TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Dwarka, V. (2026). Towards energy-efficient scientific computing: Reversible numerical linear algebra kernels in floating-point arithmetic. *Sustainable Computing: Informatics and Systems*, *49*, Article 101261. https://doi.org/10.1016/j.suscom.2025.101261

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Sustainable Computing: Informatics and Systems 49 (2026) 101261

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Towards energy-efficient scientific computing: Reversible numerical linear algebra kernels in floating-point arithmetic

V. Dwarka [a]

[a] Delft University of Technology, Mekelweg 4, Delft, 2628 CD, The Netherlands

ARTICLE INFO

Keywords:
Reversible computing
Numerical linear algebra
Floating-point arithmetic
Energy efficiency
AI energy
Ill-conditioned systems
Krylov methods
LU factorization

ABSTRACT

Frontier scientific and AI workloads now reach $10^{19} - 10^{25}$ fused multiply–add (FMA) operations per run (on the order of $2 \times 10^{19} - 2 \times 10^{25}$ FLOPs). At today's ∼10 pJ per FMA, this corresponds to approximately $10^{8} - 10^{14}$ joules of arithmetic energy. At this scale, energy becomes the limiting resource for continued growth in computational workloads, motivating a re-evaluation of long-standing algorithmic assumptions. It is often assumed that reversible computing only matters near the Landauer limit. Building on prior physical arguments that full energy recovery is only possible when computation preserves information, we demonstrate that this same requirement governs floating-point numerical kernels: overwriting state enforces a non-zero energy floor, even under ideal recovery. Thus, eliminating this wall in practice requires that the numerical algorithm itself be injective. We therefore present the *first* reversible floating-point realizations of core dense numerical kernels—matrix multiplication, LU factorization, and conjugate-gradient iteration—that retain rounding information rather than discarding it. Implemented directly in IEEE arithmetic, they achieve machine-precision forward–reverse agreement on well- and ill-conditioned problems with minimal auxiliary state. A toggle-based model with measured switching costs and realistic recovery factors predicts 10^{3}–$10^{4}\times$ reductions in arithmetic energy. These results establish injective floating-point kernels as a foundation for energy-recovering numerical computation, and indicate that realizing this potential will require sustained co-design across applied mathematics, computer science, and hardware engineering.

1. Introduction

Scientific computing underpins progress across domains, from plasma physics and climate modeling to large-scale machine learning, with core workloads dominated by iterative and direct solvers in numerical linear algebra [1,2]. As simulation fidelity and model size expand, leading systems now execute on the order of $10^{19} - 10^{25}$ FMA-operations per run [3,4]. Historically, the energy required for these arithmetic operations has not posed a hard constraint: algorithmic design has been driven primarily by accuracy, stability, and asymptotic complexity, while device scaling absorbed the computational burden. That assumption is beginning to change. With Moore's Law slowing and Dennard scaling long concluded [5,6], the growth of scientific and AI workloads is increasingly limited by power delivery and thermal budgets. In this regime, in settings where energy efficiency is a constraint, the energy cost of floating-point operations is no longer negligible.

While quantum computing remains a long-term prospect for general-purpose scientific and AI workloads, reversible computing provides a classical and near-term path towards ultra-low-energy operation, grounded in Landauer's principle, linking information erasure to heat dissipation. A common misconception is that such considerations matter only when approaching the thermodynamic limit. Both theory and experiment indicate otherwise. CMOS-compatible adiabatic logic [7,8] and superconducting logic families [9–11] have demonstrated energy-recovering switching, and when the computational state is reset rather than preserved, measurable dissipation is observed far above the Landauer scale. These devices can reduce dynamic losses even for conventional irreversible kernels, but overwriting the state still forces reset and dissipation, leaving a non-zero energy floor. In contrast, injective kernels avoid reset events and are, in principle, the only route towards asymptotically vanishing energy per operation.

Advanced numerical methods, such as optimized preconditioners, multigrid schemes, and Krylov solvers used in both scientific simulation and large-scale machine learning, substantially reduce computational cost, but they do not change the destructive update structure of classical linear algebra algorithms. For example, dense general matrix–matrix multiplication (GEMM) and LU factorization overwrite $\mathcal{O}(n^2)$ entries over $\mathcal{O}(n)$ accumulation or elimination steps, yielding $\mathcal{O}(n^3)$ in-place updates, and conjugate-gradient (CG) and related Krylov methods repeatedly overwrite $\mathcal{O}(n)$-dimensional vectors across m iterations,

E-mail address: v.n.s.r.dwarka@tudelft.nl.

https://doi.org/10.1016/j.suscom.2025.101261
Received 1 August 2025; Received in revised form 3 November 2025; Accepted 23 November 2025
Available online 20 December 2025
2210-5379/© 2025 The Author. Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

giving $\mathcal{O}(nm)$ such updates. Each such overwrite discards information, and even on energy-recovering hardware, this forces a reset operation and imposes a non-zero energy floor. Thus, lowering FLOP count alone cannot eliminate irreversibility. As problem sizes scale in scientific computing and AI, and as degrees of freedom increase in high-resolution PDE models and high-dimensional learning systems, the fixed energy costs of irreversible arithmetic are amplified. Linear algebra kernels sit at the core of these workloads: dense matrix multiplication and LU factorization underpin direct solvers and dense linear–algebra pipelines, while the CG method remains the standard choice for large sparse systems in PDE simulation and increasingly appears in optimization–driven machine–learning settings (e.g. Laplacian systems in graph learning, diffusion models, kernel methods, and Hessian–vector products in second-order or constrained optimization). The reversible formulations developed here therefore target the arithmetic core shared by PDE solvers, numerical optimization, and learning systems.

Although energy-recovering hardware is advancing, numerical algorithms have historically not been formulated to preserve state. Prior work on reversible computing has focused on logic-level techniques or integer arithmetic, while floating-point numerical solvers continue to rely on in-place updates and irreversible rounding. As a result, no framework exists for injective floating-point kernels that maintain numerical stability, while avoiding Bennett-style memory growth. An instructive parallel, however, does exist in quantum computing: well before large-scale quantum processors were available, algorithmic advances such as Harrow–Hassidim–Lloyd (HHL) [12] established mathematical foundations ahead of hardware maturity. In the same spirit, developing injective counterparts to core numerical kernels is a timely foundational step towards a co-designed ultra-low-energy scientific computing stack.

1.1. Related works and state-of-the-art

The foundations of reversible computing were laid by Landauer [13], who established the thermodynamic cost of bit erasure as $k_B T \ln 2$. Bennett [14] extended this by introducing reversible Turing machines and full history-based logging and computation to eliminate entropy loss.

Frank's seminal PhD thesis and work [15,16] explored general-purpose reversible computing architectures, including reversible linear algebra subroutines and simulation of quantum systems. His generalized reversible computing framework later formalized that deterministic operations eject no entropy *if and only if* their state-transition relation is injective, establishing logical one-to-one mappings as necessary for asymptotically vanishing dissipation [17]. His more recent work on adiabatic logic furthered CMOS-compatible energy-recovery computing [8,18]. Complementary experimental evidence comes from superconducting adiabatic logic. Adiabatic quantum-flux-parametron (AQFP) devices achieve ultra-low-energy switching in fully reversible operation [10]. Herr shows that when information is discarded in AQFP, the circuit must either (i) explicitly terminate a signal, incurring the Landauer cost $k_B T \ln 2$ per bit, or (ii) undergo non-adiabatic reset backaction that dissipates energy several orders above $k_B T \ln 2$ [11]. Thus, even in superconducting adiabatic logic, irreversible updates introduce dissipation, and full energy recovery requires logically injective computation. An industry perspective likewise emphasizes that full adiabatic energy recovery in CMOS requires logically invertible transitions [19].

At the arithmetic level, Frank and Edwards note that efficient reversible designs for integer and floating-point adders, multipliers, and fused multiply–add (FMA) units remain in their infancy, with substantial room for innovation [19]. Their discussion concerns device- and circuit-level arithmetic primitives rather than numerical kernels or stability properties. Importantly, the existence of a reversible adder or FMA does not make classical dense numerical algorithms reversible. Running conventional numerical linear algebra algorithms on a reversible datapath would therefore still require Bennett-style checkpointing to recover erased state, leading to impractical memory growth. In reversible arithmetic, the TSG gate by Thapliyal and Srinivas [20] introduced compact full-adder functionality, and subsequent designs [21–24] implemented IEEE-754-compliant floating-point adders for FPGA or quantum datapaths. These approaches target hardware datapaths and do not address the algorithmic injectivity. Perumalla and Yoginath developed Basic Linear Algebra Subprograms (BLAS) [25], a reversible library for linear algebra with paired forward–reverse routines. Their library restores the state by replaying inverse calls. However, the underlying IEEE arithmetic still overwrites intermediate results and discards rounding information, making this software-level reversibility rather than injective floating-point arithmetic compatible with physical energy-recovery constraints.

More recently, Demaine, Lynch, and Sun [26] presented reversible algorithms for matrix multiplication and regression using injective arithmetic over finite rings. Their approach avoids floating-point arithmetic and does not engage with noise resilience, rounding error, or hardware toggle behavior. Laksmi and Sudha [27] propose an efficient reversible-logic design for a single-precision IEEE754 floating-point subtractor, including a reversible comparator, subtractor, leading-zero detector, and normalization units, implemented and synthesized in adiabatic CMOS. Goubault de Brugière et al. [28] used LU factorization in symbolic CNOT circuit synthesis, while Jain and Agrawal [24] propose one of the few gate-level reversible IEEE-754 floating-point designs, analyzing quantum cost and garbage outputs.

Finally, several reversible programming languages are available. Janus [29] is one of the earliest reversible languages, featuring paired constructs for allocation/deallocation and a syntactic discipline that guarantees every assignment is invertible. It supports integer arithmetic and pointer-style memory operations. Eel [30] extends Janus by introducing reversible control structures (loops and conditionals) with energy annotations. Like Janus, Eel remains limited to integer data and requires manual construction of all arithmetic primitives. Finally, Nilang [31] is a minimal domain-specific language targeting bit-level reversibility, where programs are composed from invertible primitives such as CNOT, XOR, and rotate. While it enables reversible circuit synthesis for small fixed-size integer kernels, Nilang does not include floating-point representations. Hence, classical reversible-logic recipes (e.g. Bennett's method) guarantee bijectivity but rely on full checkpointing and ignore floating-point semantics, leading to impractical $O(n^3)$ memory blow-up and loss of numerical fidelity.

Commercial efforts, such as Vaire Computing [32–34], are actively developing reversible CMOS chips and have recently received widespread media attention for demonstrating early-stage prototypes.

1.2. Research gap and contribution

In light of the above, this paper addresses a critical and previously unfilled gap by answering the following research question: Can core scientific computing algorithms be fundamentally redesigned to operate within reversible computing paradigms, while accounting for floating-point arithmetic and numerical rounding error? If so, what is the computational and memory overhead required to retain full information, both in theory and in practice?

This work opens a new research direction for reversible numerical computing, similar in spirit to how quantum linear solvers such as HHL catalyzed engagement between quantum hardware and numerical mathematics. This paper fills these gaps through five key contributions:

1. We introduce mathematically injective floating-point primitives Add/Subtract, Multiply, and Reciprocal on double–double pairs that capture every IEEE 754 rounding bit without ancillary logs.

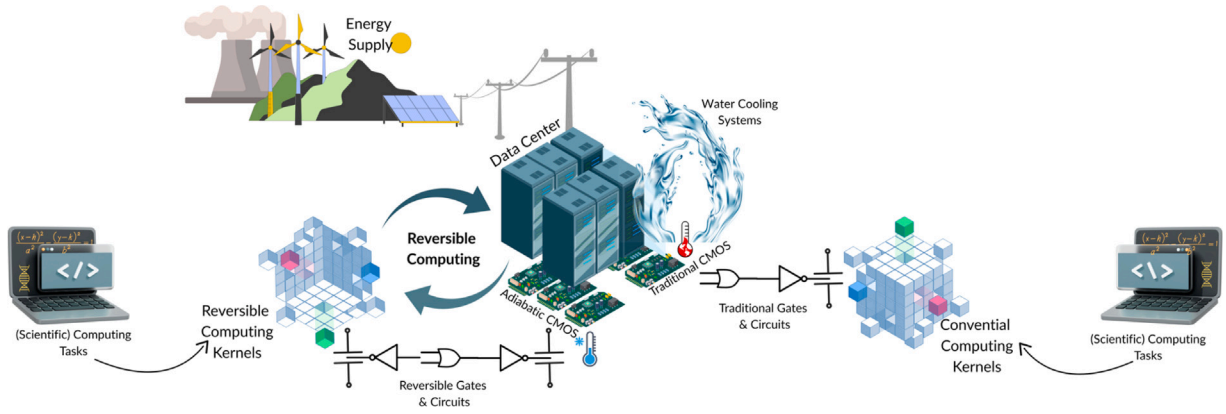


Fig. 1. Simplified depiction of the computational stack from energy supply and cooling infrastructure through high consumption algorithms underpinned by numerical linear algebra kernels (matrix matrix multiplication, matrix vector multiplication, iterative solvers and Gaussian elimination). It does not take into account network latency, storage overhead, software scheduling inefficiencies or hardware fault tolerance and resource contention.

2. We develop minimal-logging reversible variants of three core numerical kernels (GEMM, LU with partial pivoting, and the iterative CG solver) by retaining only the exact metadata needed for bijectivity and reducing memory overhead from cubic or quadratic to quadratic or linear.
3. We provide an emulator and simulator that validate numerical stability, accuracy, and exact reversibility in double–double arithmetic on both well- and severely ill-conditioned systems on 2D Poisson problems requiring hundreds of CG iterations.
4. We present a unified cost and energy projection model incorporating reversible-switching overhead η, injectivity fraction θ, recovery factor ρ, and adiabatic quality factor R. Under moderate recovery, we project $10^3 - 10^4\times$ reductions in arithmetic energy, i.e., $10^{3-4}\times$ over workloads consuming $10^8 - 10^{14}$ J.
5. We establish the first formal link between floating-point injectivity and physical energy recovery. Appendix A proves that non-injective floating-point updates necessarily incur erased-state energy even under ideal recovery, establishing an electrostatic wall far above the Landauer limit, while injective constructions avoid this term. It further derives a one-to-one correspondence between the erased state and the operational energy-model terms (θ, η, ρ), providing a mathematical foundation for energy scaling.

These advances establish the missing high-level algorithmic layer that enables future energy recovery in large-scale numerical linear algebra. By investing now, before reversible hardware is widely deployed, we lay the foundation for a co-design pipeline linking mathematics, algorithms, and physical implementation under the constraints of future energy-aware computing [5]. Together, the algorithmic contributions developed here form a critical layer in the reversible computing stack: they enable numerical solvers to be translated into gate-level circuits, where hardware-level optimizations such as adiabatic or ballistic reversible logic can be applied effectively, as depicted in Fig. 1.

2. Theoretical framework and mathematical preliminaries

Logical erasure incurs irreducible energy cost [13,14]. Frank's conditional reversibility principle [15] states that asymptotically vanishing dissipation requires injectivity on the executed inputs. Here, we express this requirement at the level of floating-point kernels by identifying the information whose loss forces nonzero erased capacitance and therefore a finite dynamic switching cost. A short argument is given below, the full derivation appears in Appendix A.

2.1. Algorithmic injectivity and physical dissipation bounds

Let $f : X \to Y$ denote the deterministic computational map on the supported input set S. If $x_1 \neq x_2 \in S$ and $f(x_1) = f(x_2)$, information is lost and zero dissipation is impossible on those inputs.

Each logical state variable is represented by a capacitor node with voltage $V_i(t)$. Let $\mathcal{E}$ be the set of nodes whose terminal voltage is independent of the input, and define the erased capacitance $C_{\text{eras}} = \sum_{i\in\mathcal{E}} C_i$. A closed forward-adiabatic-reverse cycle between two supported inputs with identical outputs yields a nonzero net capacitive–energy drop if any node is forced to an input–independent terminal value. Thus,

$$E_{\text{dyn}} \to 0 \iff C_{\text{eras}} = 0 \iff f \text{ injective on supported inputs.}$$

Adiabatic time-dilation removes transport losses but does not change C_{eras}. Consequently, whenever $C_{\text{eras}} > 0$, the erased capacitance produces **a fixed electrostatic wall** for dynamic dissipation, independent of clock slowing. Algorithmic injectivity is therefore required in addition to circuit-level adiabatic techniques.

2.2. Implication for floating-point kernels

Classical floating-point kernels, including GEMM, LU, and CG, overwrite intermediate state and therefore induce $C_{\text{eras}} > 0$, implying a nonzero electrostatic wall and dynamic energy floor even on ideal reversible hardware.

The reversible kernels developed below preserve the information identified above, so that no logical register is driven to an input-independent terminal value. Each update is injective on the executed state, giving $C_{\text{eras}} = 0$ and admitting asymptotically adiabatic execution in principle. Section 6.3 evaluates the resulting physical cost.

2.3. Canonical irreversibility

Reversible computation requires every elementary transformation to be bijective at the bit level. It requires that the mapping from input bits to output bits must admit a unique inverse. The numerical kernels that dominate large-scale simulation and optimization, however, systematically merge, round, or overwrite information. As a consequence, the forward update cannot be recovered from the available state without external logs. We illustrate this intrinsic loss of information for three workhorse algorithms: GEMMs, Gaussian elimination with partial pivoting (LU), and the CG method.

GEMM

The standard matrix–matrix product with floating-point representation (fl)

$$C = AB, \qquad c_{ij} = \mathrm{fl}\Big(\sum_{k=1}^{n} a_{ik}\, b_{kj}\Big),$$

is non-injective even in exact arithmetic, since the map $(A, B) \mapsto C$ merges two $n \times n$ inputs into one and many distinct pairs (A, B) yield the same C. In floating-point practice, each individual multiplication and each subsequent addition round their results to t-bit precision, discarding low-order bits. In modern hardware, FMA-instructions combine multiplication and addition into one operation but still round the final result to t bits, likewise erasing information.

Without recording every partial sum and rounding error, the forward update cannot be inverted. Hence, textbook GEMM destroys information both algorithmically (overwriting two matrices by one) and numerically (bit erasure in rounding) and is inherently irreversible. The algorithm for performing GEMMs is given in 1.

Algorithm 1 Standard GEMM (Matrix–Matrix Multiply)

Input: $A, B \in \mathbb{R}^{n\times n}$
Output: $C \in \mathbb{R}^{n\times n}$
$C \leftarrow 0$
for $i = 0$ **to** $n-1$ **do**
  for $j = 0$ **to** $n-1$ **do**
    $c_{ij} \leftarrow 0$
    for $k = 0$ **to** $n-1$ **do**
      $c_{ij} \leftarrow c_{ij} + a_{ik}\, b_{kj}$ {FMA}
    end for
  end for
end for
return C

LU factorization and Gaussian elimination

LU factorization is a direct method for solving a linear system $Ax = b$ by decomposing $A = LU$, where L is unit lower-triangular and U is upper-triangular, and then performing forward and backward substitution. The algorithm for performing LU factorizations is given in 2.

Unlike the iterative CG method, which builds successive approximations to the solution, Gaussian elimination with partial pivoting computes the exact solution in a finite number of steps (up to rounding error). At step k of the factorization, the update rule

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \ell_{ik}\, u_{kj}, \qquad \ell_{ik} = \frac{a_{ik}^{(k)}}{u_{kk}} \tag{1}$$

subtracts a finite precision product from the current entry. Because the diagonal entry u_{kk} can become very small—leading to large multipliers and amplified round-off error—partial pivoting is used. Pivoting swaps the current row with the one having the largest magnitude entry in column k, thereby controlling growth factors and improving numerical stability.

Once the rounded value $a_{ij}^{(k+1)}$ is stored, the low bits of both operands are irretrievably lost. Pivoting compounds the effect. Swapping two rows replaces their previous positions without keeping a copy, which eliminates the ability to infer the original ordering unless an explicit pivot log is retained. Moreover, floating-point subtraction is many-to-one, as infinitely many distinct pairs (x, y) round to the same result when $|x - y|$ is below machine resolution.

The completed LU factors and the pivot vector therefore contain insufficient information to reconstruct the initial matrix exactly, and additional states must be preserved if reversibility is desired.

Algorithm 2 Standard LU with Partial Pivoting

Input: $A \in \mathbb{R}^{n\times n}$
Output: L (unit lower-triangular), U (upper-triangular), pivot vector p

for $k = 0$ **to** $n-1$ **do**
  $p_k \leftarrow \arg\max_{i=k,\dots,n-1} |a_{ik}|$
  exchange rows k and p_k of A
  for $i = k+1$ **to** $n-1$ **do**
    $a_{ik} \leftarrow a_{ik}/a_{kk}$ $\{= \ell_{ik}\}$
    for $j = k+1$ **to** $n-1$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik}\, a_{kj}$
    end for
  end for
end for
$L \leftarrow \mathrm{tril}(A,-1) + I, \quad U \leftarrow \mathrm{triu}(A)$
return L, U, p

Conjugate gradient (CG).

CG, unlike the direct LU factorization, is an iterative Krylov subspace algorithm that solves a symmetric positive definite linear system $Ax = b$ by successively minimizing the quadratic energy

$$\phi(x) = \tfrac{1}{2} x^{\mathsf{T}} A x - b^{\mathsf{T}} x$$

over the expanding subspaces

$$\mathcal{K}_k(A, r_0) = \mathrm{span}\{r_0,\ Ar_0,\ A^2 r_0, \dots, A^{k-1} r_0\}.$$

These subspaces are called Krylov subspaces [1], and each step produces an A–orthogonal search direction p_k and advances the iterate x_k by a scalar step length α_k chosen to minimize ϕ along p_k. In exact arithmetic, the method converges in at most n iterations.

Because it requires only one sparse matrix–vector product, two inner products, and three vector axpy operations per iteration, CG is the de facto standard for large sparse symmetric positive definite (SPD) systems arising from finite element or finite volume discretizations of elliptic partial differential equations, including Poisson, linear elasticity, and thermal diffusion models.

The same kernel appears inside multigrid smoothers, implicit time stepping of parabolic problems, Hessian–vector products in second-order optimization [35], and graph Laplacian solvers in machine learning [36,37]. Its arithmetic footprint, therefore, dominates runtimes and energy budgets in a wide spectrum of scientific and engineering workloads.

The elementary operations inside one CG iteration are similarly *not* bijective.

- Dot products. Both $\langle r_k, r_k\rangle$ and $\langle p_k, Ap_k\rangle$ map an n–vector to a single scalar. The mapping is many-to-one even in exact arithmetic, and floating-point rounding coalesces infinitely many distinct inputs into the same output word. Once the scalar is stored, the original high-dimensional information is lost.
- Vector overwrites. The updates

  $$x_{k+1} = x_k + \alpha_k p_k,$$
  $$r_{k+1} = r_k - \alpha_k A p_k,$$
  $$p_{k+1} = r_{k+1} + \beta_k p_k,$$

  and replaces x_k, r_k, and p_k in memory. Unless every previous version of these vectors is logged, the state needed to invert the transformation is deleted.

Similarly, each add or multiply rounds to the nearest representable number, which merges neighboring input pairs. This rounding noise is again irreversible unless the exact rounding error is kept alongside every result.

Consequently, the textbook CG loop destroys information at every iteration. To run CG on reversible hardware without dissipation from bit erasure, one must redesign the algorithm so that all intermediate vectors or their reversible summaries remain available, or switch to an intrinsically reversible Krylov scheme that avoids irreversible reductions and overwrites.

Algorithm 3 Standard CG

Input: symmetric positive definite $A \in \mathbb{R}^{n\times n}$, right-hand side b, initial guess x_0, iteration limit m
Output: approximate solution x_m

$r_0 \leftarrow b - Ax_0$ {residual}
$p_0 \leftarrow r_0$
$\rho_0 \leftarrow r_0^{\mathsf{T}} r_0$
for $k = 0$ **to** $m-1$ **do**
  $q_k \leftarrow Ap_k$
  $\alpha_k \leftarrow \rho_k / (p_k^{\mathsf{T}} q_k)$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} \leftarrow r_k - \alpha_k q_k$
  if $\|r_{k+1}\|_2 < \mathrm{tol}$ **then break**
  $\rho_{k+1} \leftarrow r_{k+1}^{\mathsf{T}} r_{k+1}$
  $\beta_{k+1} \leftarrow \rho_{k+1}/\rho_k$
  $p_{k+1} \leftarrow r_{k+1} + \beta_{k+1} p_k$
end for
return x_{k+1}

While hardware can, in principle, be built from perfectly bijective logic gates, much of the information loss, and therefore the unavoidable energy cost, actually originates in the mathematical algorithms we run. First, almost all scientific codes today rely on standard finite-precision formats (e.g., IEEE-754), whose add, multiply or FMA-operations round away low-order bits and discard them permanently. Second, high-level solvers routinely overwrite large data structures, collapsing two matrices into one in GEMM, replacing rows during LU, or updating vectors in CG. This irreversibly discards entire degrees of freedom unless extra metadata is carried along. Even with a fully reversible processor, these algorithmic choices would continue to erase information at the software level, tying energy dissipation directly to the arithmetic workload. Only by redesigning numerical methods to preserve every bit of intermediate state, or by adopting genuinely reversible number formats and control-flow strategies, can we break this fundamental link between algorithmic irreversibility and heat generation.

2.4. Reversible reformulations of core kernels

The two sources of information loss identified above can be addressed simultaneously. We first adopt a double double representation, where each real number is stored as an ordered pair. This leads to the following arithmetic operations, as listed below.

2.4.1. Addition and subtraction

We represent every real number by a pair (h, ℓ), called a *double double* word, with $|\ell| \le 2^{-53}|h|$ and value $h + \ell$. To add two pairs we use the TwoSum identity of Dekker [38]:

$$
\begin{aligned}
s,\ e_1 &= \textsc{TwoSum}(h_1, h_2),\\
t,\ e_2 &= \textsc{TwoSum}(\ell_1, \ell_2),\\
u,\ e_3 &= \textsc{TwoSum}(e_1, t),\\
(h_s, \ell_s) &= \mathrm{Add}\big((h_1,\ell_1),\,(h_2,\ell_2)\big)\\
&= \big(s,\ (\ell_1+\ell_2) + (e_1+e_2+e_3)\big).
\end{aligned}
$$

TwoSum returns the rounded sum and its exact rounding error in a single step. All local errors e_1, e_2, e_3 are accumulated in the low word, so the pair stores the exact arithmetic sum. Because each call to TwoSum is injective on $(a, b) \mapsto (a + b, \mathrm{err})$ and both the high and low words are retained, the composite map from two input pairs to the output pair is itself injective. Subtraction is realized by addition with the second operand negated, so it inherits the same property.

The cost breakdown is as follows. A single TwoSum requires two standard floating-point operations (flops) (one add and one subtraction to extract the rounding error). Pair addition invokes three TwoSum calls, thus six flops in total, plus two additional adds to accumulate $\ell_s = (\ell_1 + \ell_2) + (e_1 + e_2 + e_3)$. That sums to 8 flops per double-double addition, yet requires no extra memory beyond the two output words.

2.4.2. Multiplication

For multiplication, we use the TwoProd algorithm of Dekker. Let (h_1, ℓ_1) and (h_2, ℓ_2) be the operands and set

$$
\begin{aligned}
p,\ e_1 &= \textsc{TwoProd}(h_1, h_2), \qquad & c_1, e_2 &= \textsc{TwoProd}(h_1, \ell_2),\\
c_2, e_3 &= \textsc{TwoProd}(\ell_1, h_2), & s,\ e_4 &= \textsc{TwoSum}(c_1, c_2),\\
t,\ e_5 &= \textsc{TwoSum}(e_1, e_2), & u,\ e_6 &= \textsc{TwoSum}(e_3, e_4),\\
v,\ e_7 &= \textsc{TwoSum}(t, s), & (h_p, e_8) &= \textsc{TwoSum}(p, v).
\end{aligned}
$$

The final pair is

$$
\begin{aligned}
(h_p, \ell_p) &= \mathrm{Mul}\big((h_1,\ell_1),\,(h_2,\ell_2)\big)\\
&= \big(h_p,\ \ell_1\ell_2 + e_5 + e_6 + e_7 + e_8\big).
\end{aligned}
$$

The low word stores every rounding residue, including the product of the two low components, so (h_p, ℓ_p) equals the exact real product. Each TwoProd and TwoSum call is injective, hence the overall mapping from the two input pairs to the result pair is injective.

The cost breakdown is as follows. Each TwoProd call implemented via Dekker splitting requires four standard multiplications and eight floating-point adds for a total of twelve flops. With three TwoProd calls, this contributes thirty-six operations. Each TwoSum costs two operations (one add and one subtract), so five calls add ten operations. Finally, accumulating the low word term $\ell_1\ell_2$ requires one extra multiply. In total, the pair multiplication consumes $36 + 10 + 1 = 47$ flops, yet introduces no additional memory beyond the two output words.

2.4.3. Division

Given a pair $x = (h, \ell)$ with $h \neq 0$ and $|\ell| \le 2^{-53}|h|$, we start from the high-word reciprocal

$$y_0 = \left(\tfrac{1}{h},\ 0\right).$$

Next, we refine this approximation three times with Newton's method [39]

$$y_{k+1} = y_k\left(2 - x\,y_k\right), \qquad k = 0, 1, 2. \tag{2}$$

All products and subtractions in (2) are carried out with the reversible pair primitives as stated above.

Hence, if we let $\varepsilon_k = x^{-1} - y_k$ denote the exact error after step k, then $\varepsilon_{k+1} = \varepsilon_k^2\, x$, so the number of correct bits doubles each iteration. Since $|\varepsilon_0|/|x^{-1}| \le 2^{-53}$, three steps yield

$$|\varepsilon_3| \le 2^{-212}\,|x^{-1}|,$$

well below the working precision of a double-double pair.

Every operation in (2) is injective on pairs, and the update variable y_k appears on both sides of the assignment. Hence the map $(x, y_k) \mapsto y_{k+1}$ is bijective. Applying the three refinements forward therefore has a unique inverse obtained by traversing the same relation backward from y_3. The reciprocal and the induced division

$$(x, y) \longmapsto x\,y^{-1}$$

are reversible without any external log.

Each Newton iteration consists of two pair multiplications, one pair subtraction, and one multiply by two. A pair multiplication costs 12 flops for each TwoProd plus 10 flops for 5 TwoSum calls, or 47 flops in total. A pair subtraction requires 3 TwoSum calls and 2 additional adds

for 8 flops. A multiply by two is one floating-point addition. Thus one iteration consumes $2\times47+8+1 = 103$ flops. Performing three iterations therefore uses $3\times103 = 309$ flops. This uses only a single double-double output slot to hold the final reciprocal (or quotient), with no additional temporary storage required.

2.4.4. Multiplication (FMA)

An FMA-instruction performs a multiplication followed by an addition with only one final rounding, thereby preserving more precision than separate multiply and add operations [40]. In an FMA-assisted two-prod, we can compute

$$p = h_1 h_2, \qquad e_1 = \mathrm{FMA}(h_1, h_2, -p),$$

to capture the exact rounding error of the high word in one multiply and one FMA. We then form

$$c = h_1\ell_2 + \ell_1 h_2 + \ell_1\ell_2 + e_1,$$

and renormalize via one TwoSum to produce the final pair (h_p, ℓ_p), which equals the exact product x_1x_2. In the Dekker two prod path each TwoProd costs four multiplies plus eight adds plus five TwoSum calls, for a total of 47 flops.

In the FMA-version, each pair multiplication requires 11 flops. First, the high-word product and its rounding error are formed using one multiply and one FMA, for three operations in total. Then, three cross-term products are computed and summed, incurring three multiplications and three adds, for six operations. Finally, a TwoSum call renormalizes the result in two operations. Each pair subtraction in the Newton update uses three TwoSum calls and two adds, for eight operations. A multiply by two costs one operation. Thus, one Newton iteration, consisting of two pair multiplications, one pair subtraction, and one multiply by two, uses 31 flops. Performing three iterations, therefore, requires 93 flops. Computing the final quotient by multiplying by the original value adds one more pair multiplication of 11 flops, bringing the total cost of the division to 104 flops.

3. Reversible numerical linear algebra and iterative solvers

The three bijective pair-arithmetic kernels introduced in Section 2.4 enable a direct re-implementation of the work-horse routines. Our guiding principle is simple:

> Replace every irreversible floating-point instruction in the textbook algorithm by its reversible pair analog, and retain the minimal metadata that makes the high-level map injective.

Because the reversible constructions preserve IEEE 754 semantics, the numerical update maps are identical to the classical algorithms, and classical backward-stability theory applies unchanged [41,42]. The numerical tests in Section 5 therefore validate bijectivity and bit-exact reversibility against classical floating-point results.

Fig. 2 visualizes the resulting dependency chain, where every intermediate kernel in both algorithms collapses, via a finite composition, to the same three colored primitives.

3.1. From classical to reversible GEMM

Following our guiding principle as outlined previously, in our reversible variant (Algorithm 4), we

1. replace each scalar multiply and add by pair-level *Multiply* and *Add* on double–double pairs (h, ℓ), so no rounding information is lost.
2. log only the final double–double output matrix $C_{\mathrm{dd}} \in (\mathbb{R}^2)^{n\times n}$, discarding all intermediate partial sums.

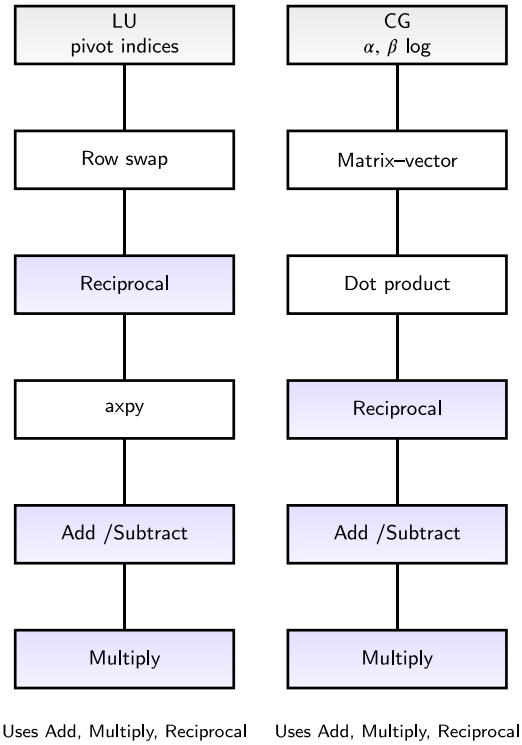


Fig. 2. Dependency scheme for the reversible solver stack. Each intermediate kernel ultimately calls only the three colored primitives (Add/Subtract, Multiply, Reciprocal), making LU and CG fully reversible.

No further state is required. The forward map (Algorithm 4, left)$(A, B) \mapsto C_{\mathrm{dd}}$ is now bijective, and its inverse is Algorithm 4 executed backwards (right).

Applying Bennett checkpointing to the standard GEMM algorithm requires logging every inner loop partial sum, resulting in an extra $O(n^3)$ storage of double-double words and an effective $O(n^4)$ work due to repeated recomputation. In contrast, our reversible variant logs only the final output matrix of size n^2, double-double words, reducing the additional memory requirement to $O(n^2)$ while performing the same $O(n^3)$ pair arithmetic operations exactly once in both the forward and reverse sweeps. We retain the final output matrix so that we can both verify numerical accuracy and use it as the starting point for the exact reverse computation

3.2. From classical to reversible LU

Classical LU overwrites two entire sub-matrices at each step and discards intermediate rounding residues. In the reversible variant (Algorithm 5) we

1. store A as a matrix of pairs and perform the update $a_{ij} \leftarrow a_{ij} - \ell_{ik}u_{kj}$ with *Add* and *Multiply* on pairs, so no rounding information is lost.
2. compute each multiplier ℓ_{ik} with a single *Reciprocal* call, then log the pivot index before every swap.

No further state is required. The forward map $(A) \mapsto (M, p)$ is now bijective, and its inverse is Algorithm 5 executed backwards.

Similarly, applying Bennett checkpointing to the standard LU factorization requires saving the entire $n\times n$ working matrix at each of the n elimination steps, leading to an extra $O(n^3)$ storage of double-double words and an effective $O(n^4)$ work due to repeated recomputation. In contrast, our reversible variant logs only the pivot list of length n, reducing the additional storage requirement to $O(n)$ while executing the

Algorithm 4 Reversible GEMM and its uncompute pass

() Reversible GEMM

Input: $A, B \in (\mathbb{R}^2)^{n\times n}$
Output: $C_{\text{dd}} \in (\mathbb{R}^2)^{n\times n}$
initialize $C_{\text{dd}}[i,j] \leftarrow (0,0)$ for all i,j
for $i = 0$ to $n-1$ **do**
  for $j = 0$ to $n-1$ **do**
    acc $\leftarrow (0,0)$
    for $t = 0$ to $n-1$ **do**
      $p \leftarrow$ dd_mul$(A_{i,t}, B_{t,j})$
      acc $\leftarrow$ dd_add(acc, p)
    end for
    $C_{\text{dd}}[i,j] \leftarrow$ acc
  end for
end for
return C_{dd}

() Uncompute

Input: $C_{\text{dd}}, A, B \in (\mathbb{R}^2)^{n\times n}$
Output: $Z \in (\mathbb{R}^2)^{n\times n}$ *(restored zero state)*
initialize $Z[i,j] \leftarrow (0,0)$ for all i,j
copy $Z[i,j] \leftarrow C_{\text{dd}}[i,j]$
for $i = n-1$ down to 0 **do**
  for $j = n-1$ down to 0 **do**
    $c \leftarrow Z[i,j]$
    for $t = n-1$ down to 0 **do**
      $p \leftarrow$ dd_mul$(A_{i,t}, B_{t,j})$
      $c \leftarrow$ dd_sub(c, p)
    end for
    $Z[i,j] \leftarrow c$
  end for
end for
return Z

same $O(n^3)$ pair arithmetic operations exactly once in both the forward and reverse sweeps.

Algorithm 5 Reversible LU with partial pivoting

Input: $A \in \mathbb{R}^{n\times n}$ (each entry a pair (h, ℓ))
Output: pair matrix M containing L (unit diagonal) and U, pivot list $p = (p_0, \ldots, p_{n-2})$
$M \leftarrow A$ {work in place}
for $k = 0$ **to** $n-2$ **do**
  $p_k \leftarrow \arg\max_{i\geq k} |M_{ik}|$ {row index with largest high word}
  swap rows k and p_k of M
  for $i = k+1$ **to** $n-1$ **do**
    $M_{ik} \leftarrow$ Reciprocal(M_{kk}) Multiply M_{ik} $\{\ell_{ik}\}$
    for $j = k+1$ **to** $n-1$ **do**
      $M_{ij} \leftarrow M_{ij}$ Add/Subtract $\left(M_{ik}$ Multiply $M_{kj}\right)$
    end for
  end for
end for
return M, p

3.3. From classical to reversible CG

In a standard CG loop, each iteration overwrites the state vectors and compresses two inner products into the scalars α_k and β_{k+1}, rendering the map non-injective. The reversible variant restores bijectivity through the following changes and is given in Algorithm 6:

1. All saxpy operations, inner products, and sparse matrix–vector products are executed with bit-preserving double–double primitives (*Add/Subtract*, *Multiply*, *Reciprocal*), so that no rounding information is discarded.
2. At the start of every iteration, a snapshot containing $(x_k, r_k, p_k, \alpha_k, \beta_{k+1})$ is appended to a log. These $3n+2$ words are exactly the data required to invert the three-term recurrence

$$x_{k+1} = x_k + \alpha_k p_k,$$
$$r_{k+1} = r_k - \alpha_k A p_k,$$
$$p_{k+1} = r_{k+1} + \beta_{k+1} p_k,$$

without re-evaluating the costly product matrix–vector product Ap_k.

Algorithm 6 Reversible CG and its uncompute pass

() Reversible CG

Input: SPD $A \in \mathbb{R}^{n\times n}$; b; initial $x \leftarrow x_0$; max iters m
Output: x_m and snapshots $\{(x_k, r_k, p_k, \alpha_k, \beta_{k+1})\}$
$r \leftarrow b - Ax$; $p \leftarrow r$
for $k = 0$ to $m-1$ **do**
  store (x, r, p) as (x_k, r_k, p_k)
  $Ap \leftarrow Ap$
  $\alpha \leftarrow \langle r, r\rangle / \langle p, Ap\rangle$
  $x \leftarrow x + \alpha\, p$
  $r_{\text{new}} \leftarrow r - \alpha\, Ap$
  $\beta \leftarrow \langle r_{\text{new}}, r_{\text{new}}\rangle / \langle r, r\rangle$
  $p \leftarrow r_{\text{new}} + \beta\, p$
  $r \leftarrow r_{\text{new}}$
  store (α, β) as (α_k, β_{k+1})
end for
return x and all stored tuples

() Uncompute

Input: x_m and tuples $(x_k, r_k, p_k, \alpha_k, \beta_{k+1})$ for $k = m-1$ to 0
Output: reconstructed initial guess x_0
for $k = m-1$ down to 0 **do**
  restore $(x_k, r_k, p_k, \alpha_k, \beta_{k+1})$
  $Ap_k \leftarrow A\, p_k$
  $r_k \leftarrow r_k + \alpha_k\, Ap_k$
  $x_k \leftarrow x_k - \alpha_k\, p_k$
  $p_k \leftarrow (p_k - r_k)/\beta_{k+1}$
end for
return x_0

Again, applying Bennett checkpointing would require saving all three state vectors of length n at each of the m iterations and recomputing them in a tree-stage unroll. This incurs $O(n\,m^2)$ extra storage and increases the overall work by a factor of $O(m)$. In contrast, our reversible variant appends exactly one snapshot of the three vectors (x_k, r_k, p_k) and the two scalars α_k and β_{k+1} per iteration, reducing the additional storage cost to $O(n\,m)$ while preserving bijectivity and executing each pair arithmetic operation exactly once in both forward and reverse sweeps.

Remarks on reversibility and memory

Algorithms 4, 5, and 6 are reversible by construction and do not rely on Bennett checkpointing (i.e. storing and later uncomputing *all* intermediate states, incurring $O(nm^2)$–$O(n^3)$ space). Each update is an injective composition of TwoSum, TwoProd (or FMA-assisted variants), and Reciprocal, and all rounding residues are retained in the low words. Only the mathematically indispensable metadata (pivot list in LU, per-iteration tuples in CG) is stored.

For GEMM (Algorithm 4), every update $c_{ij} \leftarrow$ dd_add $\big($ c_{ij}, dd_mul(A_{it}, B_{tj}) $\big)$ preserves all rounding bits inside the pair. Executing the same instructions in reverse order cancels the accumulation and restores the zero state from (C_{dd}, A, B), without intermediate logs.

In LU (Algorithm 5), the pair-level Schur updates together with the logged pivot index before each swap form a bijection, reversing the loop and undoing swaps recovers the input exactly.

In CG (Algorithm 6), the three-term recurrence $(x_k, r_k, p_k) \mapsto (x_{k+1}, r_{k+1}, p_{k+1})$ is injective in pair arithmetic. Storing $(x_k, r_k, p_k, \alpha_k, \beta_{k+1})$ per iteration is sufficient to invert the step without recomputing Ap_k, giving an $O(nm)$ reversible memory bound.

4. Cost and architectural overhead

In this section, we quantify the extra work in reversible gate counts. Throughout, n denotes the size of an $n\times n$ dense matrix, n_z the average nonzeros per row in a sparse matrix, and m the number of CG iterations.

We distinguish three complementary overhead factors:

γ (arithmetic-level runtime), η (gate-level logical energy), μ (memory-level storage).

Here, γ quantifies the slowdown relative to classical floating-point operations, η expresses the same arithmetic in reversible-gate counts, and μ

measures the increase in resident reversible state that must remain live to preserve bijectivity. All sources of overhead, including the local-error bookkeeping and resident reversible storage described in the previous subsection, are already incorporated into these factors. Consequently, every transformation in Algorithms 4–6 is exactly invertible, and no additional irreversibility is introduced anywhere in the solver stack.

4.1. Arithmetic work and complexity

As all of our reversible kernels operate in double–double arithmetic, each real number is stored as a pair (h, ℓ) whose sum equals the exact mathematical result. Accordingly, every IEEE-754 add or multiply is replaced by an injective pair-level routine for which we consider two software emulations:

- Dekker two-prod. As shown previously, a Dekker two-prod has an overhead of 47 flops. Hence, compared to a classic FMA (2 flops), Splitting each operand, one pair-multiply the Dekker overhead factor is

$$\gamma_{\text{Dekker}} = \frac{47}{2} \approx 24.$$

- FMA-two-prod. If hardware FMA is available, the emulation requires 11 raw flops per FMA. Hence, the overhead factor is

$$\gamma_{\text{FMA}} = \frac{11}{2} \approx 6.$$

Thus, as γ as the appropriate overhead factor, replacing each classical flop by γ raw flops, our three kernels require:

- GEMM Classical cost: $2n^3$ flops. Reversible cost:

$$N_{\text{pair}}^{\text{GEMM}} = \gamma\, 2n^3 = \begin{cases} 48\, n^3, & \gamma_{\text{Dekker}} = 24, \\ 12\, n^3, & \gamma_{\text{FMA}} = 6. \end{cases}$$

- LU factorization. Classical cost: $\frac{2}{3}n^3$ flops. Reversible cost:

$$N_{\text{pair}}^{\text{LU}} = \gamma\, \tfrac{2}{3} n^3 = \begin{cases} 16\, n^3, & \gamma_{\text{Dekker}} = 24, \\ 4\, n^3, & \gamma_{\text{FMA}} = 6. \end{cases}$$

- CG. One iteration uses $(n_z + 5n)$ flops plus two flops for the reciprocal. Over m iterations:

$$N_{\text{pair}}^{\text{CG}} = \gamma\, (n_z + 5n)\, m + 2\,\gamma \approx \begin{cases} 24\, (n_z + 5n)\, m, & \gamma_{\text{Dekker}} = 24, \\ 6\, (n_z + 5n)\, m, & \gamma_{\text{FMA}} = 6. \end{cases}$$

Because each γ is a fixed $O(1)$ constant, the asymptotic scaling of $O(n^3)$ for GEMM/LU and $O(nm)$ for CG remains unchanged, with only a constant multiplicative overhead. In our gate-level metric η, we therefore divide total Toffoli gates by the number of *classical FMAs*, ensuring compatibility with Roofline and hardware counter models.

4.2. Memory overhead and reversible state (μ)

The memory factor μ denotes the ratio of live reversible storage to the resident memory used in the corresponding classical algorithm. In the reversible representation, each floating-point variable is a pair $(h, \ell) \in \mathbb{R}^2$, where h stores the rounded value and ℓ accumulates the exact rounding residues returned by TwoSum or TwoProd. The low word is a permanent state rather than a temporary buffer: it holds the information that classical arithmetic discards. No auxiliary arrays or workspace buffers are introduced. Thus, one classical word becomes two reversible words, and the base factor is $\mu_{\text{pair}} = 2$.

In matrix multiplication the classical kernel holds three $n \times n$ matrices A, B, and C. In the reversible variant A and B are read only and never overwritten, and the output $C_{\text{dd}} \in (\mathbb{R}^2)^{n\times n}$ stores the result. No other state is required, so $M_{\text{GEMM}} = 2n^2$ and $\mu_{\text{GEMM}} = 2$. For LU factorization, a Bennett construction (logging all intermediate states and later uncomputing them) would require $O(n^3)$ additional words. The reversible LU kernel retains only the working pair matrix $M_{\text{dd}} \in (\mathbb{R}^2)^{n\times n}$ and the pivot list $p \in \mathbb{Z}^n$, giving $M_{\text{LU}} = 2n^2 + n$ and $\mu_{\text{LU}} \approx 2$. In CG the classical algorithm uses $O(n)$ memory for (x_k, r_k, p_k, Ap_k). Reversibility forbids overwriting, so a sufficient reversible state is one snapshot of $(x_k, r_k, p_k, \alpha_k, \beta_{k+1})$ per iteration, yielding $M_{\text{CG}} = 2(3n + 2)m = O(nm)$, with $\mu_{\text{CG}} \approx 2m$. This is a constructive upper bound.[1]

4.3. Reversible toffoli gates estimates (η)

As the previous section demonstrated that a hardware-supported FMA instruction substantially reduces computational overhead, we adopt the FMA as our baseline for further analysis. To estimate the Gate Count (GC) overhead, we now quantify the kernel cost in terms of reversible logic. For this purpose, we express all operations in Toffoli gate equivalents. Each arithmetic primitive is implemented as a bijective transformation that carries forward all bits needed to reconstruct its inputs and can be run in reverse by replaying the same gate sequence. Consequently, the quoted Toffoli count for each primitive already includes both the forward computation and its inverse uncompute.

4.3.1. FMA

Unfortunately, the literature does not provide a baseline estimate for the number of Toffoli gates required per FMA. We therefore construct our estimate by assembling known components from prior work.

The first stage is the floating-point multiplication $a \cdot b$. In double-double, the mantissas are 53-bit binary fractions, and the multiplication of two unsigned integers is known to require approximately $2n^2$ Toffoli gates in a reversible implementation for n bits [43]. For $n = 53$, this results in a contribution of $2 \cdot 53^2 = 5600$ Toffoli gates. The exponent addition uses a 2-input ripple-carry adder with signed 11-bit inputs. If we follow the example from [43], the adders require approximately $2n$ Toffoli gates, yielding an additional cost of 22 Toffoli gates. The sign logic is implemented by a single XOR gate, which is logically reversible and can be constructed without any Toffoli gates.

The second stage is the floating-point addition of the intermediate product $a \cdot b$ and the operand c. This stage begins with alignment of exponents, which requires detection and shifting of the mantissas. The aligned mantissas span approximately 106 bits, and their addition using a reversible ripple-carry adder again incurs approximately $2n = 106$ Toffoli gates. The normalization of the resulting sum, including leading-zero detection and shifting, requires additional gates. A conservative estimate for this step is 300 Toffoli gates, following [23]. Finally, IEEE-compliant rounding is applied, which contributes an additional 200 Toffoli gates, following estimates for comparable rounding units in classical reversible floating-point logic [44].

Summing the contributions across all stages, we obtain the total estimated Toffoli gate count for a logically reversible FMA:

$$\begin{aligned} GC_{\text{FMA}} = {} & \underbrace{5600}_{\text{mantissa multiplication}} + \underbrace{22}_{\text{exponent addition}} + \underbrace{106}_{\text{mantissa addition}} \\ & + \underbrace{300}_{\text{normalization}} + \underbrace{200}_{\text{rounding}} \\ = {} & 6228 \end{aligned}$$

This estimate assumes that inputs are preserved, and no explicit cleanup of ancilla or intermediate results is applied. The value of 6228 Toffoli

[1] Our implementation uses a smaller reversible log—storing only $(p_k, r_{k+1}, \alpha_k, \beta_{k+1})$ and recomputing Ap_k, reducing the practical footprint to approximately $\mu_{\text{CG}} \approx \frac{4}{3}m$ while preserving injectivity

gates serves as the baseline cost per FMA in our reversible numerical algorithms.

4.3.2. Division

Using the FMA baseline $GC_{\text{FMA}} = 6228$ and reversible addition $GC_{\text{ADD}} = 2n = 106$ based on the reversible ripple-carry adder as discussed in the previous subsection, one iteration costs

$$2 \cdot GC_{\text{FMA}} + GC_{\text{ADD}} \approx 12562.$$

Thus, the total cost of reciprocal refinement is

$$GC_{\text{recip}} \approx 3 \cdot (2 \cdot 6228 + 106) = 37686.$$

A final multiplication $a \cdot y_3$ adds one more GC_{FMA}, yielding the total reversible cost for division:

$$GC_{\text{DIV}} \approx 37686 + 6228 = 43914.$$

Note that, as explained in the previous subsection, exponent logic and normalization are already accounted for within the FMA components.

4.4. Effective cost factors of reversible linear algebra kernels η

We now compute the effective reversible cost η, defined as the number of Toffoli gates per classical FMA operation, for our three key linear algebra algorithms. In all cases, we implement each routine in a logically reversible fashion without Bennett-style logging and unrolling. This means that inputs are preserved, but intermediate registers are not uncomputed, resulting in minimal ancilla overhead. The underlying gate cost model uses the previously derived reversible instruction costs.

4.4.1. GEMM

In the dense case, the product of two $n \times n$ matrices involves n^3 FMA operations. The classical flop is $2n^3$, reflecting one multiplication and one addition per entry of the result matrix. The reversible gate count is therefore

$$\text{Toffoli}_{\text{GEMM}} = n^3 \cdot T_{\text{FMA}} = n^3 \cdot 6228.$$

Dividing by the classical FMA count gives

$$\eta_{\text{GEMM}} = \frac{n^3 \cdot 6228}{n^3} = 6228.$$

4.4.2. LU

In the forward phase of LU factorization (without pivoting), the operation count consists of both FMAs and divisions. Specifically, there are $\frac{2}{3}n^3$ FMAs and $\frac{1}{3}n^3$ divisions. The reversible Toffoli gate cost for this phase is therefore

$$\frac{2}{3}n^3 \cdot 6228 \quad \text{(FMAs)} \quad + \quad \frac{1}{3}n^3 \cdot 43914 \quad \text{(divisions)}.$$

In the backward (uncomputation) phase, only the FMAs are replayed, and divisions are not repeated. This contributes an additional

$$\frac{2}{3}n^3 \cdot 6228 \quad \text{(FMAs only)}.$$

Combining both phases, the total reversible cost is:

$$\text{Toffoli}_{\text{total}} = \left(\frac{2}{3} + \frac{1}{3} + \frac{2}{3}\right) n^3 = \frac{5}{3}n^3,$$

and thus,

$$\text{Toffoli}_{\text{LU}} = \frac{4}{3}n^3 \cdot 6228 + \frac{1}{3}n^3 \cdot 43914 = n^3 \cdot (8304 + 14638) = 22942 \cdot n^3.$$

To express this as a reversible overhead per classical FMA, we normalize by the classical FMA-equivalent operation count, which is $\frac{2}{3}n^3$. This yields:

$$\eta_{\text{LU}} = \frac{22942 \cdot n^3}{\frac{2}{3}n^3} = \frac{22942}{2/3} = 34413.$$

4.4.3. CG

Each CG iteration performs 1 sparse matrix–vector product (costing n^2 FMA primitives), 2 vector dot products (another $2n$ FMAs), 3 axpy updates ($3n$ more FMAs), and 2 scalar divisions. Hence, the forward sweep executes

$$N = n^2 + 5n$$

FMA primitives and $D = 2$ divisions. The reverse (uncompute) sweep replays the same N FMAs but requires only $D - 1 = 1$ division. Hence, the Toffoli-gate counts per iteration are

$$\begin{aligned}
\text{Toffoli}_{\text{CG,F}} &= 6228\,N \;+\; D \times 43914, \\
\text{Toffoli}_{\text{CG,R}} &= 6228\,N \;+\; (D-1) \times 43914, \\
\text{Toffoli}_{\text{CG}} &= \text{Toffoli}_{\text{CG,F}} + \text{Toffoli}_{\text{CG,R}}.
\end{aligned}$$

Because $\text{Toffoli}_{\text{CG}}$ covers both the forward and reverse sweeps, we normalize by the total number of FMA primitives, $2N$, giving

$$\eta_{\text{CG}} = \frac{\text{Toffoli}_{\text{CG}}}{2\,N} = \frac{2 \cdot 6228\,N + 3 \cdot 43914}{2\,N} = 6228 \;+\; \frac{3 \cdot 43914}{2\,N}.$$

In the limit $n \to \infty$, the extra division term vanishes and $\eta_{\text{CG}} \to 6228$, matching the GEMM and LU kernels. Even for moderate n, the correction term $3 \cdot 43914/(2N)$ is negligible, so CG's Toffoli-per-FMA overhead is effectively 6228.

The table below summarizes the effective η values across the three algorithms:

Algorithm	η (Toffoli per FMA)
GEMM	6228
LU	34413
CG (asymptotically)	6228

These values quantify the logical overhead of reversibly simulating numerical routines and form the basis for computing energy consumption and hardware feasibility bounds in later sections.

4.4.4. Future optimizations

Beyond the basic reversible primitives, several hardware-level refinements could further reduce the constant-factor overhead without altering their bijective structure. First, one could design wide reversible registers that natively hold an entire double–double pair in a single 128-bit word, rather than two separate 64-bit words. The theoretical foundation for arbitrarily wide reversible registers and ALUs was laid out in Bennett's early work on logical reversibility [14] and later developed in scalable reversible synthesis frameworks [45].

Second, carry-save or ripple-carry networks built from reversible full-adder cells can eliminate some of the temporary ancilla and correction steps in our TwoSum implementations. Quantum-circuit designers demonstrated how to cascade and pipeline reversible full-adders, effectively a carry-save adder, in Cuccaro et al.'s ripple-carry adder construction [43], and Thapliyal & Srinivas introduced the TSG gate as a compact reversible full-adder building block [20]. Adapting these patterns to classical reversible CMOS would let us remove the second correction addition in every pair-sum.

Finally, in many classical mixed-precision schemes, one stores the high-order mantissa in 64 bits and the low-order rounding residue in 32 bits, then recombines them only when needed [41]. A reversible datapath that carries the 'low' part in a 32-bit integer register alongside the 64-bit high word would cut the gate and energy cost of each low-word multiply and add roughly in half.

Taken together, these directions could offer a reduction in our effective flop-overhead factor η, while preserving the full injectivity and minimal-logging.

5. Numerical validation: Bit-exact reversibility

In this section, we evaluate our reversible implementations of the three core kernels by measuring three unified metrics: forward error, reverse error, and Toffoli gate count. All experiments use double–double arithmetic (unit round-off $u = 2^{-106}$).

The *Forward Error (FE)* quantifies how closely the reversible algorithm matches its classical floating-point analog. For GEMM we collapse each computed double–double pair (h, ℓ) to $h+\ell$ and report the relative Frobenius norm $\|C_{\mathrm{dd}} - A\,B\|_F / \|A\,B\|_F$. For LU we form L and U, apply the pivot permutation P, collapse to standard floats, and measure the infinity-norm $\|P\,A - L\,U\|_\infty$. For CG we compare the final iterate x_m against the exact residual by $\|A\,x_m - b\|_2 / \|b\|_2$.

The *Reverse Error (RE)* tests exact reversibility: after the forward pass, we run the inverse (uncompute) routine and compare the reconstructed state to the original input. In GEMM, the uncompute pass should restore the accumulator to zero, so we report $\max_{i,j} |Z_{ij}|$ in collapsed form. In LU we undo all multiplies, subtracts and row-swaps to recover A exactly, measuring $\|A_{\mathrm{rec}} - A\|_\infty$. In CG, we replay the snapshots backward to recover the initial guess x_0 and report $\|\hat{x}_0 - x_0\|_\infty$.

All code was written in Python 3.11 and executed in Google Colab, where the native IEEE-754 FMA-instruction is exposed via `math.fma`. We bind directly to `math.fma` so that our two-product primitive is truly error-free in one round, ensuring the fast FMA-assisted path is exercised throughout the experiments.

Since our reversible kernels retain every rounding bit and log only the minimal metadata required for injectivity, they inevitably incur greater memory usage (storing double–double pairs and, in CG, per-iteration snapshots) and higher arithmetic cost than their irreversible counterparts. This overhead is an expected trade-off for full bit-reversibility, not a competitive optimization against contemporary classical solvers. Hence, no optimizations or preconditioning methods have been used.

5.1. Reversible GEMM

The data in Table 1 confirm that our fully-injective, minimal-logging reversible GEMM exactly inverts every computational step up to the limits of double–double precision. After the uncompute sweep, the residual accumulator is driven back to machine noise-level ($\lesssim 10^{-30}$), demonstrating that no "garbage" remains beyond the 106-bit significance. Because we perform $2n^3$ FMA-style operations (one forward multiply-add and one reverse subtraction per entry per inner-loop), each of which rounds once in double–double arithmetic, we expect a worst-case residual on the order of $O(n^3 u)$ with $u = 2^{-106}$. Substituting $n \le 256$ into this model predicts reversibility errors below 10^{-28}, in agreement with the measured increase from 3×10^{-33} at $n = 4$ to 4×10^{-30} at $n = 256$. Note that in GEMM with random Gaussian inputs, each multiply–add and subtraction operates on generic floating-point pairs whose low-order bits do not cancel perfectly under a two-round algorithm. As a result, a small $O(n^3 u)$ remainder survives the uncompute pass—but only at the level of bits far below the double-precision unit round-off.

Similarly, the forward Frobenius-norm error remains $\sim 10^{-16}$ and always lies below the classical rounding bound $n\,\varepsilon$ (which grows from 4.44×10^{-16} at $n = 2$ to 5.68×10^{-14} at $n = 256$). Thus, our double–double reversible GEMM matches the numerical stability of a standard GEMM implementation without introducing any additional rounding artifacts.

Table 1
Reversible GEMM results on random Gaussian matrices ($A, B \sim \mathcal{N}(0,1)$).

n	RE	FE	RB ($n\varepsilon$)
2	0.00×10^{0}	3.11×10^{-17}	4.44×10^{-16}
4	3.08×10^{-33}	6.99×10^{-17}	8.88×10^{-16}
8	5.37×10^{-32}	1.30×10^{-16}	1.78×10^{-15}
16	1.36×10^{-31}	1.68×10^{-16}	3.55×10^{-15}
32	4.28×10^{-31}	2.13×10^{-16}	7.11×10^{-15}
64	8.75×10^{-31}	2.82×10^{-16}	1.42×10^{-14}
128	1.55×10^{-30}	3.79×10^{-16}	2.84×10^{-14}
256	3.83×10^{-30}	5.39×10^{-16}	5.68×10^{-14}

Table 2
Reversible LU on Hilbert (H_n) and Wilkinson (W_n) matrices.

n	H_n RE	H_n FE	W_n RE	W_n FE
4	0.00×10^{0}	5.55×10^{-17}	0.00×10^{0}	0.00×10^{0}
8	6.24×10^{-17}	1.67×10^{-16}	0.00×10^{0}	0.00×10^{0}
16	1.20×10^{-16}	3.89×10^{-16}	0.00×10^{0}	0.00×10^{0}
32	1.25×10^{-16}	8.33×10^{-16}	0.00×10^{0}	0.00×10^{0}
64	2.13×10^{-16}	1.72×10^{-15}	0.00×10^{0}	0.00×10^{0}
128	2.84×10^{-16}	3.50×10^{-15}	0.00×10^{0}	0.00×10^{0}
256	4.13×10^{-16}	7.05×10^{-15}	0.00×10^{0}	0.00×10^{0}

5.2. Reversible LU

To demonstrate both numerical accuracy and full reversibility, we tested our reversible LU factorization on two classical *stress-test* matrices:

Hilbert matrices. are defined by

$$(H_n)_{ij} = \frac{1}{i+j+1}, \quad i, j = 0, \ldots, n-1.$$

They are symmetric positive definite but become extremely ill-conditioned as n grows: $\kappa_2(H_n) \sim O\big((1+\sqrt{2})^{4n}/\sqrt{n}\big)$, reaching 10^{10} by $n = 8$ and 10^{20} by $n = 25$. A large condition number amplifies rounding errors in the computed factors by κ_2.

Wilkinson matrices. have entries

$$(W_n)_{ij} = \begin{cases} 1, & i \ge j, \\ -1, & i < j, \end{cases}$$

and only linear condition number $\kappa_2(W_n) = n$ but force worst-case pivot growth under partial pivoting.

Numerical results are reported in Table 2. For Wilkinson matrices, all pivot elements are ± 1, as every division, multiplication, and subtraction is exactly representable in IEEE-754. The reversible Newton-reciprocal incurs zero low-word residue, so both FE(W_n) and RE(W_n) are identically zero.

For Hilbert matrices, despite a huge condition number, the forward error stays below 10^{-14}, The reversible implementation maintains full control over low-order bits explicitly and avoids hidden loss of significance by silently dropped bits. Additionally, pivoting suppresses element growth, and the LU factors remain well scaled. Hence, the realized error remains moderate due to the structured nature of the Hilbert matrix, cancellation effects, and the bit-exact reversibility of the double–double arithmetic, which maintains bijective propagation of rounding information.

5.3. Reversible CG

We now validate the reversible CG method through numerical experiments. The objective is not to compete with classical implementations in speed or memory usage, but to demonstrate a working prototype that (i) preserves convergence, (ii) achieves bit-exact reversibility in ideal arithmetic and bounded reversibility error in floating-point, and (iii) provides a tractable platform for further development of fully reversible numerical solvers.

Unlike Bennett's tree-based scheme, which suffers from quadratic storage and runtime overhead due to repeated checkpointing and recomputation, our reversible CG stores only one micro-snapshot per iteration and avoids exponential growth in memory.

Table 3
Reversible CG on 2D Poisson systems using a stopping criterion for the iterations of 10^{-6}.

n_x	ε	Iterations	FE	RE
4	1	9	5.53×10^{-6}	1.90×10^{-33}
8	1	21	6.43×10^{-7}	2.58×10^{-32}
16	1	24	8.69×10^{-7}	2.23×10^{-32}
32	1	24	1.81×10^{-6}	1.88×10^{-32}
4	10^{-8}	9	1.52×10^{-7}	4.40×10^{-26}
8	10^{-8}	34	3.84×10^{-7}	1.64×10^{-25}
16	10^{-8}	86	5.90×10^{-7}	2.38×10^{-25}
32	10^{-8}	174	2.66×10^{-6}	3.59×10^{-25}

To maintain full reversibility, each iteration records the state tuple (x_k, r_k, p_k) and the scalars α_k, β_{k+1}, enabling exact reconstruction of the full trajectory. Since the memory cost is $\mathcal{O}(nm)$, where n is the number of unknowns and m is the iteration count, we restrict our validation to grids of size $n_x = 32$, or $n = 1024$. For $n_x = 64$, the logged vectors already exceed 1.5 GB in double–double precision, making pure Python implementations impractical without substantial memory optimization. This limit is not a fundamental barrier to reversibility, but a prototype constraint due to in-memory vector storage.

The test problem is the two-dimensional Poisson equation on the unit square, discretized using a standard 5-point finite difference stencil. This yields a sparse symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$, where $n = n_x^2$. Poisson-type problems appear in large-scale simulations of heat flow, electrostatics, image processing, and pressure correction steps in fluid and plasma dynamics. We consider two problem classes: a well-conditioned version using the Laplace operator with a shift $\varepsilon = 1$, yielding condition number $\kappa = \mathcal{O}(10^2)$, and an ill-conditioned variant with $\varepsilon = 10^{-8}$, for which $\kappa = \mathcal{O}(10^8)$.

Table 3 reports the number of iterations until convergence, forward residual error (compared to a reference solve), and reversal error after running the algorithm backward.

The results show that the number of iterations matches classical CG behavior: for well-conditioned systems, convergence is achieved in roughly 24 steps for $n_x = 32$, while ill-conditioned systems require up to 174 iterations. In all cases, forward errors remain small and consistent with the convergence tolerance of 10^{-7}, scaling predictably with grid size and condition number.

Crucially, the reverse error stays bounded even in the ill-conditioned regime and does not grow with the number of iterations. For instance, at $n_x = 32$ and $\varepsilon = 10^{-8}$, the reversal error remains below 10^{-24}, despite nearly 200 iterations and over 400 million reversible gates. Note that the condition number in this case, while large, is significantly smaller than in the case of Wilkinson matrices. This is immediately reflected in the smaller magnitude of the error compared to the LU results from Table 2.

Having established injective floating-point realizations of the core kernels and verified their numerical stability and reversibility, we next relate these constructions to an operational energy model that captures their physical implications.

6. From injectivity conditions to energy scaling

In this section, we introduce the operational energy model that is used in the main results and connects each of its terms directly to the formal proofs in Appendix A. The goal is to show that every quantity appearing in the model corresponds one-to-one with a mathematically defined object.

Note that the solvers enforce logical injectivity at the arithmetic level. Therefore, the memory subsystem need not maintain state history, as it requires only adiabatic, non-erasing updates, so that memory writes do not contribute to the erased capacitance C_{eras} below. Under this assumption, memory participates solely in the adiabatic transport term ε_{ad} and does not enter the electrostatic erasure term in the operational energy model.

6.1. Formal derivation of the operational energy model

Appendix A shows that any deterministic realization of $f : X \to Y$ dissipates only through (i) adiabatic transport loss that vanishes under time-dilation, (ii) an electrostatic term from forcing state to an input–independent value, and (iii) thermodynamic bit-erasure. The closed-cycle identity (Theorem A.10) isolates the electrostatic contribution, while Theorem A.3 and Lemma A.11 ensures the transport term vanishes under slowing. Proposition A.7 and Theorem A.12 identify the erased capacitance C_{eras} as the unique term invariant under slowing, and strictly positive whenever $f|_S$ is not injective. We summarize the resulting operational model here.

Let C_{eras} and C_{keep} denote the erased and preserved capacitances. From Theorem A.12 together with Lemma A.11 and Theorem A.3, the mean dynamic energy satisfies

$$E_{\text{dyn}} = \bar{\beta}\, \tfrac{1}{2} C_{\text{eras}} V^2 + \varepsilon_{\text{ad}}\, \tfrac{1}{2} C_{\text{keep}} V^2 + \alpha k_B T \ln 2, \tag{3}$$

with $\varepsilon_{\text{ad}} \to 0$ as $s \to \infty$. Logical injectivity removes the first term, and adiabatic transport removes the second.

To evaluate energy at the level of floating-point arithmetic, we express ε_{ad} and the node counts in measurable quantities. We parameterize the adiabatic factor as

$$\varepsilon_{\text{ad}} = (1 - \rho_{\text{phys}})\, R, \qquad R = 2\pi/Q,$$

so that the asymptotic limit $\varepsilon_{\text{ad}} \to 0$ corresponds to $\rho_{\text{phys}} \to 1$ and $Q \to \infty$. Here, $\rho_{\text{phys}} \in [0, 1]$ is the macroscopic energy-recovery factor and Q the resonator quality factor.

An FMA activates many state elements per logical operation. Let η_{CMOS} be the effective toggle count for a CMOS FMA and η_T the reversible toggle count. We model the effective toggle count as the affine (interpolating and convex) function

$$\eta(\theta) = \eta_{\text{CMOS}} \left[\theta + (1 - \theta)\eta_{\text{ratio}}\right], \qquad \eta_{\text{ratio}} = \frac{\eta_T}{\eta_{\text{CMOS}}} > 1,$$

where $\theta = C_{\text{eras}}/C_{\text{ref}} \in [0, 1]$. The affine form is the minimal convex choice consistent with the requirement that eliminating erasure does not reduce switching activity.

Define the CMOS reference energy

$$E_{\text{base}} = \eta_{\text{CMOS}}\, \tfrac{1}{2} C_{\text{ref}} V^2, \qquad \kappa = 1 - \theta = \frac{C_{\text{keep}}}{C_{\text{ref}}}.$$

Substituting into (3) yields the normalized model

$$\frac{E_{\text{tot}}}{E_{\text{base}}} = (1 - \rho_{\text{phys}})\, R\, \eta_{\text{ratio}}(\theta)\, \kappa + \frac{\bar{\beta}}{\eta_{\text{CMOS}}}\, \theta + \alpha\, \frac{k_B T \ln 2}{E_{\text{base}}/\eta_{\text{CMOS}}}, \tag{4}$$

which governs the continuous transition from non-injective to injective execution.

Evaluating (4) at $\theta = 1$ gives the irreversible CMOS limit

$$\frac{E_{\text{irr}}}{E_{\text{base}}} = (1 - \rho_{\text{phys}})\, R_{\text{irr}} + \frac{\bar{\beta}}{\eta_{\text{CMOS}}} + \alpha\, \frac{k_B T \ln 2}{E_{\text{base}}/\eta_{\text{CMOS}}}, \tag{5}$$

and at $\theta = 0$ the injective reversible limit

$$\frac{E_{\text{rev}}}{E_{\text{base}}} = (1 - \rho_{\text{phys}})\, R_{\text{rev}}\, \eta_{\text{ratio}}, \qquad \alpha = 0. \tag{6}$$

In (5), the electrostatic and thermodynamic contributions remain finite. In (6) both vanish and the energy tends to zero as $\rho_{\text{phys}} \to 1$. Eqs. (4)–(6) are used for the normalized plots and energy-saving factors in Section 6.3.

6.2. Empirical parameter initialization and regime separation

Eq. (4) contains only measurable physical parameters. Representative values are assigned from CMOS and reversible implementations across three regimes: irreversible, reversible, and transitional.

Irreversible CMOS baseline

The irreversible regime represents conventional CMOS logic operating near its practical efficiency limit. It defines the empirical baseline for normalization of all subsequent quantities. FPnew [46] reports a fully pipelined IEEE 754 FMA datapath with a throughput of one operation per clock cycle (Table IV) and an implementation size of $N_{\mathrm{GE,FMA}} = 1.6 \times 10^5$ NAND2-equivalent gates (Table III). Static CMOS logic exhibits an average switching activity of approximately $\alpha_{\mathrm{sw}} = 0.1$ [47, p. 184]. The corresponding average number of bit-level transitions per FMA-operation is therefore

$$\eta_{\mathrm{CMOS}} = \alpha_{\mathrm{sw}} N_{\mathrm{GE,FMA}} = 0.1 \times 1.6 \times 10^5 = 1.6 \times 10^4 \quad \text{bit-toggles per FMA.} \tag{7}$$

Empirical measurements place the energy cost of a double-precision FMA between 8 and 15 pJ for 7 nm GPUs [48], and around 50 pJ for earlier 40 nm devices [49]. We take $E_{\mathrm{CMOS}} \in \{10^{-11}, 10^{-12}\}$ J/FMA, corresponding to current and projected advanced nodes. At room temperature ($T = 300$ K), these values correspond to approximately 3.5×10^9–3.5×10^{10} times the Landauer limit per FMA. They serve as the empirical reference point for the normalization constant E_{base} introduced earlier.

Reversible and adiabatic baseline

The reversible regime corresponds to logically injective implementations that avoid bit erasure. Each operation can, in principle, approach the adiabatic limit where dissipation vanishes as the recovery factor approaches unity. A reversible FMA performs $\eta_T = 6228$ Toffoli-level toggles per operation for GEMM and CG (see Section 4.4). Because each Toffoli drives multiple bit lines, its switching cost cannot be directly compared with η_{CMOS}. No fixed conversion exists in the literature, since the mapping depends on circuit-level details such as fan-out and ancilla reuse. We therefore introduce a model-level proportionality constant s to express Toffoli activity in NAND2-equivalent bit-toggles for consistent comparison.

We define

s = number of NAND2-equivalent bit-toggles represented by one Toffoli toggle.

The bit-normalized reversible switching activity is therefore $\eta_{\mathrm{eq,rev}} = s\,\eta_T$.

Experimental demonstrations of adiabatic and reversible logic show energy dissipation far below CMOS levels. Yamae et al. [10] report about 1.4 aJ per full adder in reversible quantum flux parametron logic at 5 GHz, while Frank [8] projects 1–10 aJ for adiabatic CMOS operated near thermodynamic limits.

The parameter R denotes the fractional energy loss per resonant transfer and refines the adiabatic loss term of the model. It is linked to the resonator quality factor Q by $R = 2\pi/Q$, where Q is the ratio of stored to dissipated energy per radian of oscillation [50, Sec. 6.4]. Reported values range from $Q \approx 3 \times 10^3$ for adiabatic CMOS oscillators [51] to $Q \sim 10^6$ for superconducting AQFP logic [10], corresponding to $R \in [6 \times 10^{-6}, 6 \times 10^{-3}]$.

Transition domain and transport ratio

The transition regime continuously connects irreversible and reversible operations by interpolating their respective switching activities. To compare both families on the same physical scale, the transport ratio is defined as

$$\eta_{\mathrm{ratio}} = \frac{\eta_{\mathrm{eq,rev}}}{\eta_{\mathrm{CMOS}}} = \frac{s\,\eta_T}{\alpha_{\mathrm{sw}} N_{\mathrm{GE,FMA}}}. \tag{8}$$

Substituting $\eta_T = 6{,}228$, $\alpha_{\mathrm{sw}} = 0.1$, and $N_{\mathrm{GE,FMA}} = 1.6 \times 10^5$ from Eq. (7) yields $\eta_{\mathrm{ratio}} = 0.38925\,s$. The parity point at which reversible and CMOS designs exhibit equal total switching activity follows as

$$s_{\mathrm{eq}} = \frac{\eta_{\mathrm{CMOS}}}{\eta_T} = 2.569. \tag{9}$$

For $s > 2.569$, the reversible implementation exhibits higher aggregate switching activity, whereas for $s < 2.569$ it exhibits lower activity. Eqs. (7)–(9) thus establish an empirical conversion between the CMOS baseline and the reversible toggle count. This relation defines the transport ratio entering the normalized energy expression of Eq. (4).

The minimum irreversibly erased information for a 64-bit FMA is $\alpha \geq 3 \cdot 64 - 64 = 128$ bits. Discarded flags or rounding metadata can only increase this. We therefore take $\alpha = 128$ for the irreversible case and $\alpha = 0$ for the reversible case.

Bit erasure

For a 64-bit IEEE-754 FMA, three 64-bit inputs are consumed and one 64-bit output is produced. The minimum irreversibly erased information for a 64-bit FMA is $\alpha \geq 3 \cdot 64 - 64 = 128$ bits. Discarded flags or rounding metadata can only increase this. We therefore take $\alpha = 128$ for the irreversible case and $\alpha = 0$ for the reversible case. This represents the fundamental theoretical erasure bound (Landauer limit). Capacitive and recovery losses dominate today, but as adiabatic techniques improve the erasure contribution becomes increasingly relevant, consistent with the observation that logically irreversible updates remain the ultimate source of dissipation [17].

6.3. Energy savings across kernels and baselines

We evaluate the normalized model of Eq. (4) by sweeping the recovery parameter ρ_{phys} and the injectivity parameter θ. We take $\bar{\beta} = 1$ as a conservative worst case. Workload-dependent $\bar{\beta} < 1$ would simply scale the erase term linearly without changing the qualitative conclusions. The CMOS toggle count is fixed at $\eta_{\mathrm{CMOS}} = 1.6 \times 10^4$ (FPnew, Section 6.2). α is set to 128 bits per IEEE 754 FMA. $\rho_{\mathrm{phys}} = 0$ and $R = 1$ correspond to conventional CMOS, $\rho_{\mathrm{phys}} \approx 0.5$–$0.9$ and $R \approx 0.1$–0.5 represent demonstrated adiabatic-CMOS prototypes, and $R \leq 0.01$ proxy superconducting devices. Performance is reported as the normalized energy and the corresponding savings factor

$$S(\rho_{\mathrm{phys}}) = \frac{E_{\mathrm{CMOS}}}{E(\rho_{\mathrm{phys}}, \theta)},$$

with $S > 1$ indicating improvement. After normalization, the energy expression is dimensionless and depends only on the erased-capacitance term and the recovery-dependent transport term. The absolute CMOS baseline cancels, except through the explicit Landauer coefficient.

For the GEMM/CG case (Fig. 3), when $R = 0.5$, the fully injective line ($\theta = 0$) and the partial-injective regime ($\theta \in [0.2, 0.8]$) lie above the CMOS reference for all $\rho_{\mathrm{phys}} \in [0, 1]$, yielding net savings across the entire recovery range. For $R = 1.0$, the reversible curve crosses $S = 1$ near $\rho_{\mathrm{phys}} \approx 0.75$, and for $R = 1.5$, break-even shifts to $\rho_{\mathrm{phys}} \approx 0.85$. In both cases, the partially injective region crosses later and approaches the irreversible baseline, reflecting the nonzero erased-capacitance term. At high recovery ($\rho_{\mathrm{phys}} \gtrsim 0.9$), fully injective execution yields order-of-magnitude reductions, while partial injectivity offers only modest improvements.

The right panel considers more aggressive adiabatic devices. In this regime, all fully injective curves lie above the CMOS line for the entire ρ_{phys} interval. In this regime, the energy savings exceed a factor of ten. Further reductions in R would push the savings into the 10^2–10^3 range on more advanced hardware. By contrast, the partial-injectivity band remains close to the irreversible limit and does not improve with smaller R, making the energy floor almost horizontal. This separation between the solid curves and the shaded band is the electrostatic wall: without $\theta \to 0$, the erased-capacitance term prevents further scaling even when transport losses are nearly eliminated.

Fig. 4 repeats the analysis for the LU kernel, which has a higher reversible toggle factor ($\eta_{\mathrm{ratio}} \approx 20.17$). The larger switching activity overhead raises the reversible branch and shifts the break-even point to a higher ρ_{phys}. In the near-term panel (left), reversible execution exceeds the CMOS baseline only for sufficiently high recovery: for

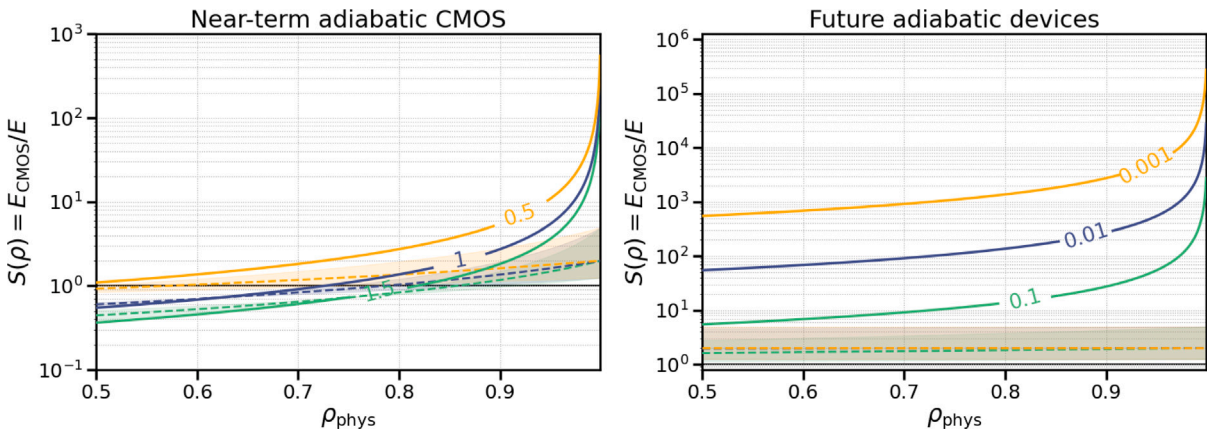


Fig. 3. Normalized energy savings factor $S(\rho_{phys})$. The reversible toggle count at $\eta_T = 6228$ for GEMM/CG, yielding a ratio $\eta_{ratio} = 3.65$. Solid curves correspond to fully injective execution ($\theta = 0$). Dashed curves represent the irreversible limit ($\theta = 1$). Shaded bands indicate the transition regime of partial injectivity ($\theta \in [0.2, 0.8]$), where erased capacitance remains and only partial adiabatic benefit is available. Each curve sweeps the resonant-loss parameter R (smaller R corresponds to higher-Q operation and stronger energy recovery).

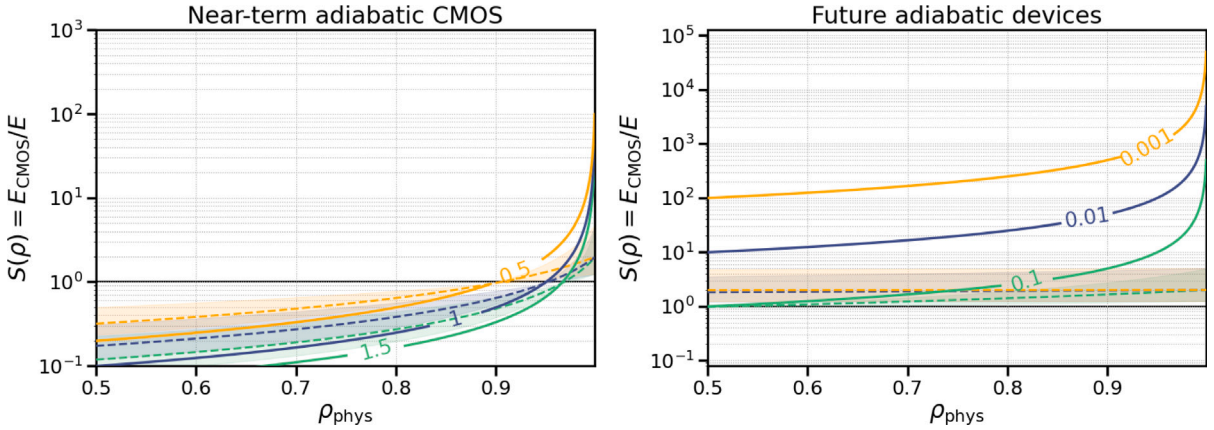


Fig. 4. Normalized energy savings factor $S(\rho_{phys})$. The reversible toggle count at $\eta_T = 34413$ for LU, yielding a ratio $\eta_{ratio} = 20.17$. Solid curves correspond to fully injective execution ($\theta = 0$). Dashed curves represent the irreversible limit ($\theta = 1$). Shaded bands indicate the transition regime of partial injectivity ($\theta \in [0.2, 0.8]$), where erased capacitance remains and only partial adiabatic benefit is available. Each curve sweeps the resonant-loss parameter R (smaller R corresponds to higher-Q operation and stronger energy recovery).

$R = 0.5$, savings emerge at $\rho_{phys} \approx 0.9$, while for $R = 1 - 1.5$ the curves remain below unity across the range shown. The electrostatic wall remains visible: the irreversible branch ($\theta = 1$) stays pinned near $S \approx 1$ regardless of ρ_{phys}, while fully-injective operation ($\theta = 0$) eventually escapes this limit once recovery is high enough.

In the future-device panel (right), the qualitative structure persists, but the attainable gains increase. For $R = 0.1$, the reversible branch already reaches break-even across the full recovery range, and for $R = 0.01$ and $R = 0.001$ the savings exceed one order of magnitude as $\rho_{phys} \to 1$. Although the magnitude of savings is lower than for GEMM, the key qualitative behavior remains: injective execution can surpass the electrostatic wall, whereas irreversible execution cannot, even under aggressive energy recovery.

6.4. Real-world potential and savings

While most research on reversible computing focuses on theoretical limits or circuit primitives, its practical implications for high-performance workloads remain less explored. This section evaluates the full-system energy impact of reversible linear algebra kernels when scaled to representative large-scale scientific and machine learning workloads. To contextualize the scale of FMA use in real-world scientific computing, we consider four examples: large-scale generative AI training, dense spectral climate emulation, sparse iterative Navier–Stokes (turbulence) solvers, and full exascale-class throughput. These domains typify the primary arithmetic structures, such as dense GEMMs and sparse matrix–vector (spMV), that dominate energy and information transfer in high-performance workloads. In all cases, the true number of FMA instructions is not reported in the cited references. However, based on the standard arithmetic model in which one FMA operation corresponds to two floating-point operations (FLOPs), we estimate the FMA count from documented FLOP figures. All assumptions are stated explicitly, and all estimates are interpreted as lower bounds. These are simplified estimates that omit memory effects, communication, and hardware-specific optimizations. This abstraction is deliberate, as the focus of this work is the algorithmic-level FMA demand in the absence of architectural tuning.

Generative AI training (chatgpt-4 scale)

Large transformer-based language models such as ChatGPT-4 require substantial dense linear algebra operations during training, primarily in the form of batched GEMMs. Estimates place the total training compute of GPT-4 at approximately 2.5×10^{25} FLOPs [3]. Since the dominant kernel is GEMM, which consists almost entirely of FMAs, we use the estimate:

$$\frac{2.5 \times 10^{25}}{2} = 1.25 \times 10^{25} \text{ FMAs.}$$

Climate emulator at exascale

[4] report a sustained performance of 0.976 EFLOP/s on the Frontier supercomputer using a climate emulation code based on spherical harmonic transforms and dense matrix operations, implemented in BLAS3. The run used 36,100 AMD MI250X GPUs. While the total wall-clock duration of the run is not disclosed, we model the total FMA count by assuming the measured performance of 0.976 EFLOP/s is sustained for 10 s, yielding:

$$\frac{9.76 \times 10^{17} \cdot 10}{2} = 4.88 \times 10^{18} \text{ FMAs.}$$

While Cholesky decomposition, which is similar to the LU factorization, is mentioned, the runtime is dominated by GEMM operations, as stated by the authors.

Sparse iterative solver for Navier–Stokes

In large-scale PDE simulations, particularly incompressible Navier–Stokes problems, sparse matrix–vector products dominate the arithmetic. Iterative methods such as CG, BiCGStab, or GMRES are commonly employed, depending on the symmetry and definiteness of the system matrix. We consider a case with 10^{13} degrees of freedom, 30 nonzeros per row, 50 SpMVs per time step, and 10^3 time steps:

$$10^{13} \cdot 30 \cdot 50 \cdot 10^4 = 1.5 \times 10^{20} \text{ FMAs.}$$

This includes only SpMV costs, excluding preconditioners and nonlinear terms.

General exascale computing

Exascale-class systems sustain at least 10^{18} FLOP/s across the full machine. Assuming a 10-second sustained run with fully FMA-dominated kernels, we obtain a workload of

$$\frac{10^{18} \cdot 10}{2} = 10^{19} \text{ FMAs.}$$

Fig. 5 illustrates that across all scientific computing workloads, the reversible curves (non-normalized model) drop well below CMOS baselines. Near-term adiabatic regimes ($R = 10^{-2}$, $\rho = 0.75$) yield about one order of magnitude reduction relative to a 1 pJ/FMA CMOS reference, and aggressive recovery ($R = 10^{-3}$, $\rho = 0.95$) yields two to three orders. The trend is uniform: decreasing R and increasing ρ suppress transport dissipation and move execution towards the adiabatic limit, even under the highest measured reversible switching overhead ($\eta_{\text{ratio}} = 20.17$, LU case).

To make the scale concrete, the Dutch data-center sector consumed 9.9×10^{15} J in 2019 [52], and U.S. data centers consumed $\sim 6.3 \times 10^{17}$ J in 2023 [53]. A *single* 10^{25}-FMA generative-AI workload at 1 pJ/FMA consumes $\sim 10^{13}$ J. That is roughly 0.1% of Dutch and 0.002% of U.S. annual data-center energy for one training run. Under a fully injective reversible execution path with high adiabatic recovery in a CMOS-realistic regime ($R = 10^{-2}$, $\rho = 0.95$), the same workload requires $\sim 1.2 \times 10^{12}$ J, corresponding to an $\approx 8.3\times$ reduction in arithmetic energy (i.e., $\approx 88\%$ lower than the irreversible case).

Achieving such reductions will require continued advances in device design, clocking networks, and system integration. However, pursuing reversible and adiabatic logic is not speculative in the way that fully fault-tolerant quantum computing is. Whereas scalable quantum systems depend on breakthroughs in coherent qubit technologies, quantum error correction, and cryogenic control, the mechanisms underlying reversible classical logic, as developed in the work of Frank and others, extend established CMOS-fabrication and device physics and do not require quantum-style coherence or error-correction overheads. Reversible computing, therefore, constitutes a technologically adjacent path towards large energy reductions in arithmetic-dominated workloads, complementing quantum efforts that target asymptotic speedups in specific problem classes.

7. Limitations and discussion

While this work focuses on algorithmic injectivity and arithmetic-level energy recovery, it does not model full system costs. The analysis isolates switching and transport within the arithmetic datapath and assumes only that the memory cells used by the solver support adiabatic, non-erasing updates so that memory writes contribute no erased capacitance C_{eras}. Beyond this local assumption on the write mechanism, the model excludes memory-hierarchy, interconnect, and I/O energy, consistent with prior reversible and adiabatic-CMOS studies [7,8,18]. Recovery is represented through two parameters (R, ρ_{phys}) that abstract device- and clock-level behavior, where finite-Q effects, clock distribution, leakage, and bias networks are not included. The normalization $\hat{\beta} = 1$ in Appendix A absorbs process-specific constants into the recovery parameters and leaves hardware calibration open. Likewise, the interpolation $\theta = C_{\text{eras}}/C_{\text{ref}}$ between irreversible and injective regimes assumes a convex dependence motivated by monotonic state activation in reversible implementations, although other forms may arise in specific technologies. These choices restrict attention to the arithmetic layer and an abstract recovery interface, providing a controlled setting in which to examine the energetic implications of preserving floating-point information. In addition, the study does not consider mixed-precision arithmetic, preconditioning, or more advanced numerical methods, as the focus is on core dense kernels and canonical iterative structure.

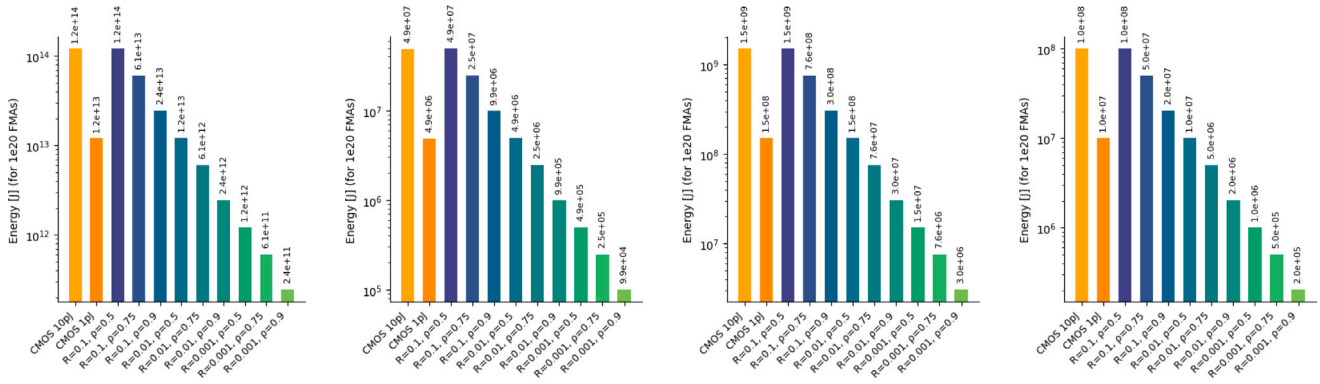
8. Outlook and future work

From the standpoint of numerical analysis and scientific computing, reversible floating-point kernels open a new paradigm in algorithm design. Rather than focusing solely on devices or gates, this perspective emphasizes how reversible logic affects the structure and cost of full numerical methods. In particular, it offers simulation engineers and applied mathematicians reconstructable access to the full computational history that conventional codes overwrite or round away, enabling algorithmic capabilities that are inaccessible in irreversible settings. These capabilities invite several lines of future work.

First, closer collaboration between applied mathematicians, computer architects, and device engineers will be essential.

Second, the possibility of replaying past iterates at negligible energy cost revives classes of preconditioners that were once dismissed. Preconditioners transform a linear system into an equivalent one with a significantly smaller condition number. This accelerates the convergence of Krylov methods, such as CG. They are a critical component in large-scale simulations such as plasma fusion modeling, climate prediction, structural mechanics, and reservoir simulation, where each matrix–vector product is costly. The purpose of preconditioning is to improve the spectrum of the coefficient matrix, particularly in ill-conditioned problems, so that Krylov solvers converge in fewer iterations with fewer floating-point operations. In difficult systems, preconditioning can determine whether convergence is possible at all.

Under reversible execution, any Krylov vector generated during the solve can be reconstructed through iterative uncomputation with only adiabatic transport cost and without storing the full subspace explicitly. This makes polynomial, spectral, and deflation-based preconditioners practical, since any required basis vector can be recovered without dissipative overhead. Because such preconditioners can, in principle, reduce the number of iterations needed for convergence, they can shorten wall-clock time while retaining near-zero energy dissipation. In classical irreversible solvers, the same capability requires either explicit storage or recomputation of discarded Krylov vectors, both of which incur substantial energy due to irreversible arithmetic and memory traffic. Consequently, these methods become less efficient in future energy-constrained workloads, where memory movement dominates the power budget.



(a) GenAI Training (b) Climate Emulator (c) Navier–Stokes Solver (d) Exascale

Fig. 5. Energy consumption for four representative workloads. The two leftmost bars in each panel correspond to classical irreversible execution at 10 pJ and 1 pJ per FMA. The remaining bars show reversible execution under the non-normalized physical model using the measured worst-case LU switching overhead $\eta_{\text{ratio}} = 20.17$. We sweep resonant-loss parameters $R \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ and recovery factors $\rho \in \{0.50, 0.75, 0.95\}$, spanning current room-temperature CMOS through aggressive future superconducting adiabatic operation. All panels share a logarithmic y–axis.

Third, extending the reversible approach from dense arithmetic to sparse matrix–vector products, multigrid, and fast transforms is essential for progressing from isolated kernel demonstrations to full-scale scientific applications. These extensions will also require addressing systems-level questions such as reversible parallelism and clocked coordination.

9. Conclusion

To our knowledge, this work provides the first reversible formulations of the core kernels underpinning modern scientific computing and large-scale learning workloads, such as dense matrix multiplication, LU factorization, and iterative methods, that preserve IEEE 754 semantics and use only minimal auxiliary state. By retaining all rounding residues through error-free pair arithmetic and storing only pivots or scalar coefficients, the forward and reverse sweeps form a bit-exact bijection with memory overhead limited to a constant factor of the original problem size.

An analytically grounded energy model informed by recent adiabatic CMOS measurements suggests that such reversible realizations can reduce arithmetic-level energy by several orders of magnitude under moderate to high recovery. The analysis further shows that only fully injective execution eliminates residual electrostatic cost, establishing an energy floor for partial reversibility that is consistent with adiabatic-logic studies.

These results provide an algorithmic basis for evaluating reversible and adiabatic hardware. Practical deployment will require advances in adiabatic memory interfaces and data transport, interconnect, and clocking, together with a clearer understanding of reversible parallelism and systems-level coordination. As simulation and learning workloads grow, architectures that minimize dissipation merit close study, and sustained progress will require coordinated advances in numerical algorithms, hardware and circuit design, device physics, and systems engineering.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Vandana Dwarka reports was provided by Dutch Research Council. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The author gratefully acknowledges Dr. Michael Frank (Vaire Computing) for valuable guidance and discussions that helped shape the direction of this research. The author also thanks Dr. Jayson Lynch (Massachusetts Institute of Technology (MIT)) for input during the development of this work. We thank the reviewers for their thoughtful and detailed feedback, which materially strengthened the clarity and rigor of this work. This research was supported by the Dutch Research Council (NWO) under the Veni Grant 2024.

Appendix A. Mathematical foundations and physical bounds

This appendix records the mathematical and physical foundations used in the paper, cites known results without rederiving them, and provides full proofs of the new results. In particular, the closed-cycle identities and electrostatic wall are established here. These results justify the injectivity requirements enforced by the reversible floating-point kernels in the main text. In this analysis, we do not explicitly track the Landauer term, as its magnitude is negligible relative to capacitive and recovery-related contributions at contemporary scales and does not affect the asymptotic bounds established here.

A.1. Passive reversible circuit model and energy axioms

We assume a finite-state lumped dynamical system describing a circuit with capacitive, inductive, and resistive energy storage and dissipation, governed by Kirchhoff conservation and passive element laws. Computation is encoded in capacitor states, where passive time-reversible evolution of the system realizes reversible computation.

Let $C_i > 0$ be fixed capacitances, $L_j > 0$ inductances, and $R_k \geq 0$ resistances. Voltages and currents satisfy Kirchhoff conservation and passive constitutive relations. Logical values are represented by two distinct voltage levels $V_{\text{ref}} \neq V$. For each logical register i and each supported input $x \in S$, the initial and terminal logical states satisfy

$$v_i(0; x),\ v_i(T; x) \in \{V_{\text{ref}}, V\}.$$

The level V_{ref} is the designated reset potential, as any other two-level encoding is equivalent up to affine rescaling and does not affect the invariants introduced below.

Capacitor and inductor energies are

$$U_C = \sum_i \frac{1}{2} C_i v_i^2, \qquad U_L = \sum_j \frac{1}{2} L_j i_j^2,$$

with total resistive dissipation $Q_R = \int_0^T \sum_k R_k i_k^2 \, dt \geq 0.$

Axiom A.1 (*Energy Balance*). For every operation on $[0,T]$ with work input W,

$$W = \Delta U_C + \Delta U_L + Q_R.$$

Axiom A.2 (*Closed Cycle*). If sources and inductor states return to their initial values and clocks are periodic, then $W = 0$ and $Q_R = -\Delta U_C$.

These identities follow from standard passive network theory and energy balance in lumped circuits [54,55] and from the first-law formulation for cyclic physical processes [56].

A.2. Logical reversibility and thermodynamic lower bounds

Theorem A.1 (*Landauer [13]*). *Let X be a finite random variable and f a deterministic map. Any physical implementation of f in thermal equilibrium at temperature T satisfies*

$$\mathbb{E}[Q] \geq k_B T \ln 2 \cdot (H(X) - H(f(X))).$$

Proof. See [13]. □

Theorem A.2 (*Bennett 1973 [14]*). *A deterministic computation can, in principle, be performed with asymptotically zero thermodynamic dissipation if and only if every computational step is logically reversible, i.e. it is one-to-one on the computational state space.*

Proof. See [14]. □

Equivalently, for a function f acting on a supported input set S, logical reversibility means

$$x_1 \neq x_2 \in S \quad \Rightarrow \quad f(x_1) \neq f(x_2), \qquad \text{i.e. } f|_S \text{ is injective.}$$

Theorem A.3 (*Adiabatic Time–Dilation Limit*). *Consider a lumped passive switching circuit with nonzero resistances, driven periodically by one or more clock waveforms. For each dilation factor $s > 0$, let the clocks be time-stretched by $u_s(t) = u(t/s)$ over period sT, and let $Q_R(s)$ be the total Joule heat dissipated over $[0, sT]$. Assume:*

- *(A1) passive element laws, time-reversal compatibility in the quasistatic limit.*
- *(A2) cyclic operation: sources and inductive states return to initial values.*
- *(A3) $\{u_s\}_{s>0}$ has uniformly bounded amplitude and total variation.*
- *(A4) non-Joule nonidealities are $o(1)$ under time dilation.*

Then there exists $\varepsilon(s) \to 0$ as $s \to \infty$ such that $Q_R(s) \leq \varepsilon(s)$.

Proof. The cited works construct passive adiabatic switching protocols in which slowing the clock waveforms makes the per-cycle dissipation arbitrarily small. See [7] for adiabatic CMOS demonstrations, [9,15–17] for the general reversible/adiabatic switching framework, and [10] for superconducting adiabatic logic. These references establish the qualitative limit $Q_R(s) \to 0$ as $s \to \infty$. They do not provide a general quantitative rate. □

A.3. Logical classes and erased-capacitance invariant

We now formalize how logical many-to-one structure in f manifests as forced state identification in any physical realization. Throughout, let $f : X \to Y$ be the realized deterministic map over $[0,T]$ and $S \subseteq X$ the supported input set: define $x_1 \sim x_2$ iff $f(x_1) = f(x_2)$, and let $\mathcal{C}$ denote the set of equivalence classes in S.

Definition A.4 (*Erased Capacitance*). For each $C \in \mathcal{C}$ with $|C| \geq 2$ define

$$\mathcal{E}_C = \{\, i \in \{1,\dots,n\} : \ v_i(T;x) = V_{\text{ref}} \text{ for all } x \in C \,\},$$

and set $\mathcal{E}_C := \varnothing$ if $|C| = 1$. Define

$$\mathcal{E} = \bigcup_{C \in \mathcal{C}} \mathcal{E}_C, \qquad \mathcal{K} = \{1,\dots,n\} \setminus \mathcal{E}, \qquad C_{\text{eras}} = \sum_{i \in \mathcal{E}} C_i, \qquad C_{\text{keep}} = \sum_{i \in \mathcal{K}} C_i.$$

Lemma A.5 (*Partition*).

$$\mathcal{E} \cap \mathcal{K} = \varnothing, \qquad \mathcal{E} \cup \mathcal{K} = \{1,\dots,n\}, \qquad C_{\text{eras}} + C_{\text{keep}} = \sum_{i=1}^{n} C_i.$$

Proof. Immediate from the definitions and finite additivity. □

Definition A.6 (*Pair erased set*). For $x, y \in S$ with $f(x) = f(y)$ define

$$\mathcal{E}(x,y) = \{\, i : \ v_i(T;x) = v_i(T;y) = V_{\text{ref}} \,\}.$$

Proposition A.7 (*Logical merge ⇒ erased capacitance*). *(i) If $f|_S$ is injective, then $C_{\text{eras}} = 0$.*

(ii) If $f|_S$ is not injective and the realization assigns $v_i(T;x) = V_{\text{ref}}$ for all $x \in C$ on a set of nodes of positive total capacitance for some class C with $|C| \geq 2$, then $C_{\text{eras}} > 0$ for that realization.

Proof. (i) If $f|_S$ is injective, every C meeting S has $|C| = 1$, so $\mathcal{E}_C = \varnothing$ by Definition A.4. Hence $\mathcal{E} = \varnothing$ and $C_{\text{eras}} = 0$.

(ii) Let $C \in \mathcal{C}$ satisfy $|C| \geq 2$ and suppose there exists a set $A \subseteq \{1,\dots,n\}$ with $\sum_{i \in A} C_i > 0$ such that $v_i(T;x) = V_{\text{ref}}$ for all $i \in A$ and all $x \in C$. Then $A \subseteq \mathcal{E}_C \subseteq \mathcal{E}$, hence

$$C_{\text{eras}} = \sum_{i \in \mathcal{E}} C_i \ \geq \ \sum_{i \in A} C_i \ > \ 0. \quad \square$$

C_{eras} measures how much physical state the circuit is forced to forget when distinct supported inputs map to the same output. If no such merging occurs, then $C_{\text{eras}} = 0$. If merging does occur, then $C_{\text{eras}} > 0$ and this erased capacitance necessarily contributes a fixed electrostatic energy cost that cannot be eliminated by slowing the computation. This establishes the electrostatic wall that will appear explicitly in Appendix A.6.

A.4. Closed cycle dissipation bound

We derive a closed-cycle identity showing that whenever two supported inputs produce the same output, the physical realization incurs a fixed electrostatic dissipation on the erased subspace. Hence, for $x, y \in S$ with $f(x) = f(y)$, we define

$$\mathcal{E}(x,y) := \{\, i : \ v_i(T;x) = v_i(T;y) = V_{\text{ref}} \,\}, \qquad \mathcal{K}(x,y) := \{1,\dots,n\} \setminus \mathcal{E}(x,y).$$

Lemma A.8 (*Adiabatic kept-node interpolation [7,15,16]*). *Let $x, y \in S$ with $f(x) = f(y)$. There exists an ideal quasi-static kept-node interpolation that acts only on $\mathcal{K}(x,y)$ while clamping every $i \in \mathcal{E}(x,y)$ at V_{ref}, and*

$$Q_R^{\text{B}} = 0, \qquad \Delta U_{C,\mathcal{E}}^{\text{B}} = 0, \qquad \Delta U_{C,\mathcal{K}}^{\text{B}} = U_{C,\mathcal{K}}(T;y) - U_{C,\mathcal{K}}(T;x).$$

Here $U_{C,\mathcal{K}}(t) := \sum_{i \in \mathcal{K}(x,y)} \frac{1}{2} C_i v_i(t)^2$ denotes the capacitor energy on the kept nodes.

Proof. As in Theorem A.3, we use time-dilated clock waveforms that admit a quasi-static limit, so $Q_R \to 0$ in principle. The $s \to \infty$ limit itself is invoked only later in Appendix A.6. Define the kept-node path

$$\gamma : [0,1] \to \mathbb{R}^{|\mathcal{K}(x,y)|}, \qquad \gamma(0) = (v_i(T;x))_{i \in \mathcal{K}(x,y)}, \quad \gamma(1) = (v_i(T;y))_{i \in \mathcal{K}(x,y)}.$$

Hold all erased nodes fixed: $v_i(t) \equiv V_{\text{ref}}$ for $i \in \mathcal{E}(x,y)$. Standard adiabatic switching constructions allow quasi-static voltage transfer on selected nodes while others are held fixed, with arbitrarily small dissipation, see Athas–Svensson [7] and Frank [15,16]. Applying these constructions to γ yields a quasi-static kept-node interpolation with

$Q_R = 0$. Since each $i \in \mathcal{E}(x,y)$ is clamped, $\Delta U_{C,\mathcal{E}} = 0$. On $\mathcal{K}(x,y)$ the capacitor energy depends only on the endpoints, hence

$$\Delta U_{C,\mathcal{K}} = \sum_{i\in\mathcal{K}(x,y)} \tfrac{1}{2}C_i\big(v_i(T;y)^2 - v_i(T;x)^2\big) = U_{C,\mathcal{K}}(T;y) - U_{C,\mathcal{K}}(T;x).$$

This proves the three identities. □

This lemma formalizes that only the non-erased part of the state can move reversibly. The erased nodes are fixed at the reset level and cannot give energy back. Reversible evolution happens on the kept nodes, given that erased nodes act as a one-way energy sink. To formulate the closed-cycle bound, we define three stages to compare the energetics of two inputs.

Definition A.9 (*Forward–Adiabatic–Reverse Cycle*)**.** Let $x, y \in S$ with $f(x) = f(y)$.

Stage A(x): forward evolution on x over $[0,T]$, yielding terminal voltages $(v_i(T;x))_{i=1}^n$.

Stage B: quasi-static kept-node interpolation (Lemma A.8), holding $v_i(t) = V_{\mathrm{ref}}$ for all $i \in \mathcal{E}(x,y)$ and mapping $(v_i(T;x))_{i\in\mathcal{K}(x,y)} \mapsto (v_i(T;y))_{i\in\mathcal{K}(x,y)}$ with $Q_R^{\mathrm{B}} = 0$.

Stage C(y): exact time-reversal of Stage A on input y, returning the system to the initial state for y.

Hence $\mathrm{A}(x) \to \mathrm{B} \to \mathrm{C}(y)$ forms a closed thermodynamic cycle restoring sources and inductive states. By Axioms A.1–A.2 and Lemma A.8 (which gives $Q_R^{\mathrm{B}} = 0$), the total dissipation of the cycle reduces to a capacitor-energy balance across the three stages. The next result formalizes this closed-cycle identity and its erased/kept decomposition.

Theorem A.10 (*Closed-cycle Identity and Decomposition*)**.** *Let $x, y \in S$ satisfy $f(x) = f(y)$. Consider Stage A (forward on x over $[0,T]$), Stage* B *as in Lemma A.8, and Stage C (exact time reverse of Stage A on y). Then*

$$Q_R(x) + Q_R(y) = \sum_{i=1}^{n} \frac{1}{2} C_i\Big(v_i(0;x)^2 - v_i(0;y)^2\Big) =: \Delta U_{\mathrm{tot}}(x,y), \tag{10}$$

and hence

$$\begin{aligned} Q_R(x) + Q_R(y) = &\underbrace{\sum_{i\in\mathcal{E}(x,y)} \frac{1}{2} C_i\Big(v_i(0;x)^2 - v_i(0;y)^2\Big)}_{=:\ \Delta U_{\mathrm{erased}}(x,y)} \\ &+ \underbrace{\sum_{i\in\mathcal{K}(x,y)} \frac{1}{2} C_i\Big(v_i(0;x)^2 - v_i(0;y)^2\Big)}_{=:\ \Delta U_{\mathrm{keep}}(x,y)}. \end{aligned} \tag{11}$$

In particular, if $v_i(0;x) = v_i(0;y)$ for all $i \in \mathcal{K}(x,y)$, then

$$Q_R(x) + Q_R(y) = \Delta U_{\mathrm{erased}}(x,y). \tag{12}$$

Moreover, since $Q_R(x), Q_R(y) \geq 0$,

$$\max\{Q_R(x), Q_R(y)\} \ \geq\ \tfrac{1}{2}\big|\Delta U_{\mathrm{tot}}(x,y)\big|. \tag{13}$$

Proof. Apply Axiom A.1 to each stage:

$$W^{\mathrm{A}} = \Delta U_C^{\mathrm{A}} + \Delta U_L^{\mathrm{A}} + Q_R(x), \quad W^{\mathrm{B}} = \Delta U_C^{\mathrm{B}} + \Delta U_L^{\mathrm{B}} + Q_R^{\mathrm{B}},$$
$$W^{\mathrm{C}} = \Delta U_C^{\mathrm{C}} + \Delta U_L^{\mathrm{C}} + Q_R(y).$$

The composite uses periodic clocks and restores sources and inductor states, so $W^{\mathrm{A}} + W^{\mathrm{B}} + W^{\mathrm{C}} = 0$ and $\Delta U_L^{\mathrm{A}} + \Delta U_L^{\mathrm{B}} + \Delta U_L^{\mathrm{C}} = 0$. By Lemma A.8, $Q_R^{\mathrm{B}} = 0$. Summing the three balances gives

$$Q_R(x) + Q_R(y) = -(\Delta U_C^{\mathrm{A}} + \Delta U_C^{\mathrm{B}} + \Delta U_C^{\mathrm{C}}). \tag{14}$$

Compute the right-hand side nodewise. For any node i,

$$\Delta U_{C,i}^{\mathrm{A}} = \tfrac{1}{2}C_i\big(v_i(T;x)^2 - v_i(0;x)^2\big), \quad \Delta U_{C,i}^{\mathrm{B}} = \tfrac{1}{2}C_i\big(v_i(T;y)^2 - v_i(T;x)^2\big),$$
$$\Delta U_{C,i}^{\mathrm{C}} = \tfrac{1}{2}C_i\big(v_i(0;y)^2 - v_i(T;y)^2\big).$$

Hence $\Delta U_{C,i}^{\mathrm{A}} + \Delta U_{C,i}^{\mathrm{B}} + \Delta U_{C,i}^{\mathrm{C}} = \tfrac{1}{2}C_i\big(v_i(0;y)^2 - v_i(0;x)^2\big)$.

Define, for each $i \in \{1,\dots,n\}$,

$$T_i \ :=\ \tfrac{1}{2}\,C_i\big(v_i(0;x)^2 - v_i(0;y)^2\big).$$

Then from (14) and the nodewise computation preceding it,

$$Q_R(x) + Q_R(y) \ =\ -\sum_{i=1}^{n}\big(\Delta U_{C,i}^{\mathrm{A}} + \Delta U_{C,i}^{\mathrm{B}} + \Delta U_{C,i}^{\mathrm{C}}\big) \ =\ \sum_{i=1}^{n} T_i \ =:\ \Delta U_{\mathrm{tot}}(x,y). \tag{15}$$

Since $Q_R(x) \geq 0$ and $Q_R(y) \geq 0$, (15) implies $\Delta U_{\mathrm{tot}}(x,y) \geq 0$. Partition the index set by $\mathcal{E}(x,y)$ and $\mathcal{K}(x,y)$:

$$\sum_{i=1}^{n} T_i \ =\ \sum_{i\in\mathcal{E}(x,y)} T_i + \sum_{i\in\mathcal{K}(x,y)} T_i \ =\ \Delta U_{\mathrm{erased}}(x,y) + \Delta U_{\mathrm{keep}}(x,y), \tag{16}$$

which is exactly (11). If $v_i(0;x) = v_i(0;y)$ for every $i \in \mathcal{K}(x,y)$, then $T_i = 0$ for all $i \in \mathcal{K}(x,y)$ and hence $\Delta U_{\mathrm{keep}}(x,y) = 0$. Together with (16) this yields

$$Q_R(x) + Q_R(y) \ =\ \Delta U_{\mathrm{erased}}(x,y), \tag{17}$$

which is (12). Finally, from (15) and $Q_R(x), Q_R(y) \geq 0$,

$$\max\{Q_R(x), Q_R(y)\} \ \geq\ \tfrac{1}{2}\big(Q_R(x) + Q_R(y)\big) \ =\ \tfrac{1}{2}\,\Delta U_{\mathrm{tot}}(x,y), \tag{18}$$

which is (13). □

A.5. Adiabatic charge-transport bound for kept nodes

We record a standard adiabatic scaling fact for a single RC branch in the notation of this appendix, see [7–10,15,16]. Note that η used here is different from the main energy model, as the two parameters play distinct roles.

Lemma A.11 (*Adiabatic Charge-transport Bound on Kept Nodes*)**.** *Let a kept node be modeled by $R > 0$, $C > 0$, driven by $u \in C^1([0,T])$. Let v solve $C\dot{v} + \frac{1}{R}v = \frac{1}{R}u$, $v(0) = u(0)$, and for $s \geq 1$ let $u_s(t) = u(t/s)$ on $[0,sT]$ with solution v_s. Define $E_R(sT) := \int_0^{sT} R\,(C\dot{v}_s(t))^2\,dt$. Then*

$$E_R(sT) \ \leq\ \frac{C}{2s^2}\int_0^T (T-\eta)\big(\dot{u}(\eta)\big)^2\,d\eta \ \leq\ \frac{CT}{2s^2}\int_0^T \big(\dot{u}(\eta)\big)^2\,d\eta, \tag{19}$$

hence $E_R(sT) = \mathcal{O}(1/s^2)$ as $s \to \infty$ and in particular $E_R(sT) = \mathcal{O}(1/s)$.

Proof. Set $w_s := v_s - u_s$. Then $\dot{w}_s + \frac{1}{RC}w_s = -\dot{u}_s$, $w_s(0) = 0$, so $w_s(t) = -\int_0^t e^{-(t-\xi)/(RC)}\dot{u}_s(\xi)\,d\xi$. With $\dot{u}_s(\xi) = \frac{1}{s}\dot{u}(\xi/s)$ and $t = s\theta$, $\xi = s\eta$,

$$E_R(sT) = \frac{1}{R}\int_0^{sT} w_s(t)^2 dt = \frac{1}{Rs}\int_0^T \Big[\int_0^\theta e^{-s(\theta-\eta)/(RC)}\dot{u}(\eta)\,d\eta\Big]^2 d\theta.$$

By Cauchy–Schwarz and $\int_0^\theta e^{-2s(\theta-\eta)/(RC)}\,d\eta \leq \frac{RC}{2s}$,

$$E_R(sT) \leq \frac{1}{Rs}\int_0^T \frac{RC}{2s}\Big(\int_0^\theta (\dot{u})^2\Big)d\theta = \frac{C}{2s^2}\int_0^T (T-\eta)(\dot{u}(\eta))^2\,d\eta,$$

which gives (19). Thus

$$E_R(sT) \leq \frac{C}{2s^2}\int_0^T (T-\eta)(\dot{u}(\eta))^2\,d\eta.$$

Since $T - \eta \leq T$ for $\eta \in [0,T]$, we obtain

$$E_R(sT) \leq \frac{CT}{2s^2}\int_0^T (\dot{u}(\eta))^2\,d\eta,$$

so $E_R(sT) = \mathcal{O}(1/s^2)$ and, for $s \geq 1$, also $E_R(sT) = \mathcal{O}(1/s)$. Consequently, $E_R(sT) \to 0$ as $s \to \infty$, i.e., the resistive dissipation vanishes in the adiabatic limit. □

For multiple kept nodes, the total adiabatic loss is the sum of branch losses and therefore obeys the same decay.

A.6. Dynamic dissipation law and electrostatic wall

We now pass from the per-input identities to the mean energy per operation. Following the standard reversible-thermodynamics formalism, the input is modeled as a random variable X supported on S, and energetic quantities are evaluated in expectation; see [14,57]. Each logical register assumes one of the two voltage levels $\{V_{\mathrm{ref}}, V\}$ at $t = 0$ and $t = T$ by the model of Appendix A. For each erased node $i \in \mathcal{E}$, define the erased-indicator and its probability

$$B_i(x) = \mathbf{1}\{\, v_i(0;x) = V,\ v_i(T;x) = V_{\mathrm{ref}}\,\}, \qquad \beta_i = \mathbb{P}[B_i(X) = 1].$$

Define the capacitance-weighted erased-high fraction

$$\bar{\beta} = \begin{cases} \dfrac{1}{C_{\mathrm{eras}}} \sum_{i\in\mathcal{E}} \beta_i C_i, & C_{\mathrm{eras}} > 0, \\ 0, & C_{\mathrm{eras}} = 0. \end{cases}$$

This sets up the expectation-level dissipation expression in which the irreversible contribution appears explicitly through C_{eras} and $\bar{\beta}$. This formulation makes the irreversible erase term and the adiabatic term explicit in circuit capacitances and voltages. We are not aware of an equivalent expression in the prior literature, and we therefore state the resulting dynamic dissipation law below.

Theorem A.12 (*Dynamic Dissipation Law and Electrostatic Wall*). *Let $E_{\mathrm{dyn}} := \mathbb{E}[Q_R(X)]$ be the mean dynamic energy per operation. Then*

$$E_{\mathrm{dyn}} = \bar{\beta}\,\frac{1}{2}\,C_{\mathrm{eras}}(V^2 - V_{\mathrm{ref}}^2) + \varepsilon_{\mathrm{ad}} \cdot \frac{1}{2}\,C_{\mathrm{keep}}\,(\Delta V_{\max})^2, \qquad \varepsilon_{\mathrm{ad}} = \mathcal{O}(1/s) \to 0, \tag{20}$$

where $\Delta V_{\max} := \max\{|V|, |V_{\mathrm{ref}}|\}$ and $\varepsilon_{\mathrm{ad}}$ is the adiabatic factor supplied by Lemma A.11. In particular, under the normalization $V_{\mathrm{ref}} = 0$,

$$\lim_{s\to\infty} E_{\mathrm{dyn}} = \bar{\beta}\,\frac{1}{2}\,C_{\mathrm{eras}} V^2. \tag{21}$$

Proof. **Erased part.** For any node i and input x, the nodewise capacitor-energy change is

$$\Delta U_{C,i}(x) = \tfrac{1}{2}\,C_i\big(v_i(T;x)^2 - v_i(0;x)^2\big).$$

By Definition A.4, for every $i \in \mathcal{E}$ and every x in the equivalence class considered, $v_i(T;x) = V_{\mathrm{ref}}$. Hence

$$\Delta U_{C,i}(x) = \begin{cases} \tfrac{1}{2}\,C_i\,(V_{\mathrm{ref}}^2 - V^2), & \text{if } v_i(0;x) = V, \\ 0, & \text{if } v_i(0;x) = V_{\mathrm{ref}}. \end{cases}$$

During a forward operation, the energy drop on erased nodes cannot be recovered by the quasi-static kept reconfiguration of Lemma A.8. Stage B clamps $\mathcal{E}$ and contributes $\Delta U^{\mathrm{B}}_{C,\mathcal{E}} = 0$ and $Q^{\mathrm{B}}_R = 0$. Therefore, the erased-node drop realized in Stage A is dissipated as heat during that operation. Taking expectation over X and summing over $i \in \mathcal{E}$,

$$\mathbb{E}[Q_{R,\mathrm{eras}}] = \sum_{i\in\mathcal{E}} \beta_i\,\tfrac{1}{2}\,C_i\,(V^2 - V_{\mathrm{ref}}^2) = \bar{\beta}\,\tfrac{1}{2}\,C_{\mathrm{eras}}\,(V^2 - V_{\mathrm{ref}}^2).$$

Kept part. For $i \in \mathcal{K}$, terminal voltages depend on x and are transported by the quasi-static kept reconfiguration of Lemma A.8. Applying the adiabatic bound of Lemma A.11 branchwise and summing over $i \in \mathcal{K}$ yields, for each x,

$$Q_{R,\mathrm{keep}}(x) \ \le\ \varepsilon_{\mathrm{ad}} \cdot \tfrac{1}{2}\,C_{\mathrm{keep}}\,(\Delta V_{\max})^2, \qquad \varepsilon_{\mathrm{ad}} = \mathcal{O}(1/s).$$

Taking expectation over X preserves this bound and gives $E_{\mathrm{keep}} := \mathbb{E}[Q_{R,\mathrm{keep}}(X)] \le \varepsilon_{\mathrm{ad}} \cdot \tfrac{1}{2}\,C_{\mathrm{keep}}\,(\Delta V_{\max})^2$.

Sum. Since $Q_R = Q_{R,\mathrm{eras}} + Q_{R,\mathrm{keep}}$, linearity of expectation gives (20). Letting $s \to \infty$ and using $\varepsilon_{\mathrm{ad}} \to 0$ from Lemma A.11 yields (21). □

A.7. Physical criterion for reversible computation

We now combine the erased-capacitance invariant of Appendix A.3, the closed-cycle identity of Theorem A.10, and the adiabatic scaling result of Appendix A.5 to state the physical counterpart of logical reversibility.

Corollary A.13 (*Physical reversibility criterion*). *Under the passive lumped model and Axioms A.1–A.2, the following are equivalent:*

$$\exists\ \textit{realization with } \lim_{s\to\infty} E_{\mathrm{dyn}} = 0 \iff f|_S \textit{ is injective} \iff \exists\ \textit{realization with } C_{\mathrm{eras}} = 0.$$

Proof (⇒). If there exists a realization with $\lim_{s\to\infty} E_{\mathrm{dyn}} = 0$, then by Theorem A.12

$$E_{\mathrm{dyn}} = \bar{\beta}\,\tfrac{1}{2}\,C_{\mathrm{eras}} V^2 + \varepsilon_{\mathrm{ad}} \cdot \tfrac{1}{2}\,C_{\mathrm{keep}} V^2, \qquad \varepsilon_{\mathrm{ad}} \to 0,$$

so the limit $E_{\mathrm{dyn}} \to 0$ forces $C_{\mathrm{eras}} = 0$. By Proposition A.7, $C_{\mathrm{eras}} = 0$ implies $f|_S$ is injective.

(⇐) If $f|_S$ is injective, then Proposition A.7 gives $C_{\mathrm{eras}} = 0$. Substituting into Theorem A.12 yields

$$E_{\mathrm{dyn}} = \varepsilon_{\mathrm{ad}} \cdot \tfrac{1}{2}\,C_{\mathrm{keep}} V^2, \qquad \varepsilon_{\mathrm{ad}} = \mathcal{O}(1/s).$$

By Lemma A.11 and Theorem A.3, $\varepsilon_{\mathrm{ad}} \to 0$ as $s \to \infty$, hence $E_{\mathrm{dyn}} \to 0$. The middle equivalence follows directly from Proposition A.7. □

Data availability

The code used in this study is included as supplementary material.

References

[1] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[2] L.N. Trefethen, D. Bau, Numerical Linear Algebra, SIAM, 2022.
[3] Our World in Data, Artificial intelligence training computation by researcher affiliation, 2024, https://ourworldindata.org/grapher/artificial-intelligence-training-computation-by-researcher-affiliation. (Accessed August 2025).
[4] S. Abdulah, A.H. Baker, G. Bosilca, Q. Cao, S. Castruccio, M.G. Genton, D.E. Keyes, Z. Khalid, H. Ltaief, Y. Song, et al., Boosting earth system model outputs and saving petabytes in their storage using exascale climate emulators, in: SC24: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2024, pp. 1–12.
[5] J. Shalf, The future of computing beyond Moore's law, Phil. Trans. R. Soc. A 378 (2166) (2020) 20190061.
[6] A. Ho, E. Erdil, T. Besiroglu, Limits to the energy efficiency of CMOS microprocessors, in: 2023 IEEE International Conference on Rebooting Computing, ICRC, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 1–10, http://dx.doi.org/10.1109/ICRC60800.2023.10386559, URL https://doi.ieeecomputersociety.org/10.1109/ICRC60800.2023.10386559.
[7] W.C. Athas, L. Svensson, Reversible logic issues in adiabatic CMOS, in: Proceedings Workshop on Physics and Computation. PhysComp'94, IEEE, 1994, pp. 111–118.
[8] M.P. Frank, R.W. Brocato, B.D. Tierney, N.A. Missert, A.H. Hsia, Reversible computing with fast, fully static, fully adiabatic CMOS, in: 2020 International Conference on Rebooting Computing, ICRC, IEEE, 2020, pp. 1–8.
[9] M.P. Frank, Asynchronous ballistic reversible computing, in: 2017 IEEE International Conference on Rebooting Computing, ICRC, IEEE, 2017, pp. 1–8.
[10] N. Takeuchi, D. Ozawa, Y. Yamanashi, N. Yoshikawa, An adiabatic quantum flux parametron as an ultra-low-power logic device, Supercond. Sci. Technol. 26 (3) (2013) 035010.
[11] Q. Herr, Landauer-limited dissipation in quantum-flux-parametron logic, 2025, arXiv preprint arXiv:2504.04284.
[12] A.W. Harrow, A. Hassidim, S. Lloyd, Quantum algorithm for linear systems of equations, Phys. Rev. Lett. 103 (15) (2009) 150502.
[13] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 183–191.
[14] C.H. Bennett, Logical reversibility of computation, IBM J. Res. Dev. 17 (6) (1973) 525–532.
[15] M.P. Frank, T.F. Knight Jr., Reversibility for Efficient Computing (Ph.D. thesis), Massachusetts Institute of Technology, Dept. of Electrical Engineering and ..., 1999.
[16] M.P. Frank, Introduction to reversible computing: motivation, progress, and challenges, in: Proceedings of the 2nd Conference on Computing Frontiers, 2005, pp. 385–390.
[17] M.P. Frank, Foundations of generalized reversible computing, in: International Conference on Reversible Computation, Springer, 2017, pp. 19–34.

[18] M.P. Frank, Perfectly Adiabatic CMOS Logic, Technical Report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2021.
[19] M.P. Frank, A.J. Edwards, Industry perspective: Limits of energy efficiency for conventional CMOS and the need for adiabatic reversible computing, APL Electron. Devices 1 (3) (2025).
[20] H. Thapliyal, S. Kotiyal, M. Srinivas, Novel BCD adders and their reversible logic implementation for IEEE 754r format, in: 19th International Conference on VLSI Design Held Jointly with 5th International Conference on Embedded Systems Design, VLSID'06, IEEE, 2006, pp. 6–pp.
[21] M. Nachtigal, H. Thapliyal, N. Ranganathan, Design of a reversible floating-point adder architecture, in: 2011 11th IEEE International Conference on Nanotechnology, IEEE, 2011, pp. 451–456.
[22] N. Keerthika, M.M. Rajmohan, A novel design of reversible floating point adder architecture, 2014.
[23] T.D. Nguyen, R. Van Meter, A resource-efficient design for a reversible floating point adder in quantum computing, J. Emerg. Technol. Comput. Syst. 11 (2014) http://dx.doi.org/10.1145/2629525.
[24] J. Jain, R. Agrawal, Design and development of efficient reversible floating point arithmetic unit, in: 2015 Fifth International Conference on Communication Systems and Network Technologies, IEEE, 2015, pp. 811–815.
[25] K.S. Perumalla, S.B. Yoginath, Towards reversible basic linear algebra subprograms: A performance study, in: Transactions on Computational Science XXIV: Special Issue on Reversible Computing, Springer, 2014, pp. 56–73.
[26] E.D. Demaine, J. Lynch, J. Sun, An efficient reversible algorithm for linear regression, in: 2021 International Conference on Rebooting Computing, ICRC, IEEE, 2021, pp. 103–108.
[27] A. Anantha Lakshmi, G. Sudha, Design of a reversible single precision floating point subtractor, SpringerPlus 3 (1) (2014) 11.
[28] T.G. De Brugière, M. Baboulin, B. Valiron, S. Martiel, C. Allouche, Gaussian elimination versus greedy methods for the synthesis of linear reversible circuits, ACM Trans. Quantum Comput. 2 (3) (2021) 1–26.
[29] C. Lutz, H. Derby, Janus: a time-reversible language, Lett. To R. Landau. 2 (1986).
[30] N. Tyagi, J. Lynch, E.D. Demaine, Toward an energy efficient language and compiler for (partially) reversible algorithms, in: International Conference on Reversible Computation, Springer, 2016, pp. 121–136.
[31] J.G. Liu, T. Zhao, Differentiate everything with a reversible embeded domain-specific language, 2020, arXiv preprint arXiv:2003.04617.
[32] Vaire Computing, Reversible Processor Roadmap, White paper, Vaire Computing, 2025, URL https://vaire.co/whitepapers.
[33] IEEE Spectrum, Startup vaire computing's reversible chips aim to slash energy use, 2024, URL https://spectrum.ieee.org/reversible-computing. [Online].
[34] MIT Technology Review, This chip could bring computing closer to its thermodynamic limits, 2024, URL https://www.technologyreview.com/2024/04/11/vaire-reversible-chip. [Online].
[35] J. Nocedal, S.J. Wright, Numerical Optimization, Springer, 2006.
[36] D.A. Spielman, S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, in: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 2004, pp. 81–90.
[37] M. Belkin, P. Niyogi, V. Sindhwani, Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, J. Mach. Learn. Res. 7 (11) (2006).
[38] T.J. Dekker, A floating-point technique for extending the available precision, Numer. Math. 18 (3) (1971) 224–242.
[39] C.T. Kelley, Solving Nonlinear Equations with Newton's Method, SIAM, 2003.
[40] E. Quinnell, E.E. Swartzlander, C. Lemonds, Floating-point fused multiply-add architectures, in: 2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, IEEE, 2007, pp. 331–337.
[41] N.J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, 2002.
[42] G.H. Golub, C.F. Van Loan, Matrix computations, JHU Press, 2013.
[43] S.A. Cuccaro, T.G. Draper, S.A. Kutin, D.P. Moulton, A new quantum ripple-carry addition circuit, 2004, ArXiv Preprint Quant-Ph/0410184.
[44] D. Maslov, G.W. Dueck, D.M. Miller, Toffoli network synthesis with templates, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 24 (6) (2005) 807–817.
[45] R. Wille, R. Drechsler, BDD-based synthesis of reversible logic for large functions, in: Proceedings of the 46th Annual Design Automation Conference, 2009, pp. 270–275.
[46] S. Mach, F. Schuiki, F. Zaruba, L. Benini, FPnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (4) (2020) 774–787.
[47] N.H. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Pearson Education India, 2015.
[48] S. Bhalachandra, B. Austin, S. Williams, N.J. Wright, Understanding the impact of input entropy on fpu, cpu, and gpu power, 2022, arXiv preprint arXiv:2212.08805.
[49] J.W. Choi, D. Bedard, R. Fowler, R. Vuduc, A roofline model of energy, in: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IEEE, 2013, pp. 661–672.
[50] D.M. Pozar, Microwave Engineering, fourth ed., Wiley, 2011.
[51] M.P. Frank, Energy-efficient computing with reversible logic: Status and roadmap, in: Invited talk, International Conference on Rebooting Computing (ICRC), 2022, URL https://www.sandia.gov/app/uploads/sites/210/2022/06/ICRC20-talk-2v4SAND.pdf. slides. Simulated resonator quality factors Q 1000–3000 and target dissipation goals reported.
[52] Dutch Data Center Association, Facts & figures 2023: Key data centre statistics, 2023, https://www.dutchdatacenters.nl/en/factsheet/. (Accessed 2 November 2025).
[53] A. Shehabi, A. Newkirk, S.J. Smith, et al., United States Data Center Energy Usage Report 2023, Technical Report, Lawrence Berkeley National Laboratory, 2024, http://dx.doi.org/10.71468/P1WC7Q, URL https://escholarship.org/uc/item/32d6m0d1. (Accessed 2 November 2025).
[54] L.O. Chua, C.A. Desoer, E.S. Kuh, Linear and Nonlinear Circuits, McGraw-Hill, 1987.
[55] J.A. Svoboda, R.C. Dorf, Introduction to Electric Circuits, John Wiley & Sons, 2013.
[56] A. Van Der Schaft, Classical thermodynamics revisited: A systems and control perspective, IEEE Control Syst. Mag. 41 (5) (2021) 32–60.
[57] O.J. Maroney, Generalizing Landauer's principle, Phys. Rev. E—Statistical, Nonlinear, Soft Matter Phys. 79 (3) (2009) 031105.