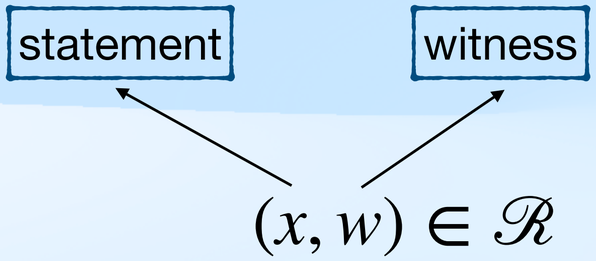
Efficiently Provable Approximations for Non-Polynomial Functions

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

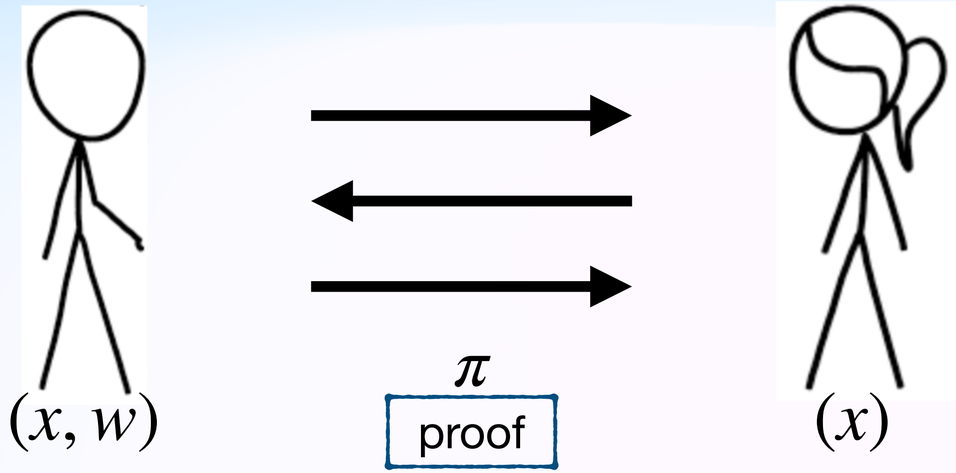
Usenix Security ’26

Succinct proof systems / SNARKs

Definition



Prover wants to convince a verifier that
they know w such that $(x, w) \in \mathscr{R}$



Succinct proof systems

Properties

- **Completeness** —
  If prover is honest and statement is true $\Longrightarrow$ verifier always accepts
- **Soundness/Extractability** —
  If statement is false/prover does not “know” a valid witness $\Longrightarrow$ verifier rejects w.h.p.
- **Succinctness** —
  Proof size and verifier time $\ll |w|$

Succinct proof systems

Practical considerations

Frontend

Information-theoretic

Compiles arbitrary program (in high-level language) to specialized circuit representation (R1CS/Plonkish/etc.)

==>circom

gnark

NOIR

Backend

Cryptographic

Proves the satisfiability of the compiled circuit using a succinct proof system

Groth16

PLONK

STARK

Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications

Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications



Applications

Machine learning

- Non-linear layers
- Positional encodings

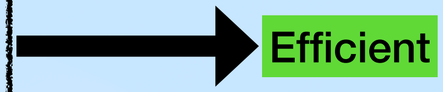
DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications



Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications

→ Efficient

Non-polynomial

Exponential

$$\exp(\pm x) := e^{\pm x}$$

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

Trigonometric

$$\sin(x), \cos(x)$$

$$\sin^{-1}(x), \cos^{-1}(x)$$

… and more

Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications

→ Efficient

Non-polynomial

Exponential

$$\exp(\pm x) := e^{\pm x}$$

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

Trigonometric

$$\sin(x), \cos(x)$$

$$\sin^{-1}(x), \cos^{-1}(x)$$

… and more

→

Applications

Machine learning

- Non-linear layers
- Positional encodings

DeFi

- Bancor formula for token pricing
- Custom pricing curves for Uniswap

Polynomial

Additions

Multiplications

→ Efficient

Non-polynomial

Exponential

$$\exp(\pm x) := e^{\pm x}$$

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

Trigonometric

$$\sin(x), \cos(x)$$

$$\sin^{-1}(x), \cos^{-1}(x)$$

… and more

→ Inefficient

Encoding real numbers inside finite fields

- Most proof systems work over (large) finite fields and require witness elements to be encoded as a field element.
- Non-polynomial functions require computations with **real numbers**
    - These cannot be expressed as field elements efficiently without losing some information

Encoding real numbers inside finite fields

- Most proof systems work over (large) finite fields and require witness elements to be encoded as a field element.
- Non-polynomial functions require computations with **real numbers**
    - These cannot be expressed as field elements efficiently without losing some information

Fixed-point encoding

Encoding real numbers inside finite fields

- Most proof systems work over (large) finite fields and require witness elements to be encoded as a field element.
- Non-polynomial functions require computations with **real numbers**
    - These cannot be expressed as field elements efficiently without losing some information

Fixed-point encoding

Floating-point encoding

Encoding real numbers inside finite fields

- Most proof systems work over (large) finite fields and require witness elements to be encoded as a field element.
- Non-polynomial functions require computations with **real numbers**
    - These cannot be expressed as field elements efficiently without losing some information

Fixed-point encoding vs **Floating-point encoding**

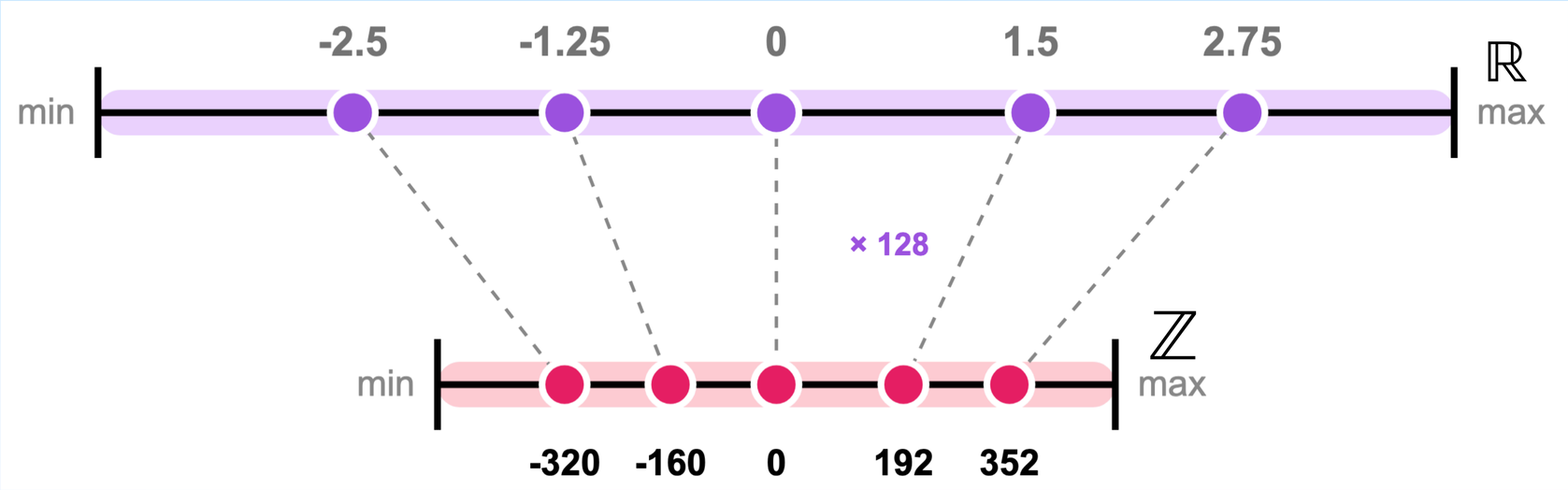
Encoding real numbers inside finite fields

Fixed-point encoding

Pick a scaling factor (which determines the precision we can get)

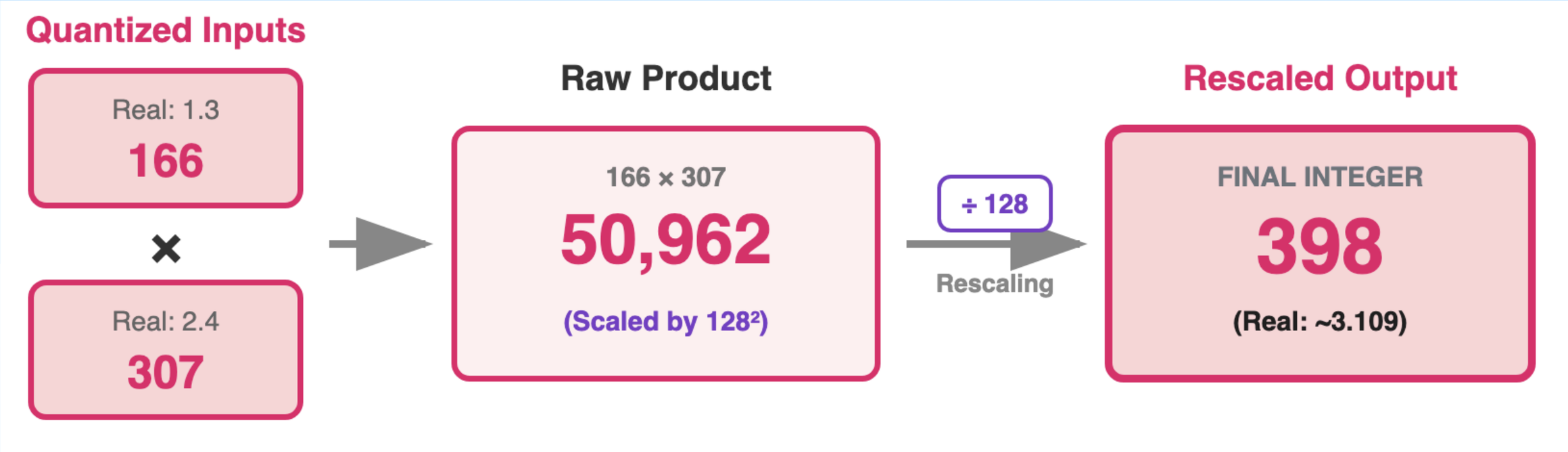
Map real numbers to integers inside a range

The field needs to be large enough to avoid overflows



Encoding real numbers inside finite fields

Operations (Add/Mul)



We need to rescale after each multiplication

Errors accumulate with each multiplication

Can we prove $y \approx f(x)$
for non-polynomial functions

Can we prove $y \approx f(x)$
for non-polynomial functions
with low prover overhead

Can we prove $y \approx f(x)$ for non-polynomial functions with low prover overhead and low absolute error?

Results

Results

T1 - Padé approximants

Drop-in replacement for Taylor-series approximations

Up to 20x better prover time

Results

T1 - Padé approximants

Drop-in replacement for Taylor-series approximations

Up to 20x better prover time

T2 - Quadrature-based approximations

Constant multiplicative depth as accuracy increases
Supports most non-polynomial functions

Up to 256x lower error

Results

T1 - Padé approximants

Drop-in replacement for Taylor-series approximations

Up to 20x better prover time

T2 - Quadrature-based approximations

Constant multiplicative depth as accuracy increases
Supports most non-polynomial functions

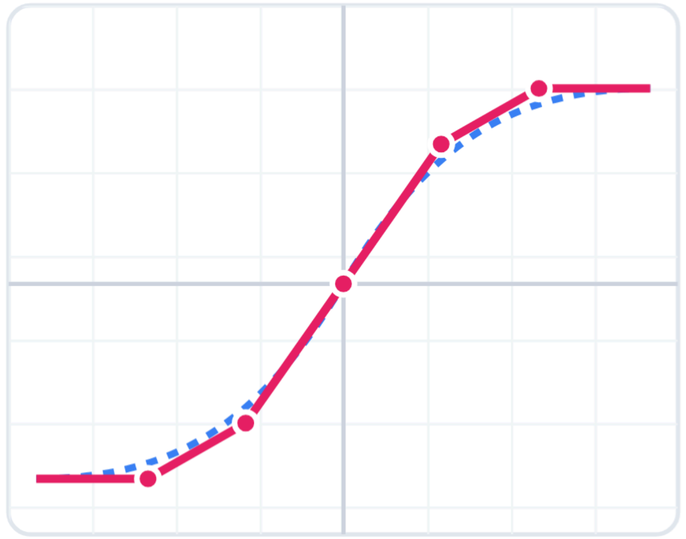
Up to 256x lower error

Applications

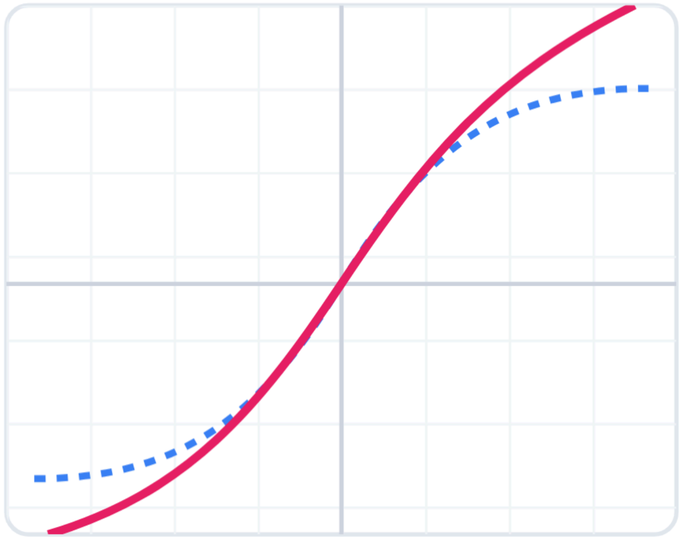
Accuracy-sensitive applications in orbital mechanics and DeFi token pricing calculations (Up to 64x lower error)

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

Existing solutions



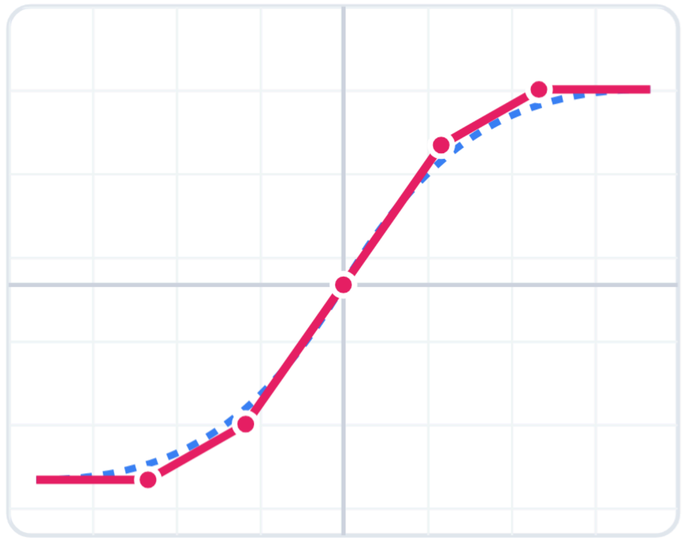
Piecewise Linear

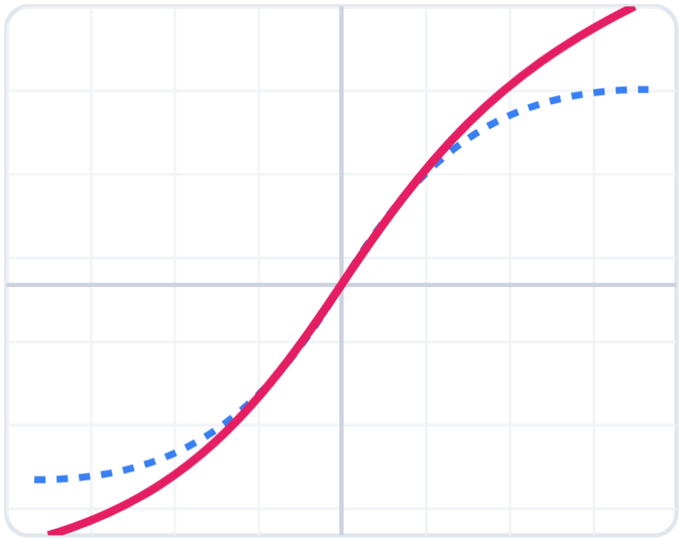


Taylor series

x	σ(x)
...	...
37	73
...	...
...	...

Lookup tables



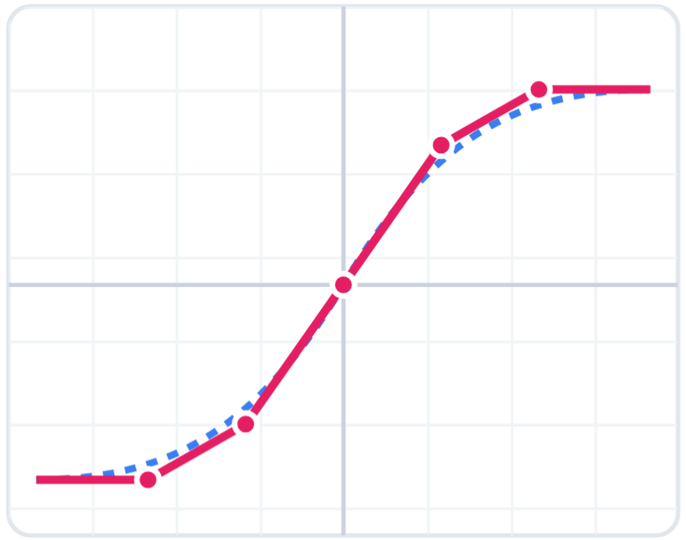


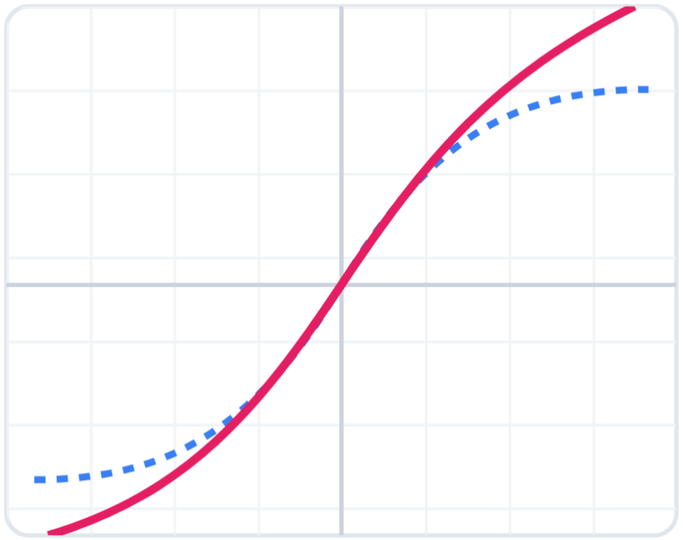
x	σ(x)
...	...
37	73
...	...
...	...

Piecewise Linear

Taylor series

Lookup tables





x	σ(x)
...	...
37	73
...	...
...	...

Piecewise Linear

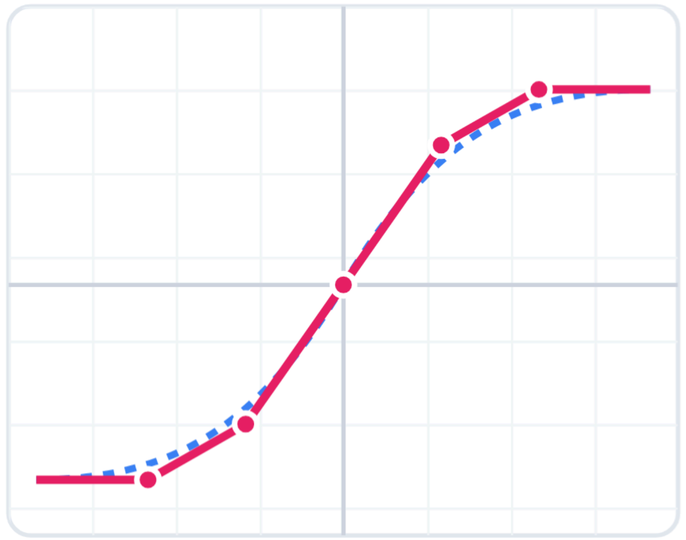
Works for all functions

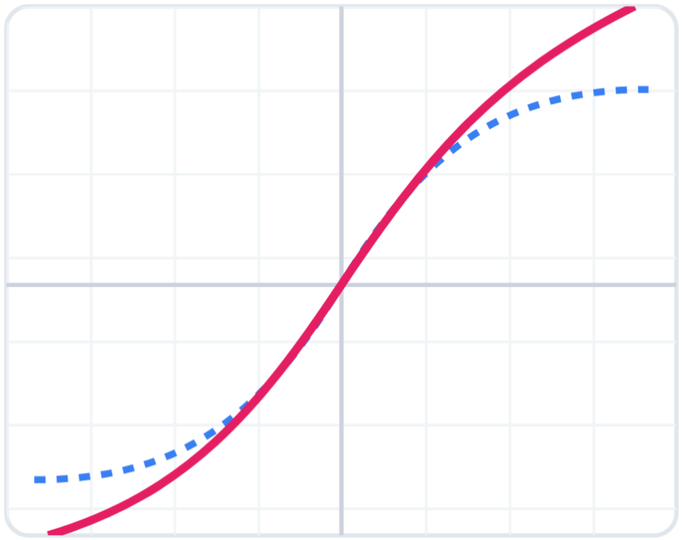
More pieces for more accuracy

Proving requires a lot of conditionals

Taylor series

Lookup tables





x	σ(x)
...	...
37	73
...	...
...	...

Piecewise Linear

Works for all functions

More pieces for more accuracy

Proving requires a lot of conditionals

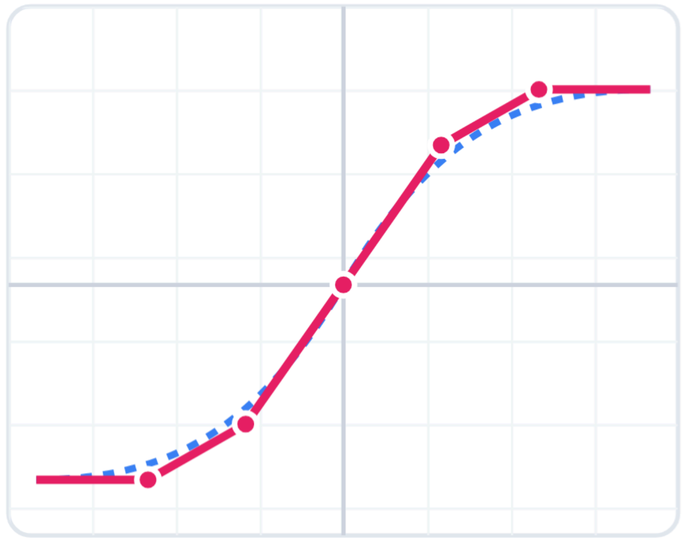
Taylor series

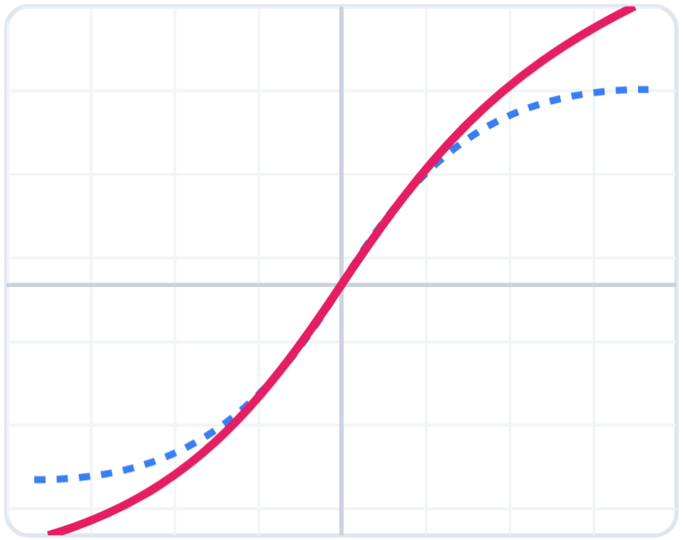
Works for all differentiable functions

Higher degree for more accuracy

Proving requires computing powers

Lookup tables





x	σ(x)
...	...
37	73
...	...
...	...

Piecewise Linear

Works for all functions

More pieces for more accuracy

Proving requires a lot of conditionals

Taylor series

Works for all differentiable functions

Higher degree for more accuracy

Proving requires computing powers

Lookup tables

Works for all functions

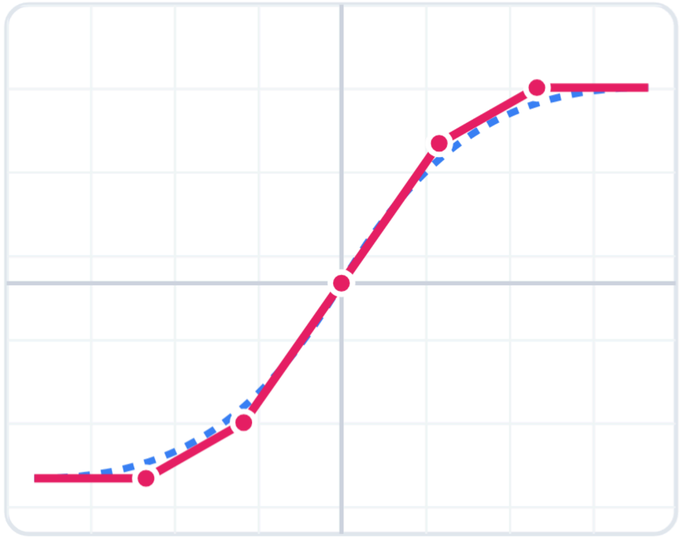
Larger table for more accuracy

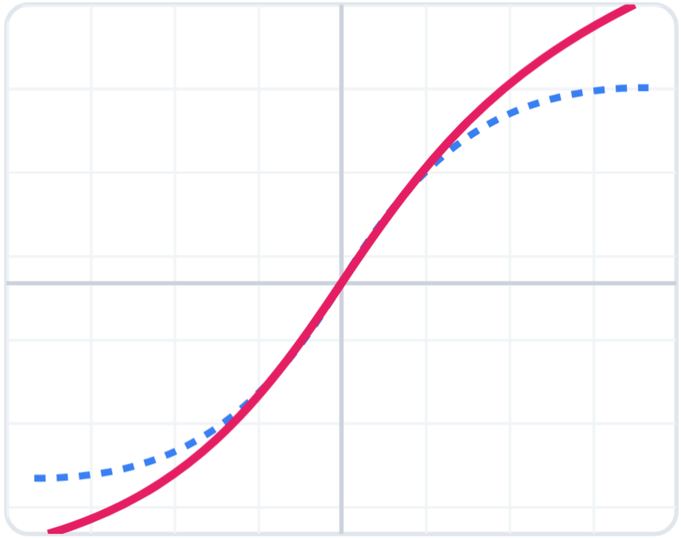
Proving requires a lookup protocol

Increasing the desired accuracy

For better accuracy, we need more terms in our approximations

More terms $\Longrightarrow$ More multiplications $\Longrightarrow$ More errors / More proving time





x	σ(x)
...	...
37	73
...	...
...	...

T1 - Padé approximation

- Padé approximants are as good as Taylor series expansions of twice the degree
- Drop-in replacement for Taylor series based approximations
- Up to 20x faster proving than SOTA

$$T_6(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480}$$

$$R_{3,3}(x) = \frac{\frac{1}{2} + \frac{1}{4}x + \frac{1}{20}x^2 + \frac{1}{240}x^3}{1 + \frac{1}{10}x^2}$$

Multiplications still grow as the desired accuracy increases

T2 - Quadrature-based approximations

Observation

Consider the inverse of $y = \sigma(x)$

$$y = \frac{1}{1 + e^{-x}}$$

$$x = \ln(y) - \ln(1 - y)$$

T2 - Quadrature-based approximations

Observation

Consider the inverse of $y = \sigma(x)$

$$y = \frac{1}{1 + e^{-x}}$$

$$x = \ln(y) - \ln(1 - y)$$

Notice that $\ln y = \int_1^y \frac{dy}{y}$, hence we have

$$y = \sigma(x) \iff x = \int_{1-y}^{y} \frac{dy}{y}$$

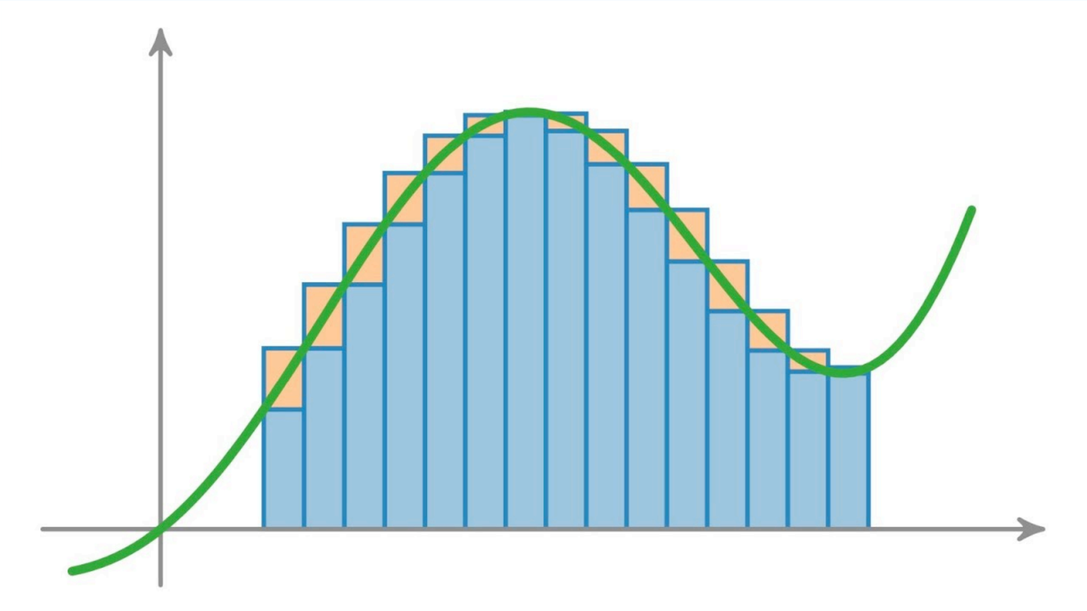
T2 - Quadrature-based approximations

Numerical integration/Quadrature

We can approximate integrals using summations

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

(for some fixed w_i, x_i)



Number of additions grows as the desired accuracy increases

T2 - Quadrature-based approximations

Numerical integration/Quadrature

$$y = \frac{1}{1 + e^{-x}}$$

$$\iff x = \ln\left(\frac{y}{1 - y}\right)$$

$$\iff x = \int_{1-y}^{y} \frac{1}{t} dt$$

$$\iff x = \int_{-1}^{1} \frac{1}{t + \frac{1}{2y - 1}} dt \qquad \implies x \approx \sum_{i=1}^{n} \frac{w_i}{x_i + \frac{1}{2y - 1}}$$

T2 - Quadrature-based approximations

Generality

Works for all functions where

the inverse of the function has an algebraic derivative

$$\int_{-1}^{1} \frac{dt}{2t + \frac{2}{y}} = x \iff y = \tanh(x)$$

$$\int_{-1}^{1} \frac{dt}{t + \frac{1+y}{1-y}} = x \iff y = e^{-x}$$

$$\int_{-1}^{1} \frac{dt}{\frac{y}{2}(t+1)^2 + \frac{2}{y}} = x \iff y = \tan(x)$$

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

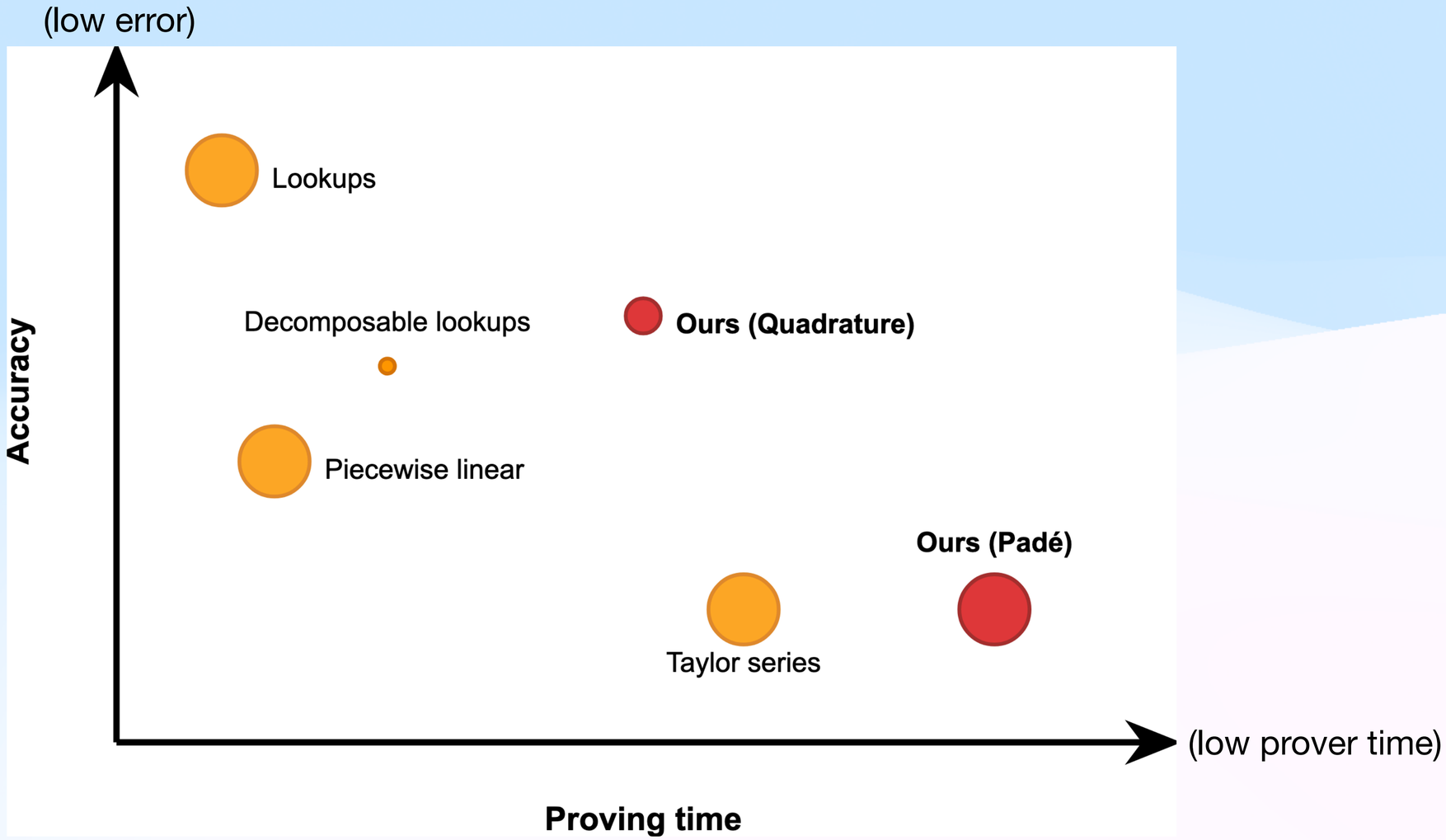
and more

Comparison

Expressiveness

Accuracy

Performance



Some applications

Bancor protocol for automated token pricing

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

Uniswap V4 supports custom curves

- Weighted product curves
- Lambert curves

$$R_A^{w_A} \cdot R_B^{1-w_A} = K$$

$$y = \frac{1}{c} \cdot W\left(\frac{c \cdot e^{2c-x}}{x} \right)$$

Orbital mechanics (Kepler equation)

$$M = E - e \sin E$$

Non-linear layers for DNNs

$$\sigma(x), \ \mathrm{SoftMax}(x)$$

Bancor Formula

- Automated token exchange protocols rely on these formulas to determine the exchange rate between tokens
    - Smart contracts don’t natively support floating point ops, so we need to approximate computations
    - Prove off-chain, verify on-chain

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

Reserve token amount: E

Token amount: T

Fractional reserve ratio: F

Reserve supply: R_0

Token supply: S_0

Bancor Formula

Statement

"Given (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"

$$y = x^k \iff \ln y = k \ln x \iff \int_{-1}^{1} \frac{dt}{t + \frac{1+y}{1-y}} = k \cdot \int_{-1}^{1} \frac{dt}{t + \frac{1+x}{1-x}}$$

Up to 64x lower error with a 120-bit scaling factor

Summary

We present two approximation techniques (Padé, Quadrature) for a large class of non-polynomial functions.

These are efficiently provable using ZKPs and scale well with increasing accuracy — additive terms instead of multiplicative.

We obtain up to 256x lower errors than comparable baselines for commonly used non-polynomial functions.