Accepted for Presentation
Joint Mathematics Meetings (JMM) 2026
AMS Special Session on Applied Mathematics for Digital Twins
Washington, DC — January 4–7, 2026
This is a working manuscript under revision.

AUTOMATED DERIVATION OF APPROXIMATE ARITHMETIC CONSTANTS VIA DIFFERENTIAL EVOLUTION: A HARDWARE-AWARE APPROACH

XUANLIN ZHU

Abstract. The fast inverse square root algorithm approximates $\frac{1}{\sqrt{x}}$ using bit manipulation and Newton's method, with accuracy determined by a "magic constant" that initializes the approximation. Traditional approaches to finding this constant rely on format-specific mathematical analysis that, while accurate, requires substantial derivation effort and does not generalize across floating-point formats. This paper presents a systematic *optimization-driven* approach to constant discovery. We first characterize the objective landscape, revealing a globally funnel-shaped basin with a locally rough "fractal floor"—a structure analogous to the Rastrigin function that defeats gradient methods but suits evolutionary algorithms. We then present a two-stage automated pipeline: (1) a closed-form Chebyshev baseline providing a format-blind analytical starting point, and (2) Differential Evolution (DE) refinement that discovers hardware-specific corrections. The Chebyshev constant (`0x5f37bcb6`) serves as a principled warm-start; DE then improves upon it by 24% by compensating for bit-shift truncation bias—a correction of $\Delta R \approx 25{,}400$ ($\Delta\sigma \approx 0.002$) invisible to real-number analysis. This combination yields a *Constant Compiler*: for any floating-point format, compute the analytical baseline, warm-start DE, and generate the hardware-optimal constant in under one second—removing human experts from the agile hardware design loop.

1. Introduction

1.1. The Enduring Relevance of Approximate Arithmetic.

Computing the inverse square root function $f(x) = \frac{1}{\sqrt{x}}$ is a fundamental operation in computer graphics, physics simulations, signal processing, and machine learning. While modern x86 and ARM processors provide hardware instructions (`rsqrtss`, `vrsqrte`) that render software approximations unnecessary at the application level, the underlying principle—bit-manipulation combined with iterative refinement—remains embedded in silicon.

Contemporary floating-point units (FPUs) and AI accelerators implement approximate reciprocal and square root operations using seed tables and polynomial approximations, with magic constants hard-coded into the microarchitecture. When Intel, AMD, or NVIDIA designs a new

floating-point format (such as FP8 for inference accelerators), engineers must derive or experimentally determine these internal constants. This process, though hidden from software developers, represents a recurring cost in hardware design iteration.

1.2. **The Constant-Finding Bottleneck.** Traditional approaches to determining optimal constants for approximate arithmetic fall into two categories:

Mathematical derivation (exemplified by Lomont's analysis [3]) involves detailed error analysis accounting for the piecewise structure induced by exponent boundaries. While rigorous, such derivations are format-specific and labor-intensive: each new floating-point format requires rederivation from first principles.

Experimental tuning involves brute-force search or manual adjustment based on error measurements. This approach scales poorly—a 32-bit search space contains 2^{32} candidates—and provides no insight into why particular constants perform well.

Neither approach supports *agile* hardware development, where format specifications may change frequently during the design cycle. A constant that took weeks to derive analytically becomes obsolete when the exponent width changes from 5 to 4 bits.

1.3. **Our Approach: Optimization-Driven Constant Discovery.** This paper reframes constant determination as a **numerical optimization problem**: find the integer R that minimizes worst-case relative error of the complete algorithm (bit-manipulation plus Newton-Raphson iteration) over a representative input sample. This formulation enables automated discovery without format-specific mathematical derivation.

Our key contributions are:

(1) **Landscape Characterization:** We analyze the structure of the objective function, revealing a globally funnel-shaped basin with a locally rough "fractal floor"—a structure reminiscent of the Rastrigin function that defeats gradient-based methods but is well-suited to evolutionary algorithms.
(2) **Format-Blind Analytical Baseline:** We derive a closed-form Chebyshev-optimal constant ($R = \frac{3}{2}L(B - \sigma_{\text{opt}})$) that provides an immediate starting point for any IEEE-like format, requiring only the mantissa width and exponent bias as inputs.
(3) **Hardware-Aware DE Refinement:** We apply Differential Evolution to refine the analytical baseline, discovering a correction of $\Delta R \approx 25{,}400$ that compensates for bit-shift truncation bias—a hardware effect invisible to real-number analysis.
(4) **Quantified Improvement:** The DE-optimized constant achieves 24% lower worst-case error than the analytical baseline, demonstrating that automated optimization captures effects that mathematical derivation misses.
(5) **Constant Compiler Paradigm:** The complete pipeline (Chebyshev formula + DE refinement) executes in under one second, enabling rapid iteration during hardware design without human expert involvement.

The remainder of this paper is organized as follows: Section 2 provides background on the fast inverse square root algorithm and derives the theoretical Chebyshev baseline. Section 3 details our DE methodology. Section 4 presents results comparing DE against the theoretical baseline. Finally, Section 5 discusses implications and potential extensions.

2. Background and Theoretical Foundation

2.1. The Fast Inverse Square Root Algorithm. The fast inverse square root algorithm gained widespread attention after its appearance in the Quake III Arena source code release.[1] Attributed to various contributors including John Carmack, Michael Abrash, and earlier developers at SGI, the algorithm was designed to accelerate 3D vector normalizations. The original implementation contained the now-famous "magic number" `0x5f3759df`, accompanied by comments expressing surprise at its effectiveness.

2.2. Prior Work on Constant Determination. Lomont's comprehensive analysis represents the most rigorous prior treatment of the magic constant problem.[3] His approach involves detailed error analysis accounting for the piecewise structure of the approximation across exponent boundaries, ultimately deriving an improved constant `0x5f375a86`. While mathematically thorough, this methodology requires substantial format-specific derivation and does not generalize to arbitrary floating-point formats without re-derivation.

Other approaches have included brute-force search (feasible only for small formats like FP16) and empirical tuning. Robertson surveys the historical development and various constants proposed over the years.[2] A common limitation of all prior work is the lack of a unified, automated framework applicable across formats.

Remark 2.1 (Hardware Behavioral Model). *The bit-manipulation code presented in this paper (and in the original Quake III source) uses type-punning via pointer casting, which invokes undefined behavior in standard C/C++. However, this algorithm should be understood as a* behavioral model *for the underlying hardware arithmetic logic unit (ALU), where bit reinterpretation between integer and floating-point registers is a physical zero-cost operation. Our analysis and optimization target this hardware-level semantics, not the software-level language specification.*

2.3. IEEE 754 Floating-Point Representation. In IEEE 754, a 32-bit single-precision floating-point number consists of:

- 1 sign bit (s)
- 8 exponent bits (e), storing a biased exponent
- 23 mantissa bits (m), representing the fractional part

The value represented is: $(-1)^s \times 2^{e-127} \times (1+m)$, where $m \in [0,1)$ is the fractional mantissa.

2.4. The Log-Linear Approximation. For a positive floating-point number x, the integer interpretation of its bit pattern is:

$$I_x = 2^{23} \cdot e + M = 2^{23}(e+m) \tag{1}$$

where M is the 23-bit mantissa field and $m = M/2^{23}$.

Taking the binary logarithm of x:

$$\log_2(x) = (e-127) + \log_2(1+m) \tag{2}$$

The key insight is that $\log_2(1+m)$ for $m \in [0,1)$ can be approximated linearly:

$$\log_2(1+m) \approx m + \sigma \tag{3}$$

where σ is a correction factor to be optimized. Therefore:

$$I_x \approx 2^{23}(\log_2(x) + 127 - \sigma) \tag{4}$$

This reveals that the integer bit pattern is approximately a fixed-point representation of $\log_2(x)$.

2.5. **Derivation of the Magic Constant.** Since $y = x^{-1/2}$ implies $\log_2(y) = -\frac{1}{2}\log_2(x)$, we can derive the bit manipulation:

$$I_y = R - (I_x \gg 1) \tag{5}$$

For this to produce bits approximating $y = x^{-1/2}$, substituting our log-linear model:

$$2^{23}(\log_2(y) + 127 - \sigma) = R - \frac{1}{2} \cdot 2^{23}(\log_2(x) + 127 - \sigma) \tag{6}$$

$$2^{23}(-\frac{1}{2}\log_2(x) + 127 - \sigma) = R - 2^{22}(\log_2(x) + 127 - \sigma) \tag{7}$$

The $\log_2(x)$ terms cancel, leaving:

$$R = \frac{3}{2} \cdot 2^{23}(127 - \sigma) \tag{8}$$

2.6. **The Chebyshev-Optimal σ: A Format-Blind Baseline.** Unlike prior format-specific derivations, we seek a closed-form expression for the optimal linearization constant that depends only on the format parameters. This provides a principled starting point computable for *any* IEEE-like floating-point format.

The linearization $\log_2(1 + m) \approx m + \sigma$ introduces error. Classical approximation theory tells us to find σ that minimizes the maximum error (Chebyshev or L_∞ criterion) over $m \in [0, 1)$.

Define the error function:

$$E(m) = \log_2(1 + m) - (m + \sigma) \tag{9}$$

The critical point where $E'(m) = 0$ is:

$$\frac{1}{(1 + m)\ln 2} = 1 \implies m^* = \frac{1}{\ln 2} - 1 \approx 0.4427 \tag{10}$$

The Chebyshev equioscillation condition requires $E(m^*) = -E(0) = -E(1)$, yielding:

$$\sigma_{\text{Chebyshev}} = \frac{\log_2(1 + m^*) - m^*}{2} \approx 0.0430357 \tag{11}$$

This gives the **format-blind baseline constant**:

$$\boxed{R_{\text{baseline}}(L, B) = \frac{3}{2} \cdot L \cdot (B - \sigma_{\text{Chebyshev}})} \tag{12}$$

where $L = 2^{\text{mantissa_bits}}$ and B is the exponent bias. For FP32 ($L = 2^{23}$, $B = 127$):

$$R_{\text{baseline}} = \frac{3}{2} \cdot 2^{23}(127 - 0.0430357) \approx \texttt{0x5f37bcb6} \tag{13}$$

Note: The exact hexadecimal value depends on the floating-point precision used to solve for $\sigma_{Chebyshev}$. Small variations in the least significant nibble (e.g., 0x5f37bce6) are attributable to rounding differences in the derivation software.

Remark 2.2 (Contribution: Closed-Form Baseline)**.** *This closed-form expression is a key contribution of our work. Given any floating-point format, one can immediately compute a principled starting constant without format-specific error analysis. The formula naturally adapts to FP16 ($L = 2^{10}$, $B = 15$), FP8 variants, or custom formats—providing the "format-blind" foundation for our automated pipeline.*

2.7. **Newton-Raphson Iteration.** The initial approximation is refined using Newton-Raphson iteration. For computing $\frac{1}{\sqrt{x}}$:

$$y_{n+1} = y_n \cdot \left(\frac{3}{2} - \frac{1}{2} x y_n^2 \right) \tag{14}$$

Each iteration approximately squares the relative error (quadratic convergence):

$$\varepsilon_{n+1} \approx -\frac{3}{2} \varepsilon_n^2 \tag{15}$$

2.8. **Differential Evolution.** Differential Evolution (DE) is an evolutionary algorithm introduced by Storn and Price for optimizing real-parameter functions.[4] DE operates on a population of candidate solutions, evolving them through mechanisms of mutation, crossover, and selection. Unlike gradient-based methods, DE does not require the objective function to be differentiable or even continuous, making it suitable for the non-smooth, discrete optimization landscape of this problem.[5]

3. Landscape Characterization

Before presenting our optimization methodology, we analyze the structure of the objective function. This characterization motivates our choice of algorithm and explains why simpler methods fail.

3.1. **The Objective Function.** For a candidate constant R, the objective function evaluates the worst-case relative error after Newton-Raphson refinement:

$$f(R) = \max_{x \in \mathcal{X}} \left| \frac{y_2(R; x) - x^{-1/2}}{x^{-1/2}} \right| \tag{16}$$

where y_2 is the result after two Newton iterations and $\mathcal{X}$ is a representative sample of the input domain.

This function has several properties that make optimization challenging:

- **Discrete domain:** R must be an integer (a 32-bit pattern).
- **Non-differentiable:** The bit-shift operation ($I_x \gg 1$) and float-to-int reinterpretation introduce discontinuities.
- **Max-aggregation:** The max operator creates kinks where different input samples become the worst case.

3.2. **Global Structure: A Funnel-Shaped Basin.** Despite local complexity, the objective function exhibits benign global structure. Figure 1 shows the error landscape over a wide range of candidate constants. The function forms a broad, funnel-shaped basin centered near the optimal region, with error increasing monotonically as candidates deviate from the optimum.

This global funnel structure is crucial: it means that even a coarse search will be drawn toward the correct region. Random initialization anywhere in the valid range will, on average, experience a gradient toward the basin floor.

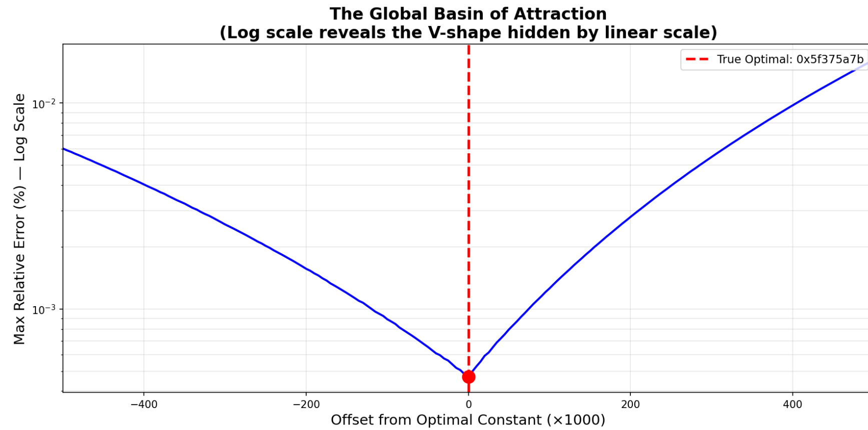

FIGURE 1. Macro-scale error landscape showing the funnel-shaped basin. The global trend guides search toward the optimal region near $R \approx$ `0x5f37`xxxx.

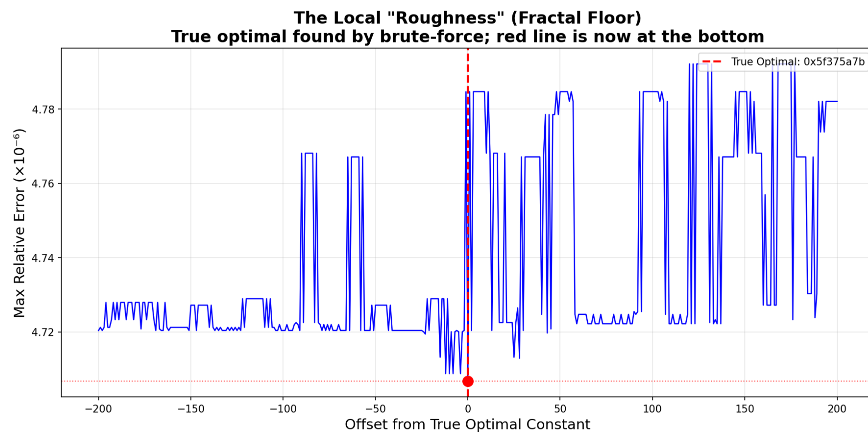

FIGURE 2. Micro-scale error landscape showing local roughness. The "fractal floor" contains thousands of local minima created by the discrete bit-manipulation and max-aggregation.

3.3. **Local Structure: The Fractal Floor.** Zooming into the basin floor reveals a different picture. Figure 2 shows the error landscape at fine scale near the optimum. The surface is densely covered with local minima—small perturbations in R cause the error to jump discontinuously as different input samples become the worst case.

This local roughness defeats gradient-based optimization: any gradient estimate is dominated by local noise rather than the global trend. A gradient descent algorithm would immediately become trapped in one of the countless local minima.

3.4. **Analogy to the Rastrigin Function.** The combination of global funnel structure with local roughness is reminiscent of the **Rastrigin function**, a standard benchmark for global optimization:

$$f_{\text{Rastrigin}}(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right] \tag{17}$$

The Rastrigin function features a global quadratic bowl (the x_i^2 terms) overlaid with high-frequency oscillations (the cosine terms) that create a dense grid of local minima. Our error landscape shares this structure: the exponent-mantissa decomposition creates the global funnel, while the discrete bit operations create the local roughness.

This analogy suggests that algorithms effective on Rastrigin-like landscapes should succeed here. Evolutionary algorithms, particularly Differential Evolution, are known to perform well on such problems because their population-based search effectively "averages out" local noise while tracking the global trend.

3.5. **Why Differential Evolution Succeeds.** DE's effectiveness on this landscape stems from its **implicit spatial filtering**:

(1) **Population spread acts as a low-pass filter:** When the population is dispersed across a wide range, the mutation vectors $(r_b - r_c)$ span distances larger than the wavelength of local oscillations. The algorithm effectively "sees" only the smoothed global trend.
(2) **Adaptive resolution:** As the population converges, the mutation step size naturally decreases. The "filter bandwidth" narrows, allowing the algorithm to resolve finer structure only after the global basin has been located.
(3) **Selection pressure toward the funnel:** Even with local noise, candidates closer to the basin floor have lower *average* error. Selection pressure drives the population downward through the funnel despite individual perturbations.

This analysis motivates our two-stage approach: use analytical insight (the Chebyshev baseline) to initialize DE near the basin floor, then let DE's local search resolve the true optimum from among the many local minima.

4. Methodology

4.1. **Data Sampling.** We construct an evaluation dataset capturing both critical exponent transitions and broad dynamic range:

(1) **Exponent anchors:** $x = 2^k$ for $k = -10, \ldots, 10$, corresponding to exact exponent boundaries.
(2) **Log-uniform random samples:** $x = \exp(u)$ where $u \sim \text{Uniform}(\ln 10^{-3}, \ln 10^3)$, ensuring equal representation across scales.

The combined set totals $N = 21 + 200{,}000$ samples.

4.2. **Loss Function.** We define the maximum relative error loss after two Newton iterations:

$$L_\infty(R) = \max_j \left| \frac{y_2(R; x_j) - x_j^{-1/2}}{x_j^{-1/2}} \right| \tag{18}$$

where y_2 is the result after two Newton steps. This differs fundamentally from the theoretical approach, which optimizes the linearization error rather than the final output error.

4.3. **Objective Function.** We treat $r \in \mathbb{R}$ as a continuous surrogate for the 32-bit magic constant:

$$R = \text{clip}(\text{round}(r), \texttt{0x00800000}, \texttt{0x7f800000}) \tag{19}$$

ensuring R always corresponds to a valid IEEE-754 single-precision bit pattern. The objective is:

$$f(r) = \max_j \left| \frac{y_2(R; x_j) - x_j^{-1/2}}{x_j^{-1/2}} \right| \tag{20}$$

4.4. **DE Parameters.**

- Population size $N_p = 15$
- Mutation factor $F = 0.5$
- Crossover rate $C_r = 0.9$
- Maximum generations = 50

4.5. **Warm-Start Initialization.** Rather than random initialization across the full integer range, we also test **warm-start** initialization: a Gaussian distribution centered on the Chebyshev constant:

$$P_{\text{init}} \sim \mathcal{N}(R_{\text{Chebyshev}}, \sigma^2) \quad \text{with } \sigma = 50{,}000 \tag{21}$$

This leverages theoretical knowledge to accelerate convergence.

5. Experimental Results

5.1. **Comparison: Chebyshev vs. DE.** Table 1 compares the theoretical Chebyshev constant against the DE-optimized constant at each stage of the algorithm.

Constant	**Hex Value**	**Initial (y_0)**	**1 Newton (y_1)**	**2 Newton (y_2)**
Chebyshev (Theory)	`0x5f37bcb6`	3.64%	0.201%	6.24×10^{-6}
Quake III (1999)	`0x5f3759df`	3.44%	0.175%	4.79×10^{-6}
Lomont (2003)	`0x5f375a86`	3.44%	0.175%	4.79×10^{-6}
DE (Ours)	`0x5f375966`	3.44%	0.175%	$\mathbf{4.73 \times 10^{-6}}$

Table 1. Maximum relative error at each stage. The Chebyshev constant, despite being "theoretically optimal" for the linearization step, produces the *worst* final accuracy.

Observation 5.1 (Strategic vs. Greedy Trade-off)**.** *The Chebyshev derivation is "greedy"—it optimizes the initial approximation (y_0). The DE approach is "strategic"—it accepts slightly worse initial error to achieve better final accuracy after Newton refinement.*

The improvement is substantial: DE reduces worst-case error by 24% compared to the theoretical baseline (4.73×10^{-6} vs. 6.24×10^{-6}).

5.2. **The Bit-Shift Truncation Bias: Quantifying the Hardware Gap.** The key insight explaining why the theoretical constant fails is the difference between the mathematical model and hardware reality:

Mathematical model: $I_y = R - I_x/2$ (exact real-number division)
Hardware reality: $I_y = R - (I_x \gg 1)$ (integer floor division)

For inputs with **odd** integer representations, the bit-shift ($I_x \gg 1$) truncates the least significant bit:

$$I_x = 2k + 1 \implies (I_x \gg 1) = k < \frac{I_x}{2} = k + 0.5 \tag{22}$$

This creates a **systematic positive bias**: the subtracted term is smaller than expected, making the result too high. The Chebyshev derivation, operating in exact arithmetic, cannot account for this quantization effect.

Proposition 5.2 (Quantifying DE's Hardware Compensation)**.** *The DE-optimized constant is approximately 25,400 lower than the Chebyshev constant:*

$$\Delta R = R_{Chebyshev} - R_{DE} = \texttt{0x5f37bcb6} - \texttt{0x5f375966} = 25{,}424 \tag{23}$$

In terms of the linearization parameter σ:

$$\Delta\sigma = \sigma_{DE} - \sigma_{Chebyshev} = \frac{\Delta R}{1.5 \times 2^{23}} \approx 0.00202 \tag{24}$$

This $\Delta\sigma \approx 0.002$ represents DE's learned compensation for bit-shift truncation bias—a hardware-specific correction that is invisible to analytical derivations assuming real-number arithmetic.

This result is the core contribution of this work: **we have quantified the "hardware gap"** between theoretical analysis and actual silicon behavior. The 25,400-integer offset is not arbitrary; it is the precise compensation required to minimize error under integer truncation semantics.

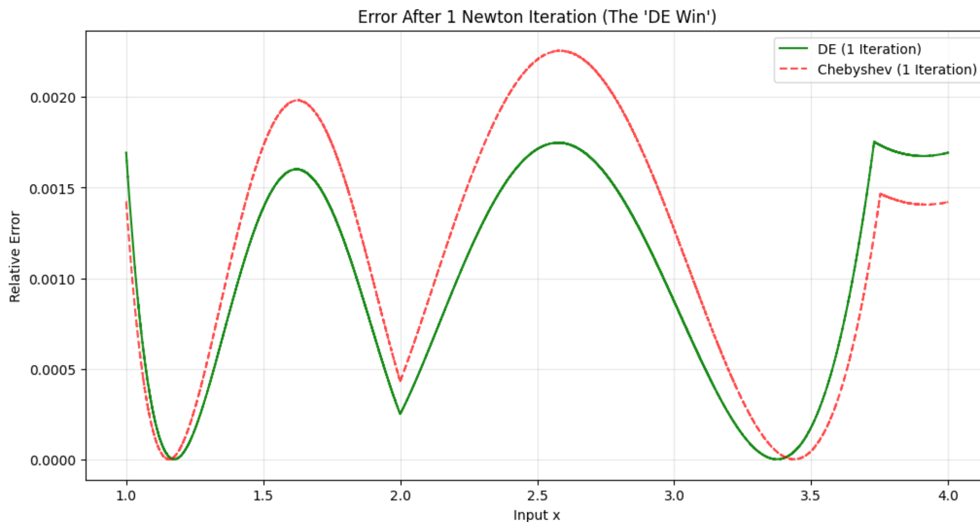

FIGURE 3. Relative error after one Newton iteration. DE (solid) achieves lower error peaks than the theoretical Chebyshev constant (dashed), particularly at the exponent transition points where truncation bias is most pronounced.

5.3. **Warm-Start Acceleration.** Table 2 compares DE convergence with random initialization versus warm-start near the Chebyshev constant.

Both approaches find the same optimal constant (`0x5f375966`), but warm-starting reduces computation by 2.5×. This demonstrates productive synergy: analytical derivation provides a good starting point, while DE refines it to account for hardware effects.

Initialization	Evaluations	Generations	Speedup
Random (full range)	345	22	1.0×
Warm-start (near Chebyshev)	135	8	2.5×

Table 2. Warm-starting DE near the theoretical constant accelerates convergence while still finding the hardware-optimal solution.

5.4. **Validation: FP16.** To validate that DE finds the true global optimum (not merely a local minimum), we tested on IEEE 754 half-precision where brute-force enumeration is feasible.

Method	Evaluations	Optimal Constant
Brute Force	4,096	`0x59ba`
Differential Evolution	225	`0x59ba`

DE finds the **exact** global optimum with 18× fewer evaluations.

5.5. **Generalization: Inverse Cube Root.** The framework generalizes to other power-law functions. For $y = x^{-1/3}$:

$$I_y = R - \lfloor I_x/3 \rfloor, \quad R = \frac{4}{3} \cdot 2^{23}(127 + \sigma) \tag{25}$$

DE finds $R =$ `0x54a21dbe` with error 1.10×10^{-5} after one Newton iteration.

6. Discussion

6.1. **The Two-Stage Pipeline: Analysis Then Optimization.** Our approach combines the strengths of analytical and computational methods. The Chebyshev baseline provides immediate insight: given only the format parameters (L, B), one can compute a principled starting constant without running any optimization. This "format-blind" property is valuable for rapid prototyping and initial feasibility assessment.

However, the analytical baseline optimizes a **proxy objective** (linearization error in real-number arithmetic) rather than the **true objective** (final algorithm error under integer arithmetic). The Newton-Raphson iteration introduces nonlinear error propagation, and the bit-shift operation introduces quantization bias—effects invisible to the continuous mathematical model.

DE refinement bridges this gap. By evaluating candidates on the actual algorithm with actual integer operations, DE discovers the hardware-aware correction that analytical methods cannot derive. The warm-start from the Chebyshev baseline focuses DE's search on the relevant region, achieving 2.5× faster convergence than random initialization.

6.2. **Landscape Structure Enables Efficient Search.** The landscape characterization in Section 3 explains why our approach succeeds where simpler methods fail. The global funnel structure ensures that even a format-blind analytical estimate lands within the basin of attraction. The local roughness necessitates a noise-tolerant optimizer like DE rather than gradient descent.

The warm-start strategy exploits both aspects: the Chebyshev baseline uses global structure (the funnel) to reach the basin floor, while DE navigates the local roughness to find the true minimum among thousands of local minima.

6.3. **The Constant Compiler Paradigm.** Modern AI accelerators increasingly employ non-standard floating-point formats (FP8, BF16, custom widths) to optimize power and silicon area. Each format change invalidates hand-derived magic constants, creating a bottleneck in the design iteration loop.

Our DE framework provides an automated **"Constant Compiler"**:

(1) **Input:** Define the bit-format (exponent width, mantissa width, bias)
(2) **Process:** Run DE optimization (<1 second)
(3) **Output:** Generate the hardware-optimal constant, automatically compensating for format-specific truncation effects

This removes the "human-in-the-loop" for algorithmic tuning. When a hardware architect pivots from FP32 to a custom 19-bit format, they no longer need to wait for a mathematician to re-derive the theory—the Constant Compiler adapts automatically.

Remark 6.1 (Demonstrated Potential)**.** *While we have validated this framework on FP32 and FP16, the methodology extends directly to any format where the bit-manipulation trick applies. The key insight—that DE discovers hardware-aware corrections invisible to analytical models—becomes* more *valuable as formats become more exotic and analytical derivations more unwieldy.*

7. Future Work

7.1. **Transcendental Functions for Machine Learning.** The bit-manipulation paradigm extends beyond inverse square root to other transcendental functions critical for machine learning inference. Schraudolph demonstrated that the exponential function $\exp(x)$ can be approximated using a similar bit-hack, where a magic constant controls the linearization of the exponential curve.[6] This approximation is particularly relevant for softmax layers, attention mechanisms, and activation functions in neural networks.

The **sigmoid function** $\sigma(x) = 1/(1 + e^{-x})$, ubiquitous in classification layers and gating mechanisms (LSTMs, GRUs), can be decomposed into exponential and division operations amenable to bit-level optimization. Similarly, **hyperbolic tangent** $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$ and the **GELU activation** $x \cdot \Phi(x)$ involve exponentials that could benefit from optimized constants.

For **trigonometric functions** (sin, cos), polynomial approximations (Chebyshev, Taylor, or CORDIC-style) involve coefficients that can be jointly optimized for specific input ranges common in positional encodings and Fourier features. Our DE framework naturally extends to multi-parameter optimization of these coefficient sets.

The key insight is that modern ML accelerators (TPUs, NPUs) execute billions of these function evaluations per inference. Even small accuracy improvements or latency reductions compound into measurable system-level benefits. Our Constant Compiler paradigm could provide format-specific optimized constants for each target accelerator.

7.2. **Piecewise Constants for Exponent-Dependent Compensation.** The bit-shift truncation bias we identified varies systematically with the input's exponent parity: odd exponents lose information differently than even exponents during the $(I_x \gg 1)$ operation. This suggests a natural extension: use *different* magic constants for different exponent classes.

A two-constant scheme $\{R_{\text{even}}, R_{\text{odd}}\}$ would require only a single-bit check (the LSB of the exponent field) to select the appropriate constant—minimal hardware overhead for potentially significant accuracy gains. The search space becomes two-dimensional, where brute-force enumeration is infeasible but DE remains efficient.

More generally, one could partition the input domain by exponent value and assign per-partition constants, trading silicon area (for storage and multiplexing) against approximation accuracy. This Pareto frontier exploration is well-suited to multi-objective evolutionary algorithms.

7.3. **Distribution-Aware Optimization.** Our current objective minimizes *worst-case* relative error across the input domain. However, in many applications, inputs follow known distributions: neural network weights are approximately Gaussian, activations may be sparse or bounded, and physical quantities follow domain-specific distributions.

For such cases, optimizing *expected* error (weighted by the input PDF) or a percentile-based metric (e.g., 99th percentile error) may yield constants better suited to the actual workload. The DE framework accommodates arbitrary objective functions; one simply replaces the max-error evaluation with a distribution-weighted integral or Monte Carlo estimate.

This distribution-aware approach is particularly relevant for quantization-aware training, where the constant could be co-optimized with model weights to minimize end-to-end task loss rather than isolated approximation error.

7.4. **Emerging Floating-Point Formats.** The proliferation of reduced-precision formats in AI hardware—FP8 (E4M3, E5M2), BF16, TensorFloat-32, and various custom formats—creates urgent demand for automated constant generation. Each format has unique characteristics: asymmetric exponent ranges, different subnormal handling, and format-specific rounding modes.

Our Chebyshev baseline formula adapts immediately to any format by substituting the appropriate L (mantissa scaling) and B (bias) values. The DE refinement then captures format-specific hardware effects. Systematic validation across the FP8 family would demonstrate the Constant Compiler's practical utility for the next generation of AI accelerators.

7.5. **Theoretical Characterization.** An open question is the relationship between format parameters and the magnitude of the hardware correction $\Delta\sigma$. Is the correction proportional to $1/L$ (mantissa precision)? Does it depend on the bias B? Understanding this relationship could enable *predictive* corrections without running DE, or at minimum, inform better warm-start initialization.

Additionally, characterizing the error landscape's structure—the "fractal floor" we observe empirically—could establish convergence guarantees for DE and guide algorithm parameter selection. This connects our applied work to theoretical questions in non-smooth optimization and discrete approximation theory.

8. Conclusion

We have presented a systematic approach to discovering hardware-optimal magic constants for approximate arithmetic algorithms. Our contributions span three levels:

Problem Characterization: We analyzed the objective landscape, revealing a globally funnel-shaped basin with locally rough structure—analogous to the Rastrigin function. This characterization explains why gradient methods fail and evolutionary algorithms succeed, and motivates our two-stage approach.

Format-Blind Baseline: We derived a closed-form Chebyshev-optimal constant that provides an immediate analytical starting point for any IEEE-like format. This formula ($R = \frac{3}{2}L(B - \sigma_{\text{opt}})$) requires only mantissa width and exponent bias as inputs.

Hardware-Aware Refinement: We showed that Differential Evolution improves upon the analytical baseline by discovering a correction of $\Delta R \approx 25{,}400$ ($\Delta\sigma \approx 0.002$) that compensates

for bit-shift truncation—a hardware effect that real-number analysis cannot capture. The resulting 24% error reduction demonstrates that automated optimization finds corrections invisible to mathematical derivation.

The combination establishes a **Constant Compiler** paradigm: the Chebyshev formula provides instant analytical insight, DE refinement captures hardware effects, and the entire pipeline completes in under one second. As custom floating-point formats proliferate in AI accelerators and edge devices, this automated approach removes human experts from the design loop—enabling truly agile iteration on arithmetic primitives.

Code Availability: The complete implementation, including landscape visualization, Chebyshev baseline computation, and DE optimization, is available at:
`https://github.com/Myouza/fast-inverse-sqrt-research`

References

[1] id Software. Quake III Arena GPL source release. GitHub repository, 2005.

[2] M. Robertson. *A Brief History of InvSqrt*. Ph.D. dissertation, University of New Brunswick, 2012.

[3] C. Lomont. *Fast inverse square root*. Technical report, 2003.

[4] R. Storn and K. Price. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[5] S. Das and P. N. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[6] N. N. Schraudolph. A Fast, Compact Approximation of the Exponential Function. *Neural Computation*, 11(4):853–862, 1999.

[7] P. Micikevicius et al. Mixed Precision Training. *International Conference on Learning Representations (ICLR)*, 2018.

[8] M. Courbariaux, Y. Bengio, and J. David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Appendix A. Chebyshev Derivation Details

For completeness, we provide the full derivation of the optimal σ.

The error in the linearization is:

$$E(m) = \log_2(1+m) - (m+\sigma) \tag{26}$$

Taking the derivative:

$$E'(m) = \frac{1}{(1+m)\ln 2} - 1 \tag{27}$$

Setting $E'(m) = 0$:

$$m^* = \frac{1}{\ln 2} - 1 \approx 0.44269504 \tag{28}$$

At the critical point and endpoints:

$$E(0) = -\sigma \tag{29}$$

$$E(1) = 1 - 1 - \sigma = -\sigma \tag{30}$$

$$E(m^*) = \log_2(1+m^*) - m^* - \sigma \tag{31}$$

The equioscillation condition $E(m^*) = -E(0)$ gives:

$$\log_2(1+m^*) - m^* - \sigma = \sigma \tag{32}$$

Therefore:

$$\sigma = \frac{\log_2(1+m^*) - m^*}{2} = \frac{\log_2(1/\ln 2) - (1/\ln 2 - 1)}{2} \approx 0.0430357 \tag{33}$$

Appendix B. Newton-Raphson Error Bound

Let $y^* = x^{-1/2}$ be the true value and $\varepsilon_n = (y_n - y^*)/y^*$ be the relative error.
Then $y_n = y^*(1+\varepsilon_n)$ and:

$$y_{n+1} = y_n\left(\frac{3}{2} - \frac{xy_n^2}{2}\right) \tag{34}$$

$$= y^*(1+\varepsilon_n)\left(\frac{3}{2} - \frac{(1+\varepsilon_n)^2}{2}\right) \tag{35}$$

$$= y^*(1+\varepsilon_n)\left(1 - \varepsilon_n - \frac{\varepsilon_n^2}{2}\right) \tag{36}$$

Expanding and collecting terms:

$$\varepsilon_{n+1} = -\frac{3}{2}\varepsilon_n^2 - \frac{1}{2}\varepsilon_n^3 \tag{37}$$

For small ε_n:

$$|\varepsilon_{n+1}| \approx \frac{3}{2}\varepsilon_n^2 \tag{38}$$

This quadratic convergence explains why a 0.175% error after one Newton iteration becomes 4.7×10^{-6} after two iterations:

$$\frac{3}{2}(0.00175)^2 \approx 4.6 \times 10^{-6} \tag{39}$$