determining the error growth of an approximation. We implement and evaluate our approach in Noir/Barretenberg, and we obtain absolute errors $2-256\times$ lower than comparable baselines for most non-polynomial functions with low prover overhead. We also demonstrate an efficient prover and low errors for high-accuracy applications in DeFi and astronomy that require non-polynomial functions, again obtaining errors $4-64\times$ lower than the baseline approximations.

1 Introduction

Modern zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) allow efficient and privacy-preserving proofs of correct computation. They are widely used in many areas, including blockchain scaling, decentralized identity, and privacy-preserving compliance. The vast majority of these systems operate by abstracting the computation as operations over a finite field, expressed as correctness constraints that can be provably satisfied. Hence, in practice, they excel at verifying computations that can be expressed as low-degree *polynomials* over such finite fields. However, many real-world applications require evaluating *non-polynomial* or *transcendental* [54] functions, such as trigonometric, activation, power functions, and other non-elementary functions involving real values. For instance, such functions are widely used in machine learning [28, 37, 42], finance [7, 23, 55], geolocation [10, 12], astronomy [49], scientific computing [2], statistics [39], and beyond [11].

Unfortunately, proving such computations via zk-SNARKs remains rather inefficient due to a fundamental mismatch in computational representations. General-purpose programming languages natively express computations over real numbers using floating-point operations (e.g., IEEE 754 standard [1]), whereas zk-SNARKs require expressing these computations as arithmetic circuits over *finite fields*. This requires "emulating" floating-point operations over finite field using addition/multiplication gates and corresponding constraints, which introduces significant overhead [10, 16].

Prior works in ZKPs have tried to address the issue of proving computations involving floating-point arithmetic. However, despite significant progress, the resulting performance remains far from practical. For example, initial work on IEEE-754 compliant multiplication over FP32 required upwards of eight thousand constraints per operation [47]. Later work [10] reduced this to 64 to 209 constraints based on the level of amortization and optimizations. Even when relaxing strict IEEE 754 compliance [16, 38], such as by adopting a relative error model, verifier complexity remains linear [16] or the number of constraints is still 1.7 orders of magnitude higher [38] (see Table 5). Moreover, these works are designed for single or double precision floating-point, making it non-trivial to further increase precision for high-accuracy applications [23, 49].

*Work partly conducted at Lagrange Labs

These severe performance limitations have motivated researchers and practitioners to adopt alternate representations of real numbers for use in efficient zero-knowledge proof systems. The natural solution is to emulate floating-point arithmetic using *fixed-point representation* that uses a fixed number of bits for integer and fractional parts, with the decimal point's position predetermined. This entails a *quantization* step that maps continuous values to discrete levels defined by bit width, introducing small *quantization errors*, which is the lower bound on the possible error even with an honest prover. It is worth noting that floating-point arithmetic is not strictly required for high-accuracy computations in ZKPs. In fact, fixed-point arithmetic is naturally finite-field friendly, enables fast operations, and can achieve accuracy comparable to or exceeding floating-point arithmetic with sufficient bits of precision, while being easily extendable to arbitrary precision. This combination of high performance and accuracy has led to the adoption of this approach in many real-world deployed zero-knowledge systems [2, 9, 27, 28, 37, 56, 57].

Efficiently provable evaluation of non-polynomial functions. The two classes of techniques that have been predominantly used in the literature to evaluate non-polynomial functions in ZKPs are: (1) *lookup tables*, and (2) *numerical approximation techniques*, such as piecewise linear approximations and polynomial approximations (e.g., Taylor series). These two approaches provide different trade-offs between accuracy, efficiency, and generality, as we discuss next.

Lookup tables [8, 13, 34, 40] require pre-populating a table with the function outputs for all possible inputs, so that proving the correctness of the function evaluation later can be reduced to proving the correctness of a lookup operation in the table (and the well-formedness of the latter, if needed). This can greatly speed up proving certain non-linear functions, *especially over small input domains* [21, 27, 37, 42, 56, 57]. At the same time, lookup based methods can oftentimes achieve errors as low as the (inherently unavoidable) quantization error. However, lookup table approaches suffer from the following *major* limitations: *(i)* They require large amounts of memory to store the tables, and the performance of the prover is directly affected by the table size; this makes them unsuitable for use over large input domains. For example, for a 30-bit domain over a 256-bit field, a two-column table would require 64 gigabytes of storage, making it difficult to store, let alone compute lookup proofs [40]. This can be mitigated for functions that are naturally decomposable [5, 8, 40], such as exponentiation, where exponents can be split into smaller chunks. Unfortunately, this is not generally applicable to many non-polynomial functions, such as trigonometric functions. *(ii)* Even when the function is decomposable, combining the results from individual lookups requires several multiplications in the zk-SNARK circuit, increasing its multiplicative depth *logarithmically* with the number of chunks. This in turn imposes a logarithmic increase in error because multiplying two b-bit numbers results in a $2b$-bit result, which needs to be re-quantized into b-bits by truncating the lower b-bits, thus losing information with each iterated multiplication. These shortcomings make lookup tables suitable for a limited class of functions in practice.

Numerical approximation techniques come in different forms with their own advantages and disadvantages. Piecewise linear approximations [15, 43] can approximate any non-polynomial function by segmenting the input domain into smaller intervals and approximating the result of each individual segment using a linear function. However, achieving high accuracy requires a large number of segments, which increases the prover's overhead as the correct segment needs to be selected. Polynomial approximations, such as Taylor series [26, 33], can approximate differentiable transcendental functions using polynomials, leading to better efficiency. In fact, modern CPUs natively use polynomial approximations to compute transcendental functions efficiently [22]. However, in this case too, achieving high accuracy necessitates high-degree polynomials, hampering prover performance.

This work. Motivated by the above, in this work, we focus on the following problem:

Can we prove the correct evaluation of a wide range of non-polynomial functions over large domains with low prover overhead and low absolute error?

We address this by introducing novel evaluation techniques based on *numerical integration* [17, 50] and, as an independent contribution, *rational polynomial function approximations* [32, 51]—a generalization of polynomial approximations. These approaches enable us to achieve our desired efficiency and accuracy goals.

The key insights behind our techniques are as follows:

- *Wide range of functions:* Many non-polynomial functions (e.g., $\exp(x)$) have the key property that the derivative of their inverse is an algebraic function. This means that if f is a function, f^{-1} is its inverse, and the derivative $\frac{d}{dx}f^{-1}(x)$ can be expressed using basic arithmetic operations, making it *algebraic* (Section 4.3).

  This turns complex non-polynomial functions into integrals of simple algebraic expressions, which we efficiently approximate using numerical integration (e.g., Gauss–Legendre quadrature), thus enabling us to support a wide range of non-polynomial functions in zero-knowledge proofs.

- *High accuracy:* The complexity, such as the number of terms, order, etc., of numerical approximation techniques (including ours) generally grows with increasing accuracy demands. Most existing techniques used in zero-knowledge literature translate this added complexity into further multiplications, i.e., in a circuit with larger multiplicative depth. As discussed above, iterated multiplications lead to compounding of errors, thus decreasing accuracy. Our Gauss–Legendre quadrature approach avoids this issue as it increases only the size of the summation, not affecting the

multiplicative depth. This is particularly important in applications where these functions are used iteratively (e.g., multiple activation layers in machine learning models), in which cases, even a small improvement in accuracy is extremely impactful over these multiple iterations.

In particular, although the summation has multiple terms based on the accuracy required, errors do not compound via the summation – this is due to the summation being weighted and all the weights adding to 2. Hence, as the domain size increase, the error we incur is *independent* of the domain size or the number of terms in our approximation – this is a novel property that, to the best of our knowledge, *no previously used approximation method for ZKPs achieves*.

- *Padé approximants:* Our rational approximation entails fractional terms with a numerator and a denominator such that it achieves the same accuracy as a Taylor series of total degree $m+n$, but restricting the degree of the numerator to m and that of the denominator to n, roughly halving the highest degree polynomial that needs to be evaluated. Simply put, if we fix a "target" approximation error, we can achieve it via Padé approximation with polynomials of half the degree of what is needed with a Taylor series – making it a "cost-free" drop-in improvement for *all* existing Taylor-series-based works.

- *Verification vs. Computation:* We leverage the support of non-determinism of zk-SNARKs to check a "certificate" of a computation, that is easier to prove than the computation itself. This is a classic technique, adopted in various contexts in prior zero-knowledge literature (e.g., [36] for basic arithmetic checks, [44] for matrix multiplication, and [28] for FFTs). A simple example of this is proving that b^{-1} is the multiplicative field inverse of $b \in \mathbb{F}_p$. Instead of proving the computation of b^{-1} (e.g., via a circuit representation or Euclid's Extended Algorithm), we can simply check that b^{-1} provided by the prover satisfies $b \cdot b^{-1} \equiv 1 \mod p$, which is a much simpler statement to express via an arithmetic circuit. Looking ahead, both our adopted approximation methods heavily rely on proving term "divisions", i.e., computation of multiplicative inverses, hence we aggressively leverage non-determinism to accelerate these operations.

1.1 Summary of Contributions

Our contributions can be summarized as follows:

1. As our primary contribution, we propose the use of **Gauss–Legendre quadrature** to prove the correct evaluation of non-polynomial functions in zk-SNARKs (Section 4.3). Our analysis (Section 4.5) shows that this quadrature-based approach achieves lower error than all prior numerical approximation techniques with minimal loss in efficiency. At the same time, although it does not achieve the (theoretically optimal, modulo quantization) accuracy of lookup table methods, it is applicable to arbitrary domain sizes where lookup tables become infeasible, and generally outperforms them in prover time for smaller domains (Section 5). In fact, increasing the desired accuracy (up to the limits imposed by quantization and input distribution) has minimal impact on prover's performance and *does not increase the multiplicative depth of the computation*, unlike prior numerical approximation methods or decomposable lookup tables.

2. Independently of the above, we are the first to propose the use of **Padé approximants**, rational functions that generalize Taylor series, to prove the correct evaluation of non-polynomial functions in zk-SNARKs (Section 4.2). This serves as an independent contribution and a drop-in replacement for the Taylor series approach. Our analysis (Section 4.5) and experimental evaluation (Section 5) show that this Padé approximation outperforms all prior numerical approximation methods both in terms of error and in terms of performance (while achieving slightly higher error but better performance than our novel quadrature-based approach).

3. We implement both techniques in Noir Lang [31], a leading Plonk-based framework, and evaluate them across four categories of non-polynomial functions: trigonometric, activation, power, and non-elementary functions commonly used in real-world applications. Our microbenchmarks show significant improvements in prover time and error vs. existing approaches based on lookup tables, piecewise linear approximations, and Taylor series (Section 5.1).

   - Our Padé-based approach is up to 20.1×, 15.8×, and 2.8× faster than lookup tables, piecewise linear approximations, and Taylor series methods, respectively, while remaining comparable w.r.t. the error loss.
   - Our Gauss–Legendre quadrature-based approach (for large quantization domains) achieves the best overall accuracy while remaining efficient. Specifically, it up to 8 bits more accurate than our baselines, corresponding to as much as a 256× reduction in absolute error.

4. We demonstrate the practicality of our techniques in two real-world applications, where accuracy is crucial:

   - *Orbital mechanics* (Section 5.2): Efficiently proving the correct evaluation of Kepler's equation, which involves trigonometric functions.
   - *DeFi token pricing* (Section 5.3): Efficiently verifying the Bancor algorithm [23], which uses a non-polynomial formula deployed on smart contracts to determine digital asset exchange rates.

   In both these applications, we show that even in large quantization domains, our approach achieves 2 to 6 bits smaller error than the Taylor-series-based approach, corresponding to as much as a $4-64\times$ smaller absolute error.

2 Related Work

In this section, we survey prior works on zero-knowledge proofs for non-polynomial functions and discuss their trade-offs and limitations. Existing approaches can be broadly classified into approaches based on floating-point arithmetic and fixed-point arithmetic.

Floating-point arithmetic. Prior works aim to prove the correctness of floating-point arithmetic with either (1) *full* compliance [10, 47] or (2) *quasi* compliance [16, 38] of the IEEE 754 floating-point standard.

The first line of work encodes the entire floating-point operation, including "edge cases" and special values (NaN, infinity), rounding modes, and exception handling as per the IEEE 754 standard, inside the circuit. This ensures that the circuit operations are identical to the result of the standard specification [10, 47]. However, ensuring this *full* compliance incurs a high constraint cost for each floating-point operation. As already noted in Section 1, a single multiplication requires more than eight thousand constraints in [47], while [10] reduces this to 64–209 constraints depending on the degree of amortization, even after excluding lookup overhead. Moreover, proofs in [47] are non-succinct (due to using VOLE-based ZK proofs from emp-toolkit) and it supports only a small set of activation functions.

Quasi-compliant works partially alleviate the high constraint overhead of full compliance by having the honest prover follow IEEE 754 and then prove an upper bound on the relative error at each step of the computation. They are more efficient, but still incur logarithmic accuracy-loss overhead due to slack in the constraints [16, 38]. Moreover, [16] is non-succinct for the verifier. Recognizing the high constraint cost of basic operations, the state-of-the-art IEEE 754-compliant work [38] directly optimizes GeLU and related activation functions using piecewise polynomial approximations and lookup tables. However, even this specialized approach incurs $4.8 - 50\times$ higher constraints costs than our fixed-point approach (see Table 5 and Appendix E). Furthermore, our work is more general, supporting a broad range of non-polynomial functions beyond activation functions. Additionally, these floating-point approaches are limited to single or double precision, restricting maximum accuracy to that offered by double precision. Fixed-point arithmetic can also achieve the same or higher accuracy by increasing the number of precision bits. Moreover, since floating-point approaches are designed for double precision, they cannot arbitrarily improve the accuracy even with additional bits of precision.

Fixed-point arithmetic. Fixed-point arithmetic represents real numbers using a fixed number of digits for the integer and fractional parts using a constant scaling factor. Existing works use either lookup tables, piecewise linear approximations, or polynomial approximations. Most prior work in fixed-point arithmetic prioritizes efficiency over precision, using common quantization domains of 8 or 16 bits, which *unavoidably* limit accuracy to roughly 2^{-8} or 2^{-16} respectively.

Lookup-table based. As noted in Section 1, lookup tables are widely used when the input domain is small or the function is decomposable [21, 27, 37, 42, 56, 57]. For example, for $y = e^x$, we can decompose x in base b and express $y = \prod_{i=0}^{k-1}(e^{b_i})^{x_i}$. The digits x_i then lie in a smaller range, reducing the setup from one table of size 2^{bk} to k tables of size 2^b. The final result can be computed by multiplying individual lookup results, thus needing only $\log k$ multiplications. However, decomposability is a strong property that only holds for a small class of functions and incurs compounding error with each multiplication when computing the final result. When the function is not decomposable, lookup tables do not scale to large domains.

Piecewise approximations [15, 43]. As noted in Section 1, piecewise approximations require numerous segments to achieve high accuracy. In fact, the prover time is dominated by verifying the correctness of the selected segments, rather than by performing multiplications, as in other approaches.

Polynomial approximations [26, 29, 33]. Any differentiable function can also be well-approximated by Taylor series centered at 0. These approximations are very good close to 0, but lose accuracy when moving away from the center. For high accuracy, we would need a polynomial approximation of a higher degree, again affecting the prover's overhead.

We also note that some of works in the fixed-point domain focus on efficiency and prover time at the cost of succinctness – this shows up in works like [21, 29] that are VOLE-based ZK proofs [46] that are highly efficient for the prover but non-succinct for the verifier. These works also restrict themselves to low scaling factors for smaller lookup tables / polynomials – for illustration, [42] uses fixed-point numbers with scaling factor 16, [29] uses degree-2 polynomial approximations, [21] does lookup-based approximations with a scaling factor of 12.

Other work. Approximations for non-polynomial functions are also commonly used in multi-party computation (MPC) – especially in privacy-preserving machine learning. Prior work uses the above techniques as well (piecewise-linear approximations [30], piecewise-polynomial (spline) approximations [19, 41], lookup-table based approximations [18, 45]), as well as direct computation using Newton-Raphson iterations [25], adapted for the distributed setting. We note that our core techniques (based on numerical integration) are based on verifying correct computation, and they are not compatible with the distributed MPC setting.

3 Preliminaries

In this section, we introduce the necessary preliminaries for our techniques and the baseline. In the rest of this section, *we use t to denote the input variable of a function*.

Fixed-point arithmetic. Let $\mathbb{F}_p$ be the finite field of the proof system, where p is the field modulus. We encode a real num-

ber $a \in \mathbb{R}$ as a fixed-point value in $\mathbb{F}_p$ by storing $\bar{a} = \lfloor a \cdot 2^s \rfloor$, where the scaling factor s specifies the precision. We denote the quantized value of a by $\bar{a}$. Consider $a, b, c \in \mathbb{R}$ and say that $\bar{a} = \lfloor a2^s \rfloor, \bar{b} = \lfloor b2^s \rfloor$. To verify the addition operation $c = a + b$, where $\bar{c} = \lfloor c2^s \rfloor$ it is enough to check:

$$\left| \bar{a} + \bar{b} - \bar{c} \right| < \varepsilon$$

for $\varepsilon < 1$. To see why this works, note that:

$$\begin{aligned} \bar{a} + \bar{b} - \bar{c} &= \lfloor a2^s \rfloor + \lfloor b2^s \rfloor - \lfloor c2^s \rfloor \\ &= (a2^s - \varepsilon_a) + (b2^s - \varepsilon_b) - ((a+b)2^s - \varepsilon_c) \\ &= \varepsilon_c - (\varepsilon_a + \varepsilon_b) \end{aligned}$$

where $\varepsilon_a, \varepsilon_b, \varepsilon_c \in [0, 1)$ are the rounding errors introduced by the fixed-point encoding. Thus, the absolute value of the ε is less than 1. To verify the multiplication operation $c = ab$, where $\bar{c} = \lfloor c2^s \rfloor$, it is enough to check:

$$\left| \bar{a} \cdot \bar{b} - \bar{c} \cdot 2^s \right| < \varepsilon 2^s$$

for $\varepsilon < 1$. To see why this works, note that:

$$\begin{aligned} \bar{a}\bar{b} - \bar{c} \cdot 2^s &= \lfloor a2^s \rfloor \cdot \lfloor b2^s \rfloor - \lfloor c2^s \rfloor 2^s \\ &= (a2^s - \varepsilon_a) \cdot (b2^s - \varepsilon_b) - (ab2^s - \varepsilon_c)2^s \\ &= (ab2^{2s} - a2^s\varepsilon_b - b2^s\varepsilon_a + \varepsilon_a\varepsilon_b) - (ab2^{2s} - \varepsilon_c 2^s) \\ &= \varepsilon_c 2^s - (a2^s\varepsilon_b + b2^s\varepsilon_a - \varepsilon_a\varepsilon_b) < \varepsilon 2^s \end{aligned}$$

where $\varepsilon_a, \varepsilon_b, \varepsilon_c \in [0, 1)$ are the rounding errors introduced by the fixed-point encoding. These techniques for verifying additions and multiplications will form the foundation of our techniques and are common in the literature [16].

Precision, approximation accuracy, and approximation error. *Precision* refers to the number of bits used to represent a real-valued number. *Accuracy* of an approximation refers to how close the approximated value is to the true value of the function being approximated, and is measured in terms of the number of matching higher order bits. *Error* or *accuracy loss* is the complement of accuracy and refers to the number of lower order bits that do not match between the approximated value and the true value. *Quantization error* refers to the rounding error introduced when representing a real-valued number using a fixed number of bits. This is the lower bound on the possible error, even with an honest prover. The error is the maximum number of lower-order bits that a cheating prover can manipulate undetected, and arises due to the slack allowed in the constraint system due to the approximation. We additionally note that due to our verification-based approach, an honest prover in our system will always succeed in proving with the exact function value (up to the quantization error), although an adversarial prover has more slack – which we bound. Henceforth, when the context is clear, we simply use *error* to refer to the *adversarial error*, the main quantity we aim to minimize in this work.

Piecewise linear approximation. Piecewise Linear approximations [52] break the domain of a function into smaller segments and approximate each segment using a linear function. For example, to approximate $f(t)$ over the interval $[a, b]$, we can divide it into n segments: $[t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]$, where $t_0 = a$ and $t_n = b$. Each segment can then be approximated by a linear function.

$$f(t) \approx \begin{cases} a_1 + b_1 t & t \in [t_0, t_1], \\ \vdots & \vdots \\ a_n + b_n t & t \in [t_{n-1}, t_n]. \end{cases}$$

The accuracy of the approximation improves with an increase in the number of segments n. But this also increases the prover time as this approach needs to prove that correct segment was selected. In the limiting case, piecewise linear approximations can be as expressive as lookup tables.

Taylor series approximation. Any continuous function f can be approximated by its Taylor series expansion [53] around a point a as follows:

$$f(t) = \sum_{k=0}^{\infty} c_k (t-a)^k, \quad \text{where } c_k = \frac{f^{(k)}(a)}{k!}$$

and $f^{(k)}$ denotes the k-th derivative of f. Truncating the series to n terms gives a degree $n-1$ polynomial approximation:

$$f(t) \approx \sum_{k=0}^{n-1} c_k (t-a)^k.$$

The accuracy of the approximation improves with an increase in the number of terms n. However, for points far from a, the approximation may require a large number of terms to achieve high accuracy, which increases the multiplications.

Padé approximants. The Padé approximation [51] of a function f is as follows: Given two integers $m \geq 0$ and $n \geq 1$, the *Padé approximant* of order $[m/n]$ is the rational function

$$R(t) = \frac{\sum_{j=0}^{m} a_j t^j}{1 + \sum_{k=1}^{n} b_k t^k},$$

that agrees with $f(t)$ in Taylor expansion about $x = 0$ up to degree $m+n$ (i.e., $f^{(k)}(0) = R^{(k)}(0)$ for $k = 0, \ldots, m+n$). The coefficients a_j and b_k can be determined by solving a system of linear equations derived from matching the Taylor series coefficients of $f(t)$ and $R(t)$ up to degree $m+n$ either using the Wynn's epsilon algorithm or by extended Euclidean algorithm [51].

Gauss–Legendre quadrature. Numerical integration (or *quadrature*) [50] is a collection of techniques that helps in approximating the value of a definite integral, especially useful for integrals without closed forms or to decrease the computational complexity of computing the integral. Although there

are many techniques that allows one to approximate integrals, we will focus on a specific kind of quadrature that satisfies both (1) high accuracy and (2) efficient proving.

At a high level, most quadrature rules for one dimensional integrals take the form

$$\int_a^b f(t)dt \approx \sum_{i=1}^{n} w_i f(t_i)$$

for some pre-determined weights w_i and points t_i. These weights are usually determined by the specific kind of quadrature, but sometimes can be chosen adaptively depending on the function being integrated and its behavior in the interval.

This work uses Gauss–Legendre quadrature, a *numerical integration* method that approximates definite integrals using weighted sums of function values at specially chosen points. Specifically, an order n Gauss–Legendre quadrature can accurately integrate polynomials of degree up to $2n-1$ [48]. The formula is as follows:

$$\int_{-1}^{1} f(t)\,dt \;\approx\; \sum_{i=1}^{n} w_i f(t_i)$$

where t_i are the **roots of the Legendre polynomial** $P_n(t)$, and w_i are the **weights**, determined by the derivative of $P_n(t)$. Both the roots and weights can be precomputed and are independent of the function f. Whenever, the limits of integration are over an interval $[a,b]$, we need perform change of variable to map the interval to $[-1,1]$.

4 Approximation Techniques

In this section, we discuss our approach for efficiently proving non-polynomial functions in zero-knowledge. Before diving into the details of our techniques, we first motivate the need for low-depth circuits for the approximation of non-polynomial functions in the fixed-point arithmetic setting, which we use throughout the paper. We will start with a prototypical example of a non-polynomial function, the sigmoid function, which is used in machine learning as an activation function, and show how existing techniques lead to circuits with multiplicative depth that grows with the desired accuracy. Then, we present our first solution based on Padé approximants, which concretely improves upon other polynomial-based techniques but still has depth that grows with the desired accuracy. We then present our main technique based on numerical integration techniques (specifically Gauss–Legendre quadrature), which allows us to prove the correct computation of the sigmoid function with a circuit of constant multiplicative depth, regardless of the desired accuracy. Finally, we describe how this technique also extends to almost all commonly used non-polynomial functions.

Throughout this section, we denote $\sigma(x) = \frac{1}{1+e^{-x}}$, as a running example to illustrate our techniques. Specifically, we want to prove the following (informal here) statement:

Given an input x and output y, prove that y is a "good" approximaton of $\sigma(x)$

4.1 Multiplicative depth as a proxy for error

As mentioned in the introduction and in Section 3, when emulating fixed-point arithmetic in a finite field $\mathbb{F}_p$, performing any operation incurs some amount of rounding error due to the quantization of real numbers. While addition operations can be performed with minimal error accumulation, multiplication operations can lead to significant error growth depending on the magnitude of the operands, especially when performed iteratively using results of previous multiplications.

In many applications, non-polynomial functions are used repeatedly within larger computations, which inevitably leads to accumulating errors – hence, even a small savings in the error incurred by approximating a non-polynomial function can lead to significant improvements in the overall accuracy of the computation.

There are two techniques that can be used to prove a fixed-point multiplication $\bar{c} = \bar{a} \cdot \bar{b}$ in a zero-knowledge proof system (where $\bar{a}, \bar{b}, \bar{c} \in \mathbb{F}_p$ are the fixed-point encodings of real numbers $a, b, c \in \mathbb{R}$ with scaling factor s):

- **Direct multiplication check**: The circuit computes $\bar{a} \cdot \bar{b}$ and truncates (within the circuit) the lower s bits to obtain $\bar{c}$. This requires a multiplication, as well as truncation logic in the circuit, which is expensive due to the need for bit (or digit) decompositions and range checks. The error incurred is precisely the error described in Section 3 for fixed-point multiplication.

- **Non-deterministic multiplication check**: The prover computes $\bar{c}$ locally and provides it as a witness to the circuit. The circuit then verifies that $|\bar{a} \cdot \bar{b} - \bar{c} \cdot 2^s| < \varepsilon 2^s$ for some small ε, which accounts for the rounding error due to fixed-point encoding. This only requires a single multiplication and a range check in the circuit, making it significantly cheaper than the direct multiplication check. The supported error here is essentially the maximum error bound for fixed-point multiplication (as we need to handle all possible inputs), which can be larger than the direct multiplication check above[1] Motivated by better performance, we use this technique in our constructions as well as implementations of the baseline approximation techniques.

Depth of non-deterministic verification circuits. We can use non-determinism and providing witnesses to effectively reduce the depth of *any* computation to a constant – for example, consider the computation of x^8, which would require a circuit of depth 3, but can be verified in depth 1 by checking *in parallel* all the intermediate multiplications. We note that this

[1] We stress that the honest prover would always provide the right answer here, i.e., there is no resulting error! Cheating provers can take advantage of the slack ε to provide an inaccurate witness by up to ε.

change does **not** reduce the error growth due to the original multiplicative depth – even though the checks are depth 1, the errors in each result compound via the next multiplication as the same terms are used in two or more non-deterministic checks. Hence, the error growth is proportional to the multiplicative depth of the actual computation or the number of iterated multiplications, not the depth of the flattened verification circuit.

4.2 Improving polynomial-based approximations for most functions

A common technique to approximate non-polynomial functions is to use polynomial approximations such as Taylor series, described in Section 3. However, achieving a small approximation error typically requires a polynomial of high degree d, which in turn leads to circuits with $O(\log d)$ multiplicative depth when evaluating the polynomial. Additionally, all polynomial-based approximations have an error that grows large outside the approximation interval, necessitating input range restrictions. This will be the case for all the approximation techniques discussed in this work, and we defer the details of the choices of input ranges to Section 4.5. However, due to the embedding of fixed-point numbers using quantization, we can choose the quantization domain carefully to ensure that all the inputs lie within a small interval around 0 – this is elaborated in our experiments in Section 5.

As an example, we can approximate $\sigma(x)$ using this Taylor series expansion around 0 as follows:

$$\sigma(x) \approx \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480} - \frac{17x^7}{80640} + \frac{31x^9}{1451520}$$

When trying to prove that $y = \sigma(x)$ using this approximation, we need to evaluate a degree 9 polynomial. Since we would like to take advantage of non-determinism, the prover provides additional witness elements $\bar{x}^2, \bar{x}^3, \ldots, \bar{x}^9$ to the circuit, and the circuit checks that these are consistent with $\bar{x}$ using multiplication checks via a logarithmic depth binary tree.

Now, we consider another class of approximations for smooth functions that are not polynomial, but rational functions – Padé approximants. These approximants have the same number of terms as Taylor series, but split between numerator and denominator. Formally, for a function $f(x)$, the (m,n) Padé approximant is defined as

$$R_{m,n}(x) = \frac{P_m(x)}{Q_n(x)} = \frac{a_0 + a_1 x + \cdots + a_m x^m}{1 + b_1 x + \cdots + b_n x^n}$$

where the coefficients a_i, b_j are chosen such that the Taylor series of $R_{m,n}(x)$ matches that of $f(x)$ up to degree $m+n$. The main advantage is that although both Taylor series and Padé approximants have the same number of terms, the degree of the Padé approximant can be made to be *half that of the Taylor series for the same approximation error*. To illustrate this, we can approximate $\sigma(x)$ using the $(5,5)$ Padé approximant as:

$$\sigma(x) \approx \frac{\frac{1}{2} + \frac{x}{4} + \frac{x^2}{18} + \frac{x^3}{144} + \frac{x^4}{2016} + \frac{x^5}{60480}}{1 + \frac{x^2}{9} + \frac{x^4}{1008}}$$

Here, we have the same number of terms (11), but the degree is only 5, which leads to a circuit with fewer iterated multiplications, compared to the Taylor series approximation. However, this cannot be computed directly within the circuit due to the division required, so we again use non-determinism – the prover provides the denominator as an additional witness element, and the circuit checks the following:

Padé Approximant Verification for $\sigma(x)$

Statement: $\bar{x}, \bar{y}$, where $y = \sigma(x)$, and the scaling factor is s.

Witness: Prover computes $\bar{x}^2, \bar{x}^3, \bar{x}^4, \bar{x}^5$, as well as $\bar{p}$ and $\bar{q}$, where $p = P_5(x)$ and $q = Q_5(x)$.

Constraints: The circuit checks:

1. Check if supplied $\bar{x}^i$'s are consistent with $\bar{x}$[2]:

$$\forall\, i \in [2,5],\ |\bar{x}^i \cdot 2^s - \bar{x}^{i-1} \cdot \bar{x}| < \varepsilon_i 2^s$$

2. Check if supplied $\bar{p}$ and $\bar{q}$ are consistent with $\bar{x}$:

$$\left| \bar{p} \cdot 2^s - \sum_{i=0}^{5} a_i \bar{x}^i \right| < \varepsilon_p 2^s$$

$$|\bar{q} \cdot 2^s - (2^s + b_1 \bar{x}^2 + b_2 \bar{x}^4)| < \varepsilon_q 2^s$$

3. Check if supplied $\bar{y}$ is consistent with $\bar{p}$ and $\bar{q}$:

$$|\bar{y} \cdot \bar{q} - \bar{p} \cdot 2^s| < \varepsilon_s 2^s$$

The ε values in the above constraints account for the rounding errors introduced by fixed-point encoding, which are left unspecified in this section (as they depend on the inputs) and are analyzed in detail in Section 4.5. In conclusion, Padé approximants provide a concrete improvement over Taylor series as they lead to circuits with lower multiplicative depth for the same accuracy – however, the depth still grows with the desired accuracy.

4.3 Obtaining constant-depth approximations using numerical integration

We now present our main technique based on numerical integration, which allows us for the first time to prove the computation of $\sigma(x)$ (and various other functions) with a circuit

[2] We use a binary tree structure to verify powers, but do not elaborate here for simplicity.

of constant multiplicative depth, regardless of the desired accuracy. We start with an observation about the exponential function $y = e^x$ and its inverse function $x = \ln y$. We notice that the derivative of $\ln y$ is *algebraic*, i.e., $\frac{d}{dy}\ln y = \frac{1}{y}$ and is easy to work with in a proof system (as we can leverage non-determinism to compute inverses). Based on this observation, we can express the sigmoid function as a definite integral of simple algebraic functions as follows:

$$y = \sigma(x) \iff x = \ln\left(\frac{y}{1-y}\right) \iff x = \int_{1-y}^{y} \frac{1}{t}dt$$
$$\iff x = \int_{-1}^{1} \frac{1}{t + \frac{1}{2y-1}}dt$$

As seen earlier in Section 3, we can use numerical integration techniques to approximate definite integrals using weighted sums of function evaluations. Specifically, we use Gauss–Legendre quadrature to compute roots t_i and weights w_i for some order n (that determines the accuracy of the approximation) to obtain the following approximation:

$$x \approx \sum_{i=1}^{n} \frac{w_i}{t_i + \frac{1}{2y-1}}$$

Given this approximation that only uses additions, multiplications, and inverses of fixed-point numbers, we can now design a circuit to prove the correct computation of $y = \sigma(x)$ using non-determinism as follows:

Gaussian Quadrature Verification for $\sigma(x)$

Statement: $\bar{x}, \bar{y}$, where $y = \sigma(x)$, and the scaling factor is s.

Witness: For a given order of the approximation n (and hence fixed constants t_i, w_i for $i \in [n]$), the prover computes the witness elements $\bar{r}$ and $\bar{h}_i$ for $i \in [n]$, where $r = \frac{1}{2y-1}$ and $h_i = \frac{1}{t_i + r}$.

Constraints: The circuit checks the following:

1. Check if supplied $\bar{r}$ is consistent with $\bar{y}$:

$$|\bar{r} \cdot (2\bar{y} - 2^s) - 2^{2s}| < \varepsilon_r 2^s$$

2. Check if supplied $\bar{h}_i$'s are consistent with $\bar{r}$:

$$\forall\, i \in [n] \quad |\bar{h}_i \cdot (\bar{t}_i + \bar{r}) - 2^{2s}| < \varepsilon_i 2^s$$

3. Check if the approximation holds $\bar{x} \approx \sum_{i=1}^{n} w_i \cdot h_i$:

$$\left|\bar{x} \cdot 2^s - \sum_{i=1}^{n} \bar{w}_i \cdot \bar{h}_i\right| < \varepsilon_s 2^s$$

Applicability of our technique to zero-knowledge backends. We note that, as long as the prover of a zk-SNARK system supports efficient range checks (e.g., via look-ups), there is no other restriction on whether a prover is compatible with our approximation technique– it is purely a "front-end" technique. The most expensive part of the circuit for the prover is the inverse checks for all $i \in [n]$ (which is also the only part that grows with the order n). However, these checks are independent of each other and can potentially be done *in parallel*, leading to a prover time that does not grow with n in practice. The error accumulation is also much smaller than polynomial-based techniques, as the errors from each term in the weighted sum do not compound (see Section 4.5).

4.4 Generalizing to other functions

The technique described in the previous section can be generalized to a wide variety of commonly used non-polynomial functions beyond the sigmoid. The property that we identified for the exponential function – that its inverse function has an algebraic derivative – holds for many other functions as well. Once the function is written in this form, it can be proved using similar techniques. We present these expressions for functions from four different classes of non-polynomial functions: activation functions, trigonometric functions, power functions, and certain non-elementary functions.

- **Activation functions**: These functions are used in machine learning as activation functions in neural networks, to introduce non-linearity. Examples include: Sigmoid, GeLU, etc. These are often based on trigonometric or hyperbolic functions, and hence can be expressed using similar integral representations. Vector-output functions like SoftMax can also be broken down into the individual exponentiations and a normalizing step [42], where each individual exponentiation can be done via numerical integration.
- **Trigonometric functions**: Functions such as $\cos(x), \tan(x), \tanh(x)$, etc., are often used in various applications involving coordinate transformations. These can also be expressed using integral representations by considering their inverse functions (such as $\arccos(x)$, $\arctan(x)$, $\operatorname{artanh}(x)$), which have algebraic derivatives.
- **Power functions**: These functions take the form $f(x) = x^k$ for some fractional k. They are commonly used in financial applications (see Section 5.3). This can also be extended to a multivariate version $f(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} x_i$. Although it is not immediately obvious that these functions satisfy the same property, we can write $y = x^k$ as $\ln y = k \ln x$, and use the integral representation of the logarithm to obtain the desired form. Similarly, it converts the product function to a sum of logarithms, which can be proved using similar techniques and reduces multiplicative depth.
- **Non-elementary functions**: These functions cannot be expressed in terms of elementary functions, but still have well-defined integral representations. Examples include: Error

function (erf(x)), Gamma function ($\Gamma(x)$). These functions are defined as integrals themselves, and hence can be approximated using numerical integration techniques directly.

Table 1: Functions and integral representations

Function	Integral Representation
$y = e^x$	$x = \int_{-1}^{1} \frac{1}{t + \frac{y+1}{y-1}} dt$
$y = \sigma(x)$	$x = \int_{-1}^{1} \frac{1}{t + \frac{1}{2y-1}} dt$
$y = \tanh(x)$	$x = \int_{-1}^{1} \frac{1}{2t + \frac{2}{y}} dt$
$y = \cos(x)$	$x = \int_{-1}^{1} \frac{-1}{\sqrt{1 - \left(\frac{t+1}{2} y\right)^2}} dt$
$y = \tan(x)$	$x = \int_{-1}^{1} \frac{1}{\frac{y}{2}(t+1)^2 + \frac{2}{y}} dt$
$y = x^k$	$\int_{-1}^{1} \frac{1}{t + \frac{y+1}{y-1}} dt = k \int_{-1}^{1} \frac{1}{t + \frac{x+1}{x-1}} dt$
$y = \text{erf}(x)$	$y = \frac{x}{\sqrt{\pi}} \int_{-1}^{1} e^{-\left(\frac{t+1}{2} x\right)^2} dt$

In some cases, although there exist algebraic derivatives for these functions' inverses, there may be known identities that allow us to express them in simpler forms. For instance, $y = \cos(x) \iff x = \arccos(y) \iff x = \int_{-1}^{y} \frac{-1}{\sqrt{1-t^2}} dt$, but verifying this requires computing square roots for each term in the summation, which is expensive. However, we can use the identity $\cos(x) = \frac{1}{\sqrt{1+\tan^2(x)}}$ to first compute $\tan(x)$ using our technique (which does not require square roots) and then compute $\cos(x)$ using a single multiplication and inverse, leading to a more efficient circuit. We present a generic circuit for proving the correct computation of all these functions:

Gaussian Quadrature Verification for $f(x)$

Statement: $\bar{x}, \bar{y}$, where $y = f(x)$, and the scaling factor is s.

Witness: For a given order of the approximation n (and hence fixed constants t_i, w_i for $i \in [n]$), the prover computes inverses of the denominator d_i for $i \in [n]$ used in the integral representation of $f(x)$ as well as any additional intermediate values required to compute d_i.

Constraints: The circuit checks the constraints:

1. Check the intermediate values are consistent with $\bar{y}$ (if any). (For example, for $\sigma(x)$, we check $\bar{r}$ is consistent with $\bar{y}$ as shown earlier.)
2. Check if supplied inverses are consistent with $\bar{y}$ and any intermediate values. (For example, for $\sigma(x)$, we check $\bar{h}_i$'s are consistent with $\bar{r}$ as shown earlier.)
3. Check if the approximation $\bar{x} \approx \sum_{i=1}^{n} w_i \cdot d_i^{-1}$ holds.

In the above circuit, the specific details of the witness elements and constraints in steps 1 and 2 depend on the function $f(x)$ being proved, while step 3 remains the same across all functions. Step 1 is either independent of the order n or is just parallel multiplications that do not increase the depth. Step 2 involves n inverse checks that are all independent of each other and can be parallelized as well. Hence, the overall circuit has constant multiplicative depth regardless of the desired accuracy (determined by n).

4.5 Error analysis

Here, we elaborate on the error analysis for the various approximation techniques discussed in this section. We analyze the sources of error for each method and how error accumulates throughout the circuit. In all cases, we assume that the function inputs lie in the interval $[-1, 1]$ (which can be ensured by appropriate quantization domain choice, as discussed earlier, corresponding to a change of units for the calculation).

4.5.1 Quadrature-based approximation

In this technique, the errors can compound from three sources; one for each check in the circuit Section 4.3. However, these do not compound in a straightforward manner as in polynomial approximations, due to the additive nature of the final summation as well as the inverse computations which can slightly mitigate the overall error. In the actual circuit, we modify the checks slightly to increase the numerical stability of the computations. This is required because the denominators as given in the circuit can be very small (for instance, when y is close to 0.5 in the sigmoid function, $2y-1$ is close to 0). The multiplication check would then involve multiplying by a very large number, which can amplify the errors significantly (as shown in Section 3).

To mitigate this, we write each term in the summation instead as $\frac{1}{t_i + \frac{1}{2y-1}} = \frac{2y-1}{2yt_i - t_i + 1}$, and modify the circuit to check for the consistency of the denominators in the first step, by checking that $y \cdot t_i$ is consistent with the supplied denominator witnesses for all i. This increases the computation for the prover slightly, but significantly improves the numerical stability and the worst-case error bounds we can obtain as the denominators are no longer close to zero. Before analyzing the errors, note that the term provided by the prover as witness has an intrinsic rounding error of at most 1 due to fixed-point encoding (however, in the circuit we are restricted to checking bits, which reduces the granularity of the error we can account for – in the experiments in Section 5 the actual errors are obtained by testing random inputs from the range).

- **Step 1**: This step now checks products p_i of the form $y \cdot t_i$ for all $i \in [n]$. The error incurred here is similar to that of

a fixed-point multiplication for each product. Since all the inputs here are bounded within $[-1,1]$, the error incurred is $\varepsilon_{p_i} - (\varepsilon_y * t_i + \varepsilon_{t_i} * y) < 2$ bits in the worst case, where $\varepsilon_y, \varepsilon_{t_i}$ are the intrinsic rounding errors of y and t_i respectively (at most 1 bit each). Hence, the total error from this step is at most 2 bits for each multiplication (as per Section 3) – this is also substantiated in our experiments in Section 5.

- **Step 2**: This step checks the inverses of the denominators $d_i = 2yt_i - t_i + 1$ using the p_i values from step 1. The multiplication check itself incurs an error of at most 2 bits. Combined with the previous step, the total error incurred considering both checks is 3 bits.

- **Step 3**: This step computes the weighted sum of the inverses from step 2. Although additions usually incur errors additively from each term, here each term is of the form $\frac{w_i}{d_i}$, where w_i are pre-computed weights that sum to 2 (for Gauss-Legendre quadrature over $[-1,1]$). Hence, the total additional error from compounding is just a single extra bit.

4.5.2 Polynomial and Padé approximation

For polynomial-based approximations (including Taylor series and Padé approximants), the total error can be decomposed into two components: approximation error and fixed-point arithmetic error. The approximation error arises from the inherent inaccuracy of using a polynomial to approximate a non-polynomial function – this mainly depends on the degree of the polynomial and is the contributing factor to the error growing with the desired accuracy. The fixed-point arithmetic error arises from the rounding errors incurred during the multiplications and additions in the polynomial evaluation, but is usually dominated by the approximation error for high-degree polynomials.

When using non-deterministic multiplication checks, each multiplication incurs a worst-case error of at most 2 bits (as per Section 3). For a degree d polynomial, there are $d-1$ multiplications required to evaluate it using Horner's method, leading to a total fixed-point arithmetic error of at most $2(d-1)$ bits. In this work, we optimize this further by using witness elements for powers of x as described earlier, which reduces the number of iterated multiplications (the depth) to at most $2\lfloor \log(d-1) \rfloor$ (for checking all the powers). However, evaluating the polynomial using these powers still requires d additions and d multiplications (to multiply each power by the corresponding coefficient). This again compounds the error, but since each term has a different error, only the highest error term (which is the largest power of x) contributes to the error. Hence, the total fixed-point arithmetic error for polynomial-based approximations is at most $2\lfloor \log d \rfloor + 1$, which grows with the degree of the polynomial.

The same argument applies to Padé approximants as well, albeit with half the degree for the same approximation error. The inverse computation in the Padé approximant circuit incurs an minor additional error, but is dominated by the other error sources. Hence, the total fixed-point error for Padé approximants is at most $2\lfloor \log d \rfloor + 1$ bits, where $d = \max(m,n)$ for the (m,n) Padé approximant. To optimize our circuits, we use $m = n$ to minimize the degree and hence the error.

4.5.3 Lookup arguments

Typically, lookup arguments do not incur any additional approximation error as they are exact representations of the function being computed. However, when using fixed-point encodings, there is an inherent rounding error due to quantization. This error is at most 1 bit for each input and output value in the lookup table. However, for certain functions that are decomposable [5, 40] ($y = e^x$ can be decomposed as $y = e^{\Sigma_i x_i k^i}$ where each x_i is a digit in base k), we can use multiple smaller lookup tables to compute the function in parts and then combine the results by multiplication. The error depends on the number of "chunks" the function is decomposed into and the iterated multiplications required to combine the results. From an analysis similar to that for polynomial approximations, we have a total error of at most $2\log m + 1$ bits.

4.5.4 Piecewise Linear Approximation

For piecewise linear approximations, we fix the desired error loss to 5 bits and increase the number of segments until the approximation error is within this bound. The fixed-point arithmetic error incurred during the evaluation of the linear function in each segment is at most 2 bits (due to a mul. and add.), but does not significantly contribute to the overall error.

5 Experimental Evaluation

In this section, we evaluate the accuracy and performance of our proposed techniques and compare them with relevant baselines. Furthermore, we also explore the applicability of our methods in two real-world applications.

All our implementations use Noir [31], a high-level Rust-like language for writing zk-SNARK circuits. Noir uses the Barretenberg backend, which is based on UltraPLONK [14]. Both Noir and Barretenberg are production-ready, state-of-the-art tools available to the zk-SNARK community. Our choice of the Noir-based toolkit is motivated by the following factors: First, it allows us to rapidly implement and compare different techniques on equal footing. Second, it provides native support for custom lookup tables [13], which is essential for implementing our lookup-based baseline. Finally, it supports range checks via lookups rather than binary decomposition, a requirement for both the remaining baselines and our proposed techniques. We use the BN254 curve with KZG [24] commitments for our experiments. All experiments were conducted on a 2023 MacBook Pro (Apple M3 Pro, 36 GB RAM,

Table 2: Microbenchmark comparison. For lookup and piecewise linear approaches, "–" indicates that the required table would be too large to be efficient. For Taylor series, "–" indicates that the approximation error is significantly worse than the other approaches and therefore not meaningful to compare. We abbreviate Gaussian–Legendre quadrature as GL. All results report **amortized** prover time (computed with 100 function calls each). Verifier time ranges from 3–5 ms across all experiments, and proof sizes range from 7.6–10.7 KiB.

		Error (bits)					Prover Performance (ms)				
f	Quant. (bits)	Lookups	Piecewise Linear	Taylor Series	**Ours (Padé)**	**Ours (GL)**	Lookups	Piecewise Linear	Taylor Series	**Ours (Padé)**	**Ours (GL)**
tan	16	2	5	8	2	4	20.20	4.04	1.23	1.00	1.06
	32	–	5	10	4		–	14.10	1.88	1.06	1.50
	64	–	–	12	6		–	–	4.26	1.51	2.49
	120	–	–	–	6		–	–	–	1.79	4.30
cos	16	2	5	4	2	4	13.50	4.11	0.99	1.00	1.13
	32	–	5	6	4		–	12.60	1.17	1.05	1.13
	64	–	–	8	6		–	–	1.69	1.48	2.52
	120	–	–	10	6		–	–	2.63	1.77	4.42
tanh	16	1	5	6	2	4	1.10	3.98	1.13	1.00	1.20
	32	3	5	10	4		1.50	11.20	1.84	1.05	1.71
	64	5	–	12	6		2.33	–	4.19	1.49	2.86
	120	7	–	–	6		2.98	–	–	1.78	6.85
$\sigma(x)$	16	1	5	4	2	4	1.05	4.18	1.00	0.98	1.09
	32	3	5	6	4		1.48	6.59	1.21	1.03	1.26
	64	5	–	10	4		2.28	–	2.38	1.23	2.43
	120	7	–	12	6		2.85	–	4.34	1.80	4.09
GeLU	16	1	5	6	4	5	1.18	4.04	1.05	1.03	1.19
	32	3	5	8	4		1.58	11.40	1.58	1.11	1.68
	64	5	–	10	6		2.46	–	2.77	1.57	2.77
	120	7	–	12	8		3.96	–	4.39	2.54	4.82
x^k	16	1	5	4	2	4	1.48	6.98	1.15	1.10	1.20
	32	3	5	6	4		1.68	20.80	1.59	1.31	1.77
	64	5	–	8	4		2.59	–	2.54	1.74	2.95
	120	7	–	8	6		4.36	–	4.14	2.62	6.79
e^{-x}	16	1	5	4	2	4	1.04	3.98	1.00	0.98	1.09
	32	3	5	6	4		1.46	11.50	1.20	1.03	1.53
	64	5	–	8	4		2.30	–	1.76	1.20	2.49
	120	7	–	10	6		2.82	–	2.65	1.84	4.31
erf(x)	16	2	5	6	4	5	14.20	3.95	1.07	1.01	2.37
	32	–	5	8	6		–	11.40	1.56	1.12	4.75
	64	–	–	10	8		–	–	2.49	1.65	23.20
	120	–	–	10	8		–	–	4.11	2.49	86.90

and 12 cores), with the default parallelism options offered in Barretenberg.

Real-valued inputs to the tested functions are drawn uniformly at random between $[-1,1]$ following the common configuration for approximation-based techniques; as noted in Section 4.5, this can be achieved by rescaling when deciding the quantization domain – this just corresponds to a change of units for the input values. Witness generation is performed using arbitrary-precision arithmetic (`mpmath`) in Python to ensure that the computed values are accurate enough for error measurement for all quantization domains.

5.1 Microbenchmarks

For our microbenchmarks, we compare our two proposed techniques against lookup tables, piecewise linear approximations, and Taylor series-based methods. Specifically, we implement the following non-polynomial functions: trigonometric functions (tan, cos, tanh), activation functions (sigmoid and GeLU), power functions (x^k), and non-elementary

functions (erf and e^{-x}), thus covering a wide variety of non-polynomial functions. In total, we implement and evaluate *eight* non-polynomial functions using *five* different approximation approaches, including the two proposed in this work. We present the results for these functions in Table 2. In all our microbenchmarks, we measure the **cumulative prover time** for proving 100 calls to the function and compute the **amortized** time per call in the table.

Experimental parameters. We carefully select the parameters for each function and approximation method to ensure a fair comparison. For lookup tables, in all our experiments, whenever the function is decomposable, we set each subtable to have 2^8 entries. Consequently, as the quantization domain grows, we increase the number of sub-tables accordingly. For piecewise linear approximations and Taylor series, we adjust the number of segments and order of the polynomial, respectively, such that the accuracy loss is comparable to our proposed methods, while still being efficient. For Padé approximations, we set the order of the numerator and denominator to be the same, in order to achieve the best prover time for a given error. For Gauss–Legendre quadrature, we select the order of the approximation to ensure a reasonable trade-off between accuracy and performance.

Accuracy and performance. Observe from Table 2 that as the quantization domain size increases, the error of all approaches except our proposed Gauss–Legendre quadrature increases. This is because of the constant multiplicative depth of our proposed integration-based method. For larger quantization domains (64 bits and above), our Gauss–Legendre quadrature approach has accuracy of $1-8$ bits better than all baselines, which translates to $2-256\times$ better absolute error. Thus, our solution emerges as a clear optimal choice to guarantee high accuracy while remaining efficient across different quantization domains. For smaller quantization domains (16 and 32 bits), our Gauss–Legendre quadrature approach is up to $19\times$, $11.7\times$, and $1.25\times$ faster than lookup tables, piecewise linear approximations, and Taylor series methods, respectively, while being comparable for smaller domains.

On the other hand, our Padé-based approach achieves superior performance (at least $2-3\times$) and accuracy over all baselines for quantization domains of 64 bits and above. Since the lookup table and piecewise linear are too inefficient for larger quantization domains, we expect the gain of our Padé-based approach to be even larger. For smaller quantization domains (16 and 32 bits), our Padé-based approach is up to $20.1\times$, $15.8\times$, and $1.7\times$ faster than lookup tables, piecewise linear approximations, and Taylor series methods, respectively.

Verifier time and proof size. Verifier time ranges from 3–5 ms across all experiments, and proof sizes range from 7.6–10.7 KiB, for all experiments that did not time out. This is as expected since all approaches are implemented with the same zk-SNARK protocol (UltraPLONK via Barretenberg).

Table 3: Application Benchmarks

	Quant. (bits)	Scheme	Error (bits)	Perf. (ms)
Orbital Mechanics	120	Taylor Series	10	98
		Ours (Padé)	8	98
		Ours (GL)	5	96
Bancor Protocol	120	Taylor Series	10	92
		Ours (Padé)	8	75
		Ours (GL)	4	74

5.2 Application #1: Orbital mechanics

Consider the problem of locating the position of a celestial body in its orbit around another body at a given time. This is a fundamental problem in astronomy where accurate calculations are necessary to understand the trajectories of planets, satellites, asteroids, etc. In fact, space agencies like NASA observe the positions of celestial bodies and use enormous computational resources to compute their trajectories with high precision [35] and release them publicly. To verify these calculations, one needs to spend similar computational resources. However, with zk-SNARKs, proving the correctness of these calculations can be delegated to a powerful prover.

At the core of computing orbital positions lies the well-known *Kepler's equation* [49], $M = E - e\sin(E)$, which relates the mean anomaly M to the eccentric anomaly E for a given orbital eccentricity e. Solving Kepler's equation for E given M and e is non-trivial, as there is no closed-form solution with elementary functions. Numerical methods, such as Newton-Raphson, are typically employed to find approximate solutions for E, and take several iterations to converge to a satisfactory precision. However, once computed, we can use our techniques to prove the correctness of the solution.

Consider proving the following statement for the well-known Halley's comet: *"Given the orbital period P, eccentricity e and the time since last perihelion dt, (x,y) represents the current position of the comet since it was last closest to the sun."* The circuit for this is described in Appendix C.

Benchmarking results. We present the accuracy and performance results for the above application in Table 3. Due to slow prover times of lookup table and piecewise linear approach for large domains, we limit the comparison of our work to Taylor series-based solution Table 2. The performance our approaches are comparable with Taylor series-based approach, while achieving $2-5$ bits better accuracy, which translates to $4-32\times$ better absolute error. All approaches achieve similar verifier times of around 4 ms and proof sizes of $7-8$ KiB.

5.3 Application #2: DeFi – Dynamic Pricing

Consider the Bancor protocol [23], which is a popular DeFi protocol that enables automated token exchange without an order book to match buyers and sellers. Instead, it relies on a

liquidity pool, a pool of locked tokens, that provides the liquidity whenever there is a request to exchange tokens. However, an algorithmic pricing mechanism determines the exchange rate between tokens.

At a high level, the protocol works as follows for exchanging reserve token E and token T: (1) Users lock token E into a smart contract in exchange for a small incentive. (2) When a user wishes to sell E to buy T, the smart contract computes the exchange rate using a pricing function based on the demand and supply of E and T in the liquidity pool. The amount of T received for a given payment of E computed as follows:

$$T = S_0\left(\left(1+\frac{E}{R_0}\right)^F - 1\right)$$

where S_0 and R_0 are the outstanding supply of tokens and reserves, respectively, before the transaction, and F be the constant fractional reserve ratio.

The above pricing function is applied over high precision real numbers, and involves computing a power function with a fractional exponent. Currently, smart contracts do not support floating-point arithmetic natively, and hence, the pricing function is implemented using fixed-point arithmetic with approximations for the power function. Moreover, we can envision a privacy preserving version of such DeFi protocols, where the amounts of tokens exchanged and liquidity pool sizes are kept private. These reasons make it an ideal candidate for delegating the work to the prover, who can compute the pricing function off-chain and provide a proof of its correctness. With our Gauss–Legendre quadrature-based technique, this can be proved efficiently and accurately.

Consider proving the following statement: *"Given the liquidity pool parameters (S_0, R_0, F) and the amount of reserve token E paid, the amount of token T received is computed correctly according to the pricing function."* The circuit for this statement is described in Appendix D.

Benchmarking results. We present the accuracy and performance results for the above application in Table 3. As argued in Section 5.2, due to slow prover times of lookup table and piecewise linear approach for large domains, we limit the comparison of our work to Taylor series-based solution Table 2. The performance of our approaches is comparable with Taylor series-based approach, while achieving $2-6$ bits better accuracy, which translates to $4-64\times$ better absolute error. All the approaches achieve similar verifier times of around 4 ms and proof sizes of around $7-8$ KiB.

Applications to Uniswap V4. As opposed to fixed curves such as the constant-product $(x \cdot y = k)$ and constant-sum $(x+y=k)$ used in Uniswap V3 [4], the new Uniswap V4 [3] supports usage of custom curves with different behavior, with one such high-visibility example being [6]. This work demonstrates two highly non-linear curves:

- A weighted product curve, $R_A^{w_A} \cdot R_B^{w_B} = k$ for $w_A + w_B = 1$ – this will use similar techniques as our previous application with fractional exponents.
- Lambert curve, $y = \frac{1}{c} \cdot W\left(\frac{c \cdot e^{(2c-x)}}{x}\right)$, where W is the Lambert W function that is amenable to verification-based approaches, as it has the property $y = W(x) \iff ye^y = x$ that can be easily checked using our approximation for the exponential function.

6 Conclusion & Discussion

In this work, we proposed the use of two approximation techniques that have not been adopted before for efficiently proving the correct evaluation of non-polynomial functions over real numbers, namely, Gauss–Legendre quadrature and Padé approximations. These approaches can be used to approximate diverse types of non-polynomial functions, such as trigonometric functions, activation functions, power functions (x^k), and other non-elementary functions. Our analysis and experimental evaluation show that they significantly outperform prior methods for arbitrary domains in terms of accuracy and zk-prover performance. A core feature of our approach that we go from proving statements using multiplicative accumulations to additive accumulations, which are significantly better when considering error accumulation in fixed-point arithmetic. This is what allows us to achieve circuits with constant multiplicative depth regardless of the desired accuracy. This not unlike the line of works in lookup arguments that recent advances moved from grand product arguments [8] for proving multi-set inclusion to sum-check style protocols that use log-derivative arguments [20].

One remark is that, since our techniques heavily exploit the non-determinism supported by zk-SNARKs (as we represent x in terms of y instead of approximations that compute y as a function of x), they would not be suitable for other cryptographic applications, such as privacy-preserving ML techniques that use secure multi-party computation (MPC) that cannot leverage non-determinism.

Finally, we believe our work leaves plenty of room for further research in the area. In particular, our approximation techniques rely on creating wide circuits with constant multiplicative depth. This provides an opportunity for a custom sumcheck based back-end to take advantage of the potential for parallelism and perform the sum-check efficiently to further improve the prover time.

A Ethics Considerations

Stakeholders and Impacts: The primary stakeholders are researchers and practitioners in cryptography, zero-knowledge machine learning and verifiable computing. This work advances the understanding of non-linear functions and more accurate techniques to prove them in zero-knowledge, and may benefit organizations seeking to improve the security of their proof systems for non-linear functions. This work

presents an efficient proof system for non-linear functions that are used in various applications, including machine learning, astronomy and decentralized finance. The research does not involve human subjects, does not collect data from users, and does not deploy systems that could directly impact individuals.

These technologies are also used by companies focused on developing products for verifiable machine learning. By improving accuracy and reducing potential adversarial influence on the proof system, which is a main focus of our work, we help improve robustness of existing tools.

Ethical Considerations: Following the principles of beneficence and respect for law and public interest, this research aims to reduce adversarial effects on theoretical and deployed proof systems. The publication of this work poses no foreseeable ethical concerns, as it provides theoretical foundations and does not enable malicious applications beyond what is already possible with existing zero-knowledge proof techniques. We have also provided a formal analysis bounding the maximum possible influence of any adversary that attacks our system.

Decision to Publish: Given the absence of human subjects or sensitive data, and the potential benefits to security infrastructure, we determined that publication is ethically appropriate and beneficial to the research community. We believe that sharing our findings will contribute to the advancement of knowledge in the field of cryptography.

B Open Science

We make the artifact available at a permanent location online (https://zenodo.org/records/20246456), including:

- **Source code**: Implementation of our Gauss-Legendre quadrature and Padé approximations in Noir/Barretenberg, alongside baseline approaches.
- **Evaluation scripts**: Scripts for generating witnesses, running the prover, and profiling constraints. These reproduce the performance experiments from Section 5, including the microbenchmarks for non-polynomial functions (Table 2), high-accuracy DeFi and astronomy applications (Table 3), and from Appendix E, including circuit size comparisons (Tables 4 and 5). We note that the accuracy claims in the tables were computed manually and are not part of the artifact.
- **README.md**: Instructions for installing dependencies, setting up the environment, and reproducing the claimed evaluation results.

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019.

[2] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-Knowledge Proofs of Training for Deep Neural Networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, 2024.

[3] Hayden Adams, Moody Salem, Noah Zinsmeister, Sara Reynolds, Austin Adams, Will Pote, Mark Toda, Alice Henshaw, Emily Williams, and Dan Robinson. Uniswap v4 core [draft], 2023.

[4] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. *Tech. rep., Uniswap, Tech. Rep.*, 2021.

[5] Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024.

[6] Auralshin. Custom curves: Custom invariant curve hooks for uniswap v4. ETHGlobal Bangkok Hackathon, 2024. 3rd Place Winner - Uniswap Foundation Category.

[7] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[8] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In *Advances in Cryptology – ASIACRYPT 2018*, 2018.

[9] Bing-Jyue Chen, Lilia Tang, and Daniel Kang. ZKTorch: Compiling ML Inference to Zero-Knowledge Proofs via Parallel Proof Accumulation, 2025.

[10] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-knowledge location privacy via accurate floating-point snarks. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3440–3459. IEEE, 2025.

[11] Ben Fisch, Daniel Freund, and Moni Naor. Physical Zero-Knowledge Proofs of Physical Properties. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 313–336, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[12] Alexander Frolov, Hal Triedman, and Ian Miers. zk-cookies: Continuous anonymous authentication for the web. Cryptology ePrint Archive, Paper 2025/1938, 2025.

[13] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315, 2020.

[14] Ariel Gabizon and Zachary J Williamson. The turboplonk program syntax for specifying snark programs. In *Proceedings of the 3rd ZKProof Workshop*, 2020.

[15] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with Zero-Knowledge Proofs of Training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, 2023.

[16] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct Zero Knowledge for Floating Point Computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[17] Carl Friedrich Gauss. *Methodus nova integralium valores per approximationem inveniendi*. 1823. Original Latin manuscript on Gaussian quadrature, published in Commentationes Societatis Regiae Scientiarum Gottingensis, Vol. 3.

[18] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 4:61ś79, 2024.

[19] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. Llama: A low latency math library for secure inference. *Proceedings on Privacy Enhancing Technologies*, 4:274–294, 2022.

[20] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. *IACR Cryptol. ePrint Arch.*, page 1530, 2022.

[21] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[22] J Harrison, T Kubaska, S Story, and P Tang. The computation of transcendental functions on the ia-64. *Intel Technology Journal Q*, 4.

[23] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. Bancor protocol. *Continuous Liquidity for Cryptographic Tokens through their Smart Contracts. Available online: https://storage. googleapis. com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en. pdf (accessed on 6 June 2020)*, 2017.

[24] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, 2010.

[25] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

[26] Kaarel August Kurik and Peeter Laud. Novel approximations of elementary functions in zero-knowledge proofs. In *International Conference on Availability, Reliability and Security*, pages 165–182. Springer, 2025.

[27] Lagrange-Labs. Deep prove, 2025. Accessed: 2025-11-01.

[28] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, 2021.

[29] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. An efficient and extensible zero-knowledge proof framework for neural networks. Cryptology ePrint Archive, Paper 2024/703, 2024.

[30] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[31] noir-lang contributors. Noir: A domain-specific language for zero-knowledge proofs. https://noir-lang.org, 2025. Accessed: 2025-09-08.

[32] Henri Eugène Padé. Sur la représentation approchée d'une fonction par des fractions rationnelles. *Annales scientifiques de l'École Normale Supérieure*, 9:1–93, 1892. 3e série, suppl., Thèse de doctorat.

[33] Dong Pan, Kezhen Liu, Bingtao Li, Yongsheng Zheng, and Jiren Ma. zkTaylor: Zero Knowledge Proofs for Machine Learning via Taylor Series Transformation. In *Proceedings of the 2024 2nd International Conference on Advances in Artificial Intelligence and Applications*, AAIA '24, 2025.

[34] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023.

[35] Ryan S. Park, William M. Folkner, James G. Williams, and Dale H. Boggs. The JPL Planetary and Lunar

Ephemerides DE440 and DE441. *The Astronomical Journal*, 2021.

[36] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013.

[37] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkgpt: An efficient non-interactive zero-knowledge proof framework for llm inference. In *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

[38] Arman Riasi, Haodi Wang, Rouzbeh Behnia, Viet Vo, and Thang Hoang. Zero-Knowledge AI Inference with High Precision. In *Proceedings of the 2025 on ACM SIGSAC Conference on Computer and Communications Security*, 2025.

[39] César Sabater and Jan Ramon. Zero knowledge arguments for verifiable sampling. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

[40] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the Lookup Singularity with Lasso. In *Advances in Cryptology – EUROCRYPT 2024*, 2024.

[41] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast (2+1)-pc or z2n via (2,2)-dpfs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2143–2157, New York, NY, USA, 2023. Association for Computing Machinery.

[42] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4405–4419, 2024.

[43] Gefei Tan, Adrià Gascón, Sarah Meiklejohn, Mariana Raykova, Xiao Wang, and Ning Luo. Founding zero-knowledge proofs of training on optimum vicinity. In *Proceedings of the 2025 on ACM SIGSAC Conference on Computer and Communications Security*, 2025.

[44] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.

[45] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022.

[46] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient multiparty computation toolkit, 2016. Available at https://github.com/emp-toolkit.

[47] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[48] Wikipedia contributors. Gaussian quadrature — Wikipedia, the free encyclopedia, 2025.

[49] Wikipedia contributors. Kepler's equation — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kepler%27s_equation&oldid=1309306225, 2025. [Online; accessed 12-November-2025].

[50] Wikipedia contributors. Numerical integration — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Numerical_integration&oldid=1304037985, 2025. [Online; accessed 13-November-2025].

[51] Wikipedia contributors. Padé approximant — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pad%C3%A9_approximant&oldid=1318533761, 2025. [Online; accessed 13-November-2025].

[52] Wikipedia contributors. Piecewise linear function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Piecewise_linear_function&oldid=1292516852, 2025. [Online; accessed 14-November-2025].

[53] Wikipedia contributors. Taylor series — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Taylor_series&oldid=1320116265, 2025. [Online; accessed 14-November-2025].

[54] Wikipedia contributors. Transcendental function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Transcendental_function&oldid=1327837638, 2025. [Online; accessed 23-December-2025].

[55] Jiahua Xu and Nikhil Vadgama. From banks to defi: the evolution of the lending market. In *Enabling the Internet of Value: How Blockchain Connects Global Businesses*, pages 53–66. Springer, 2022.

[56] Dmytro Zakharov, Oleksandr Kurbatov, Artem Sdobnov, Lev Soukhanov, Yevhenii Sekhin, Vitalii Volovyk, Mykhailo Velykodnyi, Mark Cherepovskyi, Kyrylo

Baibula, Lasha Antadze, Pavlo Kravchenko, Volodymyr Dubinin, and Yaroslav Panasenko. Bionetta: Efficient Client-Side Zero-Knowledge Machine Learning Proving, 2025.

[57] zkonduit. Ezkl, 2025. Accessed: 2025-11-01.

C Circuit: Orbital mechanics

Gaussian Quadrature Verification for Kepler's Equation

Statement: Given the orbital parameters (P, e, dt) and the position (x, y), prove that (x, y) is the correct position of the celestial body at time dt.

Witness: The witness consists of the solution E to Kepler's equation, the axes a, b of the elliptical orbit, the eccentricity square root $\sqrt{1-e^2}$ as well as $sin(E)$ and $cos(E)$ values used in the computation.

Constraints: The circuit checks the following constraints:

1. Verify the computation of mean anomaly $M \cdot P = 2\pi dt$
2. Verify that the provided square root $\sqrt{1-e^2}$ is correct using the relation $(\sqrt{1-e^2})^2 + e^2 = 1$.
3. **Ellipse Parameters:** Verify the semi-major axis a and semi-minor axis b of the elliptical orbit using the relations $a^3 = P^2$ and $b = a\sqrt{1-e^2}$.
4. **Trigonometric Values:** Verify the correctness of $cos(E)$ using the quadrature technique to ensure high accuracy, and that $sin(E)$ is computed correctly using the identity $sin^2(E) + cos^2(E) = 1$.
5. **Kepler's Equation:** Verify that the provided E satisfies Kepler's equation $M = E - e\sin(E)$.
6. **Position Calculation:** Finally, verify that the position (x, y) is computed correctly using the parametric equations: $x = a(cos(E) - e)$ and $y = bsin(E)$.

D Circuit: DeFi – Dynamic Pricing

Gaussian Quadrature Verification for DeFi Dynamic Pricing

Statement: Given the liquidity pool parameters (S_0, R_0, F) and the amount of reserve token E paid, prove that the amount of token T received is computed correctly according to the pricing function.

Witness: The witness consists of the intermediate values used in the computation, including the value of $\frac{E}{R_0}$ and its power with exponent F.

Constraints: The circuit checks the following constraints:

1. **Division:** Verify the computation of $\frac{E}{R_0}$ by checking that $E = R_0 \cdot \frac{E}{R_0}$.
2. **Power Function:** Verify the computation of $\left(1 + \frac{E}{R_0}\right)^F$ using Gaussian quadrature.
3. **Final Calculation:** Finally, verify that the amount of token T received is computed correctly using the relation $T = S_0\left(\left(1 + \frac{E}{R_0}\right)^F - 1\right)$.

E Fixed-point vs Floating-point

Although absolute prover times are incomparable between different works due to the usage of different backends and hardware, we can compare the circuit sizes – these can be R1CS constraints or Plonk gates depending on the arithmetization used. We provide two experiments to demonstrate the speedup that can be obtained while using fixed-point arithmetic as opposed to floating-point arithmetic. We use a scaling factor of 64 in these experiments to obtain a similar precision to that of double-precision floating-point arithmetic (which is bounded by machine epsilon).

Our first experiment in Table 4 is for arithmetic gadgets (additions, multiplications) where we compare the constraint costs in our setting to that of [10], which faithfully implements the IEEE-754 standard floating-point operations and is the state-of-the-art in its regime. As in [10], we also utilize lookup tables, and hence amortize our cost over 10^4 operations to present a fair comparison. We also compare the constraint costs for the GeLU activation function with [38] in Table 5 respectively. In both cases, we see that proving correctness of fixed-point arithmetic is significantly cheaper than floating-point arithmetic.

Table 4: Arithmetic Operations (Amortized constraint cost)

Operation	Our Work	[10]	Improvement
ADD	2.25	85	37.78×
MUL	4.56	64	14×

Table 5: GeLU PLONK gate cost

Size	Our Work	[38]	Baseline (from [38])
$n = 1$	3.5×10^3	1.7×10^4	$1.7 \cdot 10^7$
$n = 16$	7.1×10^3	2.3×10^5	2.8×10^8
$n = 256$	6.4×10^4	3.2×10^6	4.4×10^9