LAMP: Look-Ahead Mixed-Precision Inference of Large Language Models

Stanislav Budzinskiy [1] **Marian Gloser** [1] **Tolunay Yilmaz** [1] **Ying Hong Tham** [2] **Yuanyi Lin** [2] **Wenyi Fang** [2] **Fan Wu** [2] **Philipp Petersen** [1]

Abstract

Mixed-precision computations are a hallmark of the current stage of AI, driving the progress in large language models towards efficient, locally deployable solutions. This article addresses the floating-point c omputation o f compositionally-rich functions, concentrating on transformer inference. Based on the rounding error analysis of a composition $f(g(\mathbf{x}))$, we provide an adaptive strategy that selects a small subset of components of $g(\mathbf{x})$ to be computed more accurately while all other computations can be carried out with higher accuracy. We then explain how this strategy can be applied to different compositions within a transformer and illustrate its overall effect on transformer inference. We study the effectiveness of this algorithm numerically on GPT-2 models and demonstrate that already very low recompu-tation rates allow for improvements of up to two orders of magnitude in accuracy.

1. Introduction

Transformer deep neural networks (DNNs), originally introduced for sequence modeling in natural language processing (Vaswani et al., 2017), have become a standard computational paradigm across a wide range of domains, including large-scale language understanding (Devlin et al., 2019) as well as vision and multimodal learning (Dosovitskiy, 2021). From a computational perspective, a transformer can be viewed as a deep composition of simple operators, obtained by iterating attention-based mappings and pointwise nonlinear transformations across layers.

In practice, the evaluation of such deep compositions (*inference*) is performed in floating-point (FP) arithmetic, low-precision formats being routinely employed to improve performance and energy efficiency (Gupta et al., 2015; Micikevicius et al., 2018; Kalamkar et al., 2019). From a numerical analysis standpoint, FP evaluation introduces rounding errors at every stage of the computation; the cumulative effect of these local errors hinges on how they propagate through successive compositions of operators (Higham, 2002).

The bulk of operators in transformers are matrix products, and the existing approaches to mixed-precision inference largely address them based on two key principles: the input is quantized to low precision, and the output is accumulated in high precision (Xiao et al., 2023). Typically, the quantization precision is uniform across the whole input, and the accumulation precision is uniform across the whole output. While there are also mixed-precision quantization strategies (Dettmers et al., 2022), we are not aware of developments in mixed-precision accumulation for transformer inference.

In this work, we address this gap by proposing a mathematically principled mixed-precision inference strategy that is explicitly aware of compositional effects. Instead of treating the output of an intermediate computation uniformly, we follow a *look-ahead strategy* by flagging and recomputing, using higher-precision accumulation, those computations whose round-off errors will be most strongly amplified by the ensuing operator. Our method has a rigorous theoretical foundation and delivers strong empirical results in numerical experiments. We describe our contribution in detail in Subsection 1.2.

1.1. Rounding error analysis

To place our contribution in context, we briefly review how rounding error analysis has been applied to function evaluation, and why DNNs fall outside this classical setting. Rounding errors have been analyzed primarily for matrix computations (Higham, 2002; Connolly et al., 2021) and "basic" nonlinear functions such as the *elementary functions*: nth powers and roots, exponentials and logarithms, trigonometric and hyperbolic functions (Muller, 2016).

Special functions (Gil et al., 2007) are "basic" in mathematical physics, defined as solutions to specific differential equations or integrals. The complexity of their evaluation stems from the need to discretize the differential equation or integral to sufficient (typically very high) precision, after which rounding effects of FP arithmetic become noticeable (Lauter & Mezzarobba, 2015).

[1]Faculty of Mathematics, University of Vienna, Vienna, Austria [2]Huawei Technologies. Correspondence to: Stanislav Budzinskiy <stanislav.budzinskiy@univie.ac.at>.

Preprint. January 30, 2026.

DNNs, including transfusers, can be considered "basic" in the field of AI, and they differ from the aforementioned "basic" functions in two aspects: the target accuracy[1] and the source of evaluation complexity. First, the applications of DNNs do not seem to require the accuracy to be very high. Second, the evaluation complexity of DNNs is due to their extremely rich *compositional structure*, where the "building blocks" are either elementary functions or simple compositions of elementary functions (Blanchard et al., 2021; El Arar et al., 2024).

Therefore, *the algorithms of DNN inference need to pay specific attention to compositions*. Rounding error analysis applied to DNNs leads to worst-case bounds that grow exponentially with depth (El Arar et al., 2025; Budzinskiy et al., 2025). While the *global* rounding error appears to be difficult to tame, the *local* rounding error at each composition is easier to control—the core idea of our work.

1.2. Contributions

We propose a *novel adaptive inference strategy with mixed-precision accumulation* for various DNN architectures, including transformers, and establish its theoretical basis. The unit of our numerical analysis is a composition of two arbitrary functions, for which we develop a rigorous strategy to select the components of the inner function that need to be recomputed more accurately to ensure the numerical stability of the composition. The details are given in Section 2.

A theoretical investigation of the nonlinearities inherent to the transformer architecture allows us to prove that the aforementioned *selection procedure can be carried out with simple greedy algorithms*, overcoming the otherwise combinatorial nature of the problem, see Section 3.

The proposed method showcases highly convincing performance in numerical experiments with the GPT-2 XL model. In Figure 1, the key-query inner products are accumulated using μ mantissa bits, and we recompute about 8.3% of adaptively selected inner products in FP32.[2] One illustrative observation is that BF16 accumulation ($\mu = 7$) with adaptive recomputation deviates from uniform FP32 accumulation just as much as uniform TF32 accumulation ($\mu = 10$). Meanwhile, the effective number of mantissa bits used per key-query inner product is about 8.9 for the former against 10 for the latter.[3] Additionally, Figure 1 shows that our method does indeed select critical-for-performance inner products, since the same number of random recomputations has no effect. Find a detailed description of our experiments and more numerical results in Section 4.

[1] How many unit round-offs of error are tolerable, rather than the value of the unit round-off.

[2] This corresponds to $\tau = 1.2$ in our method; see Section 2.

[3] $1 \cdot 7 + 0.083 \cdot 23 = 8.909$

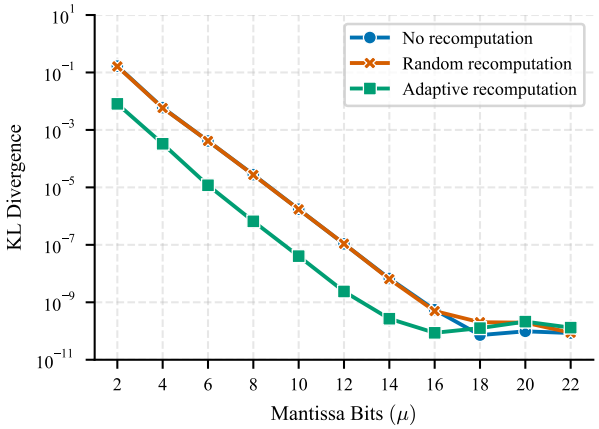

Figure 1. Performance of the proposed adaptive mixed-precision inference strategy with μ mantissa bits used for the low-precision accumulation of key-query inner products. The mixed precision strategy recomputes approximately 8.3% of the key-query inner products. The Kullback–Leibler divergence is measured against a reference model with uniform FP32 accumulation.

1.3. Impact and limitations

We demonstrate that numerical rounding effects arising from the compositional structure of DNNs can be mitigated at inference through a theoretically-grounded compositionality-aware mixed-precision strategy. Our empirical results hint at the possibility of accumulating key-query inner products in low-bitwidth formats on GPUs, packing multiple inner products in a single 32-bit register, and recovering the prediction accuracy via guided sparse recomputations. This workflow could increase the computational throughput of self-attention blocks at the cost of overhead associated with selecting the recomputations. We believe that these improvements are particularly relevant for latency-sensitive inference, where high-precision FP arithmetic is already widely employed, and even moderate gains in efficiency can have a meaningful practical impact. In addition to empirical accuracy improvements, our results highlight the importance of compositional numerical effects in deep models.

FlashAttention (Dao et al., 2022), together with its variations, is the industry standard for computing attention. Its main property is that the key-query matrix product and the softmax output probabilities are never materialized in full. For our approach, it is necessary to store and sort the vector of softmax probabilities, or at least to maintain a list of top-k highest probabilities. Therefore, our method is likely more suitable for applications with shorter context lengths.

Quantization is *not* addressed in our work. As such, the proposed approach does not involve retraining, does not modify the weights of a transformer, and does not reduce the memory footprint of its parameters (Frantar et al., 2023)—we aim to control the accuracy of *computations* instead. Conse-

quently, our method should be viewed as complementary to compression and quantization techniques.

Medium-size architectures: Our experimental evaluation focuses on the family of GPT-2 transformer models. While these models are moderate in size by current standards, they already exhibit the characteristic depth and compositional structure of modern transformers. Importantly, the numerical effects targeted by our method are present at this scale, suggesting that they are likely to become even more pronounced in larger architectures, and the effectiveness of our method likely increases.

Generalizability: The proposed mixed-precision inference strategy is derived from a theoretical analysis of the core building blocks of transformers. While the underlying idea applies to arbitrary DNN architectures, the feasibility of its implementation is architecture-dependent. Extending our approach to other DNN architectures would require a similar theoretical analysis of their constituent operations. Given the strong empirical results for transformers, we view this as a natural direction for future work.

2. Floating-point evaluation of compositions

Let $m, n, k \in \mathbb{N}$ and consider functions: $g : \mathbb{R}^k \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^m$. Given $\mathbf{x} \in \mathbb{R}^k$, our goal is to evaluate the composition $f(g(\mathbf{x}))$ in FP arithmetic.

2.1. Baseline evaluation

Let $\mathbf{y} = g(\mathbf{x}) \in \mathbb{R}^n$ be the exact output of the inner function, and let $\hat{\mathbf{y}} \in \mathbb{R}^n$ be its computed value. Assuming that g is evaluated with a mixed forward-backward stable algorithm (Higham, 2002), rounding error analysis guarantees that the relative error is bounded by

$$\frac{|\hat{\mathbf{y}} - \mathbf{y}|}{|\mathbf{y}|} \leq \mathbf{c}_g \mathrm{u}_g + \mathcal{O}(\mathrm{u}_g^2), \quad \mathbf{c}_g \in \mathbb{R}^n_+, \tag{1}$$

where the absolute value, division, and comparison of vectors are componentwise, and u_g denotes the unit round-off. The nonnegative vector $\mathbf{c}_g$ describes how round-off errors are magnified during the computation of each component of $\mathbf{y}$ and is determined by the input and output dimensions of the function g, the rounding mode (e.g., deterministic or stochastic), the condition number of g at $\mathbf{x}$, and the specific algorithm used to evaluate g.

Next, $\hat{\mathbf{y}}$ is used as input for f, which is evaluated in precision u_f. Denoting by $\hat{\mathbf{z}} \in \mathbb{R}^m$ the computed value of $f(\hat{\mathbf{y}})$, we get by the same logic as in (1) that

$$\frac{|\hat{\mathbf{z}} - f(\hat{\mathbf{y}})|}{|f(\hat{\mathbf{y}})|} \leq \mathbf{c}_f \mathrm{u}_f + \mathcal{O}(\mathrm{u}_f^2), \quad \mathbf{c}_f \in \mathbb{R}^m_+.$$

However, we need to compare $\hat{\mathbf{z}}$ with the exact $\mathbf{z} = f(\mathbf{y})$. By triangle inequality,

$$\frac{|\hat{\mathbf{z}} - \mathbf{z}|}{|\mathbf{z}|} \leq \frac{|\hat{\mathbf{z}} - f(\hat{\mathbf{y}})| + |f(\hat{\mathbf{y}}) - \mathbf{z}|}{|f(\hat{\mathbf{y}})|} \Big(1 + \mathcal{O}(\mathrm{u}_f)\Big).$$

The first term is bounded by $\mathbf{c}_f \mathrm{u}_f$ to first order. To bound the second term, we assume that f is sufficiently regular in the neighborhood of $\hat{\mathbf{y}}$ and use Taylor's theorem to get an estimate in terms of the Jacobian $\mathbf{J}_f(\hat{\mathbf{y}}) \in \mathbb{R}^{m \times n}$,

$$\frac{|f(\hat{\mathbf{y}}) - f(\mathbf{y})|}{|f(\hat{\mathbf{y}})|} \leq \frac{|\mathbf{J}_f(\hat{\mathbf{y}}) \mathrm{diag}(\hat{\mathbf{y}})| \mathbf{c}_g}{|f(\hat{\mathbf{y}})|} \mathrm{u}_g + \mathcal{O}(\mathrm{u}_g^2),$$

where $\mathrm{diag}(\hat{\mathbf{y}}) \in \mathbb{R}^{n \times n}$ is diagonal. As a result, we get

$$\frac{|\hat{\mathbf{z}} - \mathbf{z}|}{|\mathbf{z}|} \leq \mathbf{c}_f \mathrm{u}_f + \frac{|\mathbf{J}_f(\hat{\mathbf{y}}) \mathrm{diag}(\hat{\mathbf{y}})| \mathbf{c}_g}{|f(\hat{\mathbf{y}})|} \mathrm{u}_g + \mathcal{O}(\mathrm{u}_f^2 + \mathrm{u}_g^2). \tag{2}$$

2.2. Refined evaluation

The bound (2) follows from (1) and therefore depends on $\mathbf{c}_g$ (i.e., the evaluation algorithm of g) and the assumption that every component of g is computed in precision u_g. Consider a more flexible setting where we may want to compute some of the components more accurately—with a more accurate algorithm or in higher precision—to ensure that the second term in (2) is not too large. Let the nonzeros of $\mathbf{q} \in \{0, 1\}^n$ encode these components.

2.2.1. More accurate algorithm

We shall say that an evaluation algorithm is more accurate than the baseline if it leads to smaller entries in $\mathbf{c}_g$. Let us denote by $0 \leq \epsilon \leq 1$ the corresponding gain factor. Then the bound (1) becomes

$$\frac{|\hat{\mathbf{y}} - \mathbf{y}|}{|\mathbf{y}|} \leq \big(\mathbf{I} - \mathrm{diag}(\mathbf{q})\big) \mathbf{c}_g \mathrm{u}_g + \mathcal{O}(\mathrm{u}_g^2 + \epsilon \mathrm{u}_g),$$

leading to a modification of the second term in (2):

$$\frac{|\mathbf{J}_f(\hat{\mathbf{y}}) \mathrm{diag}(\hat{\mathbf{y}})| \big(\mathbf{I} - \mathrm{diag}(\mathbf{q})\big) \mathbf{c}_g}{|f(\hat{\mathbf{y}})|} \mathrm{u}_g + \mathcal{O}(\mathrm{u}_g^2 + \epsilon \mathrm{u}_g).$$

Example 2.1. Let $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$ with $\mathbf{A} \in \mathbb{R}^{n \times k}$ and $\mathbf{x} \in \mathbb{R}^k$ stored in precision u_g. Evaluating g in precision u_g, the basic multiplication algorithm has $\mathbf{c}_g = k \frac{|\mathbf{A}||\mathbf{x}|}{|\mathbf{A}\mathbf{x}|}$ for deterministic rounding (Higham, 2002) and $\mathbf{c}_g \lesssim \sqrt{k} \frac{|\mathbf{A}||\mathbf{x}|}{|\mathbf{A}\mathbf{x}|}$ with high probability for stochastic rounding (Connolly et al., 2021). Mixed-precision algorithms based on fused multiply-add can achieve $\mathbf{c}_g = \frac{|\mathbf{A}||\mathbf{x}|}{|\mathbf{A}\mathbf{x}|}$ (Blanchard et al., 2020). Therefore, the gain factor ϵ can be made small for large k.

2.2.2. Higher precision

As an alternative to a more accurate algorithm, we can use FP precision u_g^2 to compute and store the selected components. Then (1) turns into

$$\frac{|\hat{\mathbf{y}} - \mathbf{y}|}{|\mathbf{y}|} \leq \big(\mathbf{I} - \mathrm{diag}(\mathbf{q})\big) \mathbf{c}_g \mathrm{u}_g + \mathcal{O}(\mathrm{u}_g^2)$$

and leads to a similar modification in (2). This requires the evaluation algorithm of f to process mixed-precision inputs.

Example 2.2. Mixed-precision matrix multiplication (Blanchard et al., 2020) achieves $\mathbf{c}_g = \mathbf{0}$ when the output is stored in precision u_g^2, since no extra rounding is done at the end.

2.3. Look-ahead mixed-precision evaluation

The two refinement approaches of Subsection 2.2 lead to very similar rounding error bounds for the composition. The accuracy of the inner computation manifests itself in the second term, which we aim to reduce as follows: select a binary vector $\mathbf{q} \in \{0,1\}^n$ so that

$$\left\| \left| \mathrm{diag}(f(\hat{\mathbf{y}}))^{-1} \mathbf{J}_f(\hat{\mathbf{y}}) \right| (\mathbf{1} - \mathbf{q}) \right\|_\infty \leq \tau \tag{3}$$

or

$$\left\| \left| \mathrm{diag}(f(\hat{\mathbf{y}}))^{-1} \mathbf{J}_f(\hat{\mathbf{y}}) \mathrm{diag}(\hat{\mathbf{y}}) \right| (\mathbf{1} - \mathbf{q}) \right\|_\infty \leq \tau \tag{4}$$

for a given threshold $\tau \geq 0$. Let us discuss these objectives:

- Objectives (3) and (4) improve the bound (2) for suitable τ. Specifically, the unweighted (3) replaces the second term in (2) with $\mathbf{1}\tau\|\hat{\mathbf{y}}\|_\infty \|\mathbf{c}_g\|_\infty \mathrm{u}_g$, and the weighted (4) with $\mathbf{1}\tau\|\mathbf{c}_g\|_\infty \mathrm{u}_g$. Which of the two objectives to choose is a problem-specific question.
- Both (3) and (4) can always be attained with $\mathbf{q} = \mathbf{1}$, i.e., when every component of the inner function g is computed more accurately.
- When $\mathbf{q} = \mathbf{0}$, the norms in (3) and (4) are proportional to the *mixed* and *componentwise* condition numbers of f at $\hat{\mathbf{y}}$, respectively (Gohberg & Koltracht, 1993).
- The exact $f(\hat{\mathbf{y}})$ and $\mathbf{J}_f(\hat{\mathbf{y}})$ are not available in practice, so their FP computed values will be used.

If the baseline $\hat{\mathbf{y}}$ with $\mathbf{q} = \mathbf{0}$ satisfies (3) or (4), we deem the computation complete. Otherwise, we try a different $\mathbf{q}$, which entails the recomputation of $\hat{\mathbf{y}}$. Such recomputations can be costly, so we assume that the Jacobian is stable with respect to small variations in $\hat{\mathbf{y}}$ and fix the matrix $\mathbf{K} \in \{\mathbf{K}_u, \mathbf{K}_w\}$ corresponding to the baseline $\hat{\mathbf{y}}$, where[4]

$$\mathbf{K}_u = \mathrm{diag}(f(\hat{\mathbf{y}}))^{-1} \mathbf{J}_f(\hat{\mathbf{y}}),$$
$$\mathbf{K}_w = \mathrm{diag}(f(\hat{\mathbf{y}}))^{-1} \mathbf{J}_f(\hat{\mathbf{y}}) \mathrm{diag}(\hat{\mathbf{y}}).$$

Having found a suitable $\mathbf{q}$ that satisfies the bound

$$\left\| |\mathbf{K}|(\mathbf{1} - \mathbf{q}) \right\|_\infty = \left\| \mathbf{K}\big(\mathbf{I} - \mathrm{diag}(\mathbf{q})\big) \right\|_{\infty,\infty} \leq \tau,$$

we will recompute more accurately those components of $\hat{\mathbf{y}}$ that are indexed by the nonzeros of the binary vector $\mathbf{q}$. To minimize the number of recomputations, we require $\mathbf{q}$ to be sparse. This leads to our *look-ahead mixed-precision* (LAMP) problem: seek $\mathbf{q} \in \{0,1\}^n$ such that

$$\|\mathbf{q}\|_0 \to \min \quad \text{s.t.} \quad \left\| |\mathbf{K}|(\mathbf{1} - \mathbf{q}) \right\|_\infty \leq \tau. \tag{5}$$

The whole procedure of computing the inner function g is summarized in Algorithm 1. We shall call $\mathbf{K}_u$ and $\mathbf{K}_w$ the unweighted and weighted LAMP matrices, respectively.

Algorithm 1 LAMP evaluation of a composition

Input: functions f and g, variable $\mathbf{x}$, threshold $\tau \geq 0$, weighted or unweighted objective
Output: adaptively computed value $\hat{\mathbf{y}}$ of $g(\mathbf{x})$
Compute $\hat{\mathbf{y}} \approx g(\mathbf{x})$ in FP arithmetic.
Compute $\mathbf{K} \approx \mathbf{K}_u$ or $\mathbf{K} \approx \mathbf{K}_w$ in FP arithmetic.
Find a solution $\mathbf{q}$ of the LAMP problem (5).
Recompute components of $\hat{\mathbf{y}}$ indexed by nonzeros of $\mathbf{q}$ more accurately.

3. Application to transformer inference

Algorithm 1 relies on the possibility to compute individual components of g separately. Yet this is not always possible. For instance, consider the softmax function

$$\mathrm{softmax}(\mathbf{x}) = \left[\frac{\exp(x_1)}{\sum_{i=1}^k \exp(x_i)} \quad \cdots \quad \frac{\exp(x_k)}{\sum_{i=1}^k \exp(x_i)} \right]^\mathsf{T}. \tag{6}$$

If at least one component needs to be computed more accurately, every component has to be as well. Meanwhile, the required property holds for matrix-vector products: it suffices to divide the matrix into two blocks and use different multiplication algorithms for each (see Example 2.1).[5]

Matrix products are used heavily in all DNN architectures. We thus apply LAMP evaluation to the following question:

How to adapt matrix multiplication to the ensuing nonlinearity during DNN inference?

What is the "ensuing nonlinearity" in question? To target the accuracy of the end result of inference, an obvious choice is to set f to be the remaining tail of the DNN. However, the computation of its LAMP matrix becomes highly demanding. Additionally, the sparse-optimization LAMP problem (5) is in general NP-hard (Natarajan, 1995).

Instead, let f be one of the *elementary DNN nonlinearities*; for transformers, these are the activation function, layer normalization, and softmax. We prove for these three functions that *almost-the-sparsest* solutions of the LAMP problem

[4] If the Jacobian changes rapidly, the Hessian would need to be included in the analysis.

[5] When a bias term is present, it can be added to the accumulator in the required precision as well.

(5) can be obtained with a greedy algorithm of complexity $\mathcal{O}(n \log n)$—the cost of a sorting algorithm.

3.1. Activation functions

Consider an activation function $\varphi : \mathbb{R} \to \mathbb{R}$. Within DNNs, activation functions are applied componentwise to vectors:

$$f : \mathbb{R}^n \to \mathbb{R}^n, \quad f(\mathbf{y}) = \begin{bmatrix} \varphi(y_1) & \cdots & \varphi(y_n) \end{bmatrix}^\intercal.$$

As a consequence, the LAMP matrix is diagonal,

$$\mathbf{K}_u = \operatorname{diag}\left(\frac{\varphi'(y_1)}{\varphi(y_1)}, \ldots, \frac{\varphi'(y_n)}{\varphi(y_n)} \right),$$
$$\mathbf{K}_w = \operatorname{diag}\left(\frac{\varphi'(y_1)}{\varphi(y_1)} y_1, \ldots, \frac{\varphi'(y_n)}{\varphi(y_n)} y_n \right),$$

and the solution $\mathbf{q}$ of the LAMP problem (5) can be written in closed form: an entry of $\mathbf{q}$ is nonzero if and only if the corresponding diagonal entry of the LAMP matrix exceeds τ in absolute value. The diagonal structure of the Jacobian makes it possible to solve (5) immediately.

For activation functions, LAMP evaluation essentially repeats the mixed-precision accumulation idea of (El Arar et al., 2025). The novelty of the proposed LAMP problem (5) is that it applies to arbitrary compositions, and we prove below that transformer-specific compositions are particularly suitable for LAMP evaluation.

3.2. Layer normalization

Layer normalization aims to stabilize the training of DNNs (Ba et al., 2016) by "standardizing" its input via[6]

$$f : \mathbb{R}^n \to \mathbb{R}^n, \quad f(\mathbf{y}) = \sqrt{n} \frac{\mathbf{y} - \frac{1}{n}\mathbf{1}\mathbf{1}^\intercal \mathbf{y}}{\|\mathbf{y} - \frac{1}{n}\mathbf{1}\mathbf{1}^\intercal \mathbf{y}\|_2}.$$

Its placement within a transformer can vary across specific architectures (Xiong et al., 2020), the two conventional choices being *post*normalization (Vaswani et al., 2017; Devlin et al., 2019) and *pre*normalization (Brown et al., 2020; Touvron et al., 2023; Chowdhery et al., 2023). Yet neither places layer normalization right after a matrix product.

Another possible placement is between a (feedforward or attention) sublayer and a skip connection (Liu et al., 2022; OLMo et al., 2025), then layer normalization directly follows a matrix product and we can apply Algorithm 1.

Note that the shifting step in layer normalization is itself a matrix-vector product, and we can "attach" this matrix to the preceding matrix multiplication. The remaining function

$$f : \mathbb{R}^n \to \mathbb{R}^n, \quad f(\mathbf{y}) = \sqrt{n} \frac{\mathbf{y}}{\|\mathbf{y}\|_2} \tag{7}$$

is *root mean square* (RMS) layer normalization (Zhang & Sennrich, 2019), which is often used on its own (Touvron et al., 2023; OLMo et al., 2025). We analyze the LAMP problem (5) in the weighted case (4) for RMS layer normalization. Denote $[n] = \{1, \ldots, n\}$.

Lemma 3.1. *For RMS layer normalization (7), the weighted LAMP matrix equals*

$$\mathbf{K}_w = \mathbf{I} - \frac{\mathbf{1}\mathbf{y}^\intercal}{\|\mathbf{y}\|_2^2} \operatorname{diag}(\mathbf{y})$$

and satisfies

$$\|\mathbf{K}_w\|_{\infty,\infty} = 2\left(1 - \frac{\min_{i\in[n]} y_i^2}{\|\mathbf{y}\|_2^2}\right),$$
$$2\left(1 - \frac{1}{n}\right) \le \|\mathbf{K}_w\|_{\infty,\infty} \le 2.$$

Proof. See Lemma 3.6 from (Budzinskiy et al., 2025). □

Lemma 3.2. *Let $\mathbf{q} \in \{0,1\}^n$ be such that $\mathbf{q} \neq \mathbf{1}$, and denote by Ω its support. For RMS layer normalization (7), the weighted LAMP matrix satisfies*

$$\left\| |\mathbf{K}_w|(\mathbf{1} - \mathbf{q}) \right\|_\infty = 2\left(1 - \frac{\min_{j\notin\Omega} y_j^2}{\|\mathbf{y}\|_2^2}\right) - \frac{\sum_{i\in\Omega} y_i^2}{\|\mathbf{y}\|_2^2}$$

when $|\Omega| \le n - 2$ and

$$\left\| |\mathbf{K}_w|(\mathbf{1} - \mathbf{q}) \right\|_\infty = \max\left\{ \frac{y_j^2}{\|\mathbf{y}\|_2^2}, 1 - \frac{y_j^2}{\|\mathbf{y}\|_2^2} \right\}$$

when $\Omega = [n] \setminus \{j\}$.

Proof. See Appendix A.1. □

Even though Lemma 3.2 provides exact values of the matrix norm for each $\mathbf{q}$, finding the sparsest $\mathbf{q}$ to satisfy (4) can be too expensive for large n. The following proposition shows that there exist simple solutions of (5).

Proposition 3.3. *Suppose that the entries of $\mathbf{y}$ are arranged as $y_1^2 \ge \cdots \ge y_n^2$. Let $\mathbf{q} \in \{0,1\}^n$ be a solution of the weighted LAMP problem (5) for RMS layer normalization (7). If $\|\mathbf{q}\|_0 \le n - 3$ then a vector $\mathbf{q}' \in \{0,1\}^n$ given by*

$$\mathbf{q}' = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix}^\intercal, \quad \|\mathbf{q}'\|_0 = \|\mathbf{q}\|_0 + q_n,$$

also satisfies $\left\| |\mathbf{K}_w|(\mathbf{1} - \mathbf{q}') \right\|_\infty \le \tau$.

Proof. See Appendix A.2. □

Proposition 3.3 ensures that an almost-the-sparsest solution of the LAMP problem (5) can be obtained greedily: we

[6] We focus on layer normalization without scale and bias, as they can be seen as an affine function applied after the nonlinearity.

need to sort the entries of $\mathbf{y}$ in descending order according to their squares, pick the smallest s such that

$$\sum_{i=1}^{s} y_i^2 + 2y_n^2 \geq (2-\tau)\|\mathbf{y}\|_2^2,$$

and form $\mathbf{q}$ based on the initial positions of the indices $\{1,\dots,s\}$. The cost of this algorithm is dominated by the sorting step, which has the complexity $\mathcal{O}(n \log n)$.

3.3. Attention and softmax

The multi-attention mechanism is a distinctive feature of trans-formers (Vaswani et al., 2017). In the simplest case, atten-tion is computed as $\mathbf{V} \cdot \mathrm{softmax}(\mathbf{K}^\intercal\mathbf{Q})$. This is a compo-sition of three functions: the product of the key and query matrices, the softmax function (6) applied columnwise, and the product of the value matrix and the output of softmax.

When the value-softmax product is followed by layer nor-malization, its evaluation can be adapted as discussed in Subsection 3.2. We shall study the LAMP evaluation of the key-query product by analyzing the LAMP problem (5) for softmax in the unweighted case (3).

Lemma 3.4. *For softmax (6), the unweighted LAMP matrix equals*

$$\mathbf{K}_u = \mathbf{I} - \mathbf{1}\mathbf{z}^\intercal,$$

where $\mathbf{z} = \mathrm{softmax}(\mathbf{y})$*, and satisfies*

$$\|\mathbf{K}_u\|_{\infty,\infty} = 2\Big(1 - \min_{i\in[n]} z_i\Big),$$

$$2\left(1-\frac{1}{n}\right) \leq \|\mathbf{K}_u\|_{\infty,\infty} \leq 2.$$

Proof. See Lemma 3.13 from (Budzinskiy et al., 2025). □

Lemma 3.4 reveals an intriguing connection between the *weighted* LAMP matrix of RMS layer normalization and the *unweighted* LAMP matrix of softmax: in fact, they are the same matrix. To see this, it suffices to replace z_i with $y_i^2/\|\mathbf{y}\|_2^2$ and recall that $\|\mathbf{z}\|_1 = 1$ by the definition of soft-max (6). Therefore, our results on RMS layer normalization and their proofs can be extended to the softmax function.

Lemma 3.5. *Let* $\mathbf{q} \in \{0,1\}^n$ *be such that* $\mathbf{q} \neq \mathbf{1}$*, and denote by* Ω *its support. For softmax (6) with values* $\mathbf{z} = \mathrm{softmax}(\mathbf{y})$*, the unweighted LAMP matrix satisfies*

$$\big\||\mathbf{K}_u|(\mathbf{1}-\mathbf{q})\big\|_\infty = 2\left(1 - \min_{j\notin\Omega} z_j\right) - \sum_{i\in\Omega} z_i$$

when $|\Omega| \leq n-2$ *and*

$$\big\||\mathbf{K}_u|(\mathbf{1}-\mathbf{q})\big\|_\infty = \max\{z_j, 1-z_j\}$$

when $\Omega = [n] \setminus \{j\}$.

Proof. Follows from Lemma 3.2. □

Proposition 3.6. *Let* $\mathbf{z} = \mathrm{softmax}(\mathbf{y})$ *and suppose that its entries are arranged as* $z_1 \geq \cdots \geq z_n$*. Let* $\mathbf{q} \in \{0,1\}^n$ *be a solution of the unweighted LAMP problem (5) for softmax (6). If* $\|\mathbf{q}\|_0 \leq n-3$ *then a vector* $\mathbf{q}' \in \{0,1\}^n$ *given by*

$$\mathbf{q}' = \begin{bmatrix}1 & \cdots & 1 & 0 & \cdots & 0\end{bmatrix}^\intercal, \quad \|\mathbf{q}'\|_0 = \|\mathbf{q}\|_0 + q_n,$$

also satisfies $\big\||\mathbf{K}_w|(\mathbf{1}-\mathbf{q}')\big\|_\infty \leq \tau$.

Proof. The result follows readily from Proposition 3.3. □

The same greedy selection strategy applies to the softmax function as to RMS layer normalization. Let us also discuss the choice of the threshold τ and set $\tau = 2-\epsilon$. Then we seek the smallest s such that, after sorting,

$$\sum_{i=1}^{s} z_i + 2z_n \geq \epsilon,$$

and the inequality is satisfiable for $0 \leq \epsilon \leq 1$. In the case of almost equal probabilities $z_1 = \cdots = z_{n-1} = \frac{1}{n-1}$ and $z_n = 0$, we get $s = \lceil \epsilon(n-1) \rceil$. On the contrary, $s = 1$ when $z_1 = 1$. Informally, the more a transformer is "confused," the more accurate its inference should be.

4. Numerical experiments

In our experiments, we work with the GPT-2 XL model and test it on the OpenWebText[7] dataset. On the one hand, the choice of the model is dictated by available computational resources; on the other hand, GPT-2 models possess a rich compositional structure, which is the main structural prop-erty we capitalize on. Find experiments with other datasets and the GPT-2 small model in Appendix B.

Our experiments serve as a proof of concept for an adaptive approach to transformer inference: that LAMP evaluation can improve its numerical stability. The efficient implemen-tation of LAMP inference tailored to modern hardware and runtime comparisons fall beyond the scope of our article.

The code used for the experiments is publicly available.[8]

4.1. Custom floating-point format

To focus on the question of precision and put aside the issue of overflows, we simulate low-precision matrix multiplica-tions using a custom *partial single* FP format. For every $\mu \in \{1,\dots,23\}$, we define the format $\mathrm{PS}(\mu)$ to have μ mantissa bits, 8 exponent bits, and one sign bit. This for-mat is equivalent to FP32 when $\mu = 23$, to TF32 when $\mu = 10$, and to BF16 when $\mu = 7$. In code, we implement

[7] https://huggingface.co/datasets/Skylion007/openwebtext
[8] https://github.com/sbudzinskiy/LAMP-LLM

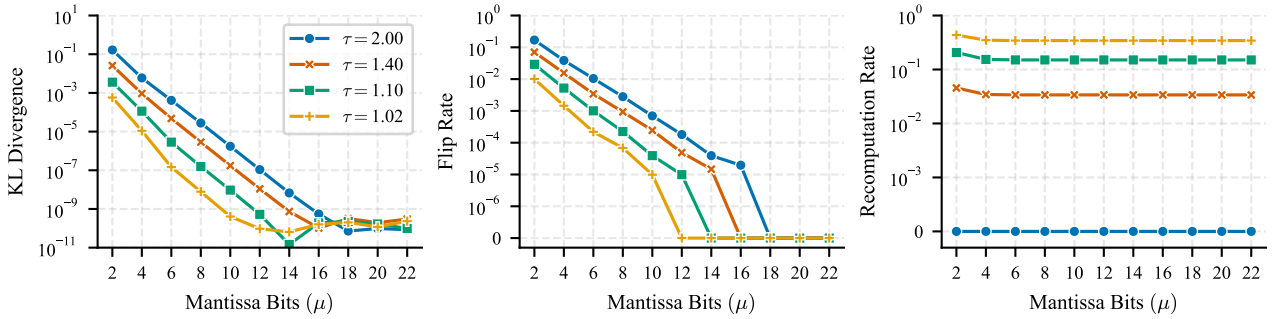

Figure 2. Performance of mixed-precision GPT-2 XL inference on the OpenWebText dataset with LAMP evaluation of the key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ).

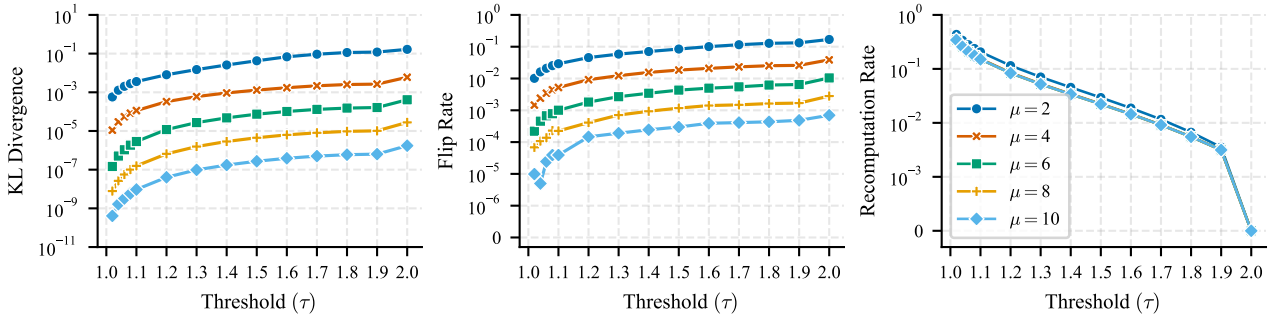

Figure 3. Performance of mixed-precision GPT-2 XL inference on the OpenWebText dataset with LAMP evaluation of the key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ).

$\mathrm{PS}(\mu)$ numbers via FP32 numbers rounded to μ mantissa bits according to the round-to-nearest-ties-to-even mode.

To perform matrix multiplication with input-output formats $\mathrm{PS}(\mu_A) \times \mathrm{PS}(\mu_B) \to \mathrm{PS}(\mu_C)$, we accumulate inner products as $\mathrm{round}(c + a \cdot b)$, where the scalar multiplication and addition are in FP32.

4.2. Experimental setting

To assess the impact of LAMP evaluation on inference, we compute the mean Kullback–Leibler (KL) divergence between the probability distributions output by a reference model and a test model over 200 sequences of 1024 tokens each. We also look at the flip rate, i.e., how often the most probable predictions of the reference and test models differ.

Our reference model uses FP32 inference uniformly for all FP operations. The test models perform the key-query products in $\mathrm{PS}(\mu)$ and recompute some of them, as selected by the LAMP problem (5), in FP32. We keep track of how many inner products are recomputed; to get the recomputation rate, we divide by the number of key-query inner products in the "causal mask."

4.3. Numerical results

In Figure 2, we fix the threshold τ of LAMP and vary the number of mantissa bits μ. The plots demonstrate clearly that our method works in practice, that is, as the threshold τ decreases, it improves the accuracy of inference and increases the number of recomputations required. Specifically, consistently for smaller μ, our method reduces the KL divergence tenfold with 3.4% of recomputations ($\tau = 1.4$), hundredfold with 15% of recomputations ($\tau = 1.1$), and thousandfold with 34.3% of recomputations ($\tau = 1.02$). Similarly consistent improvements hold for the flip rate as well, and we observe exponential-like decay of the two metrics as the number of bits μ is increased. An interesting observation is that the recomputation rate hardly depends on μ, yet we stress again that the *choice* of the components to be recomputed has a crucial impact on the performance of mixed-precision inference (cf. Figure 1 and Appendix B).

Figure 3 provides a different angle on the numerical results, fixing μ and varying τ. The plots show the KL divergence is improved at the same rate, regardless of the (small) number of bits μ, as the threshold τ decreases. The same observation holds for the flip rate, and both metrics decay faster as τ tends to one, at which point everything is selected for recomputation. We can also see that the recomputation rate

is slightly higher for extremely low-precision accumulation ($\mu = 2$) than for the rest.

The empirical results presented in Figures 2 and 3 convincingly demonstrate a strong performance of the proposed LAMP method, as it reaches high inference accuracies relative to the ground truth with the vast majority of key-query inner products performed in low precision. The additional experiments in Appendix B strengthen this conclusion.

While not studied numerically in this paper, LAMP evaluation can be applied to all transformer nonlinearities simultaneously (Section 3), which could increase the computational throughput of transformer inference even further. Together with a hardware-specific, efficient implementation of LAMP inference, this is a promising avenue for future research.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning by improving the efficiency of large language models. Our method has the potential to reduce the energy consumption required to deploy these models, contributing to "Green AI" initiatives.

Acknowledgments

This work was carried out in the framework of a research project funded by Huawei Technologies Ltd. We are grateful to El-Mehdi El Arar, Silviu-Ioan Filip, Theo Mary, and Elisa Riccietti for fruitful discussions.

Author Contributions

SB conceived the approach, formulated the research problem, and carried out the formal analysis, experimentation, and implementation. SB wrote the original draft of the manuscript. MG and TY contributed to the analysis and implementation and assisted with proofreading. YHT contributed to project administration and to reviewing and editing the manuscript. YL, WF, and FW contributed to project administration. PP supervised the project as laboratory head and reviewed and edited the manuscript.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv*, art. 1607.06450, 2016. doi: 10.48550/arXiv.1607.06450.

Blanchard, P., Higham, N. J., Lopez, F., Mary, T., and Pranesh, S. Mixed precision block fused multiply-add: Error analysis and application to gpu tensor cores. *SIAM J Sci Comput*, 42(3):C124–C141, 2020. doi: 10.1137/19M1289546.

Blanchard, P., Higham, D. J., and Higham, N. J. Accurately computing the log-sum-exp and softmax functions. *IMA J Numer Anal*, 41(4):2311–2330, 2021. doi: 10.1093/imanum/draa038.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pp. 1877–1901, 2020. doi: 10.48550/arXiv.2005.14165.

Budzinskiy, S., Fang, W., Zeng, L., and Petersen, P. Numerical error analysis of large language models. *arXiv*, art. 2503.10251, 2025. doi: 10.48550/arXiv.2503.10251.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *J Mach Learn Res*, 24(240):1–113, 2023. doi: 10.48550/arXiv.2204.02311.

Connolly, M. P., Higham, N. J., and Mary, T. Stochastic rounding and its probabilistic backward error analysis. *SIAM J Sci Comput*, 43(1):A566–A585, 2021. doi: 10.1137/20M1334796.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, volume 35, pp. 16344–16359, 2022. doi: 2205.14135.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *NeurIPS*, volume 35, pp. 30318–30332, 2022. doi: 10.48550/arXiv.2208.07339.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, volume 1, pp. 4171–4186, 2019. doi: 10.48550/arXiv.1810.04805.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. doi: 10.48550/arXiv.2010.11929.

El Arar, E.-M., Sohier, D., de Oliveira Castro, P., and Petit, E. Bounds on nonlinear errors for variance computation with stochastic rounding. *SIAM J Sci Comput*, 46(5):B579–B599, 2024. doi: 10.1137/23M1563001.

El Arar, E.-M., Filip, S.-I., Mary, T., and Riccietti, E. Mixed precision accumulation for neural network inference guided by componentwise forward error analysis. *arXiv*, art. 2503.15568, 2025. doi: 10.48550/arXiv.2503.15568.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. OPTQ: Accurate post-training quantization for generative pre-trained transformers. In *ICLR*, 2023. doi: 10.48550/arXiv.2210.17323.

Gil, A., Segura, J., and Temme, N. M. *Numerical Methods for Special Functions*. SIAM, 2007. doi: 10.1137/1.9780898717822.

Gohberg, I. and Koltracht, I. Mixed, componentwise, and structured condition numbers. *SIAM J Matrix Anal Appl*, 14(3):688–704, 1993. doi: 10.1137/0614049.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *ICML*, volume 37, pp. 1737–1746, 2015. doi: 10.48550/arXiv.1502.02551.

Higham, N. J. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2 edition, 2002. doi: 10.1137/1.9780898718027.

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., et al. A study of BFLOAT16 for deep learning training. *arXiv*, art. 1905.12322, 2019. doi: 10.48550/arXiv.1905.12322.

Lauter, C. and Mezzarobba, M. Semi-automatic floating-point implementation of special functions. In *ARITH 22*, pp. 58–65, 2015. doi: 10.1109/ARITH.2015.12.

Liu, Z., Hu, H., Lin, Y., Yao, Z., Xie, Z., Wei, Y., Ning, J., Cao, Y., Zhang, Z., Dong, L., et al. Swin transformer v2: Scaling up capacity and resolution. In *IEEE/CVF CVPR*, pp. 12009–12019, 2022. doi: 10.48550/arXiv.2111.09883.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. In *ICLR*, 2018. doi: 10.48550/arXiv.1710.03740.

Muller, J.-M. *Elementary Functions: Algorithms and Implementation*. Springer, 3 edition, 2016. doi: 10.1007/978-1-4899-7983-4.

Natarajan, B. K. Sparse approximate solutions to linear systems. *SIAM J Comput*, 24(2):227–234, 1995. doi: 10.1137/S0097539792240406.

OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., et al. 2 OLMo 2 furious. *arXiv*, art. 2501.00656, 2025. doi: 10.48550/arXiv.2501.00656.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. LLaMA: Open and efficient foundation language models. *arXiv*, art. 2302.13971, 2023. doi: 10.48550/arXiv.2302.13971.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, volume 30, pp. 5998–6008, 2017. doi: 10.48550/arXiv.1706.03762.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *ICML*, volume 202, pp. 38087–38099, 2023. doi: 10.48550/arXiv.2211.10438.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *ICML*, volume 119, pp. 10524–10533, 2020. doi: 10.48550/arXiv.2002.04745.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *NeurIPS*, volume 32, pp. 12360–12371, 2019. doi: 10.48550/arXiv.1910.07467.

A. Proofs

A.1. Proof of Lemma 3.2

Denote by s_l the lth absolute row-sum of $\mathbf{K}_w\big(\mathbf{I}-\mathrm{diag}(\mathbf{q})\big)$. By Lemma 3.1,

$$s_l = \begin{cases} \frac{1}{\|\mathbf{y}\|_2^2}\sum_{j\notin\Omega} y_j^2, & l\in\Omega,\\ \frac{1}{\|\mathbf{y}\|_2^2}\sum_{j\notin\Omega\cup\{l\}} y_j^2 + 1 - \frac{y_l^2}{\|\mathbf{y}\|_2^2}, & l\notin\Omega. \end{cases}$$

Rewriting $\sum_{j\notin\Omega} y_j^2 = \|\mathbf{y}\|_2^2 - \sum_{i\in\Omega} y_i^2$ and taking the maximum of s_l over l, we get

$$\big\|\mathbf{K}_w\big(\mathbf{I}-\mathrm{diag}(\mathbf{q})\big)\big\|_{\infty,\infty} = \max\left\{2\left(1-\frac{\min_{j\notin\Omega} y_j^2}{\|\mathbf{y}\|_2^2}\right),1\right\} - \frac{\sum_{i\in\Omega} y_i^2}{\|\mathbf{y}\|_2^2}.$$

Note that $1 > 2\Big(1-\frac{\min_{j\notin\Omega} y_j^2}{\|\mathbf{y}\|_2^2}\Big)$ if and only if $\min_{j\notin\Omega} y_j^2 > \frac{1}{2}\|\mathbf{y}\|_2^2$. When $|\Omega|\le n-2$, this would imply that

$$\sum_{j\notin\Omega} y_j^2 > \frac{1}{2}(n-|\Omega|)\|\mathbf{y}\|_2^2 \ge \|\mathbf{y}\|_2^2,$$

which is impossible. This contradiction proves the first formula. The second formula follows from the general expression.

A.2. Proof of Proposition 3.3

Denote by Ω and Ω' the supports of $\mathbf{q}$ and $\mathbf{q}'$, respectively. By Lemma 3.2, as $|\Omega'|\le n-2$, we need to show that

$$2\left(1-\frac{y_n^2}{\|\mathbf{y}\|_2^2}\right) - \frac{\sum_{i\in\Omega'} y_i^2}{\|\mathbf{y}\|_2^2} \le \tau.$$

Since $|\Omega| < n-2$ and $\mathbf{q}$ satisfies the norm constraint of LAMP, it suffices to show that

$$\sum_{i\in\Omega'} y_i^2 + 2y_n^2 \ge \sum_{i\in\Omega} y_i^2 + 2\min_{j\notin\Omega} y_j^2.$$

Note that Ω' is constructed in such a way that for every set $\tilde{\Omega}\subset[n]$ with $|\tilde{\Omega}| = |\Omega'|$, it holds that $\sum_{i\in\Omega'} y_i^2 \ge \sum_{i\in\tilde{\Omega}} y_i^2$. In case $n\notin\Omega$, we have $\min_{j\notin\Omega} y_j^2 = y_n^2$, and the desired inequality follows trivially since $|\Omega| = |\Omega'|$. When $n\in\Omega$, we have

$$\sum_{i\in\Omega} y_i^2 + 2\min_{j\notin\Omega} y_j^2 \le y_n^2 + \sum_{i\in\Omega\setminus\{n\}} y_i^2 + \min_{\substack{j,l\notin\Omega\\ j\neq l}} (y_j^2+y_l^2) \le y_n^2 + \sum_{i\in\Omega'} y_i^2 \le 2y_n^2 + \sum_{i\in\Omega'} y_i^2,$$

where in the second inequality we apply the majorizing property of Ω' to the set $\tilde{\Omega} = \Omega\setminus\{n\}\cup\{j_*,l_*\}$, with minimizing indices j_* and l_*, and note that $|\tilde{\Omega}| = |\Omega'|$.

B. Additional numerical experiments

B.1. GPT-2 XL model, three datasets

In the main text, we presented experiments with the GPT-2 XL model and the OpenWebText[9] dataset. Here, we add two more datasets for comparison: CodeParrot[10] and ArXiv.[11] The numerical results in Figures 4 and 5 show that the behavior of the proposed method remains almost the same across different datasets, signifying that our approach is "input-agnostic."

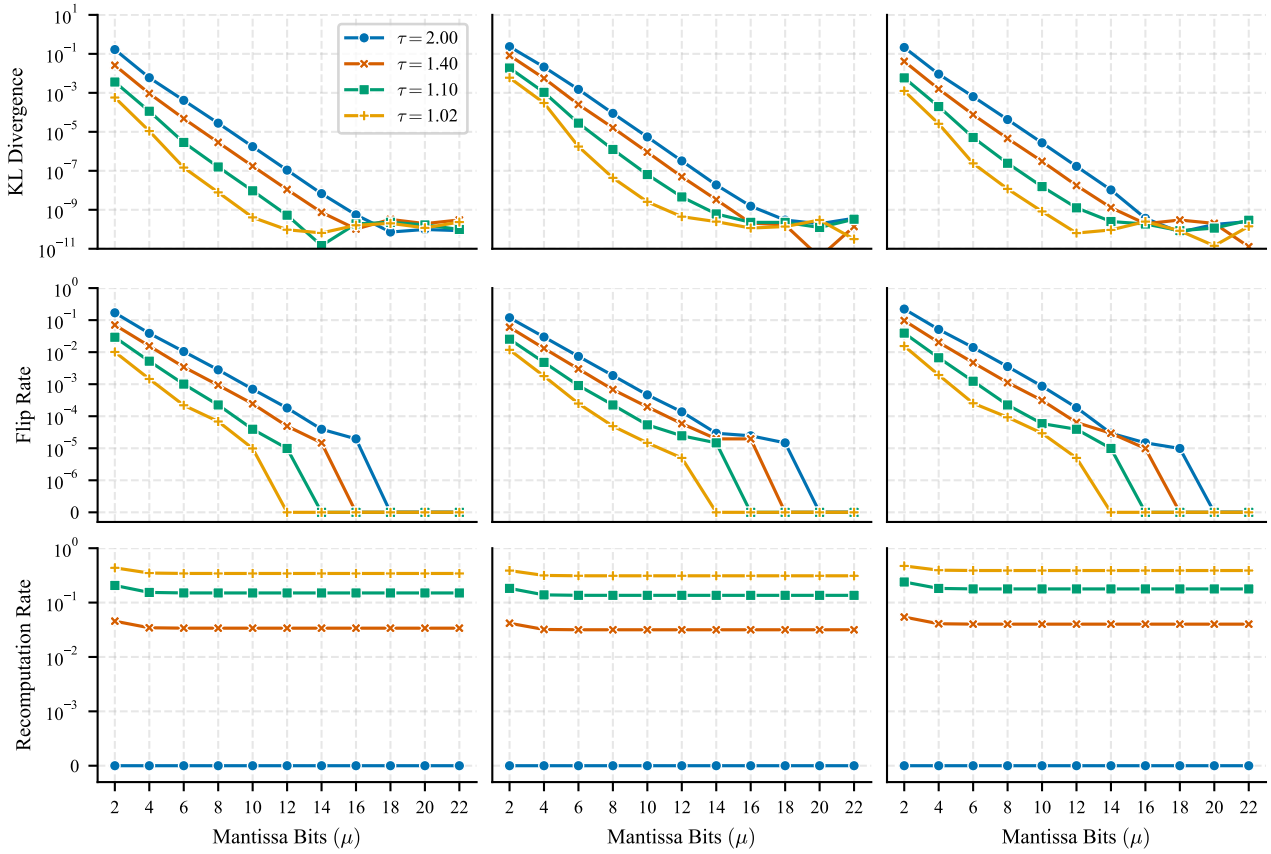

Figure 4. Performance of mixed-precision GPT-2 XL inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ).

[9] https://huggingface.co/datasets/Skylion007/openwebtext
[10] https://huggingface.co/datasets/codeparrot/codeparrot-clean
[11] https://huggingface.co/datasets/ccdv/arxiv-summarization

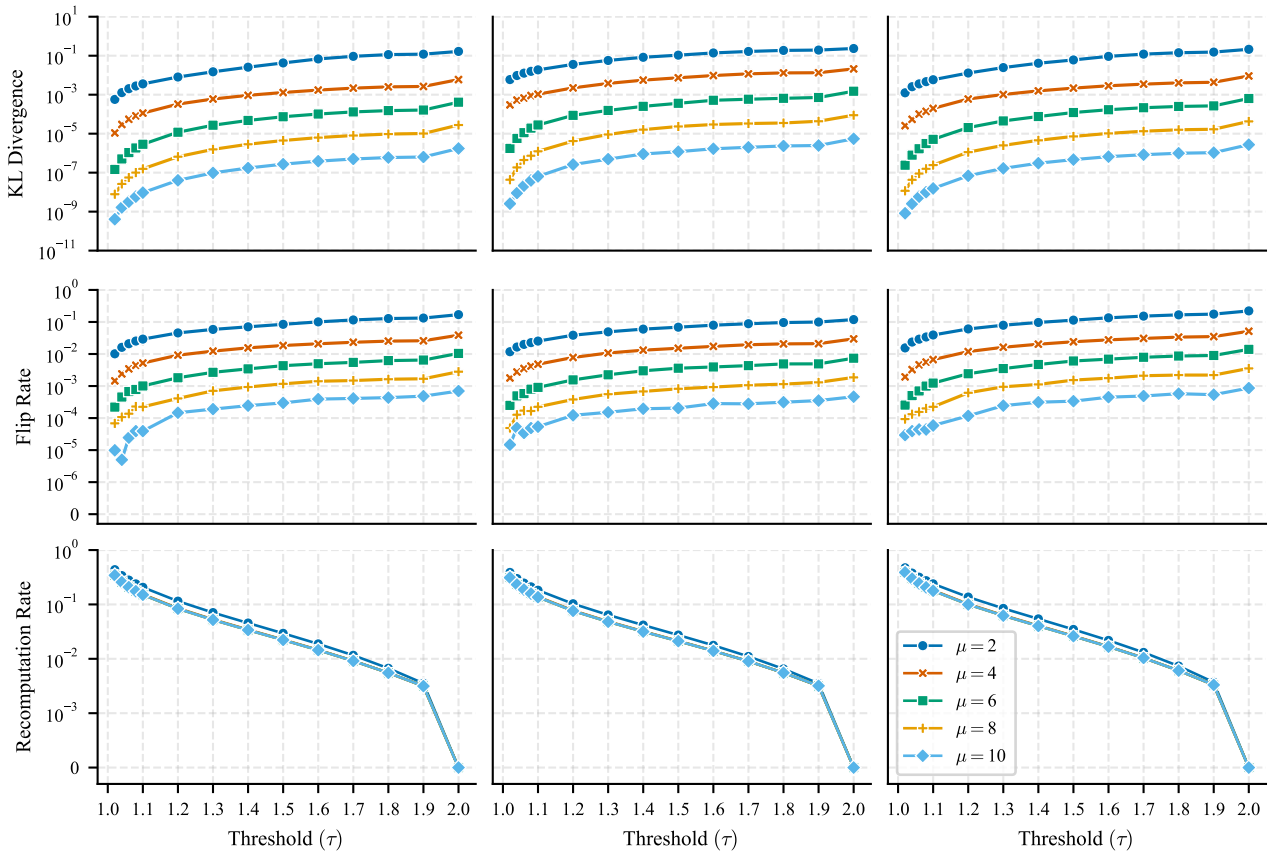

Figure 5. Performance of mixed-precision GPT-2 XL inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ).

B.2. GPT-2 small model, three datasets

Next, we validate the proposed method on a different transformer model—the GPT-2 small model—and on the same three datasets. The numerical results in Figures 6 and 7 show that low-precision accumulation has a more pronounced effect on the inference accuracy for the GPT-2 small model, as compared with the GPT-2 XL model (about 5-10 times higher KL divergence and flip rate overall). At the same time, the improvements due to high-precision recomputations of LAMP are of the same order as for the larger model. The recomputation rates are also qualitatively the same for the two models.

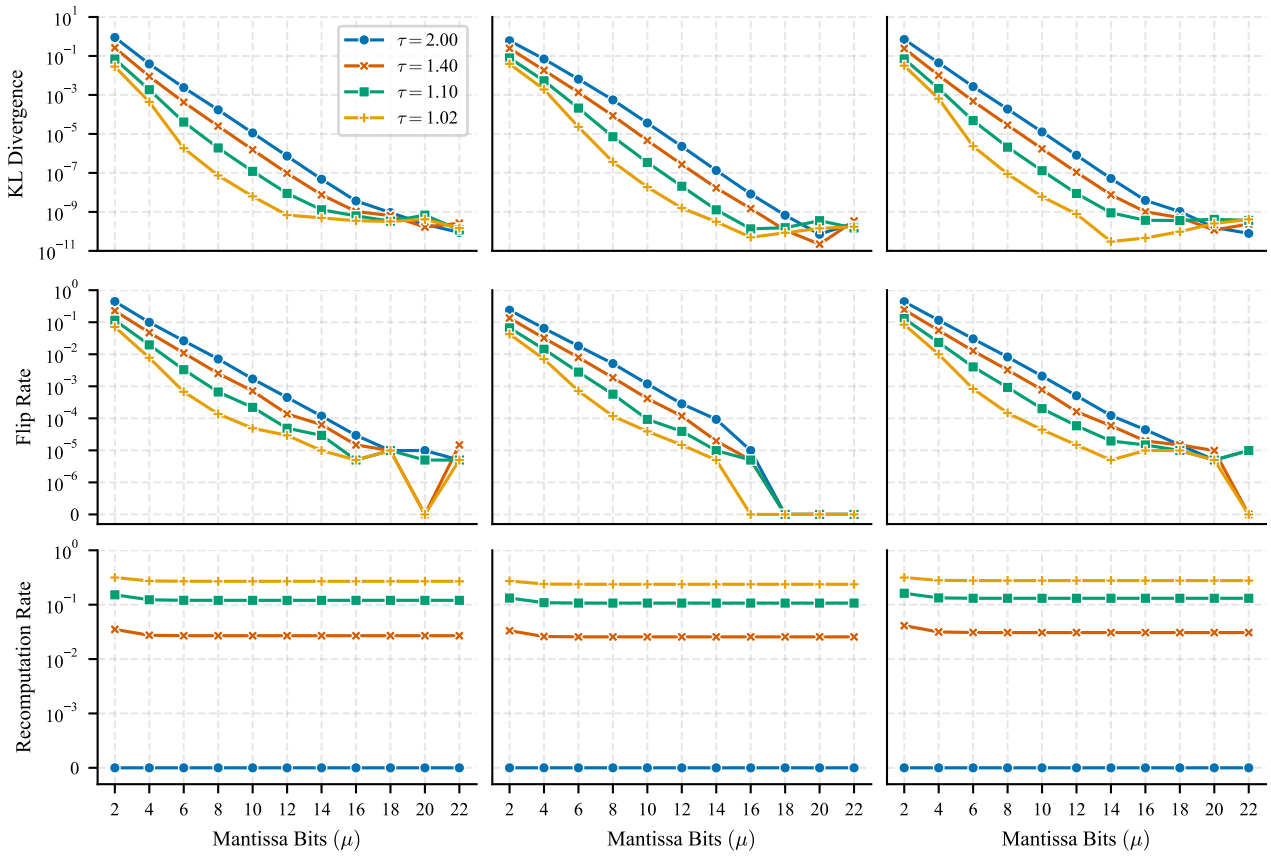

Figure 6. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ).

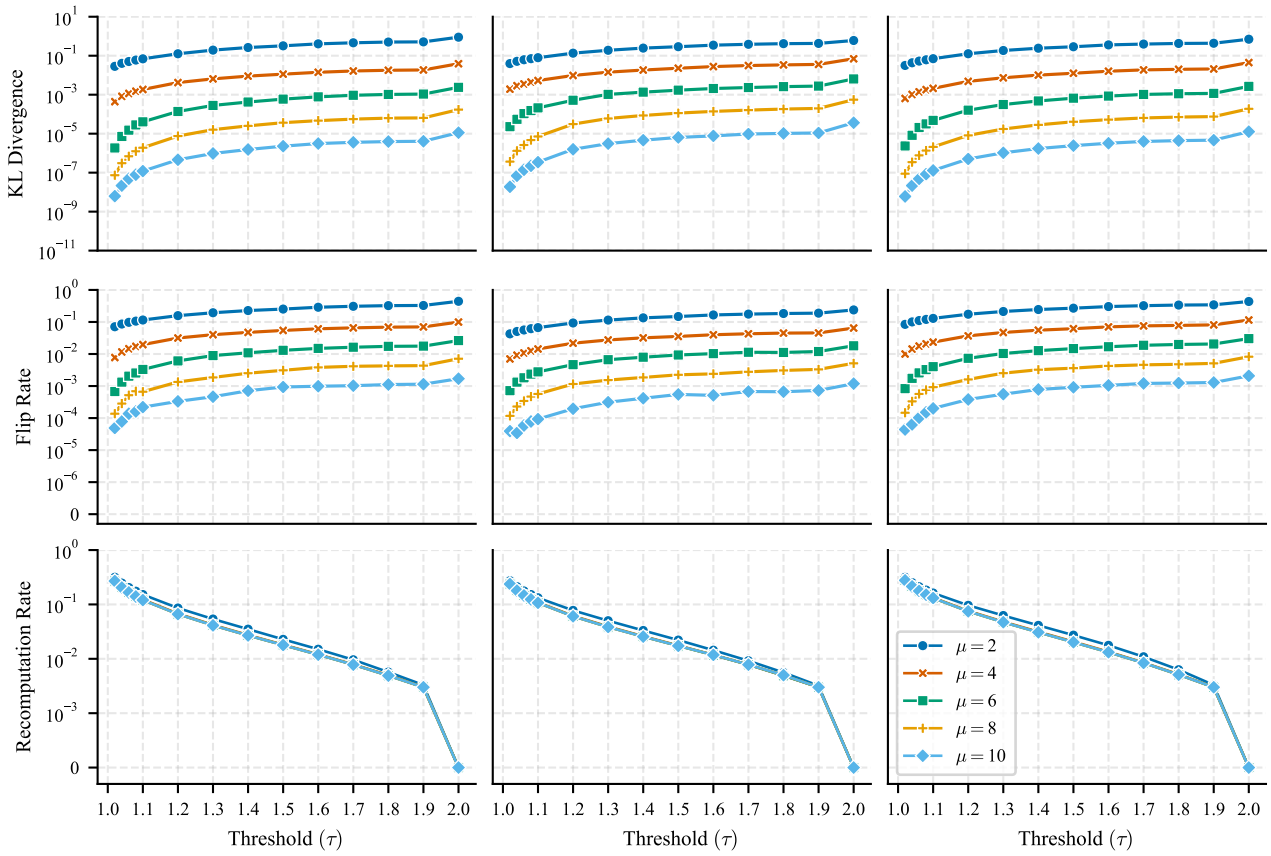

Figure 7. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ).

B.3. GPT-2 small model, three datasets, permuted sequences of tokens

To isolate the effect of word order, we construct three modified datasets by permuting the tokens in each sequence at random. This eliminates all sequential dependencies while preserving the unigram distribution, i.e., the new sequences no longer correspond to a coherent text yet consist of the same tokens in a different order. Figures 8 and 9 show that LAMP inference performs equally well on such "incoherent" data, supporting our claim that the proposed method is "input-agnostic."

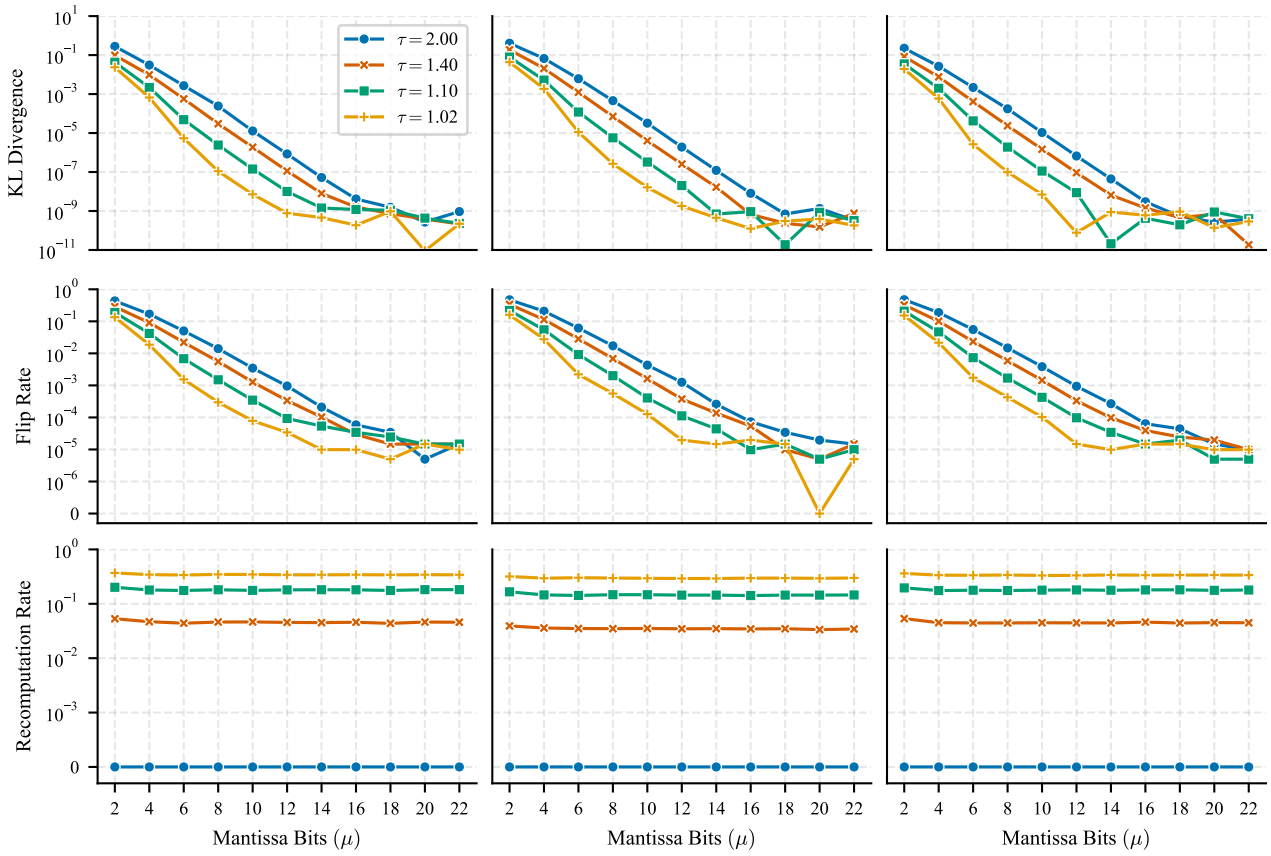

Figure 8. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ). Each sequence of tokens is randomly permuted.

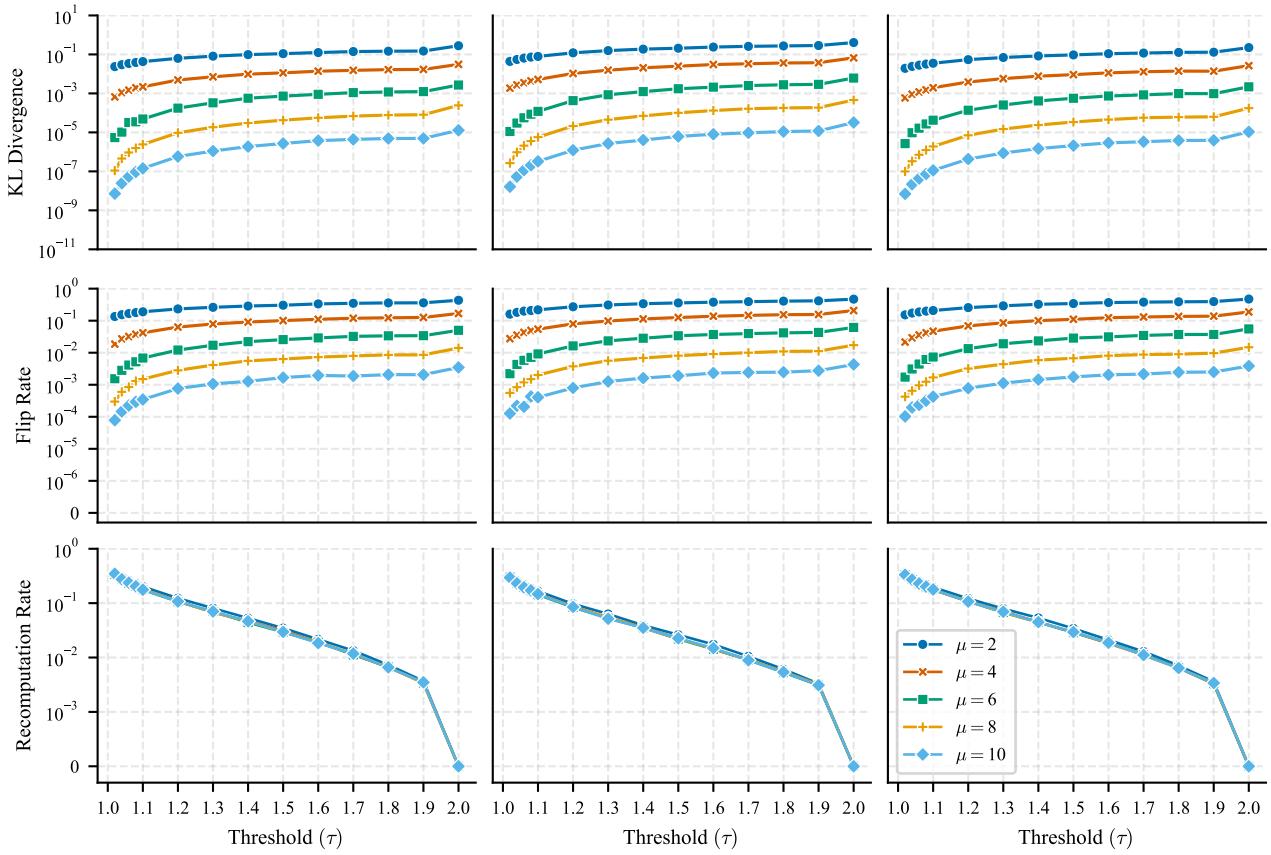

Figure 9. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with LAMP evaluation of the key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ). Each sequence of tokens is randomly permuted.

B.4. GPT-2 XL model, three datasets, random recomputation

The proposed LAMP inference adaptively chooses those key-query inner products that need to be recomputed more accurately. To show that it is not just the *number* of recomputations that is significant, but the actual *choice* of them, we perform experiments with *random* recomputations: the number of inner products to redo is chosen by LAMP, but the specific inner products are selected at random. An experiment of this sort was presented in Figure 1, and we show more results here. Figures 10 and 11 leave no doubt that the adaptive choice of the recomputations is the crux of our method, since random recomputations do not lead to any improvements.

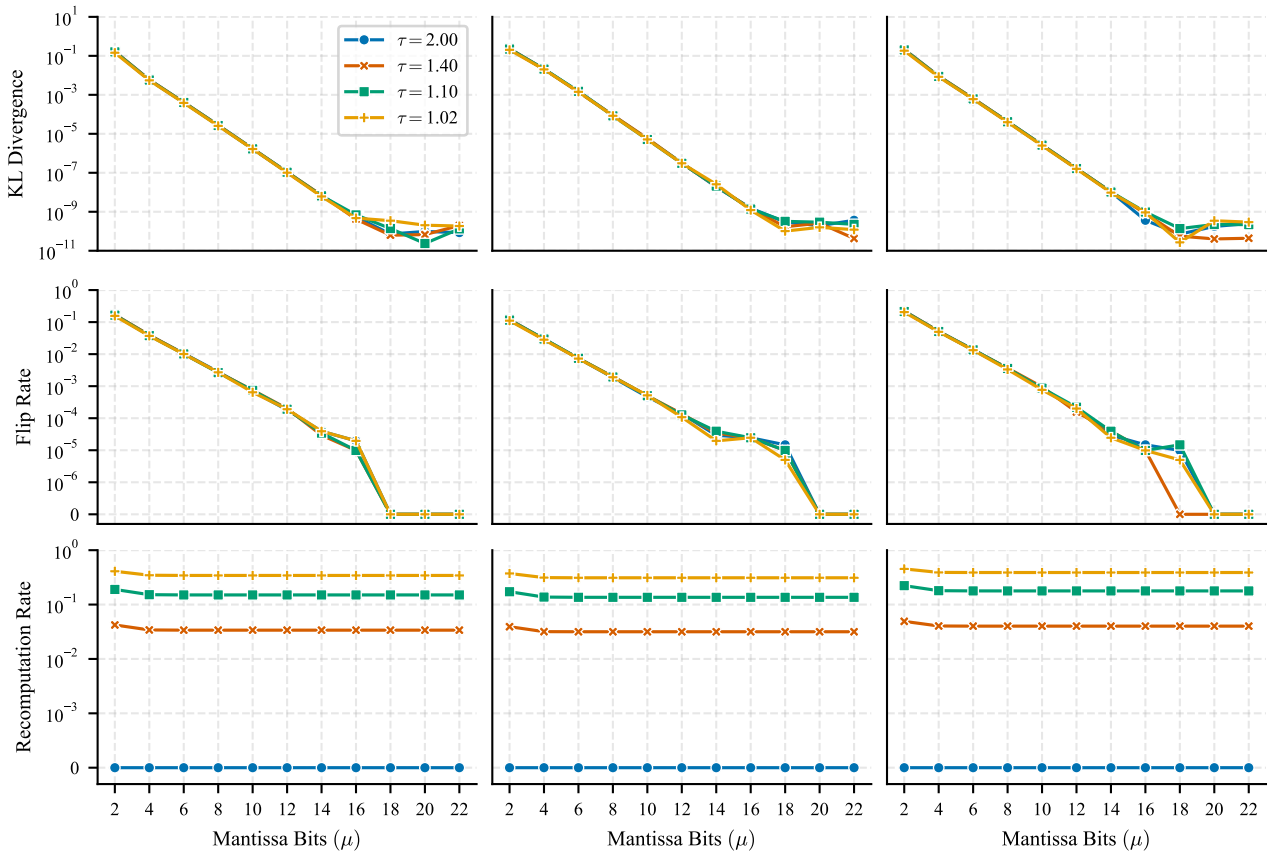

Figure 10. Performance of mixed-precision GPT-2 XL inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with recomputation of randomly chosen key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ).

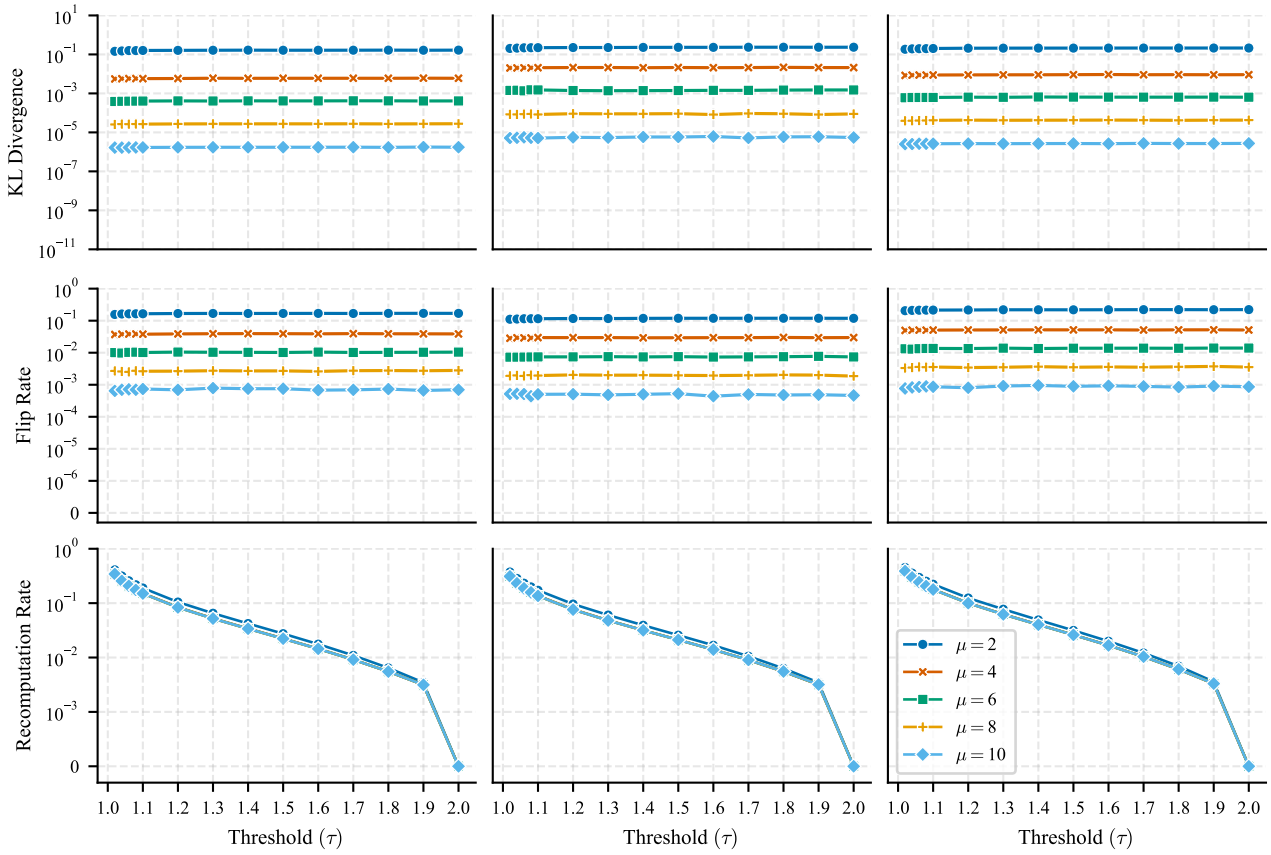

Figure 11. Performance of mixed-precision GPT-2 XL inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with recomputation of randomly chosen key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ).

B.5. GPT-2 small model, three datasets, random recomputation

Here, we repeat the experiments with random recomputations for the smaller model. The results in Figures 12 and 13 lead to the same conclusion: adaptive choice of recomputations in LAMP evaluation is critical for mixed-precision inference.

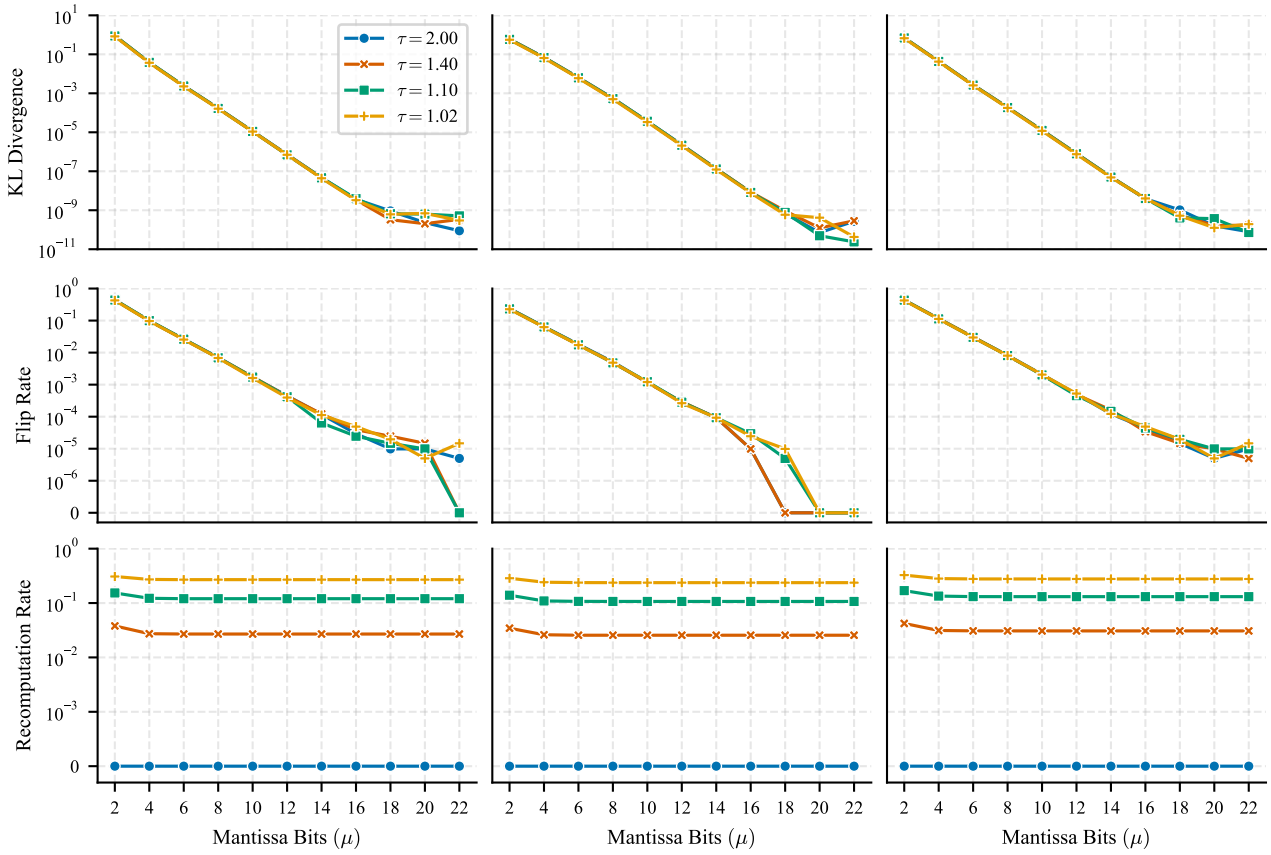

Figure 12. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with recomputation of randomly chosen key-query inner products: varying number of mantissa bits (μ) and fixed threshold of LAMP (τ).

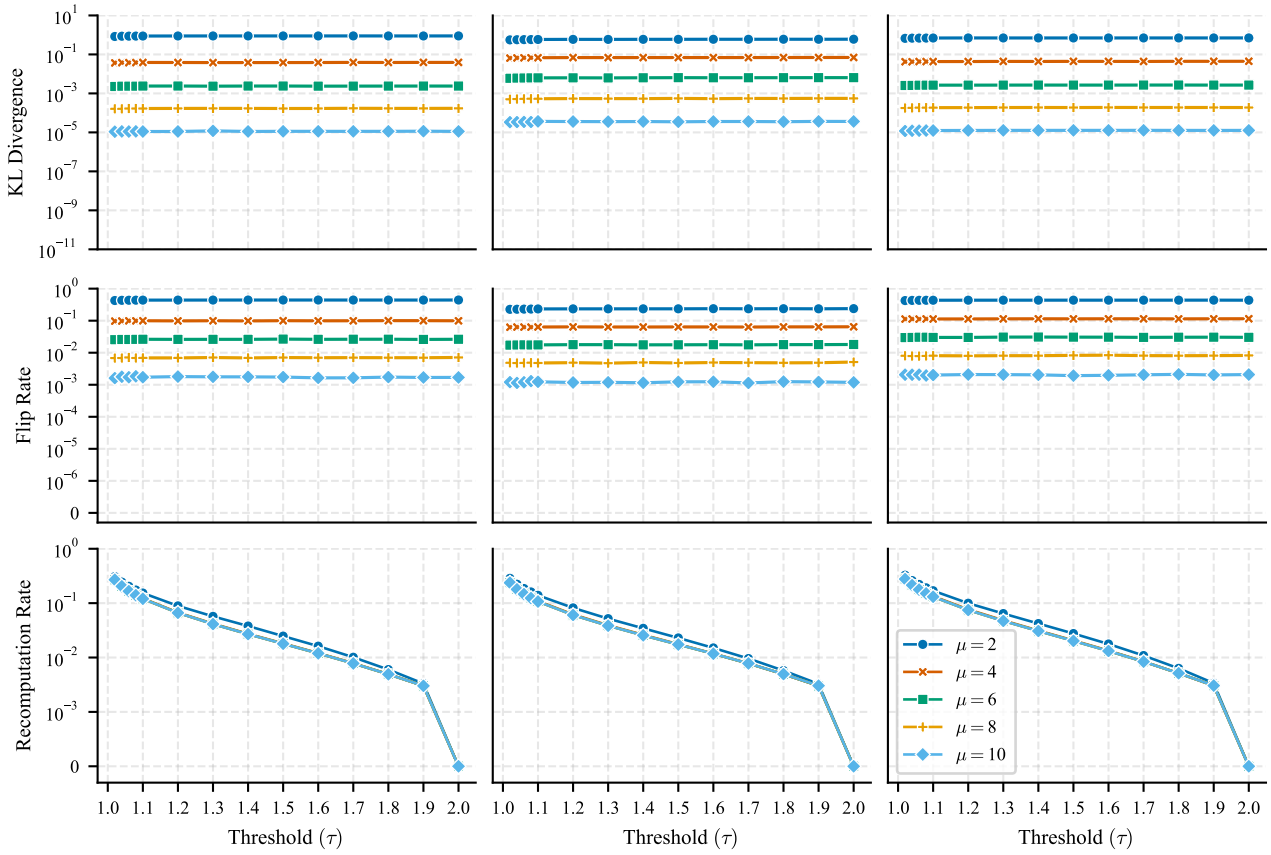

Figure 13. Performance of mixed-precision GPT-2 small inference on the OpenWebText (left), CodeParrot (center), and ArXiv (right) datasets with recomputation of randomly chosen key-query inner products: fixed number of mantissa bits (μ) and varying threshold of LAMP (τ).