

Pearl Floating Point Scheme Specification

Pearl Research Team

September 2026

Abstract

This whitepaper specifies the proof-of-useful-work protocol of the Pearl blockchain, in which the unit of mining work is FP8 matrix multiplication on GPUs, extending the integer matrix-multiplication proof of useful work of Komargodski and Weinstein [4]. At a high level, the main idea is that post-noise (non-linear) quantization removes the low-rank shortcut, so tickets can hash output tiles directly. For verification purposes, the verifier recovers the bit-exact computation using the technique of Badash et al. [5]. Security rests on the hardness of multiplying quantized, low-rank-noised matrices faster than generically. An extension of the protocol to mixture-of-experts multiplication is also described.

Contents

1	Introduction	3
2	The abstract protocol	3
3	Preliminaries	4
4	Plaintext verifier	5
4.1	Verifier overview	5
4.2	Public parameters	6
4.3	Tile computation	6
4.4	Proof contents	6
4.5	Acceptance rule	7
4.6	Verifier rationale	8
5	Jackpot policy	9
5.1	Policy checks	9
5.1.1	Entry liveness	10
5.1.2	Noise floor	10
5.1.3	Tamed products	10
5.1.4	Unpredictable summands	11
5.2	Attack vectors	11
6	Useful miner overview	12
7	Known mining speedups	13
A	Mixture-of-experts verifier	14
B	Zero-knowledge verifier	15
C	Verifier arithmetic specifications	16
C.1	Floating point arithmetic	17
C.2	E4M3FN matrix multiplication	17
C.3	Noise-line sampler	18
C.4	Noisy quantization	19
C.5	Unpredictability specification	20
C.6	Commit type (FP10)	22

1 Introduction

Proof-of-work consensus makes block production costly by design [1, 2], but classical implementations require repeated hashings that have no value outside the protocol. A proof of useful work (PoUW) replaces the artificial puzzle with a computation someone already wants performed [3]. The obstacle is permissionlessness: the miner chooses its own inputs, so the protocol must guarantee that no choice of inputs, however degenerate, makes valid tickets cheaper than the prescribed amount of work.

This whitepaper specifies a proof-of-useful-work protocol whose unit of work is FP8 matrix multiplication on GPUs. A miner multiplies matrices of its own choosing, for instance the operands of an AI training or inference workload, and every tile of the product doubles as a lottery ticket for block production. The protocol covers ordinary matrix multiplication. An optional extension covers mixture-of-experts multiplications, in which routing is conceptually part of the left operand (Appendix A). It is the consensus layer of the Pearl blockchain: the same multiplications that serve AI workloads secure the network.

The starting point of this document is the PoUW of Komargodski and Weinstein [4] for arbitrary integer matrix multiplication, whose blockchain this protocol instantiates. Their protocol perturbs the operands with low-rank noise derived from a public seed, computes the noised product, and applies a random oracle to the transcript of the computation. The low-rank structure lets an honest party remove the noise from the output at negligible cost, giving a useful protocol with $1 + o(1)$ multiplicative overhead. The analysis is set over exact arithmetic in the random-oracle model, where, under their direct-product conjecture, even the all-zeroes matrices are as hard to multiply as any other pair. This document describes an alternative to the protocol of [4] that supports floating-point matrix multiplications. Verification requires the verifier to recompute a selected output tile bit-for-bit. This is made possible because the arithmetic of the whitelisted GPUs is deterministic and reproducible via the technique of Badash et al. [5].

Organization. Section 2 presents the abstract protocol (low-rank noising plus quantization) and the core hardness assumption in its cleanest setting. Section 3 fixes notation, hashing, and the work model. Section 5 specifies the jackpot policy. Section 4 specifies the plaintext verifier, Section 6 the honest useful miner, and Section 7 the known classes of mining speedups. The appendices specify the mixture-of-experts and zero-knowledge verifiers and the bit-exact arithmetic, sampling, and quantization procedures.

2 The abstract protocol

Let

$$A \in \mathbb{R}^{m \times k}, \quad B \in \mathbb{R}^{n \times k}, \quad C := AB^\top.$$

In a common application, the user chooses A and B at higher precision than the hardware format used for multiplication. If Q denotes quantization into that format, the user would ordinarily approximate C by $Q(A)Q(B)^\top$.

The protocol replaces this by the following procedure. Derive random rank- r noise matrices $N_A \in \mathbb{R}^{m \times k}$ and $N_B \in \mathbb{R}^{n \times k}$ and compute

$$\tilde{C} := Q(A + N_A)Q(B + N_B)^\top. \tag{1}$$

Each noise matrix is chosen unpredictably after the corresponding input has been committed. The product $\tilde{C}$ determines the proof-of-work lottery. Because the noise has low rank, it also yields a useful output after the cheap correction

$$\hat{C} := \tilde{C} - AN_B^\top - N_A B^\top - N_A N_B^\top.$$

Without quantization, $\hat{C} = C$. With it, the protocol is configured so that $\hat{C}$ approximates C with accuracy comparable to the baseline product $Q(A)Q(B)^\top$. Because r is small, the three subtracted terms are efficient to compute.

While the low-rank structure would create a shortcut if it survived quantization, entrywise quantization is nonlinear in general:

$$Q(A + N_A) \neq Q(A) + Q(N_A).$$

Although N_A has rank r , its entrywise-quantized image $Q(N_A)$ can have much higher rank. The central hardness premise is that this destroys the exploitable structure: the resulting product cannot be computed accurately with substantially less work than a generic quantized product. This premise depends on the data type and quantizer, so the protocol whitelists both.

Setting $A = B = 0$ reduces the lottery product to

$$\tilde{C}_0 = Q(N_A) Q(N_B)^\top,$$

which isolates the hardness premise.

Assumption 1 (Informal quantized-subspace hardness). *For every admissible k , there is an $r \ll k$ with the following property. With high probability over random r -dimensional subspaces $U_A, U_B \subset \mathbb{R}^k$, the following holds. Draw the rows of N_A, N_B randomly from U_A, U_B , respectively. On average, computing*

$$Q(N_A) Q(N_B)^\top$$

is not substantially easier than computing

$$Q(A) Q(B)^\top,$$

where A, B have generic random rows in $\mathbb{R}^k$. The same protocol-approved Q is used in both products. Both computations require work comparable to roughly mnk multiply-accumulate operations.

Concretely, the protocol uses $r = 32$ for $k \leq 2^{16}$. The assumption compares computational difficulty, not the two input distributions.

Relation to the transcript-unpredictability assumption. Assumption 1 is the floating-point analogue of the transcript-unpredictability assumption of [4]. There, low-rank noise makes the **output** of an adversarially chosen product cheap (for $A = B = 0$ the noised product is just $N_A N_B^\top$), so hardness must rest on the transcript of the computation. Here, quantization blocks the output shortcut, and hardness can rest on the output tiles themselves. A miner controls A and B and can try to plant structure that survives noising and quantization. Section 5 specifies the policy checks that rule out such structure.

3 Preliminaries

Notation. Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{n \times k}$ be matrices over the reals. Their useful product is

$$C := AB^\top \in \mathbb{R}^{m \times n}.$$

Both operands are stored as collections of row vectors. Thus C_{ij} is the inner product of row i of A with row j of B , which accounts for the transpose on B . For selected row-index sets I_A and I_B , write A_{I_A} and B_{I_B} for the corresponding strips of rows. An index set contains no repetitions, and its strip is arranged in increasing index order. All row and column indices start at zero. The seeds `noise_seedA` and `noise_seedB` determine the noised operands $\tilde{A}$ and $\tilde{B}$. Write

$$\tilde{C}_{I_A, I_B} := \tilde{A}_{I_A} (\tilde{B}_{I_B})^\top$$

for the noised tile that the verifier recomputes. I_A indexes the rows of $\tilde{C}$, and I_B indexes its columns. The ticket digest is a keyed function `Ticket` from tiles to $\{0, 1\}^\lambda$, with $\lambda = 256$. Let $\mathcal{P}_A$ and $\mathcal{P}_B$ be the patterns that tile the rows of A and of B , hence the rows and columns of $\tilde{C}$ (e.g., a periodic tiling into blocks of consecutive rows and columns). Each tile is one (I_A, I_B) pair, and the patterns also fix how the tile's entries are grouped and ordered for hashing. We then use a randomness-extraction function `Extract` to efficiently compress a tile into a consensus 64-byte encoding that depends on every entry:

$$\text{Ticket}(\tilde{C}_{I_A, I_B}) := \text{H}_{\text{"jackpot"}}(\text{Extract}(\tilde{C}_{I_A, I_B}, \mathcal{P}_A, \mathcal{P}_B); \text{noise_seed}_A).$$

Blockchain state and the state window. A proof of work must be bound to the blockchain state, so that work cannot be prepared before the state is known or reused across blocks. In the basic setting the miner binds its commitments to the block header it is proposing, and the proposed header in turn commits to the current tip.

We index headers by depth relative to the proposed block. Write $\sigma_0 = \hat{\sigma}$ for the proposed header, σ_1 for its parent (the current tip), σ_2 for its grandparent, and so on. The context Σ carries these headers.

Consensus relaxes the basic setting by a state window of depth D (e.g., $D = 3$), allowing the miner to bind its commitment to B to any one of the D headers $\sigma_0, \dots, \sigma_{D-1}$ in the window. The commitment to A remains bound to $\hat{\sigma}$.

Domain-separated hashes. Let q be a 32-byte parent key and let ℓ be a domain-separation string, e.g. “jackpot”. For readability, each label displayed in this paper omits an implicit protocol-version component. The concrete label includes both the protocol version and the displayed role. BLAKE3 accepts one key, so q occupies its key slot. Derive the key for role ℓ by hashing the string under q :

$$\text{Subkey}(q, \ell) := \text{BLAKE3}(\ell; \text{key} = q).$$

Then

$$H_\ell(x; q) := \text{BLAKE3}(x; \text{key} = \text{Subkey}(q, \ell)).$$

When no transcript-derived parent key is available, use unkeyed $\text{BLAKE3}(\ell)$ rather than Subkey . That is,

$$H_\ell(x) := \text{BLAKE3}(x; \text{key} = \text{BLAKE3}(\ell)).$$

This preserves the domain-separation interface, without key unpredictability.

Work model. We measure work in multiply-accumulate operations (MACs). The direct cost of the selected tile is

$$T_{\text{tile}} := |I_A| \cdot |I_B| \cdot k.$$

We count MACs for concreteness only. Any model of computation serves, with T_{tile} read as the corresponding tile cost in that model. The MAC count only fixes the scale. It intentionally excludes the work required for quantization, noise generation, hashing, commitment verification, and memory movement.

4 Plaintext verifier

4.1 Verifier overview

The verifier checks one tile of the matmul, never the whole product. The proof names that tile and carries only the data the tile reads. Appendix A gives the optional MoE verifier, in which routing is conceptually part of A . Appendix B gives the zero-knowledge verifier for either mode.

1. **Receive the strips.** The proof carries the selected rows of A and B . The selected B rows become columns of the output tile.
2. **Check the trees, derive the seeds.** The strips arrive with minimal Merkle multi-proofs. The verifier recomputes the roots of A and B from the strips and their proofs, and rejects a mismatch. It derives `noise_seedB` from the root of B and the chain state, then `noise_seedA` from `noise_seedB` and the root of A . The seeds follow the commitments, so the miner cannot pick its noise.
3. **Noise the strips, compute the tile.** The seeds expand into the noise of the two strips, giving $\tilde{A}$ and $\tilde{B}$. The verifier computes $\tilde{C} = \tilde{A}(\tilde{B})^\top$ with the kernel the miner ran, so both tiles agree bit for bit.

4. **Check the difficulty.** The verifier computes the ticket digest $\text{Ticket}(\tilde{C})$, a 32-byte value keyed by a domain-separated derivative of `noise_seedA`. The tile wins if this value is at or below the effective target, which scales the network target by T_{tile} .
5. **Check the strips.** The verifier runs the **jackpot policy** on the two strips, either approving or rejecting the tile as a rich-enough matmul work.

4.2 Public parameters

The dimensions m, n, k are public. The proof also supplies a noise rank r , a **Quant**, a **Device**, operand hash identifiers `hash_idA`, `hash_idB`, and miner-chosen periodic partitions $\mathcal{P}_A, \mathcal{P}_B$. Each hashing identifier selects the precise way hashing is done, and is chosen from a consensus-allowed list. Notably, it controls the chunk size internal to BLAKE3. Each partition divides an operand’s rows into strips, thus breaking the entire $A \cdot B$ matmul deterministically into many independent mmults each consisting of a fixed number of rows of A and a fixed number of rows of B . A partition is periodic: it is described by a short pattern that repeats along the axis, so every strip has the same shape and the strips never overlap. The pattern is encoded in a few bytes, and a base is valid only if it lies on the pattern’s grid.

Device and **Quant** are enumerations, each chosen from a consensus-allowed list. **Device** names the GPU architecture, and **Quant** names the quantization scheme. This document specifies one such quantization scheme: the FP10 commit type of Appendix C.6 together with the noisy quantization of Appendix C.4. Finally, e is the number of experts of the mixture-of-experts extension (Appendix A). Ordinary matrix multiplication sets $e := 0$.

The public parameter tuples for the two operands are

$$\begin{aligned} \mathbf{p}_B &:= (n, k, r, \text{Quant}, \text{Device}, \text{hash_id}_B, \mathcal{P}_B, e), \\ \mathbf{p}_A &:= (m, \text{hash_id}_A, \mathcal{P}_A). \end{aligned} \tag{2}$$

4.3 Tile computation

After both seeds exist, the miner chooses one strip from each partition, identified by bases t_A, t_B . Write

$$I_A := \text{Rows}(\mathcal{P}_A, t_A), \quad I_B := \text{Rows}(\mathcal{P}_B, t_B) \tag{3}$$

for the selected rows, that is, the rows of the strip based at t_A (resp. t_B), in increasing order. The partitions also determine how **Extract** arranges tile values into the 64-byte input to the jackpot hash: they split the tile into 16 subtiles, and **Extract** folds the FP32 bit patterns of each subtile into one 32-bit word using a mixing step, so that every tile entry influences the digest. E.g., over the subtile’s entries x ,

$$w \leftarrow \text{rotl}_{13}(w \cdot c + \text{bits}(x)) \bmod 2^{32},$$

with c an odd constant and $\text{bits}(x)$ the 32-bit encoding of x .

4.4 Proof contents

The blockchain context supplies

$$\Sigma := (\hat{\sigma}, \sigma_1, \dots, \sigma_{D-1}),$$

where σ_j is the authenticated depth- j ancestor header. The headers $\hat{\sigma}, \sigma_j$ omit the proof commitment. The verifier rejects a proof whose carried σ_d is not a member of Σ .

The proof carries:

1. the selected ancestor header σ_d .
2. the public parameter tuples $\mathbf{p}_A, \mathbf{p}_B$.

3. tile bases t_A, t_B for the two periodic partitions.
4. openings π_A, π_B with the leaves covering the clean A and B strips, respectively, and their Merkle sibling hashes.

The proof omits the roots H_A, H_B , the noise seeds, noise factors, noised strips, product tile, extractor output, and jackpot hash.

4.5 Acceptance rule

We now specify the verifier's logical acceptance rule in detail (byte encoding and proof serialization are not specified). Appendix A replaces only the rules needed for mixture-of-experts.

The verifier rejects unless every operation below succeeds.

1. Validate the context and public arguments.

Run

$$\text{CheckPublic}(\Sigma, \sigma_d, \mathbf{p}_A, \mathbf{p}_B, t_A, t_B, \pi_A, \pi_B). \quad (4)$$

CheckPublic verifies that the parent of the proposed header $\hat{\sigma}$ connects it to the blockchain, authenticates the carried σ_d against the context Σ , and validates the header's remaining chain fields. It also sanity checks $\mathbf{p}_A$ and $\mathbf{p}_B$, and requires $e = 0$. In particular,

$$1 \leq m, n \leq 2^{24}, \quad 1024 \leq k \leq 2^{16}, \quad r = 32, \quad 32 \mid k. \quad (5)$$

For I_A, I_B defined in (3), it requires

$$I_A \subseteq \{0, \dots, m-1\}, \quad I_B \subseteq \{0, \dots, n-1\}. \quad (6)$$

Let $b_A \times b_B$, as determined by $\mathcal{P}_A, \mathcal{P}_B$, be the shape of the 32-bit word array produced by **Extract**. Its output is 64 bytes, so the verifier requires

$$b_A \cdot b_B = 16, \quad |I_A| \geq 4, \quad |I_B| \geq 16, \quad 256 \leq |I_A| \cdot |I_B| \leq 2048. \quad (7)$$

For performance, the private instance is furthermore bounded as

$$k \cdot (|I_A| + |I_B|) \leq 2^{22}. \quad (8)$$

2. Verify the openings.

With $\sigma_0 := \hat{\sigma}$, derive

$$\begin{aligned} \mathbf{key}_B &:= \mathbf{H}_{\text{"key-B"}}(\sigma_d), \\ \mathbf{key}_A &:= \mathbf{H}_{\text{"key-A"}}(\hat{\sigma}). \end{aligned}$$

For each $X \in \{A, B\}$, compute

$$(H_X, X_{I_X}) := \text{VerifyOpen}_X(\pi_X; \mathbf{key}_X, \mathbf{p}_X, I_X). \quad (9)$$

VerifyOpen_X checks that π_X opens exactly I_X . It uses $\mathbf{key}_X$ as the key for $\mathbf{hash_id}_X$. It requires the unique minimal set of leaves and sibling hashes for the operand shape and $\mathbf{hash_id}_X$. It reconstructs the Merkle roots and returns the aggregate root H_X and clean strip X_{I_X} . Opened values must use the encoding specified by **Quant**, and X_{I_X} is returned in that encoding. Decoding it is the first step of $\text{NoisyQuantize}_{\mathbf{p}_X}$ in step 4. It rejects NaN or infinity. It accepts subnormal values.

3. Derive the noise seeds.

Compute

$$\begin{aligned} \mathbf{noise_seed}_B &:= \mathbf{H}_{\text{"seed-B"}}(H_B, \mathbf{key}_B, \mathbf{p}_B), \\ \mathbf{noise_seed}_A &:= \mathbf{H}_{\text{"seed-A"}}(H_A, \mathbf{noise_seed}_B, \mathbf{key}_A, \mathbf{p}_A). \end{aligned} \quad (10)$$

4. Recompute the ticket.

The verifier uses one consensus map `SampleLine` for all E and F lines. Given a seed `noise_seedX`, an operand $X \in \{A, B\}$, a factor $Z \in \{E, F\}$, and a line index ℓ , it returns a length- r line of $\text{FP8}_{\text{E4M3FN}}$ values. The map is deterministic and does not depend on draw order: each line is addressed by (X, Z, ℓ) alone. Each output line has approximate Euclidean norm 256. The construction is in Appendix C.3.

For $X \in \{A, B\}$ and $s_X := \text{noise_seed}_X$, compute

$$\begin{aligned} E_X[\ell, :] &:= \text{SampleLine}(s_X, X, E, \ell), \quad \ell \in I_X, \\ F_X[u, :] &:= \text{SampleLine}(s_X, X, F, u), \quad 0 \leq u < k. \end{aligned}$$

Stacking the lines gives per-operand factors $E_X \in \text{FP8}^{|I_X| \times r}$ and $F_X \in \text{FP8}^{k \times r}$, so each strip receives the rank- r noise $E_X F_X^\top$ expanded on exactly the opened rows. Then compute

$$\begin{aligned} \tilde{A}_{I_A} &:= \text{NoisyQuantize}_{\mathbf{p}_A}(A_{I_A}, E_A, F_A), \\ \tilde{B}_{I_B} &:= \text{NoisyQuantize}_{\mathbf{p}_B}(B_{I_B}, E_B, F_B), \\ T &:= \text{MatMul}_{\text{Device}}(\tilde{A}_{I_A}, \tilde{B}_{I_B}), \\ z &:= \text{Extract}(T, \mathcal{P}_A, \mathcal{P}_B), \\ J &:= \text{H}^{\text{"jackpot"}}(z; \text{noise_seed}_A). \end{aligned}$$

Appendix C.4 specifies `NoisyQuantize`. Every intermediate result of the specified floating-point operations must be finite. The verifier rejects a NaN or positive or negative infinity.

5. Apply the policy and target.

Let τ be the target difficulty as specified in $\hat{\sigma}$. Accept exactly when

$$\text{JackpotPolicy}(I_A, A_{I_A}, I_B, B_{I_B}; \hat{\sigma}, \sigma_d, \mathbf{p}_A, \mathbf{p}_B, H_A, H_B) = \text{approve} \quad (11)$$

as specified in Section 5.1, and

$$\text{uint256}_{\text{LE}}(J) \leq \tau \cdot |I_A| \cdot |I_B| \cdot k. \quad (12)$$

4.6 Verifier rationale

Openings. The verifier interprets each root under one operand shape, `hash_id`, and key. A minimal Merkle proof has one expected set of leaves and sibling hashes. This rule prevents proof padding and bounds the input before arithmetic starts. An opening authenticates only the disclosed rows. It does not prove knowledge of the undisclosed matrix entries.

Opened values use the encoding selected by `Quant`. Rejecting NaN and infinity removes exceptional arithmetic inputs.

State window. The B commitment can use any header σ_d in the state window. Relative to using only σ_0 , this gives at most a D -fold opportunity to preprocess the B side. For a useful miner, the same window gives up to $D - 1$ block intervals to replace its matrices after the blockchain tip changes. It need not replace them immediately. The A commitment uses and binds the proposed header.

Challenge order. The seed chain fixes the intended Fiat–Shamir order. The miner fixes B and $\mathbf{p}_B$ before it obtains `noise_seedB`. It can then choose A , but it must fix A and $\mathbf{p}_A$ before it obtains `noise_seedA`. This order allows A to depend on the first challenge without allowing it to depend on the second challenge.

Including $\mathbf{p}_X$ in each seed hash binds the operand dimensions, noise rank, `Quant`, `Device`, hash identifiers, and partition to that noise challenge. These bindings do not prevent transcript grinding or shared work across commitments and depths. The batched and grinding security definitions must account for those choices.

Hash domains. The labels “key-A”, “key-B”, “seed-A”, and “seed-B” separate operand roles and transcript stages. The sampler uses a seed-derived key and the typed address (X, Z, ℓ) as its message. This design gives deterministic random access and makes draw order irrelevant. The seed already binds k and r , so the address does not repeat them.

The “noise-line” and “jackpot” labels derive different keys from the same seed.

Arithmetic and extraction. Quant fixes the commit type, the quantized type, the quantization algorithm, and $\text{round}_{\text{BF16} \rightarrow \text{BF16}}$. Device fixes the remaining arithmetic. This distinction is necessary because E4M3FN operands alone do not determine a bit-exact result on all devices. Reproducing a GPU matmul exactly requires pinning down rounding direction, subnormal handling, and accumulation order. Badash, Boneh, Komargodski, and Srivastava [5] characterize these behaviors across NVIDIA architectures and precisions, including FP8, and reproduce them off the GPU. The protocol uses the specified floating-point operations, not ideal real-number arithmetic. A non-finite intermediate result therefore causes verification failure.

Difficulty and eligibility. Scaling the target by $|I_A| \cdot |I_B| \cdot k$ makes the expected number of wins proportional to the direct MAC cost of the checked tile. The upper tile bound limits verifier work. The lower bounds ensure that Extract is fed with sufficient entropy.

The policy in Section 5.1 and the target have different purposes. The policy determines tile eligibility. The target sets the win rate for eligible tiles.

5 Jackpot policy

Beyond the idealized setting of Section 2, the protocol must stay secure when the matrices A, B are not zero and the arithmetic is a concrete floating-point format. This opens an attack surface, which the verifier addresses by a pointwise set of checks that either accept or reject a single lottery tile. We call it the jackpot policy.

5.1 Policy checks

The policy approves exactly when every check below passes. It runs after the verifier has rejected Infinity/NaN inputs or intermediate results.

Let $i \in I_A$ index a row of A , let $j \in I_B$ index a row of B , and let u , with $0 \leq u < k$, index a column. Write $\bar{A}$ and $\bar{B}$ for A and B after each row is normalized into quantized units. The per-row noise scales are

$$\delta := \frac{1}{2}, \quad \sigma_i^A := \delta \ell_2(\bar{A}_i), \quad \sigma_j^B := \delta \ell_2(\bar{B}_j).$$

Each row thus receives noise whose scale is the fixed fraction δ of that row’s ℓ_2 scale, so the expected noise-to-signal ratio is the same for every row (cf. Appendix C.4). The noise N^X for operand $X \in \{A, B\}$ is a rank- r matrix, already carrying the per-row scale, so that $\ell_2(N_i^X) \sim \sigma_i^X \ell_2(\bar{X}_i)$. The entrywise marginal distribution of N_{iu}^X is roughly $\mathcal{N}(0, (\sigma_i^X)^2)$, and its exact definition is specified in Appendix C.4. Let Q be the quantization into FP8 map (cf. Appendix C.4). The operands of the matmul are

$$\tilde{A}_{iu} := Q(\bar{A}_{iu} + N_{iu}^A), \quad \tilde{B}_{ju} := Q(\bar{B}_{ju} + N_{ju}^B).$$

Let $c_{ij,t}$ be the FP32 accumulator after the first t groups of Appendix C.2, and let G be the number of groups ($k/32$). The replay magnitude is

$$M_{ij} := \max \left\{ \max_{0 \leq u < k} |\tilde{A}_{iu} \tilde{B}_{ju}|, \max_{1 \leq t \leq G} |c_{ij,t}| \right\}.$$

Thus M_{ij} replays the accumulation of cell (i, j) and records the largest magnitude that appears in it, either as a single summand or as a partial accumulator, and $\text{ulp}_{\text{Device}}(M_{ij})$ is the spacing of the accumulation grid at that magnitude. The *unpredictability* of the iju summand is defined as

$$v_{iju} := \frac{(\overline{A}_{iu}^2 + (\sigma_i^A)^2)(\overline{B}_{ju}^2 + (\sigma_j^B)^2)}{2^{21}}. \quad (13)$$

Up to the fixed denominator, the numerator multiplies the second moments of the two noised entries $\overline{A}_{iu} + N_{iu}^A$ and $\overline{B}_{ju} + N_{ju}^B$, so v_{iju} tracks how widely the iju summand spreads as the noise seeds vary. The unpredictable-summands check below compares this spread against the accumulator grid the summand meets.

The policy uses the following constants:

$$\begin{aligned} \tau_{\text{idle}} &:= 8, & \varepsilon_{\text{idle}} &:= 1/64, & \sigma_{\min} &:= 1, \\ \tau_{\text{tame}} &:= 256, & \varepsilon_{\text{tame}} &:= 1/64, & \varepsilon_{\text{pred}} &:= 1/16. \end{aligned}$$

All checks in this section are phrased as if arithmetic is happening in the real numbers, but the verifier implementation is authoritative.

5.1.1 Entry liveness

For each side, define its set of idle entries:

$$\begin{aligned} \mathcal{D}_A &:= \{(i, u) : |\overline{A}_{iu}| \geq \tau_{\text{idle}} \sigma_i^A\}, \\ \mathcal{D}_B &:= \{(j, u) : |\overline{B}_{ju}| \geq \tau_{\text{idle}} \sigma_j^B\}. \end{aligned}$$

An idle entry exceeds its row's noise scale by the factor τ_{idle} , so the injected noise has standard deviation at most a $1/\tau_{\text{idle}}$ fraction of the entry's magnitude, and the quantized value of the entry is largely predictable. The check requires

$$|\mathcal{D}_A| \leq \varepsilon_{\text{idle}} |I_A| k, \quad |\mathcal{D}_B| \leq \varepsilon_{\text{idle}} |I_B| k.$$

5.1.2 Noise floor

The check requires

$$\sigma_i^A \geq \sigma_{\min} \quad (\forall i \in I_A), \quad \sigma_j^B \geq \sigma_{\min} \quad (\forall j \in I_B).$$

The scales are expressed in the quantized units of $\overline{A}$ and $\overline{B}$, so the floor requires every opened row to carry at least one quantized unit of noise.

5.1.3 Tamed products

Let $\text{ufp}_{\text{Device}}(x) = \text{ulp}_{\text{Device}}(x) \cdot 2^{W_{\text{Device}}}$ be the largest power of 2 not exceeding $|x|$. Define the portion of the tile with unusually high M_{ij} :

$$\mathcal{U} := \{(i, j) : \text{ufp}_{\text{Device}}(M_{ij}) > \tau_{\text{tame}} \sqrt{k} \sigma_i^A \sigma_j^B\} \subseteq I_A \times I_B.$$

Here $\sqrt{k} \sigma_i^A \sigma_j^B$ is the natural scale of a length- k dot product whose summands carry independent noise of scales σ_i^A and σ_j^B . A cell that exceeds it by the factor τ_{tame} indicates an unusually coherent pair of rows, in whose product the noise accounts for an unusually small relative share. The check requires

$$|\mathcal{U}| \leq \varepsilon_{\text{tame}} |I_A| |I_B|.$$

5.1.4 Unpredictable summands

Using $\text{ulp}_{\text{Device}}$ as defined in Appendix C.5, let

$$\mathcal{S} := \{(i, j, u) : v_{iju} < \text{ulp}_{\text{Device}}(M_{ij})^2\}. \quad (14)$$

The check requires the tile-wide bound

$$|\mathcal{S}| \leq \varepsilon_{\text{pred}} k |I_A| |I_B|.$$

Our verifier’s implementation slightly deviates from (14). We replace v_{iju} with an approximate lower bound of it, correspondingly making the policy slightly stricter. Appendix C.5 includes the full details.

5.2 Attack vectors

The protocol as instantiated in Sections 5 and 4 carries a considerable amount of detail, mainly because the permissionless setting leaves the miner many directions in which to attack. The paragraphs below describe each direction and the mechanism that answers it.

Degenerate arithmetic. On NaN or infinite values, device arithmetic stops resembling arithmetic over $\mathbb{R}$: a NaN propagates through every later operation, and an infinity absorbs every further finite summand. A tile driven into a non-finite state therefore collapses to a value predictable from few of its inputs. The verifier forbids NaN and infinity in the input operands and in every intermediate result, which keeps accepted tiles inside the regime that the checks of Section 5.1 model.

Noise seed grinding. The seeds are unpredictable, but a miner can grind them, hunting for structure in the noise factors that admits a faster multiplication. For this reason, structure in F_A , E_B , or F_B that yields a meaningful speedup must be rare enough that the expected cost of grinding for it is impractical. The lower bound $k \geq 1024$ and $|I_B| \geq 16$ serve this requirement. Structure in E_A tolerates a looser threshold, as E_A is derived from both A and B ’s commitments while F_A , E_B , or F_B are derived from B ’s commitment alone. This asymmetry is why some checks treat the two sides differently.

For example, if grinding makes two rows of E_B in the same I_B identical, the jackpot for $I_A \times I_B$ may cost $k|I_A|(|I_B| - 1)$ rather than $k|I_A||I_B|$. Nevertheless, the multiplicative speedup, $|I_B|/(|I_B| - 1) < 1.1$, is acceptable given the nontrivial grinding required: **SampleLine** has about 150 bits of min-entropy. Similarly, for $B = \mathbf{1}$, quantization leaves $Q(B + E_B F_B^T)_{ij} = B_{ij}$ in roughly 10% of entries. With $k \geq 1024$, grinding for a substantially larger fraction across a whole tile is impractical.

No-noise entries. A miner can commit operands whose entries dwarf the per-row noise scale. Quantization then absorbs most of the noise: such an entry’s quantized value ranges over only a few adjacent FP8 codes, so large parts of $\tilde{A}$ and $\tilde{B}$, and of every tile they produce, are predictable before the seeds are known. The entry-liveness check bounds the fraction of these entries on each side. Note that the proportion of noise-agnostic entries can be upper-bounded a priori as a function of δ alone, with no check at all, but the resulting estimates are generally looser.

Low local entropy. A miner can commit rows whose peak entry dwarfs their typical entry. The row normalization of Appendix C.4 must leave room for the peak, which compresses the rest of the row, noise included, toward zero, until the quantized noise takes only a handful of small FP8 values. Operands with so little local entropy invite specialized hardware: a circuit built for a reduced alphabet of values can multiply faster than a general FP8 kernel, breaking the premise that a multiply-accumulate costs what the work model charges for it. The noise-floor check removes this option by requiring every opened row to carry noise of scale at least $\sigma_{\min}$ in quantized units.

Coherent dot products. When a row of A aligns with a row of B , their dot product concentrates: the replay magnitude M_{ij} grows with the coherence while the noise stays near its natural scale $\sqrt{k}\sigma_i^A\sigma_j^B$. The accumulation grid follows the largest magnitude in the cell, so in a coherent cell the noise affects fewer of the retained bits, and more of the cell’s exact bits become predictable from the signal alone. The tamed-products check caps the fraction of such cells. τ_{tame} is chosen so that typical AI workloads easily fit this bound.

Unpredictability. The unpredictable-summands check is motivated by a mining strategy where the matmul is arranged so that the device accumulation truncates the noise, up to fast low-rank correction terms. For $X \in \{A, B\}$ write $S_{iu}^X = \bar{X}_{iu} + N_{iu}^X$ and $T_{iu}^X := \tilde{X}_{iu} - S_{iu}^X$. We can expand

$$\tilde{C}_{ij} = \sum_{u=0}^{k-1} \tilde{A}_{iu} \cdot \tilde{B}_{ju} = \sum_{u=0}^{k-1} (S_{iu}^A + T_{iu}^A)(S_{ju}^B + T_{ju}^B).$$

If the miner chooses A and B of low rank, then the matrices S^A and S^B have low rank too, since the noise contributes rank at most r . Every term of the expansion with a low-rank factor is cheap, so the only part of $\tilde{C}_{ij}$ that is expensive to compute is

$$\sum_{u=0}^{k-1} T_{iu}^A \cdot T_{ju}^B. \quad (15)$$

For entries $\bar{A}_{iu}$ which are not much larger than $\ell_2(\bar{A}_i)$, the marginal distribution of T_{iu}^A has variance that linearly scales with $\bar{A}_{iu}^2 + (\sigma_i^A)^2$ when letting the noise seeds vary, with a constant around $2^{-10.5}$. The unpredictable-summands check ensures that all but a small proportion of the summands in (15) survive the device’s truncation and affect the eventual dot product $\tilde{C}_{ij}$.

6 Useful miner overview

This section describes the intended workflow of an honest miner: one that mines with multiplications it wants computed anyway. The miner holds a queue of jobs, each a pair A, B of $\mathbb{R}$ matrices whose useful product is $C = AB^\top$, and interleaves the following steps with ordinary workload.

1. **Prequantize.** Encode A, B in the commitment format (FP10). The protocol does not mandate the encoding procedure. A recommended construction appears in Appendix C.6.
2. **Commit.** Hash B under `hash_idB` keyed by a recent blockchain header, and derive `noise_seedB`. Then commit A under the proposed header and derive `noise_seedA`.
3. **Noise and quantize.** Derive E_X, F_X from the seeds and compute $\tilde{X} = \text{NoisyQuantize}_{\text{p}_X}(X, E_X, F_X)$ for $X \in \{A, B\}$. This costs $O((m+n)kr)$ MACs and one pass over the operands.
4. **Multiply.** Run `MatMulDevice` once on the full noised operands. This is the dominant cost, mnk MACs, and it is the same kernel the job would run without the protocol, applied to perturbed operands.
5. **Scan for winners.** Cut the output into the tiles induced by $\mathcal{P}_A, \mathcal{P}_B$. For each tile compute the extractor output and the jackpot hash, and compare against the scaled target. Each tile wins with probability proportional to its cost T_{tile} , so the expected number of wins per job is proportional to its total MAC count. Losing tiles are discarded and reveal nothing.
6. **Open on a win.** For a winning tile, authorized by the jackpot policy, assemble the proof of Section 4.4.

Recovering the useful product. The above description computes $\tilde{C} = \tilde{A}\tilde{B}^\top$, not C . The two differ by terms the miner can reconstruct cheaply once the seeds are known: the signal–noise cross terms carry a rank- r factor and cost $O((mn + mk + nk)r)$ MACs, and the per-row descaling induced by the quantization. What cannot be removed, namely the terms involving the quantization errors (15), is of the same order as the error that directly quantizing A and B to FP8 would introduce. The miner therefore recovers the useful product at FP8-level accuracy for the cost of the noised product plus lower-order corrections.

Overhead accounting. Beyond the product itself, the miner pays for hashing the operands, noising, scanning for winning tiles, and, rarely, assembling a proof. Each of these is $O((mn + mk + nk)r)$ MACs or a single hashing pass over A, B or $\tilde{C}$, against mnk for the product, so the total overhead is a $1 + o(1)$ factor when $m, n, k \gg r$. When B is fixed in advance, its hashing and noising need only be refreshed every few blockchain-tip changes. This preprocessing improves the concrete efficiency.

7 Known mining speedups

If we drop the usability requirement, a miner may gain a speedup relative to a useful miner. The purpose of this section is not to specify the guardrails that mitigate these speedups, but to record which high-level speedup classes were in scope when the protocol was designed.

Input crafting. A miner that does not need the product can choose operands for speed rather than for use: low-rank matrices, rows dominated by a few peaks, rows whose noise quantizes away, or operand pairs with strongly correlated rows. The jackpot policy of Section 5.1 is the defense, imposing statistical properties that ensure unpredictability.

Precision shortcuts. The device arithmetic of Appendix C.2 truncates summands below the accumulation grid, so a miner can skip any summand it knows will round away and save its MACs. The unpredictable-summands check in Section 5.1 caps the fraction of skippable summands at $\varepsilon_{\text{pred}}$, and the tamed-products check keeps cells out of the magnitude regime where whole groups of summands round away at once.

Seed and commitment grinding. A miner can retry commitments to search for favorable noise: exploitable structure in the factors (Section 5.2) or otherwise favorable transcripts. Each discarded attempt costs a commitment and a seed derivation, so this class is bounded by the usual Fiat–Shamir grinding argument.

Work reuse. A miner can try to extract several tickets from one multiplication: chaining instances by resubmitting noised outputs as fresh inputs, or amortizing shared operands, noise, or preprocessing across a batch of jobs. These attacks succeed only if revealing a noised tile makes the clean tile’s ticket cheaper to predict.

Better kernels and hardware. A miner with a faster bit-exact implementation of $\text{MatMul}_{\text{Device}}$, through better scheduling or better silicon, outpaces other miners, but this is the same advantage an honest miner gains and is not an attack on the protocol: the work model charges everyone in the same units.

References

- [1] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.
- [2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

- [3] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. *IACR Cryptology ePrint Archive*, page 203, 2017.
- [4] Ilan Komargodski and Omri Weinstein. Proofs of useful work from arbitrary matrix multiplication. arXiv:2504.09971; *IACR Cryptology ePrint Archive*, Report 2025/685, 2025.
- [5] Erez Badash, Dan Boneh, Ilan Komargodski, and Megha Srivastava. Hawkeye: Reproducing GPU-level non-determinism. arXiv:2603.20421, 2026. To appear in *MLSys 2026*.
- [6] Jack O’Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O’Hearn. BLAKE3: one function, fast everywhere. Specification, 2020. <https://github.com/BLAKE3-team/BLAKE3-specs>.

A Mixture-of-experts verifier

This appendix specifies the mixture-of-experts extension as changes to the plaintext verifier.

Expert matmuls. An MoE job with activation matrix A has $e \geq 1$ experts, with each expert having $\eta := n/e$ rows in its right operand. For an expert w let $B^{(w)} \in \mathbb{R}^{\eta \times k}$ be its right operand matrix, $R[w]$ be the sorted list of rows of A routed to that expert, $s_w := |R[w]|$, and left operand

$$A_i^{(w)} := A_{R[w][i]}, \quad 0 \leq i < s_w.$$

The useful expert matmul is thus

$$C^{(w)} := A^{(w)}(B^{(w)})^\top \in \mathbb{R}^{s_w \times \eta}.$$

A token may occur in several routing lists or in none.

Protocol encoding. The protocol stacks the expert matrices as

$$B := \begin{bmatrix} B^{(0)} \\ \vdots \\ B^{(e-1)} \end{bmatrix} \in \mathbb{R}^{n \times k}.$$

This complete stack is committed and noised as one matrix, so H_B covers all experts. We remind the reader that e is part of $\mathbf{p}_B$ (as in (2)). The A -side public parameter tuple (see (2)) is extended to

$$\mathbf{p}_A := (m, \text{hash_id}_A, \mathcal{P}_A, \text{hash_id}_R, \text{hash_id}_O)$$

as detailed below:

1. Encode the routing-list lengths by $O = (O_0, \dots, O_{e-1})$ with $O_i := \sum_{w \leq i} s_w$.¹
2. Define

$$R^{\text{flat}} := R[0] \parallel \dots \parallel R[e-1]$$

where $\parallel$ denotes concatenation. Note: $|R^{\text{flat}}| = O_{e-1}$, and more generally the partition of R^{flat} to lists is pinned by O .

3. Using key_A , hash_id_R hashes R^{flat} to H_R , and hash_id_O hashes O to H_O .

Proof contents. In addition to Section 4.4, the proof includes:

1. Winning expert w and the complete cumulative-count list O .
2. Opening π_R with the leaves covering the complete list $R[w]$ and their Merkle sibling hashes.

The roots H_R, H_O are omitted, and w, t_A, t_B are miner-selected after both noise seeds exist.

¹Throughout, $O_{-1} := 0$, so that $|R[w]| = O_w - O_{w-1}$ and the interval $[O_{w-1}, O_w)$ are defined for $w = 0$ as well.

Verifier changes

1. Validate the context and public arguments.

CheckPublic accepts the extended $\mathbf{p}_A$, receives w, O, π_R , and replaces $e = 0$ with

$$1 \leq e \leq 1024, \quad e \mid n, \quad 0 \leq w < e.$$

It requires

$$|O| = e, \quad 0 \leq O_0 \leq \dots \leq O_{e-1}, \quad |R[w]| = O_w - O_{w-1} = s_w,$$

and $R[w]$ must increase strictly in $[0, m)$. No other routing list is validated. Both patterns are expressed in expert-local coordinates:

$$\begin{aligned} U_A &:= \text{Rows}(\mathcal{P}_A, t_A) \subseteq \{0, \dots, s_w - 1\}, & I_A &:= R[w][U_A], \\ U_B &:= \text{Rows}(\mathcal{P}_B, t_B) \subseteq \{0, \dots, \eta - 1\}, & I_B &:= w\eta + U_B. \end{aligned}$$

These definitions replace the original selected rows (see (3)). Partition-base and extraction-layout checks use U_A, U_B , while global row bound checks use I_A, I_B . The maps preserve cardinality, so the existing tile-shape bounds are unchanged.

2. Verify the openings.

The verifier interprets R^{flat} as containing exactly O_{e-1} token indices. Under $\mathbf{key}_A$ and $\mathbf{hash_id}_R, \pi_R$ minimally opens $[O_{w-1}, O_w)$ of R^{flat} and reconstructs H_R , using the same minimal-opening rule as π_A, π_B . It hashes the supplied O under $\mathbf{hash_id}_O$ and $\mathbf{key}_A$ to obtain H_O . It then verifies π_A, π_B against I_A, I_B .

3. Derive the noise seeds.

Keep the B -side equation in (10) and replace the A -side equation with

$$\mathbf{noise_seed}_A := \mathbf{H}_{\text{"seed-A"}}(H_A, H_R, H_O, \mathbf{noise_seed}_B, \mathbf{key}_A, \mathbf{p}_A).$$

Thus e is fixed through $\mathbf{p}_B$ before $\mathbf{noise_seed}_B$. After that seed, A , the extended $\mathbf{p}_A$, H_R , and H_O are fixed before $\mathbf{noise_seed}_A$.

4. Recompute the ticket.

For $i \in U_A$ and $j \in U_B$, the verifier samples

$$\begin{aligned} E_A[R[w][i], :] &:= \text{SampleLine}(\mathbf{noise_seed}_A, A, \mathbf{E}, R[w][i]), \\ E_B[w\eta + j, :] &:= \text{SampleLine}(\mathbf{noise_seed}_B, B, \mathbf{E}, w\eta + j). \end{aligned}$$

Because the E -lines depend only on global indices and the F -lines only on the k dimension, the noise is globally canonical: the miner may noise and quantize A once and reuse it across experts. The existing NoisyQuantize, MatMul_{Device}, Extract, and finiteness checks then apply unchanged.

5. Apply the policy and target.

Apply the existing jackpot policy (11) and target difficulty checks (12).

B Zero-knowledge verifier

Together, the outer verifier and ZK verification circuit enforce the acceptance rule of Section 4, including the changes in Appendix A when $e \geq 1$. Every plaintext condition remains unchanged. This appendix specifies only witness exposure and the ZK wrapper.

Disclosure boundary

Public inputs. Let $\mathbf{p}_{\text{ZK}}$ contain all public parameters specific to the ZK proof system. The common public-input tuple is

$$\mathcal{X}_{\text{ZK}} := (\mathbf{p}_{\text{ZK}}, \hat{\sigma}, \sigma_d, \mathbf{p}_B, \mathbf{p}_A, H_B, H_A, t_B, t_A, J).$$

For $e \geq 1$, append

$$(H_R, H_O, w, O_{w-1}, O_w, O_{e-1}, I_A).$$

For $e = 0$, append nothing.

Derived values are omitted. For non-MoE proofs, derive I_A, I_B using (3). For MoE proofs, derive η, s_w, U_A, U_B, I_B using Appendix A. Both modes also derive the keys, noise seeds, noise values, and target.

Private witness. The witness contains A_{I_A}, B_{I_B} and the complete minimal openings π_A, π_B . An MoE witness also contains all of O and $R[w]$, and π_R with every leaf of $R[w]$ and its sibling hashes. The noised strips, product tile, extractor output, and jackpot-policy intermediates are also private.

Differences from plaintext verification

1. **Bind the statement.** The outer verifier rejects disallowed $\mathbf{p}_{\text{ZK}}$ values and selects the corresponding verification circuit. Define

$$H_{\text{pub}} := \text{H}^{\text{“zk-public”}}(\mathcal{X}_{\text{ZK}}).$$

The ZK Fiat–Shamir transcript absorbs H_{pub} before its first challenge.

2. **Split verification.** The outer verifier authenticates the proof-carried σ_d is in the blockchain context Σ and performs every public-input check. It performs all protocol-level Fiat–Shamir derivations, computes the noise, and applies the target check (12) before verifying the ZK proof. The verification circuit performs every check that depends on private data and constrains the public J to the jackpot hash computed from that data.
3. **Parameterize hidden MoE openings.** For MoE, O_{w-1}, O_w identify the opened interval of R^{flat} , while O_{e-1} fixes its total length. The public I_A fixes the opened rows of A , and w, t_B determine the opened rows of B . The verification circuit checks O against H_O and against the public O_{w-1}, O_w, O_{e-1} . Furthermore, it checks $R[w], \pi_R$ against H_R , and enforces consistency with the public values, including $I_A = R[w][U_A]$.

C Verifier arithmetic specifications

We provide specifications for all the relevant routines and arithmetic operations in the verifier:

1. Floating point arithmetic: arithmetic operations and casting on FP32, BF16, and FP8, by which we always mean E4M3FN (Appendix C.1), and E4M3FN matrix multiplication (Appendix C.2).
2. Quantization procedure (Appendix C.4) and the commit type (FP10, see Appendix C.6).
3. Noise-line sampler (Appendix C.3).
4. Unpredictability specification (Appendix C.5)

The matrix multiplication operation in item (1) is device-specific hence determined by the committed hardware, item (2) depends on the Quant specification, and items (3-4) are routines based on the operations (1).

C.1 Floating point arithmetic

We start with the arithmetic operations on FP32, BF16, and FP8 excluding the matrix multiplication. These are element-wise operations with no accumulation order, so they are identical on every `Device`. The verifier rejects NaN or infinite operands and rejects any operation whose result is not finite, i.e. whose rounded result is the infinity or NaN encoding of the destination type (FP32 or BF16). The test is applied after rounding, so a value just below the largest finite magnitude that rounds up to infinity is rejected. Consequently a NaN encoding is never produced.

- **Casting:** Promoting to FP32 is straightforward as BF16 and E4M3FN are exactly representable in FP32. For downcasting we use RNE_T , round-to-nearest, ties-to-even, to the target type $T \in \{\text{BF16}, \text{FP8}\}$. In case of E4M3FN, results with $|x| > 464$, which would round to magnitude 480 or more, saturate to the largest finite magnitude 448.0 (code `0x7E`). The tie $|x| = 464$ rounds to 448 under ties-to-even. In case of BF16, overflow is rejected.
- **Binary operations:** Mul and Div are computed in FP32 according to the IEEE 754-2019 standard and downcasted.

$$\begin{aligned}\text{Mul}_{\text{BF16}}(a, b) &:= \text{RNE}_{\text{BF16}}(\text{RNE}_{\text{FP32}}(ab)), \\ \text{Div}_{\text{BF16}}(a, b) &:= \text{RNE}_{\text{BF16}}(\text{RNE}_{\text{FP32}}(a/b)), \\ \text{Min}_{\text{BF16}}(a, b) &:= \begin{cases} a, & a \leq b, \\ b, & \text{otherwise,} \end{cases} \quad \text{Max}_{\text{BF16}}(a, b) := \begin{cases} a, & a \geq b, \\ b, & \text{otherwise.} \end{cases}\end{aligned}$$

- **Min and Max:** Min and Max compare exactly and return one of their operands unchanged, so they never round. On equal values, including $+0$ and -0 , they return the first operand.
- **Clamp:** for a bound $Q > 0$,

$$\text{clamp}(x, -Q, Q) := \text{Max}_{\text{BF16}}(\text{Min}_{\text{BF16}}(x, Q), -Q).$$

- **Fused multiply-add:** $\text{FMA}_{\text{BF16}}(a, b, c) := \text{RNE}_{\text{BF16}}((a \times b) + c)$, a single rounding of the exact value directly to BF16 (IEEE 754-2019 **fusedMultiplyAdd** with destination BF16). Unlike Div, it is not computed in FP32 and downcast.

C.2 E4M3FN matrix multiplication

We now describe in detail the E4M3FN matrix multiplication based on [5]. For E4M3FN operands $U \in \text{FP8}^{a \times k}$ and $V \in \text{FP8}^{b \times k}$, the opcode $\text{MatMul}_{\text{Device}}(U, V)$ accumulates the FP32 matrix UV^T into an FP32 accumulator. Recall that a nonzero finite E4M3FN code x has sign $s \in \{-1, 1\}$, exponent field $e \in \{0, \dots, 15\}$, and fraction $m \in \{0, 1/8 \dots, 7/8\}$. Its value and stored exponent $\varepsilon(x)$ are

$$x = \begin{cases} s \cdot m 2^{-6}, & e = 0, \\ s \cdot (1 + m) 2^{e-7}, & e > 0, \end{cases} \quad \varepsilon(x) = \begin{cases} -6, & e = 0, \\ e - 7, & e > 0. \end{cases}$$

For a nonzero FP32 value the stored exponent is likewise its exponent field minus the bias, $\varepsilon(c) = \lfloor \log_2 |c| \rfloor$ for normal c and -126 for subnormal c . A zero carries no exponent and is skipped in the maximum below. The matrix multiplication is then determined by how dot-products are computed

$$\sum_{u=0}^{31} U_{i,u} V_{j,u}$$

and how they are accumulated to an FP32 accumulator (in a bit-exact manner). In [5] it is established that this computation is carried out as follows:

1. Products $U_{i,u}V_{j,u}$ are computed exactly.
2. Accumulator: has internal precision (mantissa bits) that is architecture dependent (e.g. 13 for Hopper and 25 for Blackwell), and exponent in range $[-126, 127]$. Throughout, **mantissa** excludes the implicit leading 1 and **significand** includes it, so a 14-bit (Hopper) or 26-bit (Blackwell) significand corresponds to $W_{\text{Device}} = 13$ or 25 mantissa bits, and FP32 has 23.
3. Summation: performed in groups of fixed size (architecture dependent, e.g. 32 for Hopper and Blackwell) via shifting the exponent of the products and the accumulator (see below), rounding-towards-zero to the accumulator's internal precision, adding to the accumulator, and rounding-towards-zero again to the accumulator precision.
4. Exponent shift: the exponent of every product is the sum of the stored exponents of the operands. The exponent shift is the maximum stored exponent over the nonzero products in the group and the accumulator. Products are kept un-normalized with stored exponent, e.g. $1.5 \times 1.5 = 2.25$ keeps exponent $0 + 0 = 0$ rather than re-normalizing to exponent 1. Also, subnormal inputs are not renormalized (stored exponent -6).
5. The output is rounded-towards-zero to FP32 if necessary, i.e. if $W_{\text{Device}} > 23$ (Blackwell requires rounding, but Hopper does not due to its lower accumulator precision).

In pseudocode: let W_{Device} be the internal accumulator precision and let $\text{Trunc}(x, u)$ be truncation toward zero to a multiple of u . Each group of 32 products, $a, b \in \text{FP8}^{32}$, is accumulated into the running FP32 accumulator c (which is $+0$ before the first group) as:

input:	$a, b \in \text{FP8}^{32}, c \in \text{FP32}$	(accumulator so far)
products:	$p_i \leftarrow a_i b_i \text{ for } i = 0, \dots, 31$	(exact)
exponent shift:	$\eta \leftarrow \max\{\varepsilon(c), \max_i\{\varepsilon(a_i) + \varepsilon(b_i)\}\}$	
summation:	$S \leftarrow \text{Trunc}(c, 2^{\eta - W_{\text{Device}}}) + \sum_{i=0}^{31} \text{Trunc}(p_i, 2^{\eta - W_{\text{Device}}})$	(exact)
zero canonicalization:	if $S = 0$: return $+0$	(all-zero group or exact cancellation)
rounding:	$c \leftarrow \text{Trunc}(S, 2^{\lfloor \log_2 S \rfloor - W_{\text{Device}}})$	(internal precision)
output:	if $W_{\text{Device}} > 23$: $c \leftarrow \text{Trunc}(c, 2^{\lfloor \log_2 c \rfloor - 23})$	(FP32 has 23 mantissa bits)
return	c	

A zero result is always $+0$: an exactly zero sum is canonicalized, and a nonzero sum never rounds to zero as the entries are E4M3FN.

C.3 Noise-line sampler

Let r be the noise rank, $X \in \{A, B\}$ the operand, $Z \in \{E, F\}$ the factor, and ℓ the line.

$$\text{noise_key}_X := \text{Subkey}(\text{noise_seed}_X, \text{"noise-line"}).$$

The line's r bytes are drawn from the BLAKE3 extendable output²

$$(b_0, \dots, b_{r-1}) := \text{BLAKE3-XOF}((X, Z, \ell); \text{key} = \text{noise_key}_X)[0 : r].$$

Each byte $b_u \in \{0, \dots, 255\}$ is split into its top bit $s_u \in \{0, 1\}$ and its low seven bits $m_u \in \{0, \dots, 127\}$, read as an unsigned integer, so that $b_u = 128 s_u + m_u$. The line entry is

$$x_u := (-1)^{s_u} (1 + m_u),$$

²BLAKE3 [6] is an extendable-output function (XOF): besides its default 256-bit digest it can emit an output stream of arbitrary length, whose first 32 bytes coincide with the default digest.

which is uniform on $\pm\{1, \dots, 128\}$.

$$q := \text{isqrt} \left(32^2 \sum_{u=0}^{r-1} x_u^2 \right), \quad \gamma := \text{Div}_{\text{BF16}}(8192, \text{RNE}_{\text{BF16}}(q)),$$

where isqrt is the integer floor of the square root. Since $q \approx 32\sqrt{\sum_u x_u^2}$, multiplying by $\gamma = 8192/q$ rescales the line to Euclidean norm approximately $8192/32 = 256$ before the final rounding to FP8. We write $c := 256$ for this target norm.

$$\text{SampleLine}(\text{noise_seed}_X, X, Z, \ell)_u := \text{RNE}_{\text{FP8}}(\text{Mul}_{\text{BF16}}(x_u, \gamma)),$$

where the integer x_u is exactly representable in BF16.

C.4 Noisy quantization

Let $X \in \text{BF16}^{s \times k}$ be an operand strip, decoded from the committed FP10 matrix as specified in Appendix C.6, with noise factors $E \in \text{FP8}^{s \times r}$ and $F \in \text{FP8}^{k \times r}$. This appendix defines

$$\text{NoisyQuantize}_p(X, E, F) = \tilde{X} \in \text{FP8}^{s \times k}.$$

It also fixes the per-row scale α_t and floored norm $\widehat{\ell}_2(X_t)$ that Section 5.1 uses, via $\overline{X}_t = \alpha_t X_t$ and $\sigma_t^X = \delta \alpha_t \widehat{\ell}_2(X_t)$. The scales α_t, β_t are also what a miner needs to peel the noise and recover the useful product, but that recovery is not part of the protocol.

Overview. The quantization is row-wise. Each row t of X and its noise row $N_t := (EF^\top)_t$ are scaled and added,

$$\tilde{X}_t \approx \alpha_t X_t + \beta_t N_t,$$

and the result is rounded to FP8. The scales $\alpha_t, \beta_t > 0$ are chosen so that

1. every entry of $\alpha_t X_t + \beta_t N_t$ lies within the FP8 range $[-Q, Q]$, and
2. the noise row has Euclidean norm δ times that of the scaled row, $\|\beta_t N_t\|_2 = \delta \|\alpha_t X_t\|_2$.

The noise lines have Euclidean norm c (Appendix C.3), so $|N_{tu}| \leq c^2$ and, since the lines are random, N_t has root-mean-square entry about $c^2/\sqrt{r}$. Writing $\ell_2(X_t)$ for the root-mean-square entry of X_t and $\ell_\infty(X_t)$ for its largest magnitude, requirement (2) gives

$$\beta_t = \alpha_t \ell_2(X_t) \delta \sqrt{r}/c^2,$$

and requirement (1) then becomes

$$\alpha_t (\ell_\infty(X_t) + \delta \sqrt{r} \ell_2(X_t)) \leq Q,$$

which fixes α_t by taking equality. Two caveats:

- The norm of N_t is never measured, so (2) holds only approximately and with high probability.
- Requirement (1) holds in exact arithmetic, but rounding can push an entry slightly past Q , which the clamp in the definition below handles.

Definition. Let t be a row index. The norms $\ell_2(X_t), \ell_\infty(X_t) \in \text{BF16}$ are computed from the committed representation of X rather than from its BF16 decoding, as specified in Appendix C.6. Note that $\ell_2(X_t)$ is the root-mean-square entry of the row, so $\|X_t\|_2 = \sqrt{k} \ell_2(X_t)$. It is not the Euclidean norm, which is the quantity used for the noise lines in Appendix C.3. Floor the norms at 2^{-32} :

$$\widehat{\ell}_2(X_t) := \max\{\ell_2(X_t), 2^{-32}\}, \quad \widehat{\ell}_\infty(X_t) := \max\{\ell_\infty(X_t), 2^{-32}\}.$$

Fix $Q := 448$ and $\delta := 1/2$, and let $c = 256$ be the line norm of Appendix C.3. In exact arithmetic the scales and the output are

$$\alpha_t = \frac{Q}{\widehat{\ell}_\infty(X_t) + \delta\sqrt{r}\widehat{\ell}_2(X_t)}, \quad \beta_t = \alpha_t \widehat{\ell}_2(X_t) \frac{\delta\sqrt{r}}{c^2}, \quad \tilde{X}_t = \text{RNE}_{\text{FP8}}(\alpha_t X_t + \beta_t N_t),$$

where $N := EF^\top$.

Bit-exact computation. Let

$$d_r := \text{RNE}_{\text{BF16}}(\delta \cdot \sqrt{r}), \quad e_r := \text{RNE}_{\text{BF16}}\left(\frac{\delta \cdot \sqrt{r}}{c^2}\right),$$

rounded from the exact real values.³ Every operation below is the BF16 operation of Appendix C.1, so each $+$, $\cdot$, and $/$ rounds its result to BF16. Parentheses fix the order of evaluation, and an unparenthesized $a \cdot b + c$ is the fused multiply-add $\text{FMA}_{\text{BF16}}(a, b, c)$ with a single rounding.

$$\begin{aligned} z_t &:= d_r \cdot \widehat{\ell}_2(X_t) + \widehat{\ell}_\infty(X_t), \\ \alpha_t &:= Q/z_t, \\ \beta_t &:= (\alpha_t \cdot \widehat{\ell}_2(X_t)) \cdot e_r, \\ N &:= \text{RNE}_{\text{BF16}}(\text{MatMul}_{\text{Device}}(E, F)), \\ y_{tu} &:= \alpha_t \cdot X_{tu} + (\beta_t \cdot N_{tu}), \\ \tilde{X}_{tu} &:= \text{RNE}_{\text{FP8}}(\text{clamp}(y_{tu}, -Q, Q)). \end{aligned}$$

Here $Q = 448$ is the largest finite E4M3FN magnitude, so the clamp keeps every quantized value representable.

C.5 Unpredictability specification

We recall the quantities of Section 5.1. M_{ij} is the replay magnitude of cell (i, j) , the largest magnitude appearing in its accumulation. At magnitude x , the accumulation grid of Appendix C.2 has spacing

$$\text{ulp}_{\text{Device}}(x) := 2^{\lfloor \log_2 |x| \rfloor - W_{\text{Device}}}, \quad \text{ulp}_{\text{Device}}(0) := 0.$$

Summand (i, j, u) has unpredictability (13)

$$v_{iju} := \frac{(\overline{A}_{iu}^2 + (\sigma_i^A)^2)(\overline{B}_{ju}^2 + (\sigma_j^B)^2)}{2^{21}}.$$

Finally, the unpredictable-summands check counts the set

$$\mathcal{S} := \{(i, j, u) : v_{iju} < \text{ulp}_{\text{Device}}(M_{ij})^2\}.$$

³The implementation evaluates them in double precision and rounds through FP32. For every rank $r \leq 4096$ this agrees with a single rounding of the exact value.

For efficiency, the verifier decides this inequality in the $\log_2$ domain with fixed-point integers. It computes, for each of the two factors of v_{iju} ,

$$\Lambda(x, \sigma) \approx \log_2(x^2 + \sigma^2)$$

that is a lower bound accurate to within 0.016. Then

$$\Lambda_{iju} := \Lambda(\bar{A}_{iu}, \sigma_i^A) + \Lambda(\bar{B}_{ju}, \sigma_j^B) - 21$$

satisfies $\Lambda_{iju} \leq \log_2 v_{iju} < \Lambda_{iju} + 0.032$, the -21 being $\log_2 2^{21}$, and the check counts

$$\hat{\mathcal{S}} := \{(i, j, u) : \Lambda_{iju} < \log_2(\text{ulp}_{\text{Device}}(M_{ij})^2)\}$$

in place of $\mathcal{S}$. Since Λ_{iju} is a lower bound, $\hat{\mathcal{S}} \supseteq \mathcal{S}$, and a summand lies in $\hat{\mathcal{S}} \setminus \mathcal{S}$ only when

$$\text{ulp}_{\text{Device}}(M_{ij})^2 \leq v_{iju} < 2^{0.032} \cdot \text{ulp}_{\text{Device}}(M_{ij})^2.$$

The estimate $\Lambda(x, \sigma)$. Let x be a scaled entry ($\bar{A}_{iu}$ or $\bar{B}_{ju}$) and $\sigma > 0$ its row's noise scale.⁴ First take out the larger exponent,

$$e := \max(\lfloor \log_2 |x| \rfloor, \lfloor \log_2 \sigma \rfloor), \quad x = 2^e m_x, \quad \sigma = 2^e m_\sigma,$$

so that the larger of $|m_x|, m_\sigma$ lies in $[1, 2)$ and the other in $[0, 2)$, and

$$x^2 + \sigma^2 = 2^{2e} (m_x^2 + m_\sigma^2), \quad m_x^2 + m_\sigma^2 \in [1, 8).$$

Next compute the sum of squared significands as an integer with 13 fractional bits,

$$\kappa := \left\lfloor ([2^{30} m_x^2] + [2^{30} m_\sigma^2]) / 2^{17} \right\rfloor \in [2^{13}, 2^{16}),$$

so that κ is $m_x^2 + m_\sigma^2$ rounded down onto the grid of spacing 2^{-13} , up to a negligible error:

$$2^{13}(m_x^2 + m_\sigma^2) - (1 + 2^{-16}) < \kappa \leq 2^{13}(m_x^2 + m_\sigma^2).$$

Finally, take the logarithm with 6 fractional bits:

$$\Lambda(x, \sigma) := 2e - 13 + \lfloor 64 \log_2 \kappa \rfloor / 64.$$

Accuracy. Since $\kappa / 2^{13} \leq m_x^2 + m_\sigma^2$,

$$\Lambda(x, \sigma) \leq 2e + \log_2(m_x^2 + m_\sigma^2) = \log_2(x^2 + \sigma^2).$$

In the other direction, the truncation of κ loses less than $\log_2(1 + 2^{-13}(1 + 2^{-16})) < 2^{-12}$ and the final floor loses less than 2^{-6} , for a total below $2^{-6} + 2^{-12} < 0.016$. Hence

$$\log_2(x^2 + \sigma^2) - 0.016 < \Lambda(x, \sigma) \leq \log_2(x^2 + \sigma^2).$$

Adding the two estimates and subtracting 21 gives the bound on Λ_{iju} stated above.

⁴The norm floor of Appendix C.4 gives $\widehat{\ell}_2 \geq 2^{-32}$ and hence $\sigma > 0$, so $x = \sigma = 0$ cannot occur and the estimate is always well defined.

C.6 Commit type (FP10)

An operand with n rows is committed as two matrices, with k a multiple of 8:

$$V \in \{-128, \dots, 127\}^{n \times k} \quad (\text{integer values}), \quad \Sigma \in \text{BF16}^{n \times k/8} \quad (\text{block scales}).$$

Each block of 8 consecutive entries of a row shares one scale, and entry (t, u) represents the real number

$$V_{tu} \cdot \Sigma_{t, \lfloor u/8 \rfloor}.$$

The format spends 8 bits per entry plus 16 bits per block of 8, i.e. 10 bits per entry, hence the name.

This appendix defines the decoding

$$X := \text{Open}(V, \Sigma) \in \text{BF16}^{n \times k}$$

whose opened rows Appendix C.4 takes as input, and the row norms $\ell_2(X_t)$ and $\ell_\infty(X_t)$ it uses.

Commitment. Each matrix is committed separately as a Merkle tree over its rows, under the key and hash identifier of Section 4. A row of V is its k bytes, each value in two's complement. A row of Σ is its $2k/8$ bytes, each scale as a little-endian BF16 code. Scales that are NaN or infinite are rejected. The aggregate root H_X is a hash of the two roots R_V, R_Σ together with the shapes of their matrices.

Decoding.

$$X_{tu} := \text{RNE}_{\text{BF16}}(V_{tu} \cdot \Sigma_{t, \lfloor u/8 \rfloor}).$$

The product has at most 15 significant bits, 7 from the integer and 8 from the BF16 significand, so it is exact in FP32 and X_{tu} carries a single rounding.

Row norms. The norms are computed from V and Σ directly rather than from X , so that the per-block reductions below are exact. All arithmetic in this paragraph is FP32, each operation rounding its result to FP32, and parentheses fix the order of evaluation. For row t and block $b \in \{0, \dots, k/8 - 1\}$ let

$$S_{tb} := \sum_{u=8b}^{8b+7} V_{tu}^2, \quad A_{tb} := \max_{8b \leq u \leq 8b+7} |V_{tu}|,$$

both integers with $S_{tb} \leq 8 \cdot 127^2 < 2^{17}$, so both are exact in FP32. The row's sum of squares is accumulated in block order,

$$s_{t,0} := 0, \quad s_{t,b+1} := s_{t,b} + (S_{tb} \cdot \Sigma_{tb}^2),$$

where Σ_{tb}^2 has 16 significant bits and is exact. Let $s_t := s_{t,k/8}$ be the resulting sum of squares, computed in FP32 arithmetic. A non-finite s_t is rejected. Then

$$\ell_2(X_t) := \text{Grid}\left(\text{RNE}_{\text{BF16}}\left(\sqrt{s_t/k}\right)\right), \quad \ell_\infty(X_t) := \text{RNE}_{\text{BF16}}\left(\max_b A_{tb} \cdot \Sigma_{tb}\right).$$

Here Grid rounds away the two lowest mantissa bits of a non-negative BF16, to nearest with ties going up, a carry propagating into the exponent.