Accurate Models of NVIDIA Tensor Cores

FAIZAN KHATTAK, University of Leeds, Leeds, United Kingdom of Great Britain and Northern Ireland
MANTAS MIKAITIS, School of Computer Science, University of Leeds, Leeds, United Kingdom of Great Britain and Northern Ireland

Matrix multiplication is a fundamental operation in both training of neural networks and inference. To accelerate matrix multiplication, Graphical Processing Units (GPUs) provide it implemented in hardware. Due to the increased throughput over the software-based matrix multiplication, the multipliers are increasingly used outside AI, to accelerate various applications in scientific computing. However, matrix multipliers targeted at AI are at present not compliant with IEEE 754 floating-point arithmetic behaviour, with different vendors offering different numerical features. This leads to non-reproducible results across different generations of GPU architectures, at the matrix multiply-accumulate instruction level. To study numerical characteristics of matrix multipliers—such as rounding behaviour, accumulator width, normalization points, extra carry bits, and others—test vectors are typically constructed. Yet, these vectors may or may not distinguish between different hardware models, and due to limited hardware availability, their reliability across many different platforms remains largely untested. We present software models for emulating the inner product behavior of low- and mixed-precision matrix multipliers in the V100, A100, H100 and B200 data center GPUs in most supported input formats of interest to mixed-precision algorithm developers: 8-, 16-, and 19-bit floating point. These matrix multiplier models are first approximated by determining the numerical features via test vectors designed to trigger outputs sensitive to bit level differences in the implementation, followed by semi-exhaustive comparison (randomised input vectors of 10^7 values) between the models and the actual GPU matrix multipliers—this process is repeated until the model is bit-accurate. These models enable verification of test vectors before applying them to real hardware and also support computational scientists and mixed-precision algorithm developers with easy-to-use accurate models available in MATLAB—we demonstrate their use in multi-word emulation algorithms for matrix multiplication.

Additional Key Words and Phrases: Tensor cores, mixed-precision computing, matrix multiply, inner product, IEEE 754 standard arithmetic

The software associated with this paper, the `MATLAB Tensor Core v0.5`, which includes various NVIDIA GPU tensor core models as well as a generalised model that can be used to instantiate custom tensor core variants, is available on GitHub: https://github.com/north-numerical-computing/MATLAB-tensor-core.

1 Introduction

Most recently released GPUs incorporate specialized matrix multiplier units, often referred to as *tensor cores* or *matrix cores*, which are designed to accelerate the GEneral Matrix Multiply (GEMM) operation for AI workloads. Beyond neural networks, these units are also extensively used to accelerate fundamental linear algebra kernels used in high-performance computing (HPC) applications outside of AI [7, 13, 16]. Although some applications may tolerate numerical inaccuracies, this is certainly not the case for all workloads. For example, tensor cores

Authors' Contact Information: Faizan Khattak, University of Leeds, Leeds, United Kingdom of Great Britain and Northern Ireland; e-mail: f.a.khattak@leeds.ac.uk; Mantas Mikaitis, School of Computer Science, University of Leeds, Leeds, United Kingdom of Great Britain and Northern Ireland; e-mail: m.mikaitis@leeds.ac.uk.

ACM 1544-3973/2026/7-ART
https://doi.org/10.1145/3830409

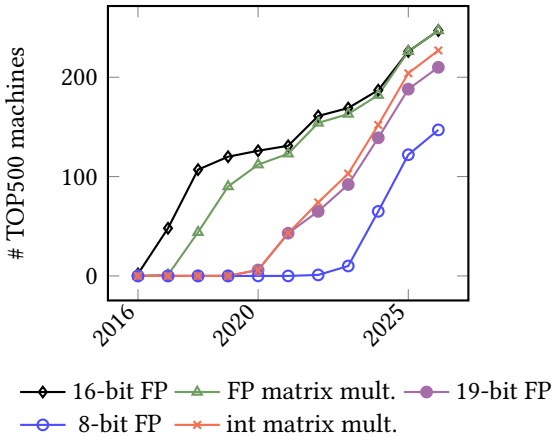

Fig. 1. Number of machines on the June 2026 TOP500 list that support low-precision floating-point formats, and low- and mixed-precision matrix multiplication operations. NVIDIA, AMD, and Intel GPUs are included in the counts.

are increasingly utilized in applications such as Fourier transforms [8, 22], beamforming [31], MR image reconstruction [24], finite element simulations [39], high-precision dense matrix multiply emulation [26], and various others [11, 37]. At the same time, almost half of the machines on the June 2026 TOP500 list[1] contain low-precision matrix multiplication operation in hardware (Fig. 1).

Support for very low-precision, 8-, 6-, and even 4-bit, floating-point formats (albeit with higher precision accumulation in the inner products) is widely available in AMD [3] and NVIDIA [30] GPU architectures. In this paper we will focus on NVIDIA matrix multipliers, informally known as *tensor cores*. Tensor cores generally do not adhere to any numerical standards, likely due to the absence of mixed-precision arithmetic standards at the time of their introduction in the NVIDIA V100 GPU in 2017. Until the establishment of relatively recent Open Compute Project [27] and IEEE P3109 [1] standardisation activities, there was no standard for low-precision number formats and arithmetic. Consequently, even today, AI hardware vendors implement mixed-precision matrix multipliers in various ways, and the computed matrix products across architectures and vendors may differ. While some of the latest hardware implements the OCP data formats [27, 34], the OCP does not standardise arithmetic behaviour.

Mixed-precision tensor cores do not conform with the IEEE 754 [18] floating point standard, because that would require relatively expensive normalisation and rounding logic to be evaluated on every fused multiply-accumulate (FMA) operation within the matrix multiply-accumulate units, which may not be justified for AI workloads that tend to tolerate rounding errors. The features of interest (that cause most impact to the matrix products) include normalization points in multi-term floating-point addition, the rounding modes at various points of computation, the block FMA size (number of additions in a dot-product-add operation), and the use of extra bits in the intermediate alignments of significands of products, amongst other design decisions. The lack of standardization across vendors makes reproducibility and explainability of numerical results particularly challenging in scientific computing applications which may rely on cross-platform consistency. It can also cause difficulties for performing accurate rounding error analysis of algorithms that utilise mixed-precision matrix multipliers.

[1]https://top500.org/lists/top500/list/2026/06/

To address these challenges, the identified numerical features can be abstracted into bit-accurate behavioral models that capture the characteristics of matrix multipliers. These models provide a software-level representation of hardware behaviour, enabling architecture research without requiring direct access to proprietary GPU implementations or specialised hardware knowledge. Furthermore, such models allow reasoning about the numerical accuracy of scientific computing applications that utilise matrix multipliers on data centre GPUs.

In addition, these models may help improve numerical reproducibility and stability in large-scale numerical simulations [37]. Undocumented hardware-level numerical effects that contribute to variability in machine learning systems [12], once characterized, can help isolate true algorithmic behavior from hardware-induced artifacts, thereby improving experimental consistency. Moreover, since matrix multiplication is a core computational component of large language model inference, variations in GPU numerical behavior can also lead to differences in inference outputs across hardware platforms [41]. The proposed models make it possible to reproduce and modify such numerical behaviors in software, enabling controlled studies of their impact on accuracy and reproducibility.

1.1 Previous work

The numerical characteristics of mixed-precision matrix multipliers are seldom documented. Nonetheless, recent studies have attempted to characterize some of these features using input vectors, compiled manually or through theorem provers, that trigger special cases in the matrix multipliers of different designs to produce unique results, allowing one to reason about the overall numerical behaviour without checking all the possible inputs. Such studies have been done for AMD, and NVIDIA GPUs [2, 9, 14, 23, 40].

These works have developed conceptual models of matrix multipliers, but we argue, using randomised testing of tensor cores and their models, that refinement is needed to improve their accuracy. The underlying limitation of these prior approaches is that they rely on an assumed feature space and subsequently determine whether such features are present in a given architecture. If a feature lies outside the assumed feature space, these methods are inherently unable to uncover it since no test is created to target it. In contrast, randomized testing compares model predictions against real hardware results over selected ensembles of inputs, enabling the discovery of hidden behaviours that may not belong to the originally assumed feature space, allowing the creation of a feedback loop in the testing methodology.

The goal of this project is to develop accurate MATLAB models of matrix multipliers, refined through a combination of targeted feature testing and randomized testing against GPU-generated results, to enable the mixed-precision research community to perform accurate numerical experiments when developing new algorithms that target these mixed-precision units. We extend the past approaches and also embed them within an iterative technique that allows to refine the models after the initial approximation is obtained.

This type of work goes back to the *Paranoia*[2] software, built in the 1980s, for testing the compliance with the IEEE 754 standard [18]. Several projects followed *Paranoia*. Hillesland and Lastra [17] developed a GPU Paranoia which allowed them to analyse R300 and NV30 devices and found that, for example, basic arithmetic operations were not optimally accurate as prescribed by the *correct rounding* of IEEE 754. FPGA Paranoia of Tan, Boland, and Constantinides [36] similarly tested the arithmetic of various FPGA devices, finding for example that Altera devices rounded division differently from Xilinx devices, by not rounding to the nearest number in some cases.

1.2 Contributions

In this work, we reapply the generalized test vectors proposed in our earlier study [2] to determine the numerical features of NVIDIA's V100, A2, A30, A40, A100, H100, H200, L40S, Ada RTX 1000, RTX PRO 6000, and B200 tensor cores across all supported input and output precisions—40 models in total. The GPU devices we target, along with their corresponding architectures and streaming multiprocessor (SM) compute capabilities, are summarized in

[2]https://www.arithmazium.org/paranoia/aaapara_toc.html

Table 1. NVIDIA GPUs whose models of matrix multipliers are determined in this work, together with their corresponding microarchitecture and SM (streaming multiprocessor) compute capability.

GPU Model	Architecture	SM Compute Capability
V100	Volta	70
A100	Ampere	80
A30	Ampere	80
A40	Ampere	86
A2	Ampere	86
L40S	Ada Lovelace	89
RTX 1000	Ada Lovelace	89
H100	Hopper	90
H200	Hopper	90
B200	Blackwell	100
RTX PRO 6000	Blackwell	120

Table 1. Unlike prior analyses [2, 9, 14, 23, 40], we extend the investigation to include the two 8-bit floating-point formats available on the L40S, Ada RTX 1000, H100, H200, RTX PRO 6000, and B200 GPUs. Based on the identified features, we develop MATLAB-based software models of tensor cores for the aforementioned GPUs, for increasing the productivity and accessibility to tensor cores by mixed-precision algorithm developers and numerical analysts. Features that we cannot find initially are identified by comparing the outputs of the emulated models against results from eleven GPUs, and subsequently verified using targeted test vectors. This iterative approach refines the MATLAB models to a higher level of accuracy than before.

In addition, our MATLAB toolbox provides a customizable tensor core model that allows users to diverge from one of the GPU tensor core models by customizing the precision and rounding behaviour. The models are developed by combining fixed-point arithmetic with the custom floating-point format simulator CPFloat [10]. We also demonstrate the use of the models for the emulation of high-precision matrix multiplication of arbitrary-sized dense matrices via tensor cores [25, 26, 32, 33].

The main contributions of this paper are summarized as follows:

- Determination of various numerical features for NVIDIA V100, A100, A2, A30, A40, H100, H200, B200, L40S, RTX PRO 6000 and RTX 1000 Ada tensor core models, including 8-bit floating-point formats. We have presented these features more precisely than any other previous work, using architectural diagrams.
- Development of MATLAB-based simulation models for GPU tensor cores, validated against GPU results through randomised matrix multiplication input space coverage.
- A customised model of tensor cores in MATLAB, which can be used to easily experiment with custom variants by specifying various features such as rounding modes and internal precision used within tensor cores.
- Experimental results on the differences that tensor cores can cause in a low-level kernel, multi-word matrix multiplication, which serves as an example of how these models can be used by the community.

2 Notations and definitions

Table 2 shows some of the characteristics of various floating-point formats available on the latest NVIDIA Blackwell architecture. Hereafter we refer to floating-point formats with the following short-hand names: fp8

Table 2. Floating-point formats that are available on the latest NVIDIA Blackwell GPUs [30]. Precision of the significand, which includes an implicit bit [18], minimum representable positive normalised value, and an approximate maximum representable positive value, are shown for each format.

Format	precision	min norm. pos.	max pos.
binary64 (double)	53	2^{-1022}	$\sim 1.798 \times 10^{308}$
binary32 (single)	24	2^{-126}	$\sim 3.403 \times 10^{38}$
tf19 (19-bit)	11	2^{-126}	$\sim 3.401 \times 10^{38}$
bfloat16	8	2^{-126}	$\sim 3.389 \times 10^{38}$
binary16 (half)	11	2^{-14}	65504
fp8-E4M3	4	2^{-6}	448
fp8-E5M2	3	2^{-14}	57344
fp6-E2M3	4	2^{0}	7.5
fp6-E3M2	3	2^{-2}	28
fp4-E2M1	2	2^{0}	6

(either of fp8-E4M3 or fp8-E5M2), fp16 (binary16 of IEEE 754 [18]), bf16 (bfloat16), tf19 (TensorFloat32), fp32 (binary32 of IEEE 754), and fp64 (binary64 of IEEE 754).

Take two matrices $A \in \mathbb{R}^{m\times k}$, and $B \in \mathbb{R}^{k\times n}$. The matrix multiply-accumulate (MMA) operation produces

$$D = AB + C \in \mathbb{R}^{m\times n}.$$

Denote with d_{ij} the element at ith row and jth column of D. We can express it as the inner product between the ith row of A and jth column of B as

$$d_{ij} = \sum_{\ell=1}^{k} a_{i\ell} b_{\ell j} + c_{ij}.$$

To focus on the inner product as an underlying operation, rather than on particular elements of D, we omit the subscripts for simplicity, and we have

$$d = \sum_{\ell=1}^{k} a_\ell b_\ell + c = \sum_{\ell=1}^{k} p_\ell + c. \tag{1}$$

The operations of the type (1) are often approximated in hardware by employing a multi-term floating-point adder [28]. In such adders, the significands of addends are aligned relative to the largest exponent either in a single global alignment or combination of local and global alignment steps [15, 21], and then subsequently added via a compressor or a tree of adders with a single rounding and normalisation step. For such many-term dot products, we define the number of product terms p_ℓ added in one go as the size of the *block fused multiply-accumulate*, following the naming convention used in the rounding error analysis of tensor cores by Blanchard et al [4], denoted with N_{FMA}. For instance, the tensor core in the A100 data center GPU has $N_{\text{FMA}} = 8$ for fp16 inputs which means $p_1 + \cdots + p_8 + c$ is computed with a single normalisation and rounding. When such units perform multi-term significand alignment, the bits of each significand that are shifted to the right are truncated or rounded with multiple sticky bits [38]. In the alignment of significands, we denote the number of extra alignment or guard bits beyond the output precision by n_{eab}, similar to our previous work [2]. For instance, it has been shown by several studies that V100 GPU tensor core performs accumulation in 23 fractional bits (the precision of the fp32 format), hence with $n_{\text{eab}} = 0$ [9, 14, 23].

Note that below, we use some abbreviations which are specific to CUDA and PTX ISA. Some of them are `HMMA`, `QMMA`, `DMMA`, `QGMMA`, `UTCHMMA`, `UTCQMMA` with meanings: matrix multiply and accumulate, fp8 matrix multiply accumulate, matrix multiply accumulate, fp8 matrix multiply and accumulate across a *warp group* (a group of parallel threads), uniform matrix multiply and accumulate, uniform matrix multiply and accumulate, respectively [5].

3 Methods

3.1 Generalised Numerical Feature Testing (GNFT)

This method for determining numerical features operates on the principle of using carefully designed expressions for forming test vectors that can trigger bit-level differences in the outputs [2]. For example, consider a test vector designed to determine whether subnormal inputs are supported in the fp16 format [18]. A test vector can be constructed by using constants: $|a_1| < 2^{-14}$ (smallest normalised value in fp16; see Table 2), $b_1 = 1$, and $a_\ell = b_\ell = 0$ for $\ell = 2, \ldots, k$. This can be generalised for any format, by using an expression for the smallest normalised value.

When applying this test vector to a tensor core, if the resulting output d is non-zero, it indicates that subnormal inputs are supported—an outcome that has to be formulated by understanding the IEEE 754 floating-point arithmetic. Similarly, $a_1 = 2^{-14}$ and $b_1 = 2^{-1}$ would demonstrate if subnormals can be produced by arithmetic operations from normalised values. Several such numerical feature–specific test vectors using constant values have been proposed in the literature [9, 14, 23]. A generalized formulation, which addresses the limitations of earlier approaches by avoiding manually deriving constants or rerunning a theorem prover that has no upper bound on the run time [40], for each format/device, has been explored by us [2]—we rely on that approach here to determine the initial approximation of numerical features of tensor cores in eleven NVIDIA GPU variants and develop accurate models of them.

3.2 Input Space Search Method (ISSM)

Once the GNFT method has identified the numerical features, a conceptual or software-based model of the hardware matrix multiplier can be constructed to simulate its behavior. To achieve higher confidence in this model, in comparison to the GPU implementation, the ISSM method is invoked to look for input vectors to (1) for which the GPU results deviate from those produced by the software model approximated by GNFT. Such discrepancies can then be analyzed to refine the model until the tests pass. When mismatches occur, a generalised test vector can be constructed to specifically detect and characterize such numerical features when analysing a new hardware. This expands the feature space and therefore, simplifies the process of determining new architectures.

It is worth noting that the input space of (1) is generally large and only a small proportion of it can be checked. For example, for $k = 16$ and 8-bit floating-point as an input format, the input space pair $\{a, b\}$ has approximately $256^{32} \approx 10^{77}$ possible inputs. For this work we have chosen to sample a, b, and c from a standard normal and uniform distributions along with carefully designed cases which are detailed in Section 4.2.

For each of the GPUs we have tested, some of them have up to five input formats: fp8 (both E4M3 and E5M2), fp16 (both fp16 and fp32 output mode), bf16, and tf19, requiring a separate verification for each. In total, we have verified 11 models of tensor cores, each with several input/output format combinations, giving 43 model verification runs in total with randomized input vectors (see Table 3).

3.3 Matrix multiplier model approximation and refinement

Our proposed algorithm for determining an accurate model of a tensor core is shown in Algorithm 1.

The step that modifies the model (line 8) causes the biggest challenge in automating the whole process, because it requires an expert to look into the mismatching outputs between the GPU and the model, and develop hypotheses

Algorithm 1 Pseudocode for approximation and refinement of models of matrix multipliers.

1: **Machine step:** Apply GNFT using generalised test vectors [2].
2: **Engineer step:** Create an initial approximation of the model.
3: **while** true **do**
4: **Machine step:** Generate an ISSM ensemble as described in Section 4.2 using Algorithm 2.
5: **Machine step:** Evaluate both the GPU and the approximate model over the entire ensemble.
6: **if** mismatches are detected between GPU and model outputs **then**
7: **Engineer step:** Inspect failure cases.
8: **Engineer step:** Refine the model.
9: **Engineer step:** Modify ISSM to include similar cases to the failure cases.
10: **Engineer step:** Add test expressions to GNFT for detecting any new features.
11: **else**
12: **Engineer interpretation:** The model is considered sufficiently accurate.
13: **break**
14: **end if**
15: **end while**

about the features of the model that need to be changed to match the GPU output. Through Algorithm 1 a lot of the manual work is removed by using the generalised testing vector to get the first approximation of the model and refinement, which means that Algorithm 1 can significantly accelerate the determination of features of future tensor cores. However, the full automation of Algorithm 1 is an open problem.

4 Results

4.1 Accurate GPU matrix multiplier models

Numerical features of matrix multiplier units on AMD and NVIDIA have been determined up to a certain degree of accuracy [9, 14, 23, 40]. Focusing on NVIDIA GPUs, we argue that previous work essentially corresponds only to the first step of Algorithm 1, albeit not using generalised tests but constant-based format-specific expressions, and that the resulting models remain inaccurate without further refinement. Previous approaches require manual porting of tests in line 1 of Algorithm 1 for each new GPU/format combination, but with our GNFT [2] this limitation is removed.

The numerical features of all eleven GPU tensor cores are summarized in Table 3, including the product alignment bits, accumulator output precision, N_{FMA}, final rounding mode to output precision, and other relevant details.

4.1.1 V100 data center GPU. Mixed-precision matrix multiplication units, or tensor cores, were introduced in the NVIDIA Volta architecture, namely, the V100 device. The V100 tensor cores support only fp16 as the input format with fp16 or fp32 as the output format [29]. Xi et al. [23] claim that the FMA size cannot be determined when there are no extra alignment bits compared with the precision of the output format. Following [2] we denote this situation with $n_{\mathrm{eab}} = 0$. However, using the FMA size, N_{FMA} (detection algorithm proposed in [2]), we were able to identify $N_{\mathrm{FMA}} = 4$, which we could determine irrespective of the value of n_{eab}. In addition, when the significands are aligned, bits that fall outside the internal accumulator's precision are truncated rather than accumulated into multiple sticky bits as is done in the algorithm by Tenca [38]. Further tests on the V100 GPU using this approach led us to the estimated tensor core model shown in Fig. 2, which differs from the model presented in [19, Fig. 2] in that the significand alignment is performed in 25 bits rather than 24, and c_1 is added together with the products rather than separately.

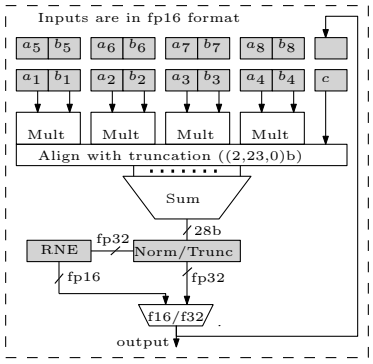

Fig. 2. A model of the inner product within the V100 GPU tensor core. Here, RNE denotes round-to-nearest ties-to-even [18] rounding mode.

In our experiments, the existent conceptual model, when simulated in MATLAB, produced different results from the GPU for certain input combinations in the inner loop of Algorithm 1. Upon analysis, we determined that in some cases, partial products are not normalized before accumulation, which can result in variations even for apparently identical products. For instance, consider the case with $c = 0$, $p_1 = 2.25$, and $p_2 = p_3 = 2^{-23}$. Now consider two alternative ways to obtain the product $p_1 = 2.25$: either by taking $a_1 = b_1 = 1.5$, or $a_1 = 1$, $b_1 = 2.25$. For the first case, the tensor core computed $d = 2.25 + 2^{-22}$, whereas for the second case, $d = 2.25$. In the first case, p_1, which has the largest magnitude, is added in a denormalised form ($10.01_2 \times 2^0$), and hence p_2 and p_3 are not truncated. In contrast, in the second case, since $p_1 = 01.001_2 \times 2^1$, the remaining two products p_2 and p_3 fall outside the representable range (beyond the 23rd fractional bit), resulting in $d = 2.25$. On the other hand, if $c = 2.25$ with $p_1 = 0$, $p_2 = p_3 = 2^{-23}$, we have $d = c$. This feature is depicted in Fig. 2 as $(2, 23, 0)$ bits at alignment stage where 2 represents integer bits, 23 fractional bits, and 0 the n_{eab} bits. It is worth noting that this test can be generalized, similar to our previous work [2] as it does not rely on the specific value such as 2.25; rather, any product greater than 2 and less than 4 can be used where the corresponding a and b have values $2 > a, b \geq \sqrt{2}$. One possible explanation for this behaviour is that it improves numerical accuracy because the products that need normalising do not lose one bit. This feature also means that the products may be allowed to exceed 2^{128} (not representable in fp32; see Table 2) and be used in the subsequent summation. This is not possible to be tested with the fp16 input format where the maximum product exponent remains capped at 30 for denormalised and 31 for normalised products.

4.1.2 A100 data center GPU. Now we turn to several tensor core variants in the A100.

In the fp32 output mode with fp16/bf16 as the input format, $N_{\text{FMA}} = 8$, and $n_{\text{eab}} = 1$ is used for the alignment and accumulation of significands of products p_ℓ. The rounding mode at both the alignment and post-normalization stages is truncation, whereas in fp16 output mode the rounding mode after normalization is Round-to-Nearest Ties-to-Even (RNE). The alignment of the significands is 26 bits wide, with 2 integer bits and 24 fractional bits, while the adder output must be

$$26 + \lceil \log_2(9) \rceil = 30 \text{ bits}$$

due to extra bits required for accumulation carries. We note that only three extra carry bits can be detected through testing [2], and although a fourth carry bit is assumed to exist by logic, to the best of our knowledge it

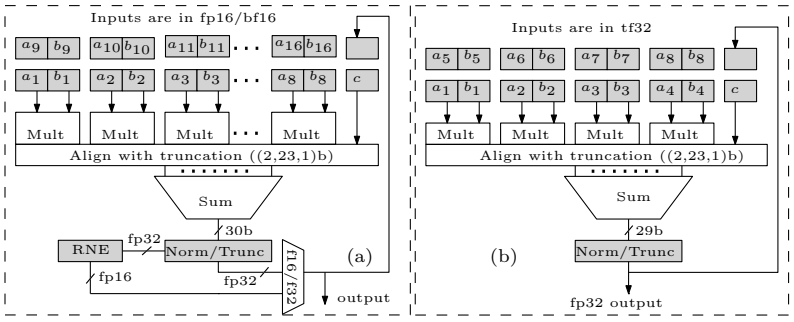

Fig. 3. A model of the inner product within the A100 GPU tensor core for the three input formats. A100 also has an fp64 tensor core, but that tensor core is compliant with the IEEE 754 FMA operation and is not shown here.

cannot be verified through numerical tests unless we assume that there is no intermediate normalisation [40] when computing (1).[3] The model that we have determined is shown in Fig. 3 (a).

For the tf19 input mode, the $N_{\text{FMA}} = 4$, with truncation as the default rounding mode. The model is shown in Fig. 3 (b). The bit-width of the denormalised sum must be 29-bit wide.

The products remain denormalised for fp16/bf16/tf19 input modes, as previously discussed in the context of V100 and fp16. This behaviour can be verified by setting $c = 0$, $p_1 = 2.25$ where $a_1 = b_1 = 1.5$, and $p_2 = 2^{-23}$, $p_3 = p_4 = 2^{-24}$. When $d = 2.25 + 2^{-22}$, the products are added in denormalised form. The difference in this test compared to the V100 lies in the presence of an extra alignment bit ($n_{\text{eab}} = 1$) in the A100, which slightly alters the accumulation behaviour, making this test dependent on knowing the n_{eab} value.

There is a nontrivial numerical behaviour in bf16 and tf19 input formats, which we discovered by exploring the mismatch between our model and the GPUs, reported on GitHub by a user. This occurs within the alignment of product significands, when all product exponents fall within the subnormal range of the fp32 format. Although product exponents are allowed to go as low as -266 for bf16 and -272 for tf19 inputs, the maximum exponent used for aligning significands during accumulation is capped from below; we denote this cap by $e_{\text{min}}^{\text{align}}$. We observed the following cases:

(1) when $c = 0$, and the product exponents are less than or equal to -132, the maximum exponent used for alignment is fixed at $e_{\text{min}}^{\text{align}} = -132$, even if the largest exponent among the products is smaller. For example:
- setting $c = 0$, $p_1 = \sum_{i=-150}^{-155} 2^i$, $p_2 = p_3 = 2^{-156}$ results in $d = 2^{-149}$.
- for $c = 0$, $p_1 = \sum_{i=-150}^{-155} 2^i$, $p_2 = 2^{-156}$, $p_3 = p_4 = 2^{-157}$, we get $d = 0$.

This shows that $-132 + 156 = 24$ fractional bits are available on the A100 tensor core accumulator. In the second case, 25 fractional bits are required to produce 2^{-149} as the output—truncation occurs and the result becomes zero.

(2) when c is a nonzero subnormal value (i.e., $0 < |c| < 2^{-126}$) and all product exponents are in the fp32 subnormal range, the maximum exponent for alignment is set to -126 instead of -132. This is verified by setting $c = 2^{-127}$, $p_1 = 2^{-150}$, $p_2 = p_3 = 2^{-151}$ which produced $d = 2^{-127}$ instead of $d = 2^{-127} + 2^{-149}$.

(3) when $c = 0$ and some or all product exponents are between -132 and -126, the maximum product exponent itself is used for alignment rather than the fixed value -132.

[3] A test vector is needed, that can distinguish normalisation followed by the addition of the final addend, versus addition of the final addend to an accumulator that is in a denormalised form with the fourth carry bit set, followed by the final normalisation and truncation instead of rounding.

In fp64 input/output mode, we have $N_{\text{FMA}} = 1$. This configuration of a tensor core is compliant with IEEE 754 with RNE as the default rounding mode, but supports RD, RU and RZ. The observed accumulation order appears to follow

$$((c + p_1) + p_2) + \ldots,$$

as indicated by permuting the values 1, 2^{-53}, and 2^{-53} among c, p_1, and p_2. Only for the case $c = p_1 = 2^{-53}$ and $p_2 = 1$ does the GPU's tensor core produce $d = 1 + 2^{-52}$; for all other permutations, it returns $d = 1$ under the default RNE rounding mode. This behavior further suggests that the FP64 tensor core performs sequential FMA operations, where each intermediate result is fed back as the new input c.

4.1.3 A2 & A30 data center GPUs. Executing the GNFT on these GPUs and using the ISSM testing, we were able to determine that the tensor cores in these GPUs behave identically to the tensor cores of the A100 GPU, except that they do not have a fp64 variant.

4.1.4 L40S data center GPU. Running the GNFT for fp16, bf16, and tf19 input formats, we determined that the L40S tensor cores exhibit behaviour identical to that of the A100, with N_{FMA} of 8, 8, and 4, respectively, and with a single extra alignment bit i.e., $n_{\text{eab}} = 1$.

The warp matrix multiply accumulate (WMMA) API does not provide support for fp8 format tensor cores available on this GPU. Therefore, the MMA instruction, specifically `mma.sync.aligned.m16n8k16/32.f32/f16.f8.f8.f32/f16`[4], is utilized which then maps to `QMMA` instruction, for computing fp8 MMA across a warp [5, Sec. 6.4]. However, although the L40S GPU belongs to the Ada architecture family, the QMMA instruction is not explicitly documented in the Ada instruction set [5, Sec. 6.2]. For both fp8 formats, fp8-E5M2 and fp8-E4M3, the accumulation takes place with 13 fractional bits. We determined this by setting $p_1 = 1$, $p_2 = p_3 = 2^{-23-n_{\text{eab}}}$ and decrementing n_{eab} from 10 until d becomes equal to $1 + p_1 + p_2$, using the methodology of Hickmann and Bradford [14]. For $n_{\text{eab}} = -10$, we obtain $d = 1 + 2^{-12}$, which indicates that there are $23 - 10 = 13$ fractional bits. Moreover, we test that the accumulation term c is summed together with the products. We verified it by setting $c = 1$ and $p_1 = p_2 = 2^{-14}$, which produced $d = 1$ in the fp32 output mode. As L40S fp16, bf16 and tf19 input format tensor cores are identical to A100 tensor core, only fp8 input format tensor core structure is shown in Figure 4. The A100 feature of limiting the maximum exponent at −132 when small subnormals are accumulated, is also available in this GPU.

4.1.5 Ada Lovelace RTX 1000 consumer-grade GPU. Tensor cores on this GPU exhibited identical numerical characteristics to those of the L40S model, and produced identical outputs for 10^5 and 10^7 random input vectors. The $e_{\text{min}}^{\text{align}}$ parameter is −132.

4.1.6 H100/H200 data center GPU. For fp16/bf16 input formats, the H100 tensor core is reported by Li et al. [23] to utilize two or more extra alignment bits, with fp32 as an output format, when the significands of the products are aligned for multi-term addition and summed. Using our generalised test vectors [2], we confirm that the H100 actually employs exactly $n_{\text{eab}} = 2$ extra alignment bits; no third alignment bit is present, and the bits of aligned significands that are out of range are truncated for fp16/bf16/tf19 input formats. As a result, the alignment stage is 26-bits wide. However, since the products are added in denormalised form, same as in the V100 and A100 devices, the width of the significand alignment is $(2, 23, 2) = 27$-bit wide, i.e., 2 integer, 23 fractional bits, and extra 2 bits of the fraction. This feature can be verified by setting $c = 0$, $p_1 = 2.25$, where $a_1 = b_1 = 1.5$, $p_2 = 2^{-23} + 2^{-24} + 2^{-25}$, and $p_3 = 2^{-25}$. If $d = 2.25 + 2^{-22}$, the largest product is kept denormalised, otherwise the output should be $d = 2.25$.

[4]The input matrices A and B may have a shared dimension of either 16 or 32. The output type can be either fp32 (denoted as `f32`) or fp16 (denoted as `f16`) in this PTX format.

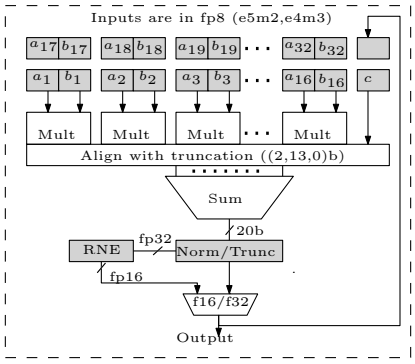

Fig. 4. A model of the inner product within the tensor cores of the L40S and Ada Lovelace RTX 1000 GPU for the fp8 input format. For fp16, bf16 and tf19, the model is identical to A100 and is not shown.

Furthermore, for the fp16 and bf16 input modes $N_{\text{FMA}} = 16$, which equals the maximum supported shared dimension of the matrix sizes available in the WMMA API. Since $N_{\text{FMA}} = 16$, the denormalized adder result has a width of $27 + \lceil \log_2(17) \rceil = 32$ bits. Upon normalization, results are truncated rather than rounded. The fp16 format for both the input and output is supported, and the default rounding mode from the internal precision to the fp16 output is RNE.

Using the WMMA API with the only available shape for the input matrix, i.e., `m16n16k8`, for tf19 input format, the instructions are mapped to `HMMA` but with fragment size of `m16n8k4` which results in $N_{\text{FMA}} = 4$ via GNFT. However, if `mma.sync.aligned` instruction is used with the matrix shape of `m16n8k8`, it is internally mapped to `HMMA.1688` which then yields $N_{\text{FMA}} = 8$ through GNFT. We report the maximum, $N_{\text{FMA}} = 8$, in the model of H100 and H200 (see Figure 5). The $e_{\min}^{\text{align}} = -133$, unlike -132 in the case of Ampere and Ada architectures, when $c = 0$ is also present. However, the test case derived for the A100 is not directly applicable to H100/H200, as the tensor cores in the Hopper architecture have $n_{\text{eab}} = 2$. We now have the following.

(1) when $c = 0$, the maximum exponent used for alignment stage is limited at -133, as shown by
   - setting $c = 0$, $p_1 = \sum_{i=-150}^{-156} 2^i$, $p_2 = 2^{-157}$, $p_3 = p_4 = 2^{-158}$ results in $d = 2^{-149}$.
   - for $c = 0$, $p_1 = \sum_{i=-150}^{-156} 2^i$, $p_2 = 2^{-157} + 2^{-158}$, $p_3 = p_4 = 2^{-159}$, we get $d = 0$.

   Since $n_{\text{eab}} = 2$, there are 25 fractional bits i.e., $-133 + 158 = 25$, which prevents truncation of the $= p_3 = p_4 = 2^{-158}$ term whereas the products less than 2^{-158} get truncated, producing $d = 0$.
(2) when c is a nonzero subnormal value, and all product exponents are in the fp32 subnormal range, the maximum exponent for alignment is set to -126. This is verified by setting $c = 2^{-127}$, $p_1 = 2^{-151}$, $p_2 = p_3 = 2^{-152}$ which produces $d = 2^{-127}$ instead of $d = 2^{-127} + 2^{-149}$, while $p_2 = p_3 > 2^{-152}$ does not result in truncation.

With the WMMA API limited to fp16, bf16, tf19 and binary64 formats, `mma.sync.aligned` instruction can be used to multiply fp8 input format matrices, as in Ada RTX 1000 and L40S. However, unlike Ada RTX 1000 or L40S where this is mapped to `QMMA`, in the case of H100/H200, we observed that fp8 inputs are first converted to fp16, after which the `HMMA` instruction is invoked at the SASS level[5] with a matrix shape of `m16n8k16`. This indicates that a direct `QMMA` instruction is not available in the Hopper architecture, and fp8 inputs issued via `mma.sync.aligned` are effectively processed using fp16 tensor cores. The internal usage of fp16 tensor core

[5]SASS is a low-level assembly language that compiles to binary microcode, which executes natively on NVIDIA GPU hardware, as per https://archive.docs.nvidia.com/gameworks/content/developertools/desktop/ptx_sass_assembly_debugging.htm

for fp8 input matrices results in interesting features for fp8. We observed that the inner product is computed in interleaved fashion where 32-element input vectors are distributed across two tensor core invocations by alternating pairs of elements. Once the products are added together, c is then added to the normalised sum of products, with RNE as the rounding mode (see Figure. 5c). This *late addition* of c is only present in this case, while in all other cases, c is summed with the products.

The interleaved input pattern is detected in both cases for `m16n8k16` and `m16n8k32` matrix shapes. It can be verified by fixing $p_1 = 1$ and $p_2 = 2^{-24}$, and then assigning the value 2^{-24} sequentially to p_3 through p_{32} while keeping all other entries at zero. If $d = 1 + 2^{-23}$, the tested product falls into the same accumulation group as p_1 and p_2. Conversely, if $d = 1$, the tested product belongs to the second group. If the two groups were not interleaved, we would observe $d = 1 + 2^{-23}$ only when assigning 2^{-24} to positions p_3–p_{16}, and $d = 1$ when assigning it to positions p_{17}–p_{32}.

The Hopper architecture supports a specific *warp-group* level `mma` instruction `wgmma.mma_async.sync`, which is internally mapped to `QGMMA`, responsible for fp8 MMA across a warpgroup (a set of four contiguous warps) [5, Sec. 6.3]—the low-level instruction which supports native fp8 format tensor core access on Hopper architecture. With this, the GNFT shows $n_{\text{eab}} = -10$, i.e., 13 fractional bits and $N_{\text{FMA}} = 32$. The behaviour of the fp8 format tensor core for H100/H200 is identical to that of the Ada Lovelace architecture, but with twice the N_{FMA}. The model diagram for H100 and H200 is shown in Figure 5d.

4.1.7 B200 data center GPU model. B200, based on the Blackwell architecture, is the latest iteration of the GPU architecture that is commercially available at the time of writing. In our testing, the B200 tensor cores demonstrated identical numerical behaviour to the H100/H200 models (Fig. 5) for 16- and 19-bit input formats. For fp32 outputs, with fp16, bf16 or tf19 inputs, the accumulator provides 2 extra bits, and the FMA tile sizes are 16, 16 and 8, respectively. The product exponent limitation during significand alignment, when the product lies in the subnormal region, is identical to that observed on H100 and H200 GPUs. When we tested the fp8 tensor core through the `mma.sync.aligned` PTX instruction, we observed an identical behaviour as that in the case of H100/H200, where fp8 input vectors are converted to fp16 and `HMMA` instruction is called at `SASS` level which means that fp16 tensor cores are utilized instead of fp8 cores. Identical interleaving pattern is detected (see Figure 5(c)). Even though `QMMA` is specifically mentioned in the instruction set of Blackwell GPU [5, Sec. 6.4], our experiments show that `mma.sync.aligned` instructions are mapped to `HMMA` not `QMMA` and `wgmma.mma_async` is not supported. This confirms that `QMMA` requires PTX instructions other than `mma.sync.aligned`. Our experience is consistent with the findings of Jarmusch et al. [20].

To access the 5th generation tensor cores available on the B200, PTX instruction of `tcgen05.mma.cta_group_1/2::kind` is invoked where `kind` defines the input type and takes on values f16 (supporting both fp16 and bf16), tf32 (tf19), and f8f6f4 (all combinations of fp8, fp6, and fp4, of which the latter two we did not address in our work). For `kind` set to f16/tf32, it maps to `UTCHMMA`, having the same numerical behaviour as that of the `HMMA`. While for `kind` set to f8f6f4 with input in fp8 format we detect $N_{\text{FMA}} = 32$ with 25 fractional bits (i.e., $n_{\text{eab}} = 2$), the fp8 tensor cores in the previously discussed H100/H200 and Ada architectures provide only 13 fractional bits (i.e., $n_{\text{eab}} = -10$). The fp8 tensor core PTX instructions map to `UTCQMMA` in SASS.

4.1.8 RTX PRO 6000 workstation GPU (Blackwell). This GPU is based on the Blackwell architecture and is designed for desktop/workstation platforms. Similar to the B200, it features fifth-generation tensor cores. The tensor core numerical features are identical to those of the B200; however, fp8 input format tensor core is accessed through the `mma.sync.aligned` PTX instruction, as in the Ada architecture, rather than the `tcgen05.mma` instruction used on the B200. Furthermore, it then internally maps these operations to `QMMA` instructions instead of `UTCQMMA` at SASS level.

Table 4 summarizes the CUDA/PTX instructions used in this work for feature extraction and randomized testing, together with their corresponding mappings to SASS instructions.

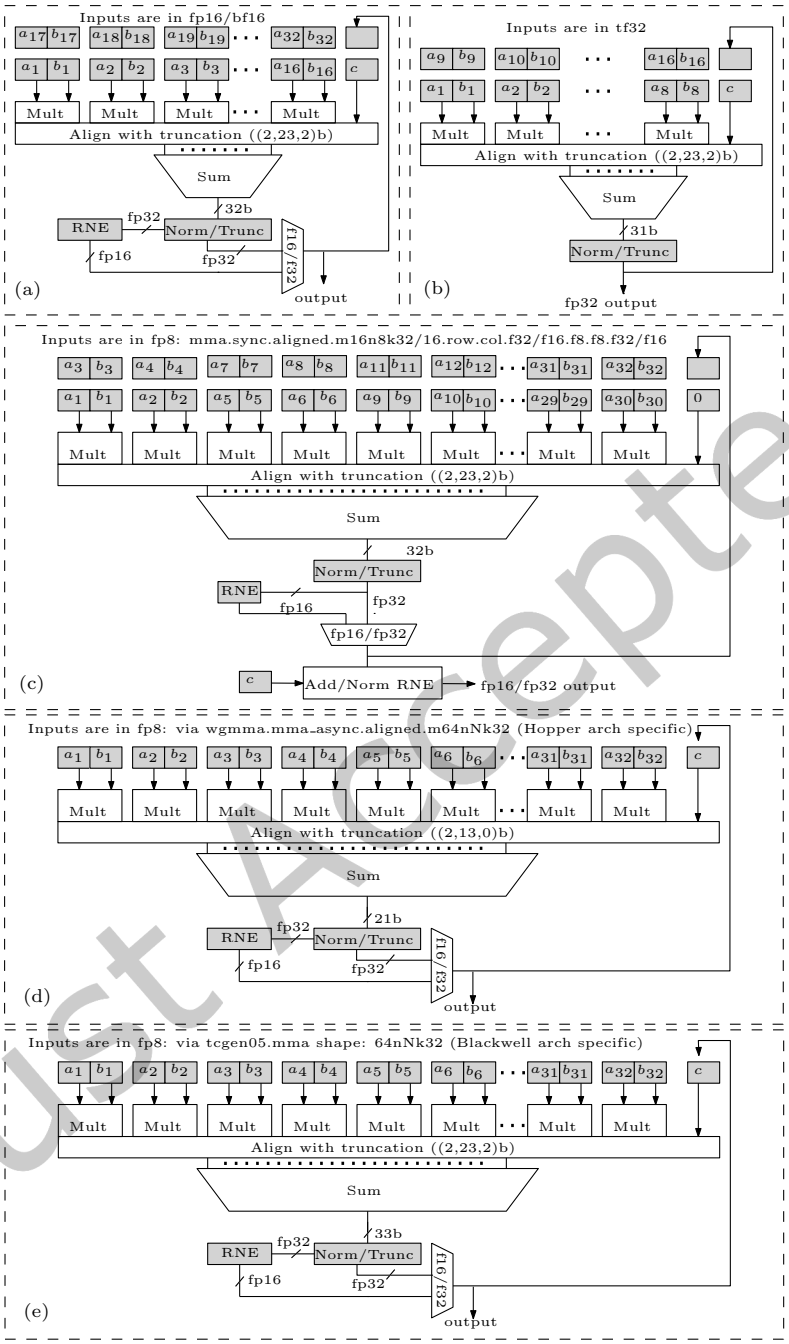

Fig. 5. A model of the inner product within the tensor cores of the H100/H200/B200 GPUs for (a) fp16/bf16, (b) tf19, (c) fp8 input format provided via `mma.sync` (which internally uses fp16 tensor core with interleaved input pattern), (d) fp8 tensor core accessed via `wgmma.mma_async` (specific to Hopper architecture), and (e) fp8 input format 5th generation tensor core in Blackwell architecture. The fp64 tensor core is compliant with IEEE 754 FMA operation and is not shown.

Table 3. Summary of the numerical features of several generations of NVIDIA tensor cores (mixed-precision matrix multipliers) in eleven different GPUs spanning years of release from 2017 to 2024.

In	Out	GPUs	Prd. Align.	Acc.	N_{FMA}	$e_{\text{min}}^{\text{lim}}$	Final Round.
fp8	fp32	B200, RTX PRO	(2,25)	33	32	-	Trunc.
		H100, H200	(2,13)	21	32	-	Trunc
		L40S, Ada RTX 1000	(2,13)	20	16	-	Trunc
fp8	fp16	B200, RTX PRO	(2,25)	33	32	-	RNE
		H100, H200	(2,13)	21	32	-	RNE
		L40S, Ada RTX 1000	(2,13)	20	16	-	RNE
fp16/bf16	fp32	H100/H200/B200/RTX PRO	(2,25)	32	16	−133	Trunc
		L40S, Ada RTX 1000	(2,24)	30	8	−132	Trunc
		A100, A2, A30	(2,24)	30	8	−132	Trunc
		V100	(2,23)	28	4	-	Trunc
tf19	fp32	H100/H200/B200/RTX PRO	(2,25)	31	8	−133	Trunc
		L40S, Ada RTX 1000	(2,24)	29	4	−132	Trunc
		A100, A2, A30	(2,24)	29	4	−132	Trunc
fp64	fp64	H100/H200/B200/RTX PRO	-	-	1	-	all 4
		A100	-	-	1	-	all 4

Note: subnormal in/out supported; output in fp16 format is also supported for fp16 input format with RNE as the final rounding. The products remain denormalised in alignment and accumulation, reflected via 2 integer bits. Extra alignment bits n_{eab} are included in the fractional bits of the product alignment precision. Accumulation output precision is a sum of extra carry bits and the product alignment bits. RTX PRO refers to RTX PRO 6000.

4.2 Comparison of software models against GPU results: sufficiency of ensemble testing and minimum amount of testing vectors

For increased assurance that tensor core models are bit-equivalent to GPU tensor core and no additional feature remains undetected, we have compared the tensor core models against the GPU results over ensembles containing vectors of following cases.

(1) Sampling a, b, and c from independent and identically distributed (i.i.d) uniform distribution with upper and lower bounds denoted as a_{min}, a_{max}, b_{min}, b_{max}, c_{min}, c_{max}, respectively, for each term with ensemble size denoted as N_{ens}. While c is a scalar, a and b are vectors of length k, where k is the inner dimension supported by a single MMA instruction. Note that k is not necessarily equal to N_{FMA}, because some of the PTX instructions support larger inner dimension than that supported by hardware tensor cores. For instance $k = 16$ for Ampere and Ada in fp16/bf16 input format but $N_{\text{FMA}} = 8$.
- In input formats bf16 and tf19, we generate ensemble via Algorithm 2, where $k = 16$ for bf16, and 8 for tf19, with following settings.

(a) $N_{\text{ens}} = 10^7$, $a_{\text{min}} = b_{\text{min}} = c_{\text{min}} = -2^{15}$, $a_{\text{max}} = b_{\text{max}} = c_{\text{max}} = 2^{15}$
(b) $N_{\text{ens}} = 10^7$, $a_{\text{min}} = b_{\text{min}} = -2^{50}$, $a_{\text{max}} = b_{\text{max}} = 2^{50}$, $c_{\text{min}} = -2^{100}$, $c_{\text{max}} = 2^{100}$
(c) $N_{\text{ens}} = 10^7$, $a_{\text{min}} = b_{\text{min}} = -2^{-70}$, $a_{\text{max}} = b_{\text{max}} = 2^{-70}$, $c_{\text{min}} = -2^{-126}$, $c_{\text{max}} = 2^{-126}$
(d) $N_{\text{ens}} = 10^7$, $a_{\text{min}} = b_{\text{min}} = -2^{-70}$, $a_{\text{max}} = b_{\text{max}} = 2^{-70}$, $c = 0$

First ensemble tests the basic working accuracy within narrow range i.e. $\pm 2^{15}$ of the mentioned format, and thereafter, models are tested within extended range, but avoiding overflows both in input and output.

Table 4. CUDA/PTX instruction mapping to SASS instructions

Input	CUDA/PTX Instr.	GPUs	SASS Instr.
fp8	`mma.sync.aligned.m16n8k32`	H100/H200/B200	`HMMA.16816`
		RTX 6000	`QMMA.16832`
		L40S/Ada RTX 1000	`QMMA.16816`
	`wgmma.mma_async.sync.m64nNk32`	H100/H200	`QGMMA.16832`
	`tcgen05.mma.cta_group`	B200	`UTCQMMA`
fp16/bf16	WMMA API	H100/H200/B200/RTX PRO	`HMMA.16816`
		L40S/Ada RTX 1000	`HMMA.1688`
		A100/A2/A30/A40	`HMMA.1688`
		V100	`HMMA.844`
	`tcgen05.mma.cta_group.kind=f16`	B200	`UTCHMMA`
tf19	WMMA API	H100/H200/B200	`HMMA.1684`
		L40S/Ada RTX 1000	`HMMA.1684`
		A100/A2/A30/A40	`HMMA.1684`
	`mma.sync.aligned.m16n8k4/8`	H100/H200/B200/RTX PRO	`HMMA.1684/8`
		L40S/Ada RTX 1000	`HMMA.1684`
		A100/A2/A30	`HMMA.1684`
	`tcgen05.mma.cta_group.kind=tf32`	B200	`UTCHMMA`
fp64	WMMA API	H100/H200/B200/RTX PRO	`DMMA.884`
		A100	`DMMA.884`

WMMA API for tf19 maps to HMMA.1684 for GPUs that support tf19 as input, but H100/H200/B200 and RTX PRO (refers to RTX PRO 6000) also support HMMA.1688 via `mma.sync.aligned.m16n8k8`. The `wgmma.mma` PTX instruction is specific to the Hopper architecture, whereas the `tcgen05.mma` PTX instruction is specific to Blackwell data-center GPUs.

In the last two cases, subnormal region is explored where the product exponents are intentionally made smaller than $e_{\min}^{\lim}$ with c either in subnormal region or set to 0.

- In input formats fp16 and fp8-e5m2, ensembles are generated with $k = 16$ for fp16 and $k = 32$ for fp8-e5m2, for all GPUs that support these formats.

(a) $N_{\text{ens}} = 10^7$, $a_{\min} = b_{\min} = c_{\min} = -2^7$, $a_{\max} = b_{\max} = c_{\max} = 2^7$
(b) $N_{\text{ens}} = 10^7$, $a_{\min} = b_{\min} = -2^{15}$, $a_{\max} = b_{\max} = 2^{15}$, $c_{\min} = -2^{100}$, $c_{\max} = 2^{100}$
(c) $N_{\text{ens}} = 10^7$, $a_{\min} = b_{\min} = -2^{-15}$, $a_{\max} = b_{\max} = 2^{-15}$, $c_{\min} = -2^{-126}$, $c_{\max} = 2^{-126}$

Similar to the bf16 and tf19 case, these three cases test fp16 and fp8-e5m2 in normal and subnormal regions.

- In fp8-e4m3 input format, for which $L = 32$, we have

(a) $N_{\text{ens}} = 10^7$, $a_{\min} = b_{\min} = -2^7$, $a_{\max} = b_{\max} = 2^7$, $c_{\min} = -2^{100}$, $c_{\max} = 2^{100}$
(b) $N_{\text{ens}} = 10^7$, $a_{\min} = b_{\min} = -2^{-6}$, $a_{\max} = b_{\max} = 2^{-6}$, $c_{\min} = -2^{-126}$, $c_{\max} = 2^{-126}$

- In bf16 format, we generate an extended ensemble with $k = 16$ for Ampere, Hopper, and Blackwell GPUs with $N_{\text{ens}} = 10^8$, $a_{\min} = b_{\min} = -2^{63}$, $a_{\max} = b_{\max} = 2^{63}$, $c_{\min} = -2^{122}$, $c_{\max} = 2^{122}$. This setup enables testing of the model across all possible output categories, including Inf, NaN, normal, and subnormal values.

(2) $N_{\text{ens}} = 10^5$ samples drawn via Algorithm 2 from a normal distribution in all input formats.
(3) Edge Cases

- Test vectors to evaluate behaviour for ±Inf and NaN inputs.
- Scenarios where $c = 0$, or where some or all partial products are zero.
- Extreme dynamic range cases, e.g., $c = \pm 2^{127}$ with products $-2^{-126} \leq p_i \leq 2^{-126}$, and vice versa.
- Subnormal cases, with $c = 0$ or a subnormal value in fp32 format and $p_i \in [-2^{-126}, 2^{-126}]$.
- Cases where partial products exceed 2^{128}, but the final sum is less than 2^{128}.
- Mixed input configurations involving zero, normal, subnormal, and special values.
- Choose a, b, and c such that the dot product generates the maximum number of carries during accumulation (see [2]).

Algorithm 2 Ensemble generation pseudo-code for randomized testing of tensor core models

Choose k as per input format and GPU model
mt19937 rng(0) % Mersenne Twister with seed 0
choose N_{ens}, $a_{\min}$, $a_{\max}$, $b_{\min}$, $b_{\max}$, $c_{\min}$, $c_{\max}$
for $i = 1 : N_{\text{ens}}$ **do**
    $a_\ell \overset{\text{i.i.d.}}{\sim} \mathcal{U}(a_{\min}, a_{\max})$, $\ell = 1, \ldots, k$
    $b_\ell \overset{\text{i.i.d.}}{\sim} \mathcal{U}(b_{\min}, b_{\max})$, $\ell = 1, \ldots, k$
    $c \overset{\text{i.i.d.}}{\sim} \mathcal{U}(c_{\min}, c_{\max})$
end for

For the hardware execution, WMMA API, mma.sync, wgmma.mma_async, and tcgen05.mma instructions were invoked in CUDA. To further verify that tensor cores were active, the cuobjdump tool was used to inspect the compiled binary and confirm the presence of HMMA, QMMA, QGMMA, UTCHMMA, UTCQMMA, DMMA instructions. Finally, we note that DMMA operations are not modelled or tested via randomized testing, nor are they included in the model package, since, as reported by Fasi et al. [9], they behave as sequential FMAs and are fully IEEE compliant. Therefore, one can directly rely on MATLAB's built-in fma command to emulate the behaviour of DMMA.

For Ada GPUs, CUDA toolkit 12.9 was used, while for all other GPUs, we used CUDA toolkit 12.8, while the GCC version for Ada GPUs and B200 was 14.2.0, and 13.1.0, respectively, while for remaining GPUs, it was 11.5.0.

4.2.1 Evolution of software models across iterations of model refinement. Here we describe how the models presented in this paper evolved over the iterations of Algorithm 1.

In the first iteration, using GNFT, we determined the FMA sizes, extra alignment and carry bits, alignment rounding mode, and final rounding modes (truncation in fp32, and RNE for fp16 output format). However, the GNFT-only model failed for certain test cases in the ensemble, which subsequently led to the identification of the denormalised product feature. Consequently, the GNFT was extended with dedicated test vectors capable of detecting this behaviour. This feature also revealed that products are computed as the product of significands multiplied by two raised to the sum of exponents, thereby allowing intermediate products to exceed 2^{128} without overflow, provided that the final accumulated result remains representable in the output format.

After incorporating the necessary updates into the software models, the second iteration showed that the V100 model satisfied the entire test ensemble, as it only supports fp16 input formats. However, all other models failed within regions of the ensemble involving subnormal a, b, and c values in bf16 and tf19 formats (prompted by a GitHub issue report from a community member, username *dfyz*). Detailed analysis revealed that the minimum exponent permitted during product alignment is -132 for Ampere and Ada, and -133 for Hopper and Blackwell in both formats, with $c = 0$ treated as a special case, i.e., ignored rather than interpreted as a subnormal value. This observation prompted the construction of dedicated tests to identify exponent limitations, which were subsequently incorporated into the GNFT. In the third iteration, the GPU and MATLAB model outputs matched

across the entire test ensemble, at the bit level, across all supported input and output precision formats across 11 GPUs for 8-bit and higher precision formats (Table 3).

Moreover, the emulated models were also updated to match the GPU outputs in exceptional cases (`NaN`, `Inf`, or `-Inf`) via expressions such as $\infty \pm \infty$, $\pm(\texttt{Inf} \times \texttt{Inf})$, and $\texttt{NaN} \pm \texttt{Inf}$, both as direct inputs and as overflow scenarios where finite accumulations lead to $\pm$`Inf` in the supported output formats. Consistent with GPU behavior, `NaN` takes precedence whenever it appears.

4.2.2 Sufficiency of the constructed ensemble. The sufficiency of the randomized testing strategy is supported by the structural simplicity of the proposed models. In particular, the truncation of aligned significands beyond $(23 + n_{\text{eab}})$−fractional bits eliminates many complex corner cases, such as partially or completely out-of-range shifted significands, thereby significantly reducing the effective search space. Consequently, broad randomized sampling, combined with targeted stress testing, provides high coverage of relevant numerical behaviours.

The constructed ensemble consists of complementary randomized and targeted tests designed to exercise the models across normal, subnormal, and special-value operating regions. The ensemble includes evaluations over broad dynamic ranges, targeted subnormal-region analysis for different tensor core formats, and extended stress testing capable of triggering all output categories, including normal, subnormal, Inf, and NaN outputs. In addition, edge cases were incorporated to evaluate behaviours involving zero-valued accumulators or products, exponent alignment limitations, extreme dynamic ranges, denormalised products, and mixed combinations of normal, subnormal, and special values.

Together, the combination of randomized sampling, targeted edge-case evaluation, and iterative refinement through Algorithm 1 provides strong empirical evidence that the proposed models accurately capture the numerical behaviour of the hardware. While exhaustive validation over the complete input space is infeasible, the constructed ensemble provides a high degree of confidence in bit-level agreement between the software models and GPU implementations. The extended ensemble testing over 100 million input samples was performed and the absence of further mismatches throughout these tests therefore provides strong evidence that the resulting software models are already highly accurate.

Randomized testing can also be replaced by a minimum set of test vectors for any tensor core architecture, provided that the assumed feature space is a superset of all features present in a given architecture. However, this is not the case in the current scenario, as the initially assumed feature space (in the GNFT and other prior techniques) did not include denormalized products, minimum exponent limitations, and the special case of $c = 0$ instead of being considered as a subnormal case.

In the future, we will explore different strategies for ISSM; we plan to port Schryer's [35] strategy from testing scalar operations to dot products and explore covering signs, exponents and significands via separate sampling. We also plan to deploy the verification in parallel across hundreds of cores to cover a larger portion of the input space.

4.3 Example MATLAB code for using the models

This section introduces the `MATLAB Tensor Core v0.5`. The toolbox was developed using MATLAB R2026a and depends on CPFloat [10] for rounding the inputs, that are by default in binary64, to low-precision formats that MATLAB does not natively support. Here we discuss the user interface and the overall structure of the code.

The toolbox is comprised of three layers:

- `Generic_BFMA_TC.m`: provides a generalised tensor core model which can be set up with various features [2].
- `GEMM.m`: accepts α, A, B, β, and C, floating-point input and output formats, and the settings for the `Generic_BFMA_TC.m`, and approximates the GEMM $\alpha \times A \times B + \beta \times C$ using the tensor core model.
- A set of GPU tensor core models, such as `B200TC.m`, which instantiate the model parameters and call `GEMM.m`.

The v0.5 of the toolbox implements a recursive algorithm in `GEMM.m`, which is equivalent to recursively using a single tensor core to compute each inner product in the GEMM. It does not attempt to match the results of any CUDA GEMM algorithm, of which there are possibly many and they are chosen dynamically based on the input matrix shapes and numerical formats. The following shows an example that calls the B200 tensor core model with an fp8 format as an input and fp32 as the output for multiplying two 4×4 matrices.

```
inopts.format = 'fp8-e4m3', outopts.format = 'binary32';
A = cpfloat(randn(4), inopts), B = cpfloat(randn(4), inopts);
C = cpfloat(randn(4), outopts), alpha = 1, beta = 1;
B200TC(alpha, A, B, beta, C, inopts.format, outopts.format)
>> ans =
    0.1484   -0.6631   -0.1836   -1.3271
    0.8232    1.6418    0.4805    3.0227
    3.6592   -0.1250    1.4902    2.2637
    3.7432   -3.4275    0.2031    0.1663
```

Apart from calling the mentioned GPU tensor core functions, the user can call a custom model tensor core function with a set of parameters; an example is provided below.

```
params.neab = 3;       % Extra alignment bits.
params.fma = 32;       % Block FMA size.
params.frmode = 'rne'; % Final rounding mode.
CustomTC(alpha, A, B, beta, C, in_format, out_format, params)
```

The `GEMM.m` file executes the parallelized version of matrix multiplication if the `Parallel Computing Toolbox`, introduced in MATLAB R2013b, is installed and the machine supports multicore processing. This ensures that computations are efficiently distributed across available CPU cores, accelerating large-scale matrix operations. If either is not supported, a serialized version is executed. Lastly, the proposed toolbox is compatible with Octave and can also be accessed from Python using either the Oct2Py library or the MATLAB engine. In the future, we will port the back-end of the models to C and interface to MATLAB via the mex interface, to improve performance of computing with the tensor core models.

4.4 Comparison of our models with prior models

4.4.1 Numerical features. A comparison of the numerical features identified by different studies is presented in Table 5, including prior works such as [2, 9, 14, 23, 40]. Hickmann et al. [14] investigated only the V100 GPU and correctly identified most numerical features. However, they concluded that the c term is added late to the product sum using RNE, whereas our results indicate otherwise. Furthermore, the preservation of denormalized products was not reported.

Similarly, Fasi et al. [9] examined the tensor cores of both V100 and A100 GPUs. While their reported features are largely consistent with our findings, they did not identify the preservation of denormalized products, nor the exponent limitation present in the bf16 format on A100. Li et al. [23] extended the analysis to H100 GPUs in addition to V100 and A100, but did not investigate the fp8 input format. Moreover, they did not report either the denormalized product behavior or the exponent limitation feature. Their characterization of N_{FMA} and n_{eab} for H100 with bf16 and fp16 inputs was also inconclusive, reporting only bounds, i.e., $n_{\text{eab}} \geq 2$ and $N_{\text{FMA}} \geq 16$.

The GNFT-only studies likewise did not identify the denormalized-product and exponent-limitation features and did not consider fp8 input formats. In contrast, the proposed framework combines targeted feature testing with randomized numerical testing, enabling the discovery of previously unreported behaviors and hidden numerical characteristics across multiple GPU architectures and input precisions. As shown in Table 5, the proposed approach provides the most comprehensive coverage of tensor-core numerical features among the compared works.

4.4.2 Ensemble based failure rate. Here we compare the refined model, the GNFT-only model, and prior models against GPU outputs in terms of ensemble failure rate. While no publicly available software implementations exist for any of the earlier studies, we use the proposed MATLAB model to emulate the models in [9, 14, 23], since these papers provide comprehensive descriptions of the numerical features required for accurate emulation. To simulate the model of [23], the following assumptions are made:

- $N_{\mathrm{FMA}} = 1$ for V100, since [23] reports that the FMA size is not detectable.
- Since the authors report $n_{\mathrm{eab}} \geq 2$ and $N_{\mathrm{FMA}} \geq 16$ for H100, the minimum reported values are assumed, i.e., $n_{\mathrm{eab}} = 2$ and $N_{\mathrm{FMA}} = 16$.
- Products are assumed to be normalised, and the minimum exponent limitation for alignment is set to $-\infty$, as these features were not detected.

The V100 model estimated in [14] is simulated as reported, where c is added afterwards via an RNE operation with the product normalised. For the GNFT-only model [2], the FMA size is correctly detected; however, product denormalisation and exponent-limitation behaviour are not incorporated.

The ensemble failure rate for an ensemble size of 10^5 is largest for the V100 model of [14], while the second-highest failure rate is observed for the model of [23], as reported in Table 6. Although the simulated model of [23] yields a failure rate similar to that of the GNFT-only model, the failure rate for H100 may in practice be even larger than that reported in Table 6 if simulated with $n_{\mathrm{eab}} > 2$ and $N_{\mathrm{FMA}} > 16$. Such settings would further deviate from the true H100 hardware behaviour. This is the reason for $\geq$ symbol with H100 failure rate in [23] column in Table 6. The proposed refined model achieves zero failure rate across all illustrated test cases, while the GNFT-only model remains the closest approximation among the compared methods.

It should also be noted that, depending on how the ensemble is generated, the GNFT-only and [23] models may fail to trigger certain hardware features, such as exponent limitation. Consequently, these models may exhibit zero failure rate regardless of ensemble size. An example of this behaviour is shown in the final bf16 input-format case in Table 6.

5 Multi-word algorithms for emulating high-precision GEMM on tensor core models

Multi-word arithmetic is a technique of emulating high-precision matrix multiplication with low precision tensor cores. It consists of splitting high-precision input matrices into several low-precision matrices, multiplying them with tensor cores, and adding up the products either in tensor cores or the CUDA cores. In order to demonstrate how one may use the MATLAB tensor core models, we have reproduced the experiments of Mary and Mikaitis [26, Sec. 5] via several different models. We refer the readers to [26] for a full description of the algorithms.

Figures 6–8 show the norm-wise errors, where $\|\cdot\|_\infty$ is the infinity norm of a matrix, on five different tensor core models (V100, A100, the H100 which also covers H200 and the B200, L40S, and a custom variant of the B200 which uses round to nearest instead of bit truncation) with the input format set to fp8, fp16 and bf16, and the output format set to fp32. The number of words, which defines how accurate the emulation is, is shown at the top of each of the sub-figures.

The modified B200 model has rounding to nearest instead of bit truncation—this is applied when an internal accumulator is rounded to the output format, not on the alignments of significands, which are still truncated as in the standard B200 tensor core model.

Table 5. Comparison of numerical features of tensor cores on NVIDIA GPUs reported by prior works and the present work.

GPU	Feature	[14]	[9]	[23]	[40]	[2]	Proposed
V100	N_{FMA}	✓	✓	✗	✓	✓	✓
	n_{eab}	✓	✓	✓	✓	✓	✓
	denom. prod.	NR	NR	NR	NR	NR	✓
	c is added late/early	✗	✓	✓	✓	✓	✓
A100/A30/A40/A2	N_{FMA}	-	✓	✓	✓	✓	✓
	n_{eab}	-	✓	✓	✓	✓	✓
	denom. prod.	-	NR	NR	NR	NR	✓
	exp. limit. ($e_{\text{min}}^{\text{align}}$)	-	NR	NR	NR	NR	✓
	$c = 0$ special/subnormal case	-	NR	NR	NR	NR	✓
Ada (RTX-1000/L40S)	$N_{\text{FMA}} + n_{\text{eab}}$ (fp16/bf16/tf19)	-	-	-	-	✓	✓
	$N_{\text{FMA}} + n_{\text{eab}}$ (fp8)	-	-	-	-	-	✓
	denom. prod.	-	-	-	-	NR	✓
	exp. limit. ($e_{\text{min}}^{\text{align}}$)	-	-	-	-	NR	✓
	$c = 0$ special/subnormal case	-	-	-	-	NR	✓
H100/H200	$N_{\text{FMA}} + n_{\text{eab}}$ (fp16/bf16/tf19)	-	-	✗*	-	-	✓
	$N_{\text{FMA}} + n_{\text{eab}}$ (fp8)	-	-	-	-	-	✓
	denom. prod.	-	-	NR	-	-	✓
	exp. limit. ($e_{\text{min}}^{\text{align}}$)	-	-	NR	-	-	✓
	$c = 0$ special/subnormal case	-	-	NR	-	-	✓
B200/RTX PRO 6000	$N_{\text{FMA}} + n_{\text{eab}}$ (fp16/bf16/tf19)	-	-	-	-	-	✓
	$N_{\text{FMA}} + n_{\text{eab}}$ (fp8)	-	-	-	-	-	✓
	denom. prod.	-	-	-	-	-	✓
	exp. limit. ($e_{\text{min}}^{\text{align}}$)	-	-	-	-	-	✓
	$c = 0$ special/subnormal case	-	-	-	-	-	✓

Cells marked - mean that GPU or the input format is not considered, ✗: feature tested but reported conclusion is inaccurate, NR: stands for not reported, ✗*: feature detected partially correct or reported with ambiguity.

Table 6. Ensemble-based mismatch rates of proposed and prior tensor core models compared against GPU outputs.

Ensemble setting	[14]	[23]	GNFT [2]	Proposed
(fp16): $a, b, c \in [-2^7, 2^7]$	V100: 55.4%	45%	22.5%	0%
	A100: NA	15.5%	15.5%	0%
	H100: NA	13%	13%	0%
(bf16): $a, b, c \in [-2^{15}, 2^{15}]$	A100: NA	9.6%	9.6%	0%
	H100: NA	$\geq 9.2\%$	9.2%	0%
$c = 0$, $a, b \in [-2^{-70}, 2^{-70}]$	A100: NA	0.006%	0.006%	0%
	H100: NA	$\geq 0.002\%$	0.002%	0%
$c = [-2^{-126}, 2^{-126}]$, $a, b \in [-2^{-65}, 2^{-65}]$	A100: NA	0%	0%	0%
	H100: NA	$\geq 0\%$	0%	0%

The ensemble for comparison is created with a seed of 0 via mt19937 by sampling 10^5 samples from a uniform distribution.

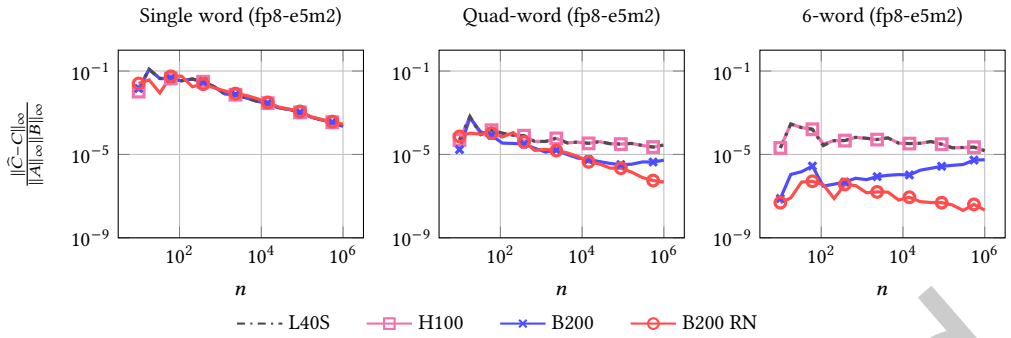

Fig. 6. Multi-word arithmetic experiment presented by Mary and Mikaitis [26, Sec. 5] on the simulation of various tensor cores. We have reproduced the experiment on four different tensor core generations modelled in MATLAB. Relative norm-wise errors of matrix multiplication, compared with a default MATLAB binary64 multiplication, are shown. The input matrices to the GEMM are $A \in R^{10\times n}$ and $B \in R^{n\times 10}$. These matrices are multiplied with a multi-word algorithm [26, Sec. 4] by splitting them into several fp8-e5m2 words.

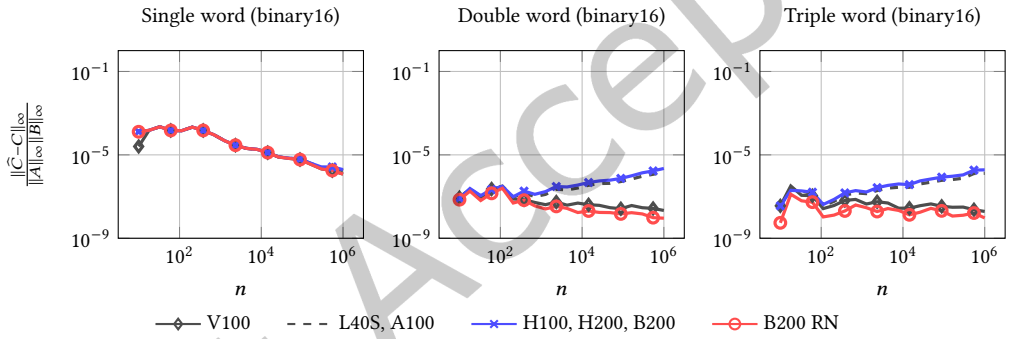

Fig. 7. Multi-word arithmetic experiment presented by Mary and Mikaitis [26, Sec. 5] on the simulation of various tensor cores. Relative norm-wise errors of matrix multiplication, compared with a default MATLAB binary64 multiplication, are shown. The input matrices to the GEMM are $A \in R^{10\times n}$ and $B \in R^{n\times 10}$. These matrices are multiplied with a multi-word algorithm [26, Sec. 4] by splitting them into several fp16 words.

Figure 6 shows the differences in accuracy between L40S/H100, which have $n_{\text{eab}} = -10$ and B200 which has $n_{\text{eab}} = 2$ in the accumulator. There is also a significant improvement to the B200 result by enabling RN. The L40S and H100/H200 GPUs are, as expected, the least accurate in fp8 input format because of the 13 fractional bits in the accumulator as discussed in Section 4.1.4. The DeepSeek-V3 Technical Report addresses this limitation in the Hopper GPU variant H800 [6, Sec. 3.3.2].

For fp16 (Figure 7), interestingly, for single-word arithmetic, all models closely match in accuracy. For double- and triple-word arithmetic, the results of Figure 7 show that the V100 tensor core provides two orders of magnitude lower error at $n = 10^6$. This may be caused by B200 tensor core model having more accurate accumulator with $n_{\text{eab}} = 2$, which is rounded to zero, making the error larger than the V100 tensor core's result. This can occur if the V100 result is above the reference result, whilst the B200 result is closer to the reference, but below it, so that

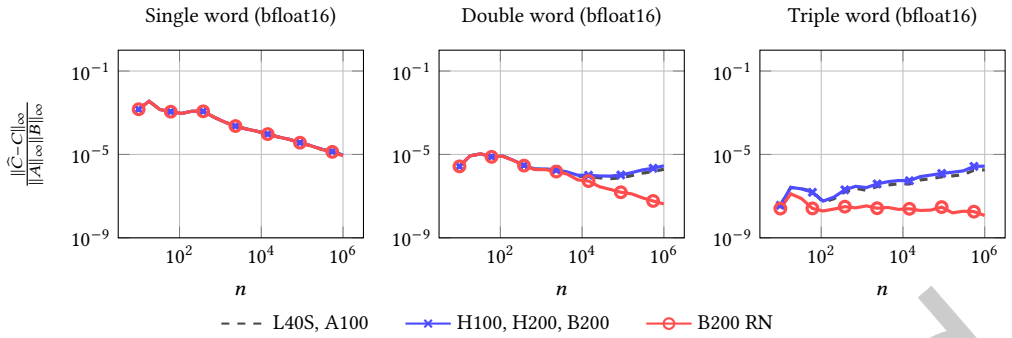

Fig. 8. Multi-word arithmetic experiment presented by Mary and Mikaitis [26, Sec. 5] on the simulation of various tensor cores. Relative norm-wise errors of matrix multiplication, compared with a default MATLAB binary64 multiplication, are shown. The input matrices to the GEMM are $A \in R^{10\times n}$ and $B \in R^{n\times 10}$. These matrices are multiplied with a multi-word algorithm [26, Sec. 4] by splitting them into several bfloat16 words.

RZ pushes it further below. We have tested this hypothesis by enabling RN in the B200, which demonstrates an improvement. Further analysis of this behavior is out of the scope of this paper and we leave it for future work.

We expect similar differences in mixed-precision iterative refinement where GEMM operations are executed using different tensor core models [13]. Having the tensor core models at hand, callable in MATLAB, allows researching how factors such as extra alignment bits, FMA size, and rounding modes influence the behavior and accuracy of such mixed-precision computations.

These experiments demonstrate that the proposed software models can be used by researchers to simulate and study the effects of NVIDIA GPU tensor cores with customized numerical features in diverse applications, such as in artificial intelligence and scientific computing, and also for authentication of new test vectors for line 1 of Algorithm 1. The data and the code for producing Figures 6–8 is available.[6]

6 Conclusion

We have presented the research behind the development of the `MATLAB Tensor Core v0.5` toolbox. The toolbox contains various NVIDIA GPU tensor core models, as well as a parameterised model that can be used to instantiate custom variants of tensor cores for research purposes. The models were verified against the hardware by large-scale randomised testing and model refinement. The proposed toolbox includes tensor core models for NVIDIA A2, A30, A100, Ada RTX 1000, L40S, H100, H200, RTX PRO 6000 (Blackwell) and B200, supporting all 8-, 16-, and 19-bit precision formats. For binary64, NVIDIA GPU tensor cores behave as sequential chains of IEEE-compliant FMAs; therefore, MATLAB's built-in `fma` function can be used to emulate such tensor core behaviour. In addition to the fixed models, we provide a custom tensor core model that enables users to simulate arbitrary configurations by adjusting the FMA size, the number of extra bits used for aligning significands, and the rounding mode (supporting `RNE`, `RZ`, `RD`, and `RU`).

In the future, this toolbox will be actively maintained through regular versioned releases on GitHub. We plan to improve both the functionality and performance, by porting the back-end to C, by porting the front-end to other languages such as Julia, and by adding new NVIDIA tensor core and AMD matrix engine models. We also plan to add different GEMM algorithms and improve the validation in Algorithm 1 by exploring different distributions of randomised test vectors and techniques such as the ones used by Schryer [35].

[6] https://github.com/north-numerical-computing/MATLAB-tensor-core/tree/main/experiments

IEEE 754 [18] and OCP [34] do not standardise reduction operations, and P3109 [1] does not require them for conformance. We hope that a simulator of GPU matrix multiplier models will allow users to understand the differences and effects on applications of current and future matrix multiply units, and impact future standardisation efforts.

7 Acknowledgment

We thank John Hodrien at University of Leeds for technical support with the Aire machine containing the NVIDIA L40S and A2, and The COSmology MAchine (COSMA) support at Durham University for providing the access to A30, V100, A100, H100, H200 and RTX PRO 6000 GPUs. We thank Argonne National Laboratory for providing access to V100, A100, H100, and B200 GPUs. We also thank Jack Dongarra, John Gunnels, Eduardo Basurto, and Eric Rife for arranging access to the B200 GPUs before they became available via ANL. We are grateful to a GitHub user *dfyz* (Ivan Komarov) for testing the tensor core models and reporting special cases. Both authors are funded by the EPSRC grant "*Informing Future Numerical Standards by Determining Features of Non-Standard Mathematical Hardware*", ref. UKRI151.

References

[1] 2026. *Interim Report on Binary Floating-point Formats for Machine Learning*. Technical Report. https://github.com/P3109/Public/blob/main/IEEE%20WG%20P3109%20Interim%20Report%20v3.2.1.pdf Version 3.2.1.

[2] Faizan A. Khattak and Mantas Mikaitis. 2025. Generalized Methodology for Determining Numerical Features of Hardware Floating-Point Matrix Multipliers: Part I. In *2025 IEEE High Performance Extreme Computing Conference (HPEC)*. Wakefield, MA, USA. doi:10.1109/HPEC67600.2025.11196657

[3] AMD. 2025. Datasheet: AMD Instinct MI355X GPU. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/product-briefs/amd-instinct-mi355x-gpu-brochure.pdf

[4] Pierre Blanchard, Nicholas J. Higham, Florent Lopez, Theo Mary, and Srikara Pranesh. 2020. Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores. *SIAM Journal on Scientific Computing* 42, 3 (2020), C124–C141. doi:10.1137/19M1289546

[5] NVIDIA Corporation. 2025. CUDA Binary Utilities, Release 13.1. https://docs.nvidia.com/cuda/pdf/CUDA_Binary_Utilities.pdf

[6] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 https://arxiv.org/abs/2412.19437

[7] Jack Dongarra, John Gunnels, Harun Bayraktar, Azzam Haidar, and Dan Ernst. 2025. Accelerating Supercomputing: AI-Hardware-Driven Innovation for Speed and Efficiency. In *2025 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7. doi:10.1109/HPEC67600.2025.11196413

[8] Sultan Durrani, Muhammad Saad Chughtai, Mert Hidayetoglu, Rashid Tahir, Abdul Dakkak, Lawrence Rauchwerger, Fareed Zaffar, and Wen-mei Hwu. 2021. Accelerating Fourier and Number Theoretic Transforms using Tensor Cores and Warp Shuffles. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 345–355. doi:10.1109/PACT52795.2021.00032

[9] Massimiliano Fasi, Nicholas J. Higham, Mantas Mikaitis, and Srikara Pranesh. 2021. Numerical behavior of NVIDIA tensor cores. *PeerJ Computer Science* 7 (2021), e330. doi:10.7717/peerj-cs.330

[10] Massimiliano Fasi and Mantas Mikaitis. 2023. CPFloat: A C Library for Simulating Low-Precision Arithmetic. *ACM Trans. Math. Softw.* 49, 2, Article 18 (June 2023), 32 pages. doi:10.1145/3585515

[11] Boyuan Feng, Yuke Wang, Guoyang Chen, Weifeng Zhang, Yuan Xie, and Yufei Ding. 2021. EGEMM-TC: accelerating scientific computing on tensor cores with extended precision. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 278–291. doi:10.1145/3437801.3441599

[12] Odd Erik Gundersen, Kevin Coakley, Christine Kirkpatrick, and Yolanda Gil. 2023. Sources of Irreproducibility in Machine Learning: A Review. arXiv:2204.07610 [cs.LG] https://arxiv.org/abs/2204.07610

[13] Azzam Haidar, Harun Bayraktar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. 2020. Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476, 2243 (2020), 20200110. doi:10.1098/rspa.2020.0110

[14] Brian Hickmann and Dennis Bradford. 2019. Experimental Analysis of Matrix Multiplication Functional Units. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. 116–119. doi:10.1109/ARITH.2019.00031

[15] Brian Hickmann, Jieasheng Chen, Michael Rotzin, Andrew Yang, Maciej Urbanski, and Sasikanth Avancha. 2020. Intel Nervana Neural Network Processor-T (NNP-T) Fused Floating Point Many-Term Dot Product. In *2020 IEEE 27th Symposium on Computer Arithmetic*

(ARITH). 133–136. doi:10.1109/ARITH48897.2020.00029

[16] Nicholas J. Higham and Theo Mary. 2022. Mixed Precision Algorithms in Numerical Linear Algebra. *Acta Numerica* 31 (May 2022), 347–414. doi:10.1017/s0962492922000022

[17] Karl E. Hillesland and Anselmo Lastra. 2004. GPU Floating-Point Paranoia. In *ACM Workshop on General-Purpose Computing on Graphics Processors (GP^2)*. ACM, Los Angeles, CA, USA.

[18] 2019. *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA. 82 pages. doi:10.1109/IEEESTD.2019.8766229

[19] Intel Corporation. 2018. BFLOAT16—Hardware Numerics Definition. Available at https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf (accessed 4 July 2026). White paper. Document number 338302-001US..

[20] Aaron Jarmusch, Nathan Graddon, and Sunita Chandrasekaran. 2025. Dissecting the NVIDIA Blackwell Architecture with Microbenchmarks. arXiv:2507.10789 [cs.DC] https://arxiv.org/abs/2507.10789

[21] Himanshu Kaul, Mark Anders, Sanu Mathew, Seongjong Kim, and Ram Krishnamurthy. 2019. Optimized Fused Floating-Point Many-Term Dot-Product Hardware for Machine Learning Accelerators. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. 84–87. doi:10.1109/ARITH.2019.00021

[22] Binrui Li, Shenggan Cheng, and James Lin. 2021. tcFFT: Accelerating Half-Precision FFT through Tensor Cores. arXiv:2104.11471 [cs.DC] https://arxiv.org/abs/2104.11471

[23] Xinyi Li, Ang Li, Bo Fang, Katarzyna Swirydowicz, Ignacio Laguna, and Ganesh Gopalakrishnan. 2024. FTTN: Feature-Targeted Testing for Numerical Properties of NVIDIA & AMD Matrix Accelerators. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 39–46. doi:10.1109/CCGrid59990.2024.00014

[24] Tianjian Lu, Thibault Marin, Yue Zhuo, Yi-Fan Chen, and Chao Ma. 2020. Accelerating MRI Reconstruction on TPUs. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–9. doi:10.1109/HPEC43674.2020.9286192

[25] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter. 2018. NVIDIA Tensor Core Programmability, Performance & Precision. In *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium Workshops*. Vancouver, BC, Canada, 522–531. doi:10.1109/IPDPSW.2018.00091

[26] Theo Mary and Mantas Mikaitis. 2025. Error Analysis of Matrix Multiplication with Narrow Range Floating-Point Arithmetic. *SIAM J. Sci. Comput.* 47, 4 (2025), B785–B800. doi:10.1137/24M1685109

[27] Paulius Micikevicius, Stuart Oberman, Pradeep Dubey, Marius Cornea, Andres Rodriguez, Ian Bratt, Richard Grisenthwaite, Norm Jouppi, Chiachen Chou, Amber Huffman, Michael Schulte, Ralph Wittig, Dharmesh Jani, and Summer Deng. 2023. *OCP 8-bit Floating Point Specification (OFP8)*. Technical Report. Open Compute Project. https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-0-2023-12-01-pdf-1 Revision 1.0.

[28] Mantas Mikaitis. 2024. Monotonicity of Multi-Term Floating-Point Adders. *IEEE Trans. Comput.* 73, 6 (Feb. 2024), 1531–1543. doi:10.1109/TC.2024.3371783

[29] NVIDIA. 2017. NVIDIA Tesla V100 GPU architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[30] NVIDIA. 2025. NVIDIA Blackwell Architecture Technical Brief. https://resources.nvidia.com/en-us-blackwell-architecture

[31] Leon Oostrum, Bram Veenboer, Ronald Rook, Michael Brown, Pieter Kruizinga, and John W. Romein. 2025. The Tensor-Core Beamformer: A High-Speed Signal-Processing Library for Multidisciplinary Use. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 582–592. doi:10.1109/IPDPS64566.2025.00058

[32] Hiroyuki Ootomo and Rio Yokota. 2022. Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance. *The International Journal of High Performance Computing Applications* 36, 4 (June 2022), 475–491. doi:10.1177/10943420221090256

[33] Louis Pisha and Łukasz Ligowski. 2021. Accelerating Non-Power-Of-2 Size Fourier Transforms with GPU Tensor Cores. In *Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium*. Portland, OR, USA, 507–516. doi:10.1109/IPDPS49936.2021.00059

[34] Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Michael Schulte, Ralph Wittig, Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Summer Deng, Maxim Naumov, Paulius Micikevicius, Michael Siu, and Colin Verrilli. 2023. OCP Microscaling Formats (MX) Specification. 16 pages. https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf Version 1.0.

[35] N. L. Schryer. 1981. *A Test of a Computer's Floating-Point Arithmetic Unit*. Technical Report Computer Science Technical Report 89. AT&T Bell Laboratories, Murray Hill, NJ, Murray Hill, NJ 07974.

[36] Xuan You Tan, David Boland, and George Constantinides. 2012. FPGA Paranoia: Testing Numerical Properties of FPGA Floating Point IP-Cores. In *Reconfigurable Computing: Architectures, Tools and Applications*, Oliver C. S. Choy, Ray C. C. Cheung, Peter Athanas, and Kentaro Sano (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 290–301.

[37] Michela Taufer, Omar Padron, Philip Saponaro, and Sandeep Patel. 2010. Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. 1–9. doi:10.1109/IPDPS.2010.5470481

[38] Alexandre F. Tenca. 2009. Multi-operand Floating-Point Addition. In *2009 19th IEEE Symposium on Computer Arithmetic*. 161–168. doi:10.1109/ARITH.2009.27

[39] Jiqun Tu, Ian Karlin, John Camier, Veselin Dobrev, Tzanio Kolev, Stefan Henneking, and Omar Ghattas. 2026. Accelerating High-Order Finite Element Simulations at Extreme Scale with FP64 Tensor Cores. arXiv:2603.09038 [cs.DC] https://arxiv.org/abs/2603.09038

[40] Benjamin Valpey, Xinyi Li, Sreepathi Pai, and Ganesh Gopalakrishnan. 2025. An SMT Formalization of Mixed-Precision Matrix Multiplication. In *NASA Formal Methods*. Springer Nature Switzerland, Cham, 360–379.

[41] Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. 2025. Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference. In *Advances in Neural Information Processing Systems*, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (Eds.), Vol. 38. Curran Associates, Inc., 169819–169851. https://proceedings.neurips.cc/paper_files/paper/2025/file/f80094a824ba5912d4a2de169c404a40-Paper-Conference.pdf

Received 6 April 2026; revised 11 June 2026; accepted 3 July 2026