Fast and Stable Triangular Inversion for Delta-Rule Linear Transformers

Aleksandros Sobczyk **Gioele Gottardo** **Christos K. Matzoros**

Mirko De Vita **Filip Skogh** **Anastasios Zouzias** **Jiawei Zhuang**

Huawei Technologies Switzerland AG
Zurich, Switzerland *

Abstract

Linear attention has emerged as a cornerstone for efficient long-context architectures, as evidenced by its integration into state-of-the-art open-source models including Qwen3.5/3.6, Kimi Linear/K3, and RWKV-7. Models that incorporate linear attention layers with the so-called Delta-Rule involve the inversion of triangular matrices as a core sub-routine. This operation often forms a performance bottleneck, and, due to its high-sensitivity to numerical errors, it can significantly deteriorate end-to-end model accuracy if it is not carefully implemented. This work provides a systematic analysis of both direct and iterative triangular inversion algorithms, targeting methods that are rich in matrix products, and, therefore, have the potential to efficiently utilize modern hardware. To that end, our analysis covers a broad spectrum of mathematical and practical aspects, with a heavy focus on numerical stability, computational complexity, and, ultimately, hardware efficiency and practical considerations. We provide a rigorous experimental evaluation to verify these properties in practical scenarios, and in low-precision floating-point representations, highlighting the strengths and limitations of each method. Performance benchmarks on NPUs reveal up to $4.3\times$ speed-up against the state-of-the-art implementations of SGLang for triangular matrix inversion, leading to significant performance improvements on the entire layer level, while maintaining end-to-end model accuracy.

1 Introduction

The attention mechanism [36] is one of the most important innovations of the last decade in deep learning, and lies at the core of Large Language Models (LLMs). The quadratic time complexity and the linearly growing KV-cache of softmax-attention have become key bottlenecks for scaling LLMs. Linear attention [20] offers a way to reduce the computational complexity, and many different variants have been developed to close the performance gap between linear and standard attention [6, 21, 25, 34, 43]. Prominent approaches towards this direction include DeltaNets [27, 44] and Gated DeltaNets (GDNs) [42], which introduce decay and gating factors.

In brief, given an input sequence of length L, the corresponding query-key-value matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times d}$, where d is the head dimension, and a causal mask (lower-triangular) matrix $\mathbf{M} \in \mathbb{R}^{L\times L}$, in

*Contacts: aleksandros.sobczyk@huawei.com gioele.gottardo2@h-partners.com christos.konstantinos.matzoros@h-partners.com mirko.de.vita@h-partners.com filip.skogh@h-partners.com anastasios.zouzias@huawei.com zhuangjiawei@huawei.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

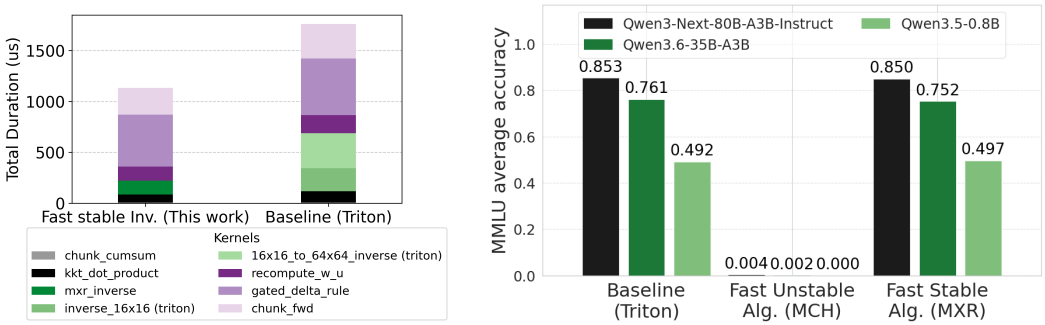

Figure 1: Left: Runtime breakdown of single GDN layer. Right: Accuracy drop in end-to-end benchmark evaluation due to inversion error accumulation.

its simplest form linear attention can be defined in two equivalent ways:

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V} \quad \text{(matrix form)}, \tag{1}$$

$$\mathbf{o}_t = \left(\sum_{j=1}^{t} \mathbf{v}_j \mathbf{k}_j^\top\right) \mathbf{q}_t = \mathbf{V}_{1:t,:}^\top \mathbf{K}_{1:t,:} \mathbf{q}_t \quad \text{(linear recurrence)}, \tag{2}$$

where $\odot$ denotes element-wise matrix product. The linear recurrence formulation requires significantly less memory than the matrix form, as it only requires the state matrix to update the model state at any time step. On the other hand, the matrix form lends itself for parallel processing. To achieve the "best of both worlds", and to optimize the computational capabilities of AI accelerators, a standard technique is to utilize *chunk-wise* parallel processing [17, 44]. In this case, the sequence is partitioned into $\lceil L/C \rceil$ "chunks" of (at most) length C each. In practice, C is typically one of $\{16, 32, 64, 128\}$. Let us denote the corresponding "chunks" of the matrix $\mathbf{K}$ as $\mathbf{K}_{[k]} := \mathbf{K}_{kC:(k+1)C,:} \in \mathbb{R}^{C \times d}$. After several non-trivial mathematical re-formulations (see [42] and Appendix A), the final output updates within each chunk require the computation of the inverse of a *unit-diagonal lower triangular matrix* $\mathbf{A}$ of size $C \times C$, which has the form:

$$\mathbf{A}_{[k]}^{-1} = \left(\mathbf{I} + \phi\left(\mathbf{K}_{[k]}, \mathbf{K}_{[k]}^\top\right) \odot \mathbf{L}^-\right)^{-1}, \quad \mathbf{L}_{i,j}^- := \begin{cases} 0, & \text{if } i \leq j, \\ 1, & \text{otherwise}, \end{cases} \tag{3}$$

where $\phi(\cdot,\cdot) \in \mathbb{R}^{C \times C}$ is a function that depends on the specific model architecture, typically involving element-wise functions of the elements of $\mathbf{K}$, such as scaling or exponentiation, $\mathbf{I}$ is the identity matrix, and $\mathbf{L}^-$ is a strictly lower triangular all-ones (mask) matrix of size C. We provide a description of the different $\phi(\cdot,\cdot)$ functions that are used by various models in Table 1.

Table 1: Examples of functions $\phi(\cdot,\cdot)$ in existing models.

Model $\phi(\cdot,\cdot)$
DeltaNet [27] $\mathrm{diag}(\beta) \cdot \mathbf{K}\mathbf{K}^\top$
DeltaFormer [41] $\exp(\mathbf{W}\mathbf{K}^\top)$
GDN [42] $\mathbf{K}\mathbf{K}^\top \odot \mathcal{A}$
Comba [16] $\mathbf{K}\mathbf{K}^\top \odot \mathcal{A}^{i-1/j}$
KDA [21] $(\mathbf{K} \odot \mathbf{\Gamma})(\mathbf{K} \oslash \mathbf{\Gamma})^\top$

Table 2: Summary of triangular matrix inversion methods.

Method (Acronym)	Type	Stability	# MatMuls	Refs.
Vector Col. Sweep (VCS)	Direct	stable (Lem. 3.1)	-	[9, 26]
Matrix Col. Sweep (MCS)	Direct	stable (Lem. 3.1)	$O(n)$	[9, 26]
Bunch-Hopcroft (MBH)	Direct	log-stable	$\approx 2\log(n)$	[2, 7]
Cayley-Hamilton (MCH)	Direct	unstable	$2\lceil\log(\frac{n}{2})\rceil$	[5]
Mixed Recursion (MXR)	Direct	log-stable	$\approx 2\log(n)$	Sec. 3.4
Newton-Schulz (NS)	Iterative	stable	$2 \times N_{\text{iter}}$	[23, 28]
Iterative Refinement (IR)	Iterative	stable (per iter)	$2 \times N_{\text{iter}}$	[14]

1.1 Motivation and Contributions

In existing model implementations, the actual inversion of the individual chunks can quickly become a performance bottleneck. For example, in Figure 1 (Left) it is shown that the triangular inversion part can take up to $\sim 40\%$ of the total layer device-runtime.

Alternative approaches for hardware-aware algorithms have been proposed [47] to better utilize matrix cores in AI accelerators and to surpass the performance bottlenecks. However, matrix cores

are usually optimized for low-precision arithmetic, and hence if these hardware-aware algorithms are not designed carefully, the errors introduced due to low-precision floating point arithmetic can easily lead to catastrophic accumulation of errors, often returning completely wrong, or even "NaN" solutions. This is clearly illustrated in Figure 1 (Right), where we show how an unstable inversion method drastically reduces the reasoning capabilities of Qwen3.6, dropping the accuracy on MMLU from 76.1% to 0.2%! Evidently, model accuracy is highly correlated to the accuracy of the underlying inversion algorithm.

This is of course not surprising. Inverting triangular matrices, or, even more importantly, solving[2] triangular systems of linear equations, is a very thoroughly studied problem in numerical analysis, dating all the way back to the early works of Wilkinson [40]. One of the main reasons is that triangular factorizations (LU, Cholesky, QR, etc) established themselves early-on as the "gold standard" procedure to solve general (dense) linear systems of equations. As such, the stability properties and computational complexity are quite well-understood, and algorithm implementations should always take them into consideration and carefully handle numerical errors.

The aforementioned observations motivated us to revisit the rich bibliography on triangular matrix inversion algorithms, in the context of Linear Transformers. We focus on deriving algorithms based on the following guidelines:

Matrix products and hardware efficiency: In order to take advantage of matrix cores in modern AI accelerators, the underlying inversion algorithms should extensively use matrix products.

Numerical stability: Fast algorithms are not useful if they return wrong results. To that end, numerical stability analysis is mandatory to ensure high-accuracy and meaningful results in real-applications.

Focus on Linear Attention: The specific properties of the matrices arising in Delta-Rule Linear Transformers should be exploited as much as possible to achieve the best performance, both from the mathematical analysis perspective, as well as practical performance.

Our main contribution is the thorough analysis of several types of direct and iterative triangular inversion algorithms, based on the aforementioned guidelines. A summary of the methods studied here and their properties is provided in Table 2.

By dedicating the analysis to the underlying types of matrices, we provide some novel mathematical properties that are also useful for practical applications. Stability properties are thoroughly tested experimentally for all algorithms, in three different precisions: float32, float16, and bfloat16. Interestingly, we show that even *abysmally unstable* algorithms (Alg. 5, MCH), if used carefully, can be efficiently exploited as sub-routines inside very fast and stable algorithms (Alg. 2, MXR). In terms of performance, the latter approach achieved up to 4.3x speedup on NPUs compared to the current state-of-the-art implementations for triangular inversion in SGLang.

2 Preliminaries

2.1 Numerical stability

Fast algorithms are not useful if the returned results are not accurate. In this section, we recall a notion of numerical stability for matrix inversion (see e.g. [8, 14]), which allows us to argue about the robustness of different algorithms against numerical errors. It is especially important in low-precision scenarios, which are becoming more and more prominent in AI workloads. We assume a standard floating-point model of computation with t bits for the significand, p bits for the exponent, and one bit for the sign. The machine precision is equal to $\mathbf{u} := 2^{-t}$. (see Appendix B for a detailed description).

Definition 2.1. *Let $\mathbf{A}$ be an invertible $n \times n$ triangular matrix. An algorithm $\mathcal{A}$ which computes an approximate inverse $\widetilde{\mathbf{A}}^{-1}$ is said to be f-stable if:*

$$\left\| \mathbf{A}^{-1} - \widetilde{\mathbf{A}}^{-1} \right\|_2 \leq c_1 \cdot n^{c_2} \cdot \kappa_2^f(\mathbf{A}) \cdot \mathbf{u} \cdot \left\| \mathbf{A}^{-1} \right\|_2 ,$$

where c_1 and c_2 are global constants (independent of the matrix size n), and $\kappa_2(\mathbf{A}) = \|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2 \geq 1$ is the spectral-norm condition number of $\mathbf{A}$.

[2] In applications where the main task is to solve a linear system of equations, the explicit computation of the entire inverse is evidently redundant, and even characterized as a "sin" [15], as it can harm both the performance and the numerical accuracy.

In the above definition, f can be a constant, or a function of n. For more details and different notions of numerical stability we refer to standard textbooks [9, 10, 14] and references therein. Throughout the paper, we might refer to $O(1)$-stable algorithms as "fully-stable" or simply "stable", and to $O(\text{polylog}(n))$-stable algorithms as "logarithmically-stable" or "log-stable", as coined in [7].

2.2 Triangular matrices and their condition number

One of the challenging and intriguing facts about triangular matrices in general, is that they can easily have an abominable condition number: Viswanath and Trefethen proved that the matrix $\mathbf{I} + \texttt{strict_tril}(\mathbf{G})$, where $\mathbf{G}_{i,j} \sim \mathcal{N}(0,1)$, almost surely has an *exponentially growing* condition number [37]. Since the floating point errors in matrix inversion depend on the condition number of the input, this fact seems very discouraging for inverting GDN matrices, since they are also triangular with unit diagonal elements. Luckily, this is not the case. One of the key observations for our analysis is that matrices of the form of Eq. (3) are actually *well-conditioned*. This is necessary to justify the use of low-precision compute units for such operations, and it also reveals why such implementations have been working well in practice to-date. In [18], it was proved that the elements of the inverse of a DeltaNet matrix are always in $[-1, 1]$. The following implication is immediate (a simple proof can be found in Appendix C.1).

Corollary 2.1. *The condition number of an $n \times n$ DeltaNet matrix $\mathbf{A}$ of Equation (3) is at most n^2.*

Note that the scaling $\phi(\cdot,\cdot)$ that takes place in Eq. (3) typically reduces the magnitude of the off-diagonal elements, in which case it is expected to further reduce the condition number as the matrix becomes more-and-more diagonally dominant. In that respect, these types of matrices are much better conditioned than random triangular matrices, which is promising towards exploiting low-precision hardware and fast (potentially less stable) algorithms.

Algorithm 1 MCS$(\mathbf{L}, n)$:

$\mathbf{L} \leftarrow 2\mathbf{I} - \mathbf{L}$
$\mathbf{X} \leftarrow \mathbf{I}$
for $k = n-1$ down to 0 **:**
  $\mathbf{M} \leftarrow \mathbf{I} + \mathbf{L}[:,k] \cdot \mathbf{e}_k^\top$
  $\mathbf{X} \leftarrow \mathbf{M} \cdot \mathbf{X}$
return $\mathbf{X}$

Algorithm 2 MXR$(\mathbf{L}, n, b_0, r = 0)$:

$\mathbf{D} \leftarrow b_0 \times b_0$ diagonal blocks of $\mathbf{L}$
$\mathbf{Y} \leftarrow$ MCH$(\mathbf{D}, b_0)$
for $i = 1, \ldots, r$ **:**
  $\mathbf{Y} \leftarrow$ IR$(\mathbf{Y}, \mathbf{L})$ ▷ Alg. 6
$\mathbf{X} \leftarrow$ MBH$(\mathbf{L}, n, \mathbf{Y}, b_0)$ ▷ Alg. 4
return $\mathbf{X}$

Algorithm 3 NS$(\mathbf{L}, n, m, \mathbf{X}_0)$:

$\mathbf{A} \leftarrow \mathbf{I} + \mathbf{L}$
$\mathbf{X} \leftarrow \mathbf{X}_0$
for $k = 0, \ldots, m-1$ **:**
  $\mathbf{Y} \leftarrow \mathbf{A} \cdot \mathbf{X}$
  $\mathbf{X} \leftarrow 2\mathbf{X} - \mathbf{X} \cdot \mathbf{Y}$
return $\mathbf{X}$

Algorithm 4 MBH$(\mathbf{L}, n, \mathbf{X}_0 = \mathbf{I}, b_0 = 1)$:

$\mathbf{X} \leftarrow \mathbf{X}_0$
for $b = b_0, 2b_0, 4b_0, \ldots, n$ **:**
  $\mathbf{D}_e \leftarrow$ even $b \times b$ diagonal blocks of $\mathbf{X}$
  $\mathbf{D}_o \leftarrow$ odd $b \times b$ diagonal blocks of $\mathbf{X}$
  $\mathbf{X} \leftarrow \mathbf{D}_e + \mathbf{D}_o - \mathbf{D}_o \cdot \mathbf{L} \cdot \mathbf{D}_e$
return $\mathbf{X}$

Algorithm 5 MCH$(\mathbf{L}, n)$:

$\mathbf{X} \leftarrow \mathbf{I} - \mathbf{L}$
$\mathbf{Y} \leftarrow \mathbf{L}$
for $k = 1, 2, \ldots, \log(\frac{n}{2})$ **:**
  $\mathbf{Y} \leftarrow \mathbf{Y} \cdot \mathbf{Y}$
  $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{X} \cdot \mathbf{Y}$
return $\mathbf{X}$

Algorithm 6 IR$(\mathbf{Y}, \mathbf{L})$:

$\mathbf{A} \leftarrow \mathbf{I} - \mathbf{L}$
$\mathbf{R} \leftarrow \mathbf{I} - \mathbf{Y}\mathbf{A}$
$\mathbf{X} \leftarrow \mathbf{Y} + \mathbf{R}\mathbf{Y}$
return $\mathbf{X}$

3 Analysis of methods

In this section, we provide the analysis of the main triangular inverse methods considered here, in terms of their stability properties and their computational complexity. Complexity is mainly focused on matrix products, which is also in-line with recent proposals on models of computation that focus on modern AI accelerators [3, 4, 32, 48]. A fine-grained "bit complexity" analysis can be inferred by the floating point precision bounds. Algorithms 1-6 provide a high-level description of the methods.

3.1 Vectorized and matrix product-based column-sweep (VCS and MCS)

The first algorithm that we discuss is the one proposed by Sameh and Brent [26], which, to our knowledge, was the first parallel algorithm to solve triangular systems with logarithmic depth. It is often referred to as the *column-sweep* method, and it can be implemented in a vectorized form, or in a matrix product-based formulation (see also [9]). The proof can be found in Appendix C.2.

Lemma 3.1. *Column-sweep is numerically-stable, as per Definition 2.1. For well-conditioned matrices (e.g. DeltaNet) that satisfy Corollary 2.1, $t = \log(1/\epsilon) + O(\log(n))$ bits of accuracy are sufficient so that the returned solution $\widetilde{\mathbf{A}}^{-1}$ satisfies $\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq \epsilon \|\mathbf{A}^{-1}\|_2$, for any $\epsilon \in (0, 1)$.*

The corresponding matrix product-based formulation is described in Algorithm 1. In general, the MCS algorithm below generates the matrices $\mathbf{M}_{n-1}\mathbf{M}_{n-2}\dots\mathbf{M}_1$ and performs the chained matrix products. One drawback of the MCS method is that, for matrix size n, it requires $n-1$ matrix products. In terms of operations, the complexity of these two algorithms is as follows:

Complexity (VCS): $O(n^2)$ **vector ops** of length $n, n-1, n-2, \dots, 2, 1$.

Complexity (MCS): $n-1$ **matmuls** of size $n \times n$, $O(n)$ **vector ops** of length $O(n)$.

3.2 Inverse via characteristic polynomial (MCH)

The main drawback of the column-sweep algorithm is that it executes $O(n)$ square matrix products of size n. The authors of [47] highlight that an alternative algorithm can be considered for such cases, which reduces inverse computation to $O(\log(n))$ matrix products. It can be straightforwardly derived from *Cayley–Hamilton theorem*, by noting that all eigenvalues are equal to one, and therefore the characteristic polynomial can be derived analytically. The inverse in this case is given by:

$$(\mathbf{I} + \mathbf{L})^{-1} = \mathbf{I} - \mathbf{L} + \mathbf{L}^2 - \mathbf{L}^3 + \cdots + (-1)^{n-1}\mathbf{L}^{n-1}, \tag{4}$$

where $\mathbf{L}$ is an arbitrary strict triangular square matrix of size n, and it can be quickly computed via repeated squaring: we first set $\mathbf{X} \leftarrow \mathbf{I} - \mathbf{L}$ and $\mathbf{Y} \leftarrow \mathbf{L}$, and, thereafter, for $i = 1, \dots, \lfloor \log(n) \rfloor - 1$, the algorithm updates first $\mathbf{Y} \leftarrow \mathbf{Y} \cdot \mathbf{Y}$, and then $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{X} \cdot \mathbf{Y}$. It finally returns the last $\mathbf{X}$. An algorithmic description is listed in Algorithm 5.

Complexity (MCH): $\approx 2\log(n)$ square matrix products of size n.

In terms of stability, this algorithm and other ones related to the characteristic polynomial are known to have terrible numerical properties (see e.g. the discussion in [5]). The main issues arise due to the rapid growth of intermediate values caused by the repeated squaring. Consider for example the matrix $\mathbf{A} = \mathbf{I} + \mathbf{L}^-$ where $\mathbf{L}^-$ is the strict lower triangular matrix with all entries equal to -1. After $\log(k)$ iterations of MCH, the elements of the matrix $\mathbf{Y}_k = (\mathbf{L}^-)^k$ can be derived analytically: $(\mathbf{L}^-)^k_{i,j} = \binom{i-j-1}{k-1}$. Specifically, the element $(\mathbf{L}^-)^{n/2}_{n,1} = \binom{n-2}{n/2-1}$, grows approximately as $\sim 2^n/\sqrt{n}$, up to constant factors. This can easily lead to overflow, especially in low-precision formats. However, as long as the algorithm is restricted to very small (constant-sized) matrices, the numerical errors can be maintained to an acceptable level. While this algorithm should be generally avoided, it can be used (carefully) as a subroutine for handling the inverses of small diagonal blocks during the execution of more stable algorithms.

3.3 A log-stable matrix recursive algorithm (MBH)

In this section, we revisit a recursive triangular inversion algorithm, which we call MBH, since (to our knowledge) the first analysis is due to Bunch and Hopcroft [2]. In [7] it was proved that the algorithm is $\mathrm{polylog}(n)$-stable. The algorithm is based on the following observation:

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{A}_{22} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}_{11}^{-1} & -\mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ 0 & \mathbf{A}_{22}^{-1} \end{pmatrix}. \tag{5}$$

where $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ are square matrices. At each step, it involves computing the inverses of two matrices of half the size, and two matrix products of half the size. In practice, when the individual sub-blocks become very small, it can become a waste of resources to launch a separate matrix product for each tiny block. Instead, it is preferable to use an "unrolled" recursion, in order to group together small matrix products that occur at the lowest levels of the recursion. The stability analysis of [7], together with Corollary 2.1, it implies a bound on the number of precision bits.

Proposition 3.1. *Fix $\epsilon \in (0, 1)$. For some absolute constant $c > 0$, it is sufficient to use $t = \log(1/\epsilon) + O(\log^c(n))$ bits of precision to invert a well-conditioned matrix $\mathbf{A}$ that satisfies Corollary 2.1 (e.g. DeltaNet), using Alg. 4, such that the returned solution $\widetilde{\mathbf{A}}^{-1}$ satisfies $\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq \epsilon \|\mathbf{A}^{-1}\|_2$.*

Complexity (MBH): $\approx 2\log(n)$ matrix products of size $n \times n$ (proof in Appendix C.3).

3.4 A mixed recursive algorithm (MXR)

One issue of the recursive algorithm of the previous section is that, for "tiny" block-sizes, it needs to execute two full $n \times n$ matrix multiplication per iteration, while most of the elements in these cases are almost zero, since the matrices are almost diagonal. To overcome this for practical purposes, we propose to use the unstable algorithm of Section 3.2 only for a few initial iterations, e.g., for block-sizes up to 16×16, in order to compute the small diagonal inverses as a starting point, and then continue the log-stable iteration for the later steps. This increases the hardware utilization, and, since the unstable algorithm is only executed for constant-sized blocks, the full algorithm in its entirety remains logarithmically stable (albeit, the "constant errors" might be large, and should still be treated with care). An example of this approach, which we later show to be extremely efficient in practice, is outlined below (see also Algorithm 2):

1. Use the algorithm of Section 3.2 in order to compute the small inverses of the diagonal blocks of $\mathbf{A}$, where each block has size $C \times C$ for some constant C (e.g., $C \in \{16, 32\}$).
2. Use the unrolled recursive algorithm of Section 3.3 to compute the final inverse.

Complexity (MXR): $\approx 2\log(n)$ matrix products of size $n \times n$.

3.5 Iterative inversion with Newton-Schulz

All aforementioned algorithms so far are *direct*, in a sense that, in exact arithmetic, they return the exact inverse after a finite number of steps. In this section we revisit the Newton-Schulz iteration [28], which, as Higham highlights in [14], "...is attractive because it involves only matrix multiplication". Given a square matrix $\mathbf{A}$, and an initial guess $\mathbf{X}_0$ for the inverse of $\mathbf{A}$, the method iteratively updates:

$$\mathbf{X}_{k+1} \leftarrow \mathbf{X}_k(2\mathbf{I} - \mathbf{A}\mathbf{X}_k), \quad k = 0, 1, 2, \ldots \tag{6}$$

This iteration has been studied in depth for computing the inverse and/or pseudo-inverse of general matrices [23, 24, 33, 45], but we are not aware of a dedicated analysis for unit-diagonal triangular matrices. Nevertheless, we can obtain quite strong convergence and stability guarantees from existing analyses. In [23], Pan and Reif provide explicit bounds on the convergence and number of iterations.

With a straightforward error analysis, one can show that a single iteration of Newton-Schulz is numerically stable. However, carrying over the analysis to multiple iterations to achieve end-to-end numerical stability for the inverse, as in Definition 2.1, is quite tedious. At this point we can refer to the recent and very rigorous analysis of Shah [30], who showed end-to-end stability guarantees of the Newton-Schulz iteration for the sign function, and proved that, indeed, the method is stable for that task. Given the existing analyses, there is no reason to suspect that Eq. (6) is unstable. However, to our knowledge, a formal proof remains an interesting open question. The next proposition directly follows from the analysis in [23] and the stability assumption. The proof is located in Appendix C.4

Proposition 3.2 (Follows from [23])**.** *Fix $\epsilon > 0$. Assume that the Newton-Schulz method is stable, in a sense that, after k iterations, the returned matrix $\widetilde{\mathbf{A}}^{-1}$ satisfies $\|\widehat{\mathbf{A}}^{-1} - \widetilde{\mathbf{A}}^{-1}\|_2 \leq \mathbf{u} \cdot (n \cdot \kappa(\mathbf{A}))^{O(1)} \cdot \|\mathbf{A}^{-1}\|_2$, where $\widehat{\mathbf{A}}^{-1}$ is the true matrix after k iterations in exact arithmetic. Then, for $\mathrm{poly}(n)$-conditioned triangular matrices, after $O(\log(n))$ iterations $\widetilde{\mathbf{A}}^{-1}$ satisfies $\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq \epsilon\|\mathbf{A}^{-1}\|_2$ using $t = \log(1/\epsilon) + O(\log(n))$ bits of precision, using $\mathbf{X}_0 = \mathbf{I}/n^{O(1)}$ as an initial guess.*

Complexity (NS): $2 \times N_{iter}$ matrix products of size $n \times n$ (typically $N_{iter} \approx 2\log(n)$).

3.6 Iterative refinement

A standard technique to improve the accuracy of numerical algorithms for solving linear systems of equations (especially in floating point arithmetic) is iterative refinement [13, 22, 39]. In the context of this work, we use the following iterative scheme. If $\widetilde{\mathbf{A}}_0^{-1}$ is the approximate inverse returned by an algorithm, then for $k = 0, 1, \ldots$ we iteratively update:

$$(i)\ \mathbf{R}_k \leftarrow \mathbf{I} - \widetilde{\mathbf{A}}_k^{-1}\mathbf{A}, \qquad (ii)\ \widetilde{\mathbf{A}}_{k+1}^{-1} \leftarrow \widetilde{\mathbf{A}}_k^{-1} + \mathbf{R}_k\widetilde{\mathbf{A}}_k^{-1}. \tag{7}$$

As it is shown later in the numerical experiments, this simple iterative scheme, which involves only two matrix products per iteration, can significantly increase the final accuracy in some cases, even with a single iteration.

4 Evaluating the numerical accuracy of the methods

In this section we evaluate experimentally the numerical accuracy of all the aforementioned methods.

4.1 Accuracy of the computed inverse

We first assess the accuracy of each method in terms of the posterior errors of the returned inverse compared to the true inverse of the input matrix. For the experimental results illustrated in Figure 2, the input matrix is equal to $\mathbf{A} = \mathbf{I} + \mathbf{L}$, where $\mathbf{L} := \texttt{strict_tril}(\mathbf{K}\mathbf{K}^\top)$, and each row of $\mathbf{K}$ is a random vector with i.i.d. uniform elements from $[0, 1]$, which is then rescaled to have unit length.

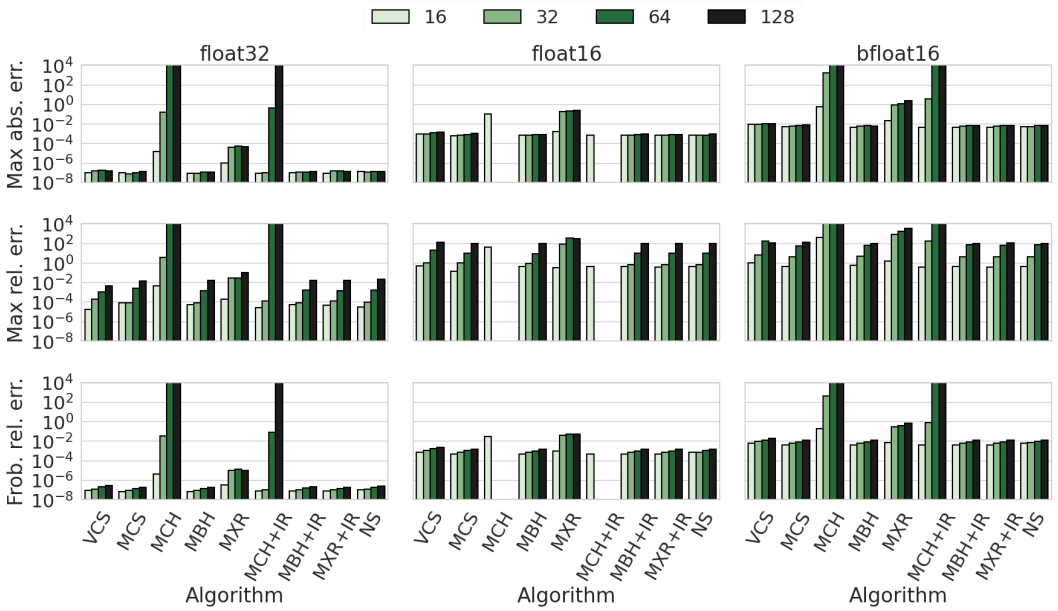

Figure 2: Comparison of the inverse accuracy of various methods, for three different input precision formats, and four matrix sizes.

The "true inverse" $\mathbf{A}^{-1}$ is obtained computationally by calling the inverse functions provided by PyTorch in double-precision, which internally calls BLAS/LAPACK routines that are highly accurate. To have a broad overview, we measure three different types of forward errors:

$$\text{Max abs.: } \max_{i,j} |\mathbf{A}^{-1}_{i,j} - \widetilde{\mathbf{A}}^{-1}_{i,j}|, \quad \text{Max rel.: } \max_{i,j \le i} \frac{|\mathbf{A}^{-1}_{i,j} - \widetilde{\mathbf{A}}^{-1}_{i,j}|}{|\mathbf{A}^{-1}_{i,j}|}, \quad \text{Frob.-norm relative: } \frac{\|\mathbf{A}^{-1} - \widetilde{\mathbf{A}}^{-1}\|_F}{\|\mathbf{A}^{-1}\|_F}.$$

We report results for three different types of numerical precision for the inputs: float16 ("half precision"), bfloat16 ("brain float"), and float32 ("single precision"). Internally in the algorithms, matrix products are always executed in float32 precision, even if they receive the input matrices in lower precision; this is standard in modern hardware, since every matrix product might need more bits to store the final result.

In single precision, all methods except MCH and MXR return very accurate results, where the relative errors are close to machine precision. MXR (without refinement) achieves about two to three fewer digits of accuracy than the other methods on average, but this is remediated with a single step of iterative refinement. The unstable algorithm, MCH, returns completely wrong results for sizes 64 and 128, but the errors are within acceptable range for size 16, and barely acceptable for size 32.

In the half-precision case, float16, the errors of all methods increase substantially. The errors are no longer within the ideal machine precision error range, but they are still within an acceptable threshold, achieving about three-to-four digits of accuracy on average. The unstable algorithm returns NaN values for all sizes above 16, while, again, MXR achieves decent accuracy, which is on-par with the other methods after a single step of iterative refinement.

The bfloat16 format slightly "improves" the accuracy of the unstable MCH algorithm: the returned values are no longer NaN, which is expected since bfloat16 has a larger exponent range than float16, and can help with extremely large values. However, the errors of MCH remain unacceptably large, and all the returned digits are completely wrong. At the same time, all other methods return higher errors in bfloat16 than in float16, since bfloat16 uses fewer bits for the mantissa. With these observations, float16 should be preferred over bfloat16 for inverting triangular matrices in such models.

4.2 The effect of numerical accuracy in end-to-end benchmarks

The motivation for our numerical accuracy study stems from Fig. 1 (right), which shows that an unstable algorithm such as MCH causes the model accuracy to collapse to zero. This is likely due to the emergence of NaNs during computation, which in turn disrupt the GDN algorithm. In Fig. 7 we show that all the stable algorithms maintain end-to-end accuracy for three different models. Surprisingly also the MXR keeps the same accuracy as more stable algorithms, making it a good candidate for fast and stable inverse method for GDNs. The end-to-end accuracy is obtained by modifying the current implementation of the inverse on SGLang [46] to use our algorithms, and then by running the MMLU [12] benchmark (MIT Licence) over 10 different subjects. The kernels of SGLang that we use as baseline can be found online [29] and such baseline is labeled in Fig. 1 (left) as `Triton`. We keep the same naming throughout the whole paper.

We next investigate numerical accuracy in more detail by analyzing how errors in computing the triangular inverse affect end-to-end model performance. We employ the Newton–Schulz method and evaluate it at different iteration counts. For each checkpoint, we measure the relative error with respect to the double-precision inverse computed using NumPy, as well as the average accuracy of several models on MMLU.

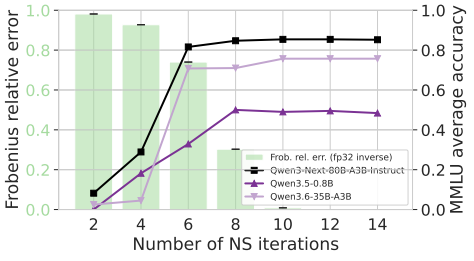

Figure 3: Number of Newton-Schulz iterations (x-axis) versus matrix inversion error using fp32 (left y-axis) and MMLU average accuracy (right y-axis). Chunk size $C = 64$.

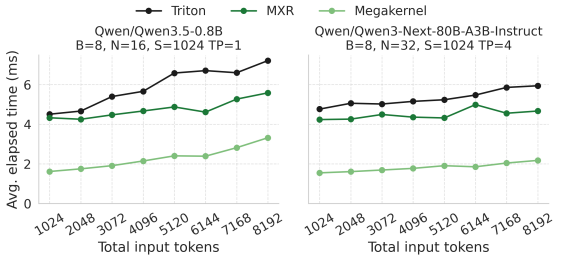

Figure 4: GDN layer runtime on device over increasing input tokens. Qwen3.5-0.8B is served using one NPU (TP=1) and Qwen3-Next-80B-Instruct is served using four NPUs (TP=4).

Interestingly, as seen in Fig. 3, even after only six iterations, the model already starts to perform relatively well, with 81.6% accuracy for Qwen3-Next-80B-A3B-Instruct. While this is an impressive result considering the relative errors, the accuracy drops are still significant: 4.9% (Qwen3.6-35B-A3B), 16.6% (Qwen3.5-0.8B), and 3.8% (Qwen3-Next-80B-A3B-Instruct) compared to the 12-iteration setting, which ensures numerical stability. This confirms the importance of our study in terms of numerical analysis.

We further observe that increasing the number of iterations beyond the optimal value (e.g., 14 iterations) does not have any noticeable effect on end-to-end performance.

5 Performance evaluation and optimization strategies

In this section we report some insights on the performance of the studied algorithms on NPUs, which have recently received significant attention especially for inference tasks. The goal is to understand to which extent the proposed algorithms can efficiently utilize the underlying hardware of modern AI accelerators, and how competitive they are with well-established implementations (Triton, TileLang [38]) on major LLM-serving frameworks.

The NPU devices used in our experimentation were Ascend AI accelerators (910B2), containing 48 SIMD units and 24 matrix multiplication cores capable of performing square matrix multiplication of

size up to 128. The theoretical memory bandwidth is 1.2 TB/s. The host is an ARM-based CPU with 48 cores. Major LLM serving frameworks, in particular, SGLang and vLLM, officially support NPUs and provide optimized inverse primitives that we use as baselines.

5.1 Scalability of individual inverse kernels

At the kernel level, we measure the execution time of the algorithms in isolation. We run 5 warm-up iterations followed by 20 measured iterations, and clear the data cache before each iteration. We set batch size 32, 4 attention heads, and varying sequence length up to $16K$. In Fig. 5, we compare our algorithms against three baselines. The Pytorch Eager reference is a basic python implementation based on the column-sweep algorithm. The Triton implementation is the official implementation for SGLang on NPUs [29] and uses a variation of the MBH algorithm. The Tilelang implementation is another kernel that is provided by the Tilelang framework for NPUs [35] as an example of optimized algorithm. They also used a similar approach to Triton's, but in a different framework, and by materializing some dimensions of the input. Our best algorithm, MXR, achieves up to **4.3**× speedup

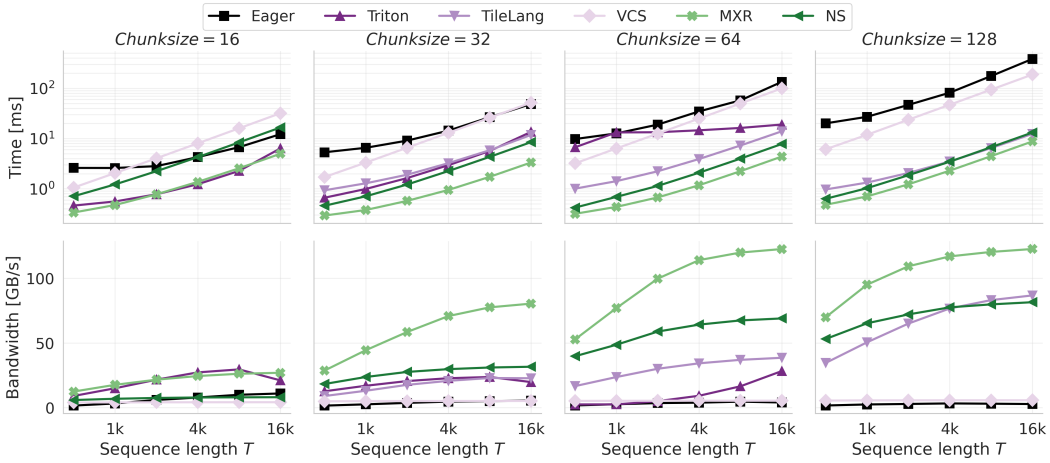

Figure 5: Kernel evaluation of different inverse algorithms. $B = 32$, $N = 4$ and varying sequence length up to $16K$.

over the official implementation and a 3× speedup over TileLang. This confirms that MXR is the best among the proposed candidates as a fast and stable triangular inverse algorithm for GDNs on NPUs.

5.2 Full-layer performance comparison

To see how our kernels perform at the layer level we embed our best performing kernel, MXR, into the GDN layer implemented in [29]. As it is well-known, however, improvements at the kernel level can easily get "absorbed" in end-to-end benchmarks by other, unoptimized kernels, due to Amdahl's law. To that end, we further improved the GDN layer by optimizing the other five kernels of the GDN layer, and combining them into a "MegaKernel", which also supports chunk size 128.

In Fig. 4 we evaluate the performance of these two approaches and compare it to the existing implementation (Triton). We provide the results for two common configurations of the GDN layer when running Qwen models on NPUs: a small model run on one NPU (Qwen3.5-0.8B) and a big model run on 4 NPUs with tensor parallelism (Qwen3-Next-80B). The figure illustrates the average runtime as the input size (number of tokens) increases. In the small model configuration we obtain up to 1.29× speedup with the inverse kernel substitution and 2.18× speedup with the MegaKernel. With the big model configuration we obtain up to 1.27× speedup with the kernel substitution and 2.74× speedup with the MegaKernel.

5.3 End-to-end inference performance evaluation

In this section we provide some end-to-end performance results, regarding the effect of our implementations in the entire inference level. We evaluated the following two models served via the `sglang`

framework: Qwen/Qwen3.5-0.8B-Base (served on 1 NPU), Qwen/Qwen3.6-35B-A3B (served on 2 NPUs). We used the `sglang.bench_one_batch` functionality, which serves the model and executes queries to report latency and throughput during the prefill and decode phase. Three different variations were tested for each model:

- default: The default GDN backend of `sglang`.
- MXR: As above, but replacing the `solve_tril` method with our implementation.
- MegaKernel: Replacing the entire GDN layer with our MegaKernel.

In the following table we report the median as well as the 25th and 75th percentile of the prefill throughput in tokens-per-second (TPS, the higher values the better) that was measured by `sglang`.

model	backend	median TPS	25% TPS	75% TPS
Qwen3.5-0.8B-Base	default	18169	18012	18344
Qwen3.5-0.8B-Base	MXR	18924	18675	19060
Qwen3.5-0.8B-Base	MegaKernel	20216	20045	20700
Qwen3.6-35B-A3B	default	8790	8691	8849
Qwen3.6-35B-A3B	MXR	8906	8837	9026
Qwen3.6-35B-A3B	MegaKernel	9814	9434	9879

For the Qwen3.5-0.8B-Base model, MXR achieves a $\approx 4\%$ prefill median throughput increase, while the MegaKernel achieves a $\approx 11\%$ increase. For the Qwen3.6-35B-A3B model, the MXR backend achieves a smaller increase (about $\approx 1.3\%$, but still outside the statistical error), while the MegaKernel demonstrates again around $\approx 11\%$ increase in the median prefill throughput.

6 Conclusion

In this work we revisited triangular inversion algorithms in the context of Linear Attention with the Delta-Rule. We studied both direct and iterative methods, focusing on algorithms that are rich in matrix products and have the potential to efficiently utilize modern hardware that supports such operations. For each method, a detailed numerical stability and complexity analysis was provided, highlighting their effectiveness and their robustness against numerical errors. Besides the theoretical properties, we provided detailed experimental evaluations, covering a wide-spectrum of properties. These included numerical accuracy benchmarks, both at the kernel-level, in different floating point formats, and at the end-to-end model accuracy level. Moreover, we implemented and empirically evaluated the performance of the methods on NPUs. At the kernel level, our fastest and stable implementation achieved up to $4.3\times$ speedup against optimized baselines, and up to $2.74\times$ speedup at the entire model-layer level, when optimized as a whole. We hope that these findings will serve as guidance for fast and robust AI kernel development, with high performance, and, importantly, reliable numerical results.

Code availability

The source code for the kernels and the scripts to reproduce the experimental evaluations are available in the following links:

- https://github.com/huawei-csl/gdn-tri-inverse
- https://github.com/huawei-csl/pto-kernels
- https://github.com/huawei-csl/megagdn-pto

Acknowledgements

We thank Efstratios Gallopoulos for helpful literature recommendations, and for suggesting the term "abominable" to characterize the condition number of random triangular matrices. We also thank the anonymous referees and area chairs of NeurIPS 2026 for their helpful feedback and constructive criticism.

References

[1] Christian Bischof and Charles Van Loan. The WY Representation for Products of Householder Matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987.

[2] James R. Bunch and John E. Hopcroft. Triangular Factorization and Inversion by Fast Matrix Multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[3] Rezaul Chowdhury, Francesco Silvestri, and Flavio Vella. A Computational Model for Tensor Core Units. In *Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 519–521, 2020.

[4] Rezaul Chowdhury, Francesco Silvestri, and Flavio Vella. Algorithm Design for Tensor Units. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 353–367. Springer, 2021.

[5] Bruno Codenotti, Mauro Leoncini, and Franco P. Preparata. The Role of Arithmetic in Fast Parallel Matrix Inversion. *Algorithmica*, 30(4):685–707, 2001.

[6] Tri Dao and Albert Gu. Transformers are SSMs: generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*. JMLR, 2024.

[7] James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, October 2007.

[8] Jeremy J. Du Croz and Nicholas J. Higham. Stability of Methods for Matrix Inversion. *IMA Journal of Numerical Analysis*, 12(1):1–19, 1992.

[9] Efstratios Gallopoulos, Bernard Philippe, and Ahmed H. Sameh. *Parallelism in Matrix Computations*. Springer Netherlands, 2016.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013.

[11] Riccardo Grazzi, Julien Siems, Jörg K.H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues. In *International Conference on Learning Representations (ICLR)*, 2025.

[12] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations (ICLR)*, 2021.

[13] Nicholas J. Higham. Iterative refinement for linear systems and LAPACK. *IMA Journal of Numerical Analysis*, 17(4):495–509, 1997.

[14] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, January 2002.

[15] Nicholas J. Higham. Seven Sins of Numerical Linear Algebra. Available online: https://nhigham.com/2022/10/11/seven-sins-of-numerical-linear-algebra/, 2022.

[16] Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Improving Bilinear RNN with Closed-loop Control. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[17] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer Quality in Linear Time. In *International Conference on Machine Learning (ICML)*, pages 9099–9117. PMLR, 2022.

[18] Su Jianlin. The elements of DeltaNet's core inverse matrix are always within [-1, 1]. Available online: https://spaces.ac.cn/archives/11563, Jan 2026.

[19] Thierry Joffrain, Tze Meng Low, Enrique S Quintana-Ortí, Robert van de Geijn, and Field G Van Zee. Accumulating Householder transformations, revisited. *ACM Transactions on Mathematical Software*, 32(2):169–179, 2006.

[20] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *International Conference on Machine Learning (ICML)*, pages 5156–5165. PMLR, 2020.

[21] Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture. *arXiv*, 2025.

[22] Cleve B. Moler. Iterative refinement in floating point. *Journal of the ACM*, 14(2):316–321, 1967.

[23] Victor Pan and John Reif. Efficient parallel solution of linear systems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 143–152, 1985.

[24] Victor Pan and Robert Schreiber. An Improved Newton Iteration for the Generalized Inverse of a Matrix, with Applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1130, 1991.

[25] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with Expressive Dynamic State Evolution. In *Second Conference on Language Modeling*, 2025.

[26] Ahmed H. Sameh and Richard P. Brent. Solving triangular systems on a parallel computer. *SIAM Journal on Numerical Analysis*, 14(6):1101–1113, December 1977.

[27] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers. In Marina Meila and Tong Zhang, editors, *International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 9355–9366. PMLR, 18–24 Jul 2021.

[28] Günther Schulz. Iterative berechung der reziproken matrix. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, January 1933.

[29] SGLang Project Contributors. SGLang-Kernel-NPU: SGLang Kernel Library for Ascend NPU. https://github.com/sgl-project/sgl-kernel-npu, 2025. Accessed: 2026-05-06.

[30] Rikhav Shah. Hermitian diagonalization in linear precision. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5599–5615. SIAM, 2025.

[31] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[32] Aleksandros Sobczyk, Giuseppe Sorrentino, and Anastasios Zouzias. Segmented Operations using Matrix Multiplications. *arXiv*, 2025.

[33] Torsten Söderström and Gilbert W. Stewart. On the Numerical Properties of an Iterative Method for Computing the Moore–Penrose Generalized Inverse. *SIAM Journal on Numerical Analysis*, 11(1):61–74, March 1974.

[34] Kimi Team. Kimi k3: Open frontier intelligence. *arXiv*, 2026.

[35] TileLang Contributors. TileLang-Ascend: Ascend TileLang Adapter. https://github.com/tile-ai/tilelang-ascend, 2025. Accessed: 2026-05-06.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[37] Divakar Viswanath and Lloyd Nicholas Trefethen. Condition Numbers of Random Triangular Matrices. *SIAM Journal on Matrix Analysis and Applications*, 19(2):564–581, 1998.

[38] Lei Wang, Yu Cheng, Yining Shi, Zhiwen Mo, Zhengju Tang, Wenhao Xie, Tong Wu, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, and Zhi Yang. TileLang: Bridge Programmability and Performance in Modern Neural Kernels. In *International Conference on Learning Representations (ICLR)*, 2026.

[39] James H. Wilkinson. *Progress report on the automatic computing engine*. Mathematics Division, Department of Scientific & Industrial Research, 1948.

[40] James H. Wilkinson. Error Analysis of Direct Methods of Matrix Inversion. *Journal of the ACM*, 8(3):281–330, 1961.

[41] Mingyu Xu, Tenglong Ao, Jiaao He, Jianqiao Lu, Guang Shi, and Shu Zhong. DeltaFormer: Unlock the state space of Transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[42] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations (ICLR)*, volume 2025, pages 29687–29707, 2025.

[43] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning (ICML)*. JMLR, 2024.

[44] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 115491–115522. Curran Associates, Inc., 2024.

[45] Malik Zaka Ullah, Fazlollah Soleymani, and Abdulrahman S. Al-Fhaid. An efficient matrix iteration for computing weighted Moore–Penrose inverse. *Applied Mathematics and Computation*, 226:441–454, January 2014.

[46] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[47] Shu Zhong, Mingyu Xu, Tenglong Ao, and Guang Shi. Understanding transformer from the perspective of associative memory. *arXiv*, 2025.

[48] Anastasios Zouzias and William F. McColl. A Parallel Scan Algorithm in the Tensor Core Unit Model. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 489–502, 2023.

A Chunkwise parallel form and triangular inverse derivation

Here we provide more details for the chunkwise parallel form. As mentioned in the introduction, the inputs are three matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L \times d}$, where L is the sequence length and the rows of $\mathbf{K}$ are normalized to have unit length. In linear attention, after preprocessing the matrices $\mathbf{Q}$ and $\mathbf{K}$ with some suitable functions (e.g. ELU), the t-th output, for $t = 1, \dots, L$, is given by Eq. (2), which can be also written in the matrix form of Eq. (1), which we repeat here for readability:

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V} \text{ (matrix form)}, \quad \mathbf{o}_t = \left(\sum_{j=1}^{t} \mathbf{v}_j \mathbf{k}_j^\top\right) \mathbf{q}_t = \mathbf{V}_{1:t,:}^\top \mathbf{K}_{1:t,:} \mathbf{q}_t \text{ (linear recurrence)}.$$

The matrix $\mathbf{S}_t = \mathbf{V}_{1:t,:}^\top \mathbf{K}_{1:t,:}$ is often referred to as the *state matrix*, and the linear recurrence formulation can be rewritten as a state update:

$$\mathbf{o}_t = \mathbf{S}_{t-1}\mathbf{q}_t + \mathbf{v}_t \mathbf{k}_t^\top \mathbf{q}_t. \tag{8}$$

Chunkwise processing. When the processing is performed chunkwise, the sequence of length L is partitioned into $\lceil L/C \rceil$ "chunks" of length C each. This gives rise to individual query-key-value matrices $\mathbf{\Xi}_{[k]} := \mathbf{\Xi}_{kC:(k+1)C,:} \in \mathbb{R}^{C \times d}$, for $k = 0, \dots, \lceil L/C \rceil - 1$, where $\mathbf{\Xi}$ can be any of $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$. The corresponding state matrix is defined as:

$$\mathbf{S}_{[k]} := \mathbf{V}_{[k]}^\top \mathbf{K}_{[k]} \in \mathbb{R}^{d \times d}. \tag{9}$$

To simplify the notation, hereafter we focus on a single chunk.

Delta rule and gating. The "Delta rule" [27] generalizes the state update definition by introducing a scalar β_t (a learning rate):

$$\begin{aligned} \mathbf{S}_t^{\text{DeltaNet}} &= \mathbf{S}_{t-1} + \beta_t(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t)\mathbf{k}_t^\top \\ &= \mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top\right) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top. \end{aligned} \tag{10}$$

For the rest of our analysis, we use the standard assumption that $\beta_t \in (0, 1)$. We highlight, however, that $\beta_t \in (0, 2)$ has been carefully studied in the literature and provides novel insights, such as state-tracking capabilities [11, 31]. Our analysis can be adapted to also cover this case, as well as the DeltaProduct generalization introduced in [31], provided the underlying key vectors are unit length (which is indeed the case in [31]).

GDNs further generalize DeltaNets by introducing a weight-decay factor $a_t \in [0, 1]$:

$$\mathbf{S}_t^{\text{GDN}} = a_t \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top. \tag{11}$$

This scalar decay can be further extended to a diagonal matrix form, as introduced in Kimi Delta Attention (KDA) [21]:

$$\mathbf{S}_t^{\text{KDA}} = \mathbf{S}_{t-1} \operatorname{diag}(\mathbf{a}_t)(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top. \tag{12}$$

The KDA update is the most general one, since it covers GDN when all the elements of $\mathbf{a}_t$ are the same, and also DeltaNet reduces to it when $\operatorname{diag}(\mathbf{a}_t)$ is the identity matrix. We will therefore focus on the KDA update for the rest of this section.

By expanding the state update formulation, we have that:

$$\mathbf{S}_t^{\text{KDA}} = \mathbf{S}_0 \prod_{i=1}^{t} \operatorname{diag}(\mathbf{a}_i)(\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top) + \sum_{i=1}^{t} \beta_i \mathbf{v}_i \mathbf{k}_i^\top \prod_{j=i+1}^{t} \operatorname{diag}(\mathbf{a}_j)(\mathbf{I} - \beta_j \mathbf{k}_j \mathbf{k}_j^\top), \tag{13}$$

where $\prod_{i=1}^{t} \operatorname{diag}(\mathbf{a}_i)(\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top) := \operatorname{diag}(\mathbf{a}_1)(\mathbf{I} - \beta_1 \mathbf{k}_1 \mathbf{k}_1^\top) \cdot \operatorname{diag}(\mathbf{a}_2)(\mathbf{I} - \beta_2 \mathbf{k}_2 \mathbf{k}_2^\top) \dots$, i.e., the non-commutative product is defined in ascending order left-to-right.

In [21, 43, 44], the authors used classic tools from Numerical Linear Algebra to reformulate the state update in a parallelizable, matrix-multiplication friendly manner. Specifically, the main techniques used trace back to the so-called WY and UT transforms, which provide a way to factorize products of

generalized Householder transformations [1, 19]. Briefly, the UT transform states that the product of Householder matrices admits a factorization of the form:

$$(\mathbf{I} - \beta_1 \mathbf{u}_1 \mathbf{u}_1^\top) \cdot (\mathbf{I} - \beta_2 \mathbf{u}_2 \mathbf{u}_2^\top) \cdot \ldots \cdot (\mathbf{I} - \beta_k \mathbf{u}_k \mathbf{u}_k^\top) = \mathbf{I} - \mathbf{U}_k \mathbf{T}_k^{-1} \mathbf{U}_k^\top,$$

where $\mathbf{U} = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \ldots \quad \mathbf{u}_k)$ has k columns, and $\mathbf{T}_k$ is a $k \times k$ upper triangular matrix.

After some algebra (see e.g. Section 3 of [21]), it can be shown that the state update reduces to few matrix products, notably, involving the inverse of the matrix $\mathbf{A}$ defined as:

$$\mathbf{A}^{-1} := \left(\mathbf{I} + \mathrm{diag}(\mathbf{b}) \cdot (\mathbf{\Gamma} \odot \mathbf{K}) \cdot (\mathbf{K} \oslash \mathbf{\Gamma})^\top \odot \mathbf{L}^-\right)^{-1},$$

where $\mathbf{\Gamma}$ is a matrix whose i-th row is equal to $\boldsymbol{\gamma}_i := \mathbf{a}_1 \odot \mathbf{a}_2 \odot \ldots \odot \mathbf{a}_i$.

B Floating point model

Floating point error analysis becomes more and more important in AI computing, especially with the recent trend of reducing the bits of precision in floating point formats. For the purposes of this note, we recall some basic definitions, and refer to the classic textbook of Higham [14] for further details.

In the standard floating point model, the floating point representation $\mathbf{fl}(x)$ of a real number x is a $(1 + p + t)$-bit number:

$$\mathbf{fl}(x) = \pm 2^e \times \left(\tfrac{m_1}{2} + \tfrac{m_2}{2^2} + \cdots + \tfrac{m_t}{2^t}\right), \tag{14}$$

where one bit stores the sign of the number $\pm$, p bits are used to store the (integral) exponent e in the so-called "biased" format, and t bits $m_1, m_2, \ldots, m_t$ are used for the *significand* (also known as the *mantissa*). For all *normalized* numbers it holds that:

$$\mathbf{fl}(x) = (1 + \theta)x, \quad |\theta| \leq 2^{-t} := \mathbf{u}, \tag{15}$$

where $\mathbf{u}$ is referred to as the *machine precision*. Normalized numbers are all numbers that can be represented within the given exponent range. Given two (normalized) floating point numbers a, b, floating point operations $\circ \in \{+, -, \times, /\}$ and square roots are assumed to satisfy:

$$\mathbf{fl}(a \circ b) = (1 + \theta)(a \circ b), \qquad \text{and} \qquad fl\left(\sqrt{a}\right) = (1 + \theta)\sqrt{a}. \tag{16}$$

C Deferred proofs

C.1 Proof of Corollary 2.1

Proof. It holds that $\max_{i,j} |\mathbf{A}_{i,j}| \leq 1$, and from [18] we know that $\max_{i,j} |\mathbf{A}^{-1}_{i,j}| \leq 1$. For any matrix $\mathbf{M}$, it holds that $\|\mathbf{M}\|_2 \leq n \cdot \max_{i,j} |\mathbf{M}_{i,j}|$, which implies that $\|\mathbf{A}\|_2 \leq n$ and $\left\|\mathbf{A}^{-1}\right\|_2 \leq n$, and thus $\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2 \leq n^2$. □

C.2 Proof of Lemma 3.1

Proof. [26] proved that, for any vector $\mathbf{b}$, the returned solution $\widetilde{\mathbf{x}}$ for the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ satisfies $(\mathbf{A} + \mathbf{E})\widetilde{\mathbf{x}} = \mathbf{b}$, where $\|\mathbf{E}\|_\infty \leq C \cdot n^2 \log(n) \cdot \kappa^2(\mathbf{A}) \cdot \|\mathbf{A}\|_\infty \cdot \mathbf{u}$. Since we apply the algorithm for every column of the identity as $\mathbf{b}$, we can use [14, Theorem 7.4], which implies that $\left\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\right\|_\infty \leq \xi \cdot \|\mathbf{A}^{-1}\|_\infty$, where $\xi := \frac{|\mathbf{A}^{-1}\|_\infty \|\mathbf{E}\|_\infty}{1 - \|\mathbf{A}^{-1}\|_\infty \|\mathbf{E}\|_\infty}$. Using standard norm inequalities we can finally argue that

$$\begin{aligned}
\left\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\right\|_2 &\leq \sqrt{n} \left\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\right\|_\infty \\
&\leq \sqrt{n} \frac{\|\mathbf{A}^{-1}\|_\infty \|\mathbf{E}\|_\infty}{1 - \|\mathbf{A}^{-1}\|_\infty \|\mathbf{E}\|_\infty} \cdot \|\mathbf{A}^{-1}\|_\infty \\
&\leq C' \cdot \|\mathbf{A}^{-1}\|_\infty \cdot n^{2.5} \log(n) \cdot \kappa^2(\mathbf{A}) \cdot \|\mathbf{A}\|_\infty \cdot \mathbf{u} \cdot \|\mathbf{A}^{-1}\|_\infty \\
&\leq C' \cdot n^{3.5} \log(n) \cdot \kappa^3(\mathbf{A}) \cdot \mathbf{u} \cdot \|\mathbf{A}^{-1}\|_2,
\end{aligned}$$

where the third step requires the assumption that the machine precision is sufficiently small, so that following bound needs to be satisfied: $\|\mathbf{A}^{-1}\|_\infty \|\mathbf{E}\|_\infty \ll 1$. □

C.3 Proof of MBH complexity

Lemma C.1. *Let $\mathbf{A} = \mathbf{I} + \mathbf{L}$ be an $n \times n$ matrix where $\mathbf{L}$ is strictly lower triangular. We can compute the inverse of $\mathbf{A}$ using $2\log(n)$ matrix products of size $n \times n$, by unrolling the algorithm of [2].*

Proof. Assume for simplicity that n is a power of 2, and let $b \in \{2, 4, 8, \ldots, n/2\}$. We define the matrix

$$\mathbf{D}(\mathbf{A}, b) = \begin{pmatrix} \mathbf{A}_{0,0} & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{A}_{1,1} & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & \mathbf{A}_{n/b-1,n/b-1} \end{pmatrix}. \tag{17}$$

Here, $\mathbf{D}(\mathbf{A}, b)$ contains only the diagonal blocks of the matrix $\mathbf{A}$. Each block $\mathbf{A}_{i,i}$ is $b \times b$, and there are n/b blocks in total. Let also $\mathbf{D}_e(\mathbf{A}, b)$ be the matrix containing the "even" diagonal blocks of $\mathbf{D}(\mathbf{A}, b)$, and $\mathbf{D}_o(\mathbf{A}, b)$ the odd blocks of $\mathbf{D}(\mathbf{A}, b)$. Clearly, $\mathbf{D}_o(\mathbf{A}, b) + \mathbf{D}_e(\mathbf{A}, b) = \mathbf{D}(\mathbf{A}, b)$. Now, we work as follows. First, we initialize $\mathbf{X} = \mathbf{I}$ and $b = 1$. Then, while $b < n$, we execute the following three steps:

1. Create $\mathbf{D}_e(\mathbf{X}, b)$ and $\mathbf{D}_o(\mathbf{X}, b)$ as defined above.
2. Update $\mathbf{X} \leftarrow \mathbf{D}_e(\mathbf{X}, b) + \mathbf{D}_o(\mathbf{X}, b) - \mathbf{D}_o(\mathbf{X}, b) \cdot \mathbf{L} \cdot \mathbf{D}_e(\mathbf{X}, b)$.
3. $b \leftarrow 2b$.

Finally, we return $\mathbf{X}$. There are $\log(n)$ iterations, and at each iteration two matrix products are performed. □

C.4 Proof of Proposition 3.2

Proof. The authors in [23] showed that, if $\mathbf{A}$ is invertible and $\mathbf{X}_0$ satisfies $\|\mathbf{I} - \mathbf{A}\mathbf{X}_0\| < q := 1 - 1/n^{O(1)}$, then, in exact arithmetic, after $O(\log(n))$ iterations, the Newton-Schulz method of Eq. (6) returns an approximate inverse $\widehat{\mathbf{A}}^{-1}$ which satisfies:

$$\left\| \widehat{\mathbf{A}}^{-1} - \mathbf{A}^{-1} \right\|_2 \leq 2^{-n^{O(1)}+1} \cdot \left\| \mathbf{A}^{-1} \right\|_2. \tag{18}$$

Now let $\mathbf{E} := \mathbf{X}_0 - \mathbf{A}^{-1}$. We have that $\|\mathbf{E}\|_2 = \|\mathbf{A}^{-1}\mathbf{A}\mathbf{X}_0 - \mathbf{A}^{-1}\|_2 \leq \|\mathbf{A}^{-1}\|_2 \cdot \|\mathbf{A}\mathbf{X}_0 - \mathbf{I}\|_2 \leq q\|\mathbf{A}^{-1}\|_2 < \|\mathbf{A}^{-1}\|_2$. Thus $\|\mathbf{X}_0\|_2 = \|\mathbf{A}^{-1} + \mathbf{E}\|_2 \leq (1+q)\|\mathbf{A}^{-1}\|_2 < 2\|\mathbf{A}^{-1}\|_2$. By the stability assumption, we have that the returned floating point matrix satisfies:

$$\|\widetilde{\mathbf{A}}^{-1} - \widehat{\mathbf{A}}^{-1}\|_2 \leq \mathbf{u} \cdot n^{O(1)} \cdot \|\mathbf{A}^{-1}\|_2.$$

Then the triangle inequality provides a coarse upper bound:

$$\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 = \|\widetilde{\mathbf{A}}^{-1} - \widehat{\mathbf{A}}^{-1} + \widehat{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq 2\mathbf{u} \cdot n^{O(1)} \cdot \|\mathbf{A}^{-1}\|_2 + 2^{-n^{O(1)}+1} \cdot \left\|\mathbf{A}^{-1}\right\|_2.$$

For reasonable values of n and for $\mathbf{u} \leq \epsilon/n^{O(1)}$, which translates to $\log(1/\epsilon) + O(\log(n))$ bits of precision, the previous inequality gives $\|\widetilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq \epsilon\|\mathbf{A}^{-1}\|_2$.

Note that for triangular matrices that satisfy Corollary 2.1, using $\mathbf{X}_0 = \mathbf{I}/n^{O(1)}$ is a simple and efficient initial guess that satisfies all the required properties. □

D Additional experiments and implementation details

D.1 Parallelization and asynchronous execution

To achieve the best possible performance, we highlight some high-level practices. First, note that, in existing implementations of linear transformers architectures, the triangular inversion part is inherently massively parallel. Typically, the input is given as a high-dimensional tensor with numerous small-sized chunks, which are all completely independent, and can be inverted without

communication. However, internally, within the inversion algorithms, there are data dependencies between each iteration (or recursion level, for the recursive ones). The effect of these dependencies can be mitigated to a minimal level by pipelining.

As an example, in the repeated-squaring algorithm (MCH), Step 1. depends on the corresponding $\mathbf{Y}$ matrix from the previous iteration, while Step 2. expects as an input the output of Step 1. However, the update of $\mathbf{Y}$ in iteration $k+1$ is independent of the update of $\mathbf{X}$ in iteration k, and therefore these two operations can be overlapped, if enough matrix cores are available. Similar observations can be made for the other algorithms as well. At the "I/O level", since individual chunks are independent, it is possible to overlap the inversion of a chunk while simultaneously loading the next chunk from the main (slow) memory to the local memory.

D.2 Stability analysis on different types of matrices

Here we add some additional experiments regarding the numerical accuracy of the methods on different classes of matrices. In Figure 6 we provide the same experiment as in Figure 2, where the strictly lower-triangular part of the input matrix is additionally scaled by a decay factor.

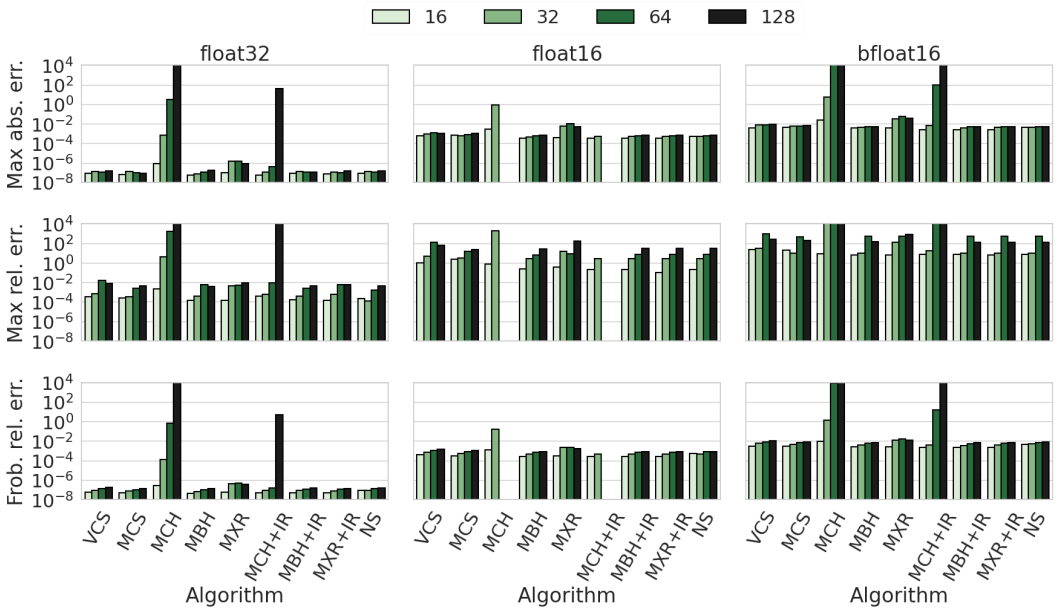

Figure 6: Comparison of the inverse accuracy of various methods, for three different input precision formats, and four matrix sizes.

D.3 MMLU Benchmarks

Figure 7 illustrates the average accuracy obtained on the MMLU benchmark, for 10 different subjects, for three different models with different inverse algorithms. All algorithms in Figure 7 are on-par with the baseline accuracy. This figure complements Figure 2, which already show-cased that the unstable algorithm MCH can lead to catastrophic accuracy drop, when used naively.

In Tab. 3 and 4 we report the full results of the MMLU benchmarks summarized in Fig. 7 and Fig. 3 respectively:

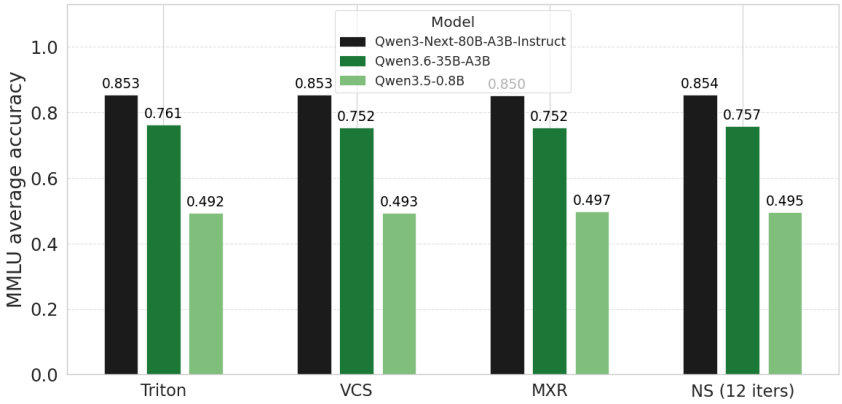

Figure 7: Comparison of the accuracy obtained on MMLU (10-subjects) of different models with different inverse algorithms.

Table 3: MMLU accuracy (10-subject subset) across inversion methods and Qwen models.

Model	Method	Avg.	*AbsAlg*	*Anatomy*	*Astron*	*BusEth*	*ClinKn*	*ColBio*	*ColChem*	*ColCS*	*ColMath*	*ColMed*
Qwen3-80B-A3B	Triton	0.853	0.800	0.807	0.934	0.810	0.894	0.972	0.670	0.850	0.780	0.861
	VCS	0.853	0.800	0.815	0.934	0.810	0.894	0.972	0.670	0.840	0.770	0.867
	MXR	0.850	0.800	0.800	0.934	0.800	0.894	0.965	0.670	0.850	0.760	0.861
	NS-12	0.854	0.800	0.807	0.941	0.810	0.891	0.972	0.680	0.860	0.770	0.861
Qwen3-35B-A3B	Triton	0.761	0.480	0.830	0.737	0.850	0.823	0.944	0.470	0.770	0.680	0.803
	VCS	0.752	0.460	0.830	0.737	0.830	0.811	0.931	0.460	0.760	0.700	0.780
	MXR	0.752	0.500	0.830	0.717	0.830	0.819	0.951	0.440	0.750	0.680	0.775
	NS-12	0.757	0.460	0.837	0.743	0.840	0.826	0.938	0.450	0.790	0.680	0.775
Qwen3.5-0.8B	Triton	0.492	0.320	0.467	0.500	0.510	0.581	0.618	0.470	0.420	0.350	0.491
	VCS	0.493	0.330	0.467	0.507	0.500	0.581	0.604	0.490	0.420	0.330	0.503
	MXR	0.497	0.330	0.452	0.507	0.540	0.600	0.590	0.490	0.440	0.310	0.503
	NS-12	0.495	0.330	0.467	0.493	0.530	0.589	0.604	0.490	0.430	0.330	0.491

Table 4: MMLU accuracy (10-subject subset) across Newton-Schulz iteration counts and Qwen models.

Model	Method	Avg.	*AbsAlg*	*Anatomy*	*Astron*	*BusEth*	*ClinKn*	*ColBio*	*ColChem*	*ColCS*	*ColMath*	*ColMed*
Qwen3-80B-A3B	NS-2	0.082	0.130	0.044	0.059	0.130	0.060	0.076	0.030	0.060	0.130	0.127
	NS-4	0.289	0.250	0.400	0.263	0.270	0.302	0.410	0.110	0.290	0.240	0.272
	NS-6	0.816	0.680	0.807	0.941	0.820	0.864	0.938	0.640	0.770	0.680	0.821
	NS-8	0.847	0.780	0.815	0.934	0.820	0.883	0.965	0.670	0.850	0.760	0.850
	NS-10	0.854	0.800	0.807	0.941	0.810	0.891	0.965	0.670	0.860	0.780	0.867
	NS-12	0.854	0.800	0.807	0.941	0.810	0.891	0.972	0.680	0.860	0.770	0.861
	NS-14	0.852	0.800	0.807	0.934	0.810	0.891	0.965	0.670	0.850	0.780	0.861
Qwen3-35B-A3B	NS-2	0.025	0.000	0.104	0.013	0.030	0.019	0.014	0.020	0.000	0.000	0.035
	NS-4	0.045	0.130	0.074	0.092	0.000	0.030	0.014	0.010	0.000	0.000	0.075
	NS-6	0.708	0.360	0.733	0.868	0.750	0.770	0.882	0.460	0.710	0.530	0.728
	NS-8	0.710	0.410	0.704	0.737	0.830	0.766	0.917	0.420	0.790	0.610	0.717
	NS-10	0.757	0.480	0.837	0.750	0.830	0.823	0.944	0.460	0.760	0.660	0.792
	NS-12	0.757	0.460	0.837	0.743	0.840	0.826	0.938	0.450	0.790	0.680	0.775
	NS-14	0.757	0.490	0.837	0.743	0.840	0.819	0.931	0.440	0.770	0.690	0.792
Qwen3.5-0.8B	NS-2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	NS-4	0.182	0.170	0.333	0.211	0.170	0.155	0.236	0.080	0.170	0.030	0.202
	NS-6	0.329	0.260	0.363	0.375	0.320	0.377	0.354	0.250	0.330	0.230	0.312
	NS-8	0.500	0.320	0.467	0.507	0.550	0.600	0.576	0.480	0.410	0.350	0.526
	NS-10	0.490	0.320	0.474	0.487	0.510	0.574	0.611	0.490	0.420	0.340	0.491
	NS-12	0.495	0.330	0.467	0.493	0.530	0.589	0.604	0.490	0.430	0.330	0.491
	NS-14	0.484	0.310	0.452	0.487	0.500	0.577	0.604	0.490	0.410	0.320	0.486

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introduction reflect both the theoretical and the experimental results regarding the studied algorithms complexity and numerical stability. Discussions about generalizations are also provided.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: For all the algorithms described we provide thorough discussions about both their strengths and limitations, and explain in which settings they are expected to perform better and when they might be less competitive.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: In all theorems, lemmas, and propositions, we clearly state all the underlying assumptions, for example, the assumed machine model and bits of precision, and the properties of the input matrices.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All the details pertaining the experimental evaluations are provided, including parameter selections, hardware description, models' architecture, and benchmarks used.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Links to the open-source repositories that contain the related source code are provided in the section "Code availability".

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All details about experiment parameters and model details are provided.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Where applicable, we describe precisely how many experiments were executed, how the data can vary in each execution, how results are aggregated and presented in the figures and tables, and how statistics are computed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware descriptions are provided, and guidelines how the experiments were executed and how time of execution was measured.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work contributes to the advancement of Machine Learning and Artificial Intelligence. There are many potential societal consequences of our work and similar works in general, but we cannot foresee any specific immediate concerns related to this specific work.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We cannot see any specific related risks besides the general risks involved in the broader field of Artificial Intelligence and Language Models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use the MMLU benchmark dataset which is under the MIT License and we explicitly declare it in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Strictly speaking, the paper itself does not really release any new assets. Our source code related to the optimized NPU implementations is already publicly available as open-source (before the paper submission), but we cannot provide a direct link to avoid violating the double-blind policy.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used for this paper for only the following purposes: writing assistance, formatting of plots and figures, and coding assistance. Core methodology, scientific rigor, or originality of the research were NOT impacted.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.