USENIX Security '26 Artifact Appendix: Efficiently Provable Approximations for Non-Polynomial Functions

Sriram Sridhar[1] Shravan Srinivasan[2] Dimitrios Papadopoulos[3] Charalampos Papamanthou[2,4]

[1]University of California, Berkeley [2]Lagrange Labs
[3]Hong Kong University of Science and Technology [4]Yale University

A Artifact Appendix

A.1 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 decreases the *adversarial error* (the maximum number of lower-order bits a cheating prover can manipulate undetected) up to the 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 adversarial error leads to increased multiplicative depth – the main factor determining the adversarial error growth of an approximation. We implement and evaluate our approach in Noir/Barretenberg, and we obtain absolute adversarial 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 adversarial errors for high-accuracy applications in DeFi and astronomy that require non-polynomial functions, again obtaining adversarial errors $4-64\times$ lower than the baseline approximations.

This submission is the artifact for the above paper. It contains source code corresponding to the experiments in Tables 2-5 of the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

No ethical concerns. The artifact does not collect personal data, access network resources, or disable any security mechanisms. It performs only local computation.

A.2.2 How to access

The artifact is available at https://zenodo.org/records/20246456.

A.2.3 Hardware dependencies

None. The artifact was tested on an Apple M3 Pro (12 cores: 6 performance + 6 efficiency) with 36 GB RAM, but should run on any machine with sufficient memory. No specialized hardware (GPU, FPGA, etc.) is required.

A.2.4 Software dependencies

- Python 3 (tested with 3.13)
- Python packages: `sympy==1.14.0`, `toml==0.10.2` (listed in `requirements.txt`)
- Noir (tested with `nargo` v1.0.0-beta.21), installed via `noirup`
- Barretenberg backend (tested with `bb` v5.0.0), installed via `bbup`

A.2.5 Benchmarks

None. All inputs are generated programmatically by the witness generation scripts.

A.3 Set-up

A.3.1 Installation

1. Clone the repository:

```
# Download and extract from Zenodo:
# https://zenodo.org/records/20246456
cd non-linear-artifact
```

2. Install Python dependencies:

```
pip3 install -r requirements.txt
```

3. Install Noir and Barretenberg:

```
noirup
bbup
```

A.3.2 Basic Test

Run a single experiment to verify the setup:

```
python3 scripts/gl_gen.py sigmoid 100 8 16
python3 scripts/run_prover.py gl_quad
```

This generates a witness for an order-8 Gauss-Legendre quadrature approximation of the sigmoid function at 16-bit fixed-point scale, then proves and verifies the computation using Noir/Barretenberg. A successful run prints proving key time, prover time, verifier time, and proof size.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Our microbenchmarks for the two approaches in our work: Padé and Gauss-Legendre quadrature approximations, vs existing approaches (polynomial, piecewise-linear, lookup table approximations) for a large class of non-polynomial functions. This is present in Table 2 of the paper.
(C2): For high-accuracy applications in DeFi and astronomy, we report prover times in Table 3.
(C3): In Tables 4,5 we compare circuit sizes of our fixed-point approaches vs floating-point based proof systems.

A.4.2 Experiments

The experiments correspond to the claims made above directly: **C1** to **E1**, **C2** to **E2** and **C3** to **E3**.
We estimate human time of about 15 minutes for setup and running the experiments, while the experiments themselves should take less than 30 minutes of compute (wall-clock) time on the hardware in Sec. A.2.3 (**E1** takes the longest at around 15-20 minutes, while **E2,E3** take less than 5 minutes each).

(E1): Run all microbenchmarks for Table 2 using `python3 scripts/run_all.py`. The results csv is stored in the `./experiments` folder. Note that some experiments marked as failed are by design and should match the empty entries in the paper; (these are either due to them taking unreasonably long or having large error, in which case we do not run them as mentioned in the paper).
(E2): Run the Orbital mechanics experiments using

```
python3 scripts/gl_gen.py kepler 1 30 120
python3 scripts/run_prover.py gl_quad
python3 scripts/poly_gen.py kepler 1 30 120
python3 scripts/run_prover.py poly
python3 scripts/pade_gen.py kepler 1 16 120
python3 scripts/run_prover.py pade
```

For the Bancor protocol:

```
python3 scripts/gl_gen.py bancor 1 22 120
python3 scripts/run_prover.py gl_quad
python3 scripts/poly_gen.py bancor 1 28 120
python3 scripts/run_prover.py poly
python3 scripts/pade_gen.py bancor 1 14 120
python3 scripts/run_prover.py pade
```

(E3): For addition and multiplication constraint costs, run

```
python3 scripts/fp_add_gen.py 10000 64
python3 scripts/run_prover.py fp_add --profile-gates
python3 scripts/fp_mul_gen.py 10000 64
python3 scripts/run_prover.py fp_mul --profile-gates
```

For GeLU comparisons, run (replace 1 by 16 or 256 as in the paper)

```
python3 scripts/gl_gen.py gelu 1 24 64
python3 scripts/run_prover.py gl_quad --profile-gates
```

A.5 Notes on Reusability

Some of the microbenchmarks may report failures even though the table in the paper reports success. This can happen due to low probability events of random input values being very close to the edge of the quantization range; in such a case, that specific or set of experiments may be run again (use `python3 run_all.py --help` to see options to run only a subset of experiments).

Note that the parameters chosen for the experiments were found programmatically by a script `./scripts/find_optimal_params.py` and by increasing degree/order of the approximations (in `./scripts/run_all.py`), the failure probability can be reduced further.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.