MATRIX MULTIPLICATION VIA FREIVALDS' ALGORITHM

1. INTRODUCTION

Freivalds' algorithm [1] provides a one-sided Monte Carlo verification (see [6, Section 1.2]) of the equation $AB = C$: the check always passes when the equality holds, and fails with high probability when it does not. In contrast to the deterministic approach described in [2], which recomputes the matrix product in full, Freivalds' method drastically reduces the number of multiplication and addition gates required in the circuit. This makes it especially attractive for large matrices or constrained proving environments.

The same idea applies to our quantized matrix multiplication circuit [3], and we will explain how to adapt Freivalds' algorithm to the quantized setting in a forthcoming document.

Freivalds' check never rejects a correct product $C = AB$, and it rejects an incorrect product $C \neq AB$ with high probability. Precisely, if $C \neq AB$, the probability that the check still passes is at most $1/p$, where p is the field characteristic. Note that this is *not* the same as saying that if the check passes, then $C = AB$ with high probability. This distinction reflects a general principle in the theory of soundness: the verifier is guaranteed to reject false claims with high probability, but acceptance does not imply that the claim is true with high probability. Soundness is a property of the protocol's behavior on incorrect inputs, not a measure of the likelihood that accepted inputs are correct.

2. MATRIX MULTIPLICATION: THE PROCESS

Freivalds' algorithm verifies the matrix identity $AB = C$ by sampling a random vector X and checking whether $A(BX) = CX$. If $AB = C$, the check always passes; if $AB \neq C$, the check fails with high probability. This reduces verification to three matrix–vector products—BX, CX, and $A(BX)$—each requiring significantly fewer multiplications and additions than computing the full matrix product AB.

In the version implemented here, the entries of X are sampled independently and uniformly at random from the field $\mathbb{Z}/p\mathbb{Z}$. In other settings—particularly when working over general rings rather than fields—it is common to sample from $\{0,1\}$ and repeat the check multiple times to amplify soundness. The circuit we describe below implements the field-based version.

2.1. Steps in the process. Each step in the process has a corresponding explanatory note in Subsection 2.2 that provides additional context and details.

(1) The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a prime.

(2) Let $A = [a_{ik}], B = [b_{kj}], C = [c_{ij}]$ be integer matrices of dimensions $\ell \times m, m \times n, \ell \times n$, respectively.

(2a) Assume there is an integer h such that for all i, k, j,

$$h - p \leqslant \sum_{k=1}^{m} a_{ik} b_{kj}, c_{ij} < h. \tag{2.1}$$

If the assumption that $\sum_{k=1}^{m} a_{ik} b_{kj} \in [h-p, h)$ cannot be justified, a range check can be performed, as illustrated below for the cases $h = p$ and $h = (p+1)/2$.

(2b) If $h = p$, assume that for all i, k, j,

$$0 \leqslant a_{ik}, b_{kj}, c_{ij} < p, \tag{2.2}$$

and perform a range check to ensure that

$$a_{ik}, b_{kj} \in [0, T), \tag{2.3}$$

for some positive integer T satisfying

$$mT^2 \leqslant p. \tag{2.4}$$

This ensures that (2.1) holds (with $h = p$) for all i, k, j.

(2c) If $h = (p+1)/2$, so that (2.1) is equivalent to

$$-\frac{p-1}{2} \leqslant \sum_{k=1}^{m} a_{ik} b_{kj},\, c_{ij} \leqslant \frac{p-1}{2}, \tag{2.5}$$

assume that for all i, k, j,

$$-\frac{p-1}{2} \leqslant a_{ik}, b_{kj}, c_{ij} \leqslant \frac{p-1}{2}, \tag{2.6}$$

and perform a range check to ensure that

$$a_{ik}, b_{kj} \in [-T, T), \tag{2.7}$$

for some positive integer T satisfying

$$mT^2 \leqslant \frac{p-1}{2}. \tag{2.8}$$

This ensures that (2.5) holds (i.e. (2.1) holds with $h = (p+1)/2$) for all i, k, j.

(3) The goal of the circuit is to verify that $AB = C$ using a probabilistic check. If

$$\sum_{k=1}^{m} a_{ik} b_{kj} = c_{ij} \tag{2.9}$$

holds for all i, j, then the circuit accepts (`True`). If not, then the circuit rejects (`False`) *with high probability*. In other words, the circuit behaves correctly on valid input and is sound with high probability on invalid input.

(4) Most frameworks work exclusively with least residue representations. Accordingly, we use $\bar{z}$ to denote the least residue of z mod p:

$$\bar{a}_{ik} \equiv a_{ik} \bmod p, \quad \bar{b}_{kj} \equiv b_{kj} \bmod p, \quad \bar{c}_{ij} \equiv c_{ij} \bmod p, \quad 0 \leqslant \bar{a}_{ik}, \bar{b}_{kj}, \bar{c}_{ij} < p. \tag{2.10}$$

Under the assumption that (2.3) holds for all i, k, j, goal (2.9) holds for all i, j if and only if

$$\sum_{k=1}^{m} \bar{a}_{ik} \bar{b}_{kj} \equiv \bar{c}_{ij} \bmod p \tag{2.11}$$

for all i, j [Proposition 4.2, Remark 4.3]. Let $\bar{A} = [\bar{a}_{ik}]$, $\bar{B} = [\bar{b}_{kj}]$, and $\bar{C} = [\bar{c}_{ij}]$.

(5) Let $X \in (\mathbb{Z}/p\mathbb{Z})^n$ be a pseudorandom vector derived deterministically (e.g., via a hash function) from public inputs [Listing 1, Step 1]. Let $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$, where each $\bar{x}_j \in \{0, 1, \ldots, p-1\}$ is the least residue representative of the j-th coordinate of X.

(6) The circuit computes the product $\bar{B}\bar{X}$ and reduces all entries modulo p to obtain $\bar{U} = (\bar{u}_1, \ldots, \bar{u}_m)$ [Listing 1, Step 2]. That is,

$$\bar{u}_k \equiv \bar{b}_{k1}\bar{x}_1 + \cdots + \bar{b}_{kn}\bar{x}_n \bmod p, \quad 0 \leqslant \bar{u}_k < p \quad (1 \leqslant k \leqslant m). \tag{2.12}$$

(7) The circuit computes the product $\bar{C}\bar{X}$ and reduces all entries modulo p to obtain $\bar{V} = (\bar{v}_1, \ldots, \bar{v}_\ell)$ [Listing 1, Step 4]. That is,

$$\bar{v}_i \equiv \bar{c}_{i1}\bar{x}_1 + \cdots + \bar{c}_{in}\bar{x}_n \bmod p, \quad 0 \leqslant \bar{v}_i < p \quad (1 \leqslant i \leqslant \ell). \tag{2.13}$$

(8) For $1 \leqslant i \leqslant \ell$, impose the constraint [Listing 1, Steps 3, 5]

$$(\bar{a}_{i1}\bar{u}_1 + \cdots + \bar{a}_{im}\bar{u}_m) - \bar{v}_i \equiv 0 \bmod p. \tag{2.14}$$

(9) If $AB = C$, then the constraints (2.14) are satisfied for all i. If $AB \neq C$, and assuming the bounds in (2.1) hold, the probability that the constraints (2.14) are satisfied for all i is at most $1/p$ [Propositions 4.2, 4.4]:

$$\mathbb{P}((2.14) \text{ holds for all } i \mid AB \neq C) \leqslant \frac{1}{p}. \tag{2.15}$$

(10) To reduce the soundness error, repeat Steps (5) through (9) independently s times using fresh random vectors $\bar{X}_1, \ldots, \bar{X}_s$. In this case, the bound in (2.15) may be replaced by $1/p^s$.

2.2. Commentary on each step.

(1) Typically, p is an n-bit prime satisfying

$$2^{n-1} \leqslant p < 2^n,$$

with $n \approx 256$. In practice, we use the prime field associated with the scalar field of the BN254 elliptic curve, a 254-bit prime that offers a good balance between security and efficiency. This field is widely supported in cryptographic applications and zk-SNARK frameworks due to the curve's pairing-friendly properties and efficient arithmetic over $\mathbb{Z}/p\mathbb{Z}$.

(2) The offset parameter h provides flexibility in defining an integer interval of length p to which all values are assumed to belong. Prior to entering the circuit, we typically assume that each term $\sum_{k=1}^{m} a_{ik}b_{kj}$ and each value c_{ij} lies within such a range. Two common conventions are:

- the *least residue range* $[0, p)$, corresponding to $h = p$;
- the *balanced residue range* $(-p/2, p/2]$, corresponding to $h = (p+1)/2$.

Because arithmetic circuits over $\mathbb{Z}/p\mathbb{Z}$ cannot distinguish between integers that differ by a multiple of p, the condition

$$h - p \leqslant c_{ij} < h$$

must be justified off-circuit.

The same applies to the term $\sum_{k=1}^{m} a_{ik}b_{kj}$. In many scenarios, such bounds may be justified off-circuit, particularly when m is small and the matrix entries are known to be bounded. However, when such justification is unavailable, we must instead enforce the bound in-circuit by verifying, for instance, that

$$a_{ik}, b_{kj} \in [-T, T)$$

for some positive integer T satisfying

$$mT^2 \leqslant \frac{p-1}{2}.$$

This guarantees that

$$\sum_{k=1}^{m} a_{ik}b_{kj} \in \left[-mT^2, mT^2\right] \subseteq \left[-(p-1)/2, (p-1)/2\right]. \tag{2.16}$$

As always, the validity of the range check depends on the off-circuit assumption that a_{ik}, b_{kj} are encoded using a known, consistent set of representatives modulo p (e.g., the balanced residue range). Without such an assumption, even a successful range check does not determine the underlying integer values uniquely. For further discussion of this subtlety and the implementation of range checks, see our companion documents [4, 5].

(3) In the direct approach of [2], the circuit verifies ℓn constraints—one for each entry of the output matrix C. This requires evaluating a full matrix product, which involves ℓmn multiplication gates (one for each term in the m-term dot product defining each of the ℓn outputs) and $\ell(m-1)n$ addition gates (to sum each dot product).

In contrast, the approach taken here follows Freivalds' algorithm [1], which rejects incorrect claims with high probability by compressing the verification task from a full matrix-matrix equality into three matrix-vector products. As we will see in Comments (6), (7), (8), this reduces the number of constraints from ℓn to just ℓ, and drastically lowers the arithmetic cost: the total number of multiplication gates becomes $\ell m + \ell n + mn$, and the number of addition gates becomes $\ell(m-1) + \ell(n-1) + m(n-1)$. When $\ell = m = n$, this translates to $O(n^2)$ multiplications and additions, in contrast to the $O(n^3)$ cost of full matrix multiplication. These counts exclude the cost of sampling and introducing the random vector.

(4) In typical usage, the prover supplies the least residue matrices $\bar{A} = [\bar{a}_{ik}]$, $\bar{B} = [\bar{b}_{kj}]$, and $\bar{C} = [\bar{c}_{ij}]$ as part of the witness, where each entry is the canonical representative of its corresponding integer modulo p.

If B represents a fixed matrix—such as a set of learned model weights—it may be hardcoded into the circuit. This eliminates the need for the prover to supply $\bar{B}$ as part of the witness and can significantly reduce prover-side workload.

As noted, the circuit's goal—stated in Step (3)—admits an equivalent formulation in terms of congruences modulo p. This equivalence follows from Proposition 4.2, but it is essential that the assumption in (2.1) holds for all i, k, j. Extending the notion of congruence to matrices entrywise, as in Definition 4.1, we may therefore write:

$$AB = C \quad \Longleftrightarrow \quad \bar{A}\bar{B} \equiv \bar{C} \bmod p, \tag{2.17}$$

provided all entries of AB and C lie in the same interval of length p.

Thus, the goal of the circuit is to verify that $\bar{A}\bar{B} \equiv \bar{C} \bmod p$ using a probabilistic check. If the congruence holds, the circuit accepts. If it does not, the circuit rejects with high probability.

(5) The vector X must be chosen uniformly at random to ensure the soundness of the protocol. In an interactive setting, this randomness would be supplied by the verifier after the prover has committed to the matrices A, B, and C. In a non-interactive proof system (e.g., a zkSNARK), X is typically derived from a cryptographic hash of the circuit inputs and outputs using the Fiat–Shamir heuristic. Accordingly, X must be deterministically derived from public data, but remain unpredictable to the prover until after their commitment. In practice, X is often treated as a *public input* to the circuit or computed within the circuit itself via a cryptographic hash function.

(6) In a non-interactive setting, the circuit is responsible for computing the product $\bar{B}\bar{X}$. This ensures soundness: if the prover were allowed to supply the result, they could choose $\bar{B}$ and $\bar{U}$ inconsistently to pass the final check without satisfying the intended matrix product relation. The verifier cannot compute $\bar{B}\bar{X}$ either, since $\bar{B}$ is not assumed to be public.

Although we phrase this step in terms of integer arithmetic—computing $\bar{B}\bar{X}$ and reducing modulo p—this is done purely for notational simplicity. In reality, the circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, and all values should be understood as elements of that field. Throughout this section, we adopt the convention of using least residue representatives from $\{0,1,\ldots,p-1\}$ to make the exposition more accessible, while recognizing that the underlying computations are carried out over the field.

To compute $\bar{B}\bar{X}$, the circuit performs m inner products of length n, requiring mn multiplication gates and $m(n-1)$ addition gates.

(7) In a non-interactive setting, the circuit—not the prover or verifier—must compute the product $\bar{C}\bar{X}$. Allowing the prover to supply the result would compromise soundness, as the prover could fabricate values of $\bar{C}$ that satisfy the check without correctly encoding a matrix product. The verifier cannot compute it either, as $\bar{C}$ is not assumed to be public.

As before, we describe the computation in terms of integer arithmetic—dot products followed by reduction modulo p—to keep the exposition elementary. However, the actual circuit performs these operations over the field $\mathbb{Z}/p\mathbb{Z}$, and values should be interpreted as field elements. We continue to use least residue representatives for clarity, but all arithmetic takes place in the field.

To compute $\bar{C}\bar{X}$, the circuit must evaluate ℓ inner products of length n, requiring ℓn multiplication gates and $\ell(n-1)$ addition gates.

(8) Each of the ℓ constraints in (2.14) involves m multiplications and $m-1$ additions (not counting the subtraction of $\bar{v}_i$).

In what follows, instead of writing $M_1 \equiv M_2 \bmod p$, we will simply write $M_1 = M_2$, with the understanding that equality holds over the field $\mathbb{Z}/p\mathbb{Z}$. Recall from Step (4) and Comment (4) that the goal of the circuit is to verify that $\bar{A}\bar{B} = \bar{C}$ using a probabilistic check: the circuit accepts if the equality holds, and rejects with high probability—that is, accepts with low probability—if it does not.

The constraints of Step (8) collectively assert that $\bar{A}\bar{U} = \bar{V}$. Substituting from (2.12) and (2.13), we find that $\bar{U} = \bar{B}\bar{X}$ and $\bar{V} = \bar{C}\bar{X}$, so the constraint becomes

$$\bar{A}\bar{B}\bar{X} = \bar{C}\bar{X}, \tag{2.18}$$

or equivalently,

$$(\bar{A}\bar{B} - \bar{C})\bar{X} = \mathbf{0}, \tag{2.19}$$

meaning that $\bar{X}$ lies in the null space of the matrix $\bar{A}\bar{B} - \bar{C}$.

Thus, the constraints (2.14) are equivalent to the constraint (2.19), and we'll refer to the latter formulation below.

Let $\bar{O}_{\ell\times n}$ denote the zero matrix in $(\mathbb{Z}/p\mathbb{Z})^{\ell\times n}$. There are two cases. If the prover is honest and provides the correct $\bar{A},\bar{B},\bar{C}$ to the circuit, then $\bar{A}\bar{B} - \bar{C} = \bar{O}_{\ell\times n}$ and (2.19) holds trivially.

(9) Continuing from Comment (8), if the prover is dishonest, so that $\bar{A}\bar{B} - \bar{C} \neq \bar{O}_{\ell\times n}$, it is still possible that the constraint (2.19) holds, resulting in a false positive verification. However, this occurs with probability at most $1/p$, as we now explain.

Given that $\bar{A}\bar{B} - \bar{C}$ is nonzero, its null space is a proper subspace of $(\mathbb{Z}/p\mathbb{Z})^n$ with dimension at most $n-1$. Therefore, the probability that a uniformly random vector $\bar{X} \in (\mathbb{Z}/p\mathbb{Z})^n$ lies in this null space is at most

$$\frac{\#N(\bar{A}\bar{B} - \bar{C})}{\#(\mathbb{Z}/p\mathbb{Z})^n} \leqslant \frac{p^{n-1}}{p^n} = \frac{1}{p}. \tag{2.20}$$

(10) If the prover is dishonest, so that $\bar{A}\bar{B} - \bar{C} \neq \bar{O}_{\ell\times n}$, then each random vector $\bar{X}_r$ lies in the null space of $\bar{A}\bar{B} - \bar{C}$ with probability at most $1/p$, independently. Therefore, the probability that all s independent tests pass is at most $(1/p)^s$.

3. Matrix Multiplication with Bias: The Process

In this section, we extend the probabilistic verification method for matrix multiplication $AB = C$ to handle affine combinations of the form $AB + C = D$. This operation arises frequently in neural networks, where A typically represents learned weights, B an input vector or matrix, and C a bias term added elementwise after the matrix multiplication. The extension is straightforward: it requires only one additional matrix-vector product to account for the bias term, and the rest of the procedure—including the use of random vector compression and Freivalds' method—remains unchanged in structure.

3.1. Steps in the process. Each step in the process has a corresponding explanatory note in Subsection 3.2 that provides additional context and details.

(1) The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a prime.

(2) Let $A = [a_{ik}], B = [b_{kj}], C = [c_{ij}], D = [d_{ij}]$ be integer matrices of dimensions $\ell \times m, m \times n, \ell \times n, \ell \times n$, respectively.

(2a) Assume there is an integer h such that for all i, k, j,

$$h - p \leqslant \left(\sum_{k=1}^{m} a_{ik} b_{kj}\right) + c_{ij}, d_{ij} < h. \tag{3.1}$$

If the assumption that $\left(\sum_{k=1}^{m} a_{ik} b_{kj}\right) + c_{ij} \in [h-p, h)$ cannot be justified, a range check can be performed, as illustrated below for the cases $h = p$ and $h = (p+1)/2$.

(2b) If $h = p$, assume that for all i, k, j,

$$0 \leqslant a_{ik}, b_{kj}, c_{ij}, d_{ij} < p, \tag{3.2}$$

and perform a range check to ensure that

$$a_{ik}, b_{kj}, c_{ij} \in [0, T), \tag{3.3}$$

for some positive integer T satisfying

$$mT^2 + T \leqslant p. \tag{3.4}$$

This ensures that (3.1) holds (with $h = p$) for all i, k, j.

(2c) If $h = (p+1)/2$, so that (3.1) is equivalent to

$$-\frac{p-1}{2} \leqslant \left(\sum_{k=1}^{m} a_{ik} b_{kj}\right) + c_{ij}, d_{ij} \leqslant \frac{p-1}{2}, \tag{3.5}$$

assume that for all i, k, j,

$$-\frac{p-1}{2} \leqslant a_{ik}, b_{kj}, c_{ij}, d_{ij} \leqslant \frac{p-1}{2}, \tag{3.6}$$

and perform a range check to ensure that

$$a_{ik}, b_{kj}, c_{ij} \in [-T, T), \tag{3.7}$$

for some positive integer T satisfying

$$mT^2 + T \leqslant \frac{p-1}{2}. \tag{3.8}$$

This ensures that (3.5) holds (i.e. (3.1) holds with $h = (p+1)/2$) for all i, k, j.

(3) The goal of the circuit is to verify that $AB + C = D$ using a probabilistic check. If

$$\left(\sum_{k=1}^{m} a_{ik} b_{kj}\right) + c_{ij} = d_{ij} \tag{3.9}$$

holds for all i, j, then the circuit accepts (`True`). If not, then the circuit rejects (`False`) *with high probability*. In other words, the circuit behaves correctly on valid input and is sound with high probability on invalid input.

(4) Most frameworks work exclusively with least residue representations. Accordingly, we use $\bar{z}$ to denote the least residue of $z \bmod p$:

$$\bar{a}_{ik} \equiv a_{ik} \bmod p, \quad \bar{b}_{kj} \equiv b_{kj} \bmod p, \quad \bar{c}_{ij} \equiv c_{ij} \bmod p, \quad \bar{d}_{ij} \equiv d_{ij} \bmod p, \quad 0 \leqslant \bar{a}_{ik}, \bar{b}_{kj}, \bar{c}_{ij}, \bar{d}_{ij} < p. \tag{3.10}$$

Under the assumption that (3.3) holds for all i, k, j, goal (3.9) holds for all i, j if and only if

$$\left(\sum_{k=1}^{m} \bar{a}_{ik} \bar{b}_{kj}\right) + c_{ij} \equiv \bar{d}_{ij} \bmod p \tag{3.11}$$

for all i, j [Proposition 4.2, Remark 4.3]. Let $\bar{A} = [\bar{a}_{ik}]$, $\bar{B} = [\bar{b}_{kj}]$, $\bar{C} = [\bar{c}_{ij}]$, and $\bar{D} = [\bar{d}_{ij}]$.

(5) Let $X \in (\mathbb{Z}/p\mathbb{Z})^n$ be a pseudorandom vector derived deterministically (e.g., via a hash function) from public inputs [Listing 2, Step 1]. Let $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$, where each $\bar{x}_j \in \{0, 1, \ldots, p-1\}$ is the least residue representative of the j-th coordinate of X.

(6) The circuit computes the product $\bar{B}\bar{X}$ and reduces all entries modulo p to obtain $\bar{U} = (\bar{u}_1, \ldots, \bar{u}_m)$ [Listing 2, Step 2]. That is,

$$\bar{u}_k \equiv \bar{b}_{k1}\bar{x}_1 + \cdots + \bar{b}_{kn}\bar{x}_n \bmod p, \quad 0 \leqslant \bar{u}_k < p \quad (1 \leqslant k \leqslant m). \tag{3.12}$$

(7) The circuit computes the product $\bar{C}\bar{X}$ and reduces all entries modulo p to obtain $\bar{V} = (\bar{v}_1, \ldots, \bar{v}_\ell)$ [Listing 2, Step 4]. That is,

$$\bar{v}_i \equiv \bar{c}_{i1}\bar{x}_1 + \cdots + \bar{c}_{in}\bar{x}_n \bmod p, \quad 0 \leqslant \bar{v}_i < p \quad (1 \leqslant i \leqslant \ell). \tag{3.13}$$

(8) The circuit computes the product $\bar{D}\bar{X}$ and reduces all entries modulo p to obtain $\bar{W} = (\bar{w}_1, \ldots, \bar{w}_\ell)$ [Listing 2, Step 5]. That is,

$$\bar{w}_i \equiv \bar{d}_{i1}\bar{x}_1 + \cdots + \bar{d}_{in}\bar{x}_n \bmod p, \quad 0 \leqslant \bar{w}_i < p \quad (1 \leqslant i \leqslant \ell). \tag{3.14}$$

(9) For $1 \leqslant i \leqslant \ell$, impose the constraint [Listing 2, Step 6]

$$(\bar{a}_{i1}\bar{u}_1 + \cdots + \bar{a}_{im}\bar{u}_m) + \bar{v}_i - w_i \equiv 0 \bmod p. \tag{3.15}$$

(10) If $AB + C = D$, then the constraints (3.15) are satisfied for all i. If $AB + C \neq D$, and assuming the bounds in (3.1) hold, the probability that the constraints (3.15) are satisfied for all i is at most $1/p$ [Propositions 4.2, 4.4]:

$$\mathbb{P}((3.15) \text{ holds for all } i \mid AB + C \neq D) \leqslant \frac{1}{p}. \tag{3.16}$$

(11) To reduce the soundness error, repeat Steps (5) through (10) independently s times using fresh random vectors $\bar{X}_1, \ldots, \bar{X}_s$. In this case, the bound in (3.16) may be replaced by $1/p^s$.

3.2. Commentary on each step.

(1) See Comment (1) of Subsection 2.2.

(2) See Comment (2) of Subsection 2.2: a similar comment applies here.

(3) In the direct approach of [2], the circuit verifies ℓn constraints—one for each entry of the output matrix D. Computing $AB + C$ requires evaluating a full matrix product and adding a bias term. This involves ℓmn multiplication gates (one for each product in the dot products), $\ell(m-1)n$ addition gates to sum the dot products, and an additional ℓn additions to incorporate the bias matrix, for a total of ℓmn additions.

In contrast, the approach taken here follows Freivalds' algorithm [1], which rejects incorrect claims with high probability by compressing the verification task from a full matrix-matrix equality into four matrix-vector products. As we will see in Comments (6), (7), (8), (9), this reduces the number of constraints from ℓn to just ℓ, and drastically lowers the arithmetic cost: the total number of multiplication gates becomes $2\ell m + \ell n + mn$, and the number of addition gates becomes $\ell(m-1) + 2\ell(n-1) + m(n-1)$. When $\ell = m = n$, this translates to $O(n^2)$ multiplications and additions, in contrast to the $O(n^3)$ cost of full matrix multiplication. These counts exclude the cost of sampling and introducing the random vector.

(4) In typical usage, the prover supplies the least residue matrices $\bar{A} = [\bar{a}_{ik}]$, $\bar{B} = [\bar{b}_{kj}]$, $\bar{C} = [\bar{c}_{ij}]$, and $\bar{D} = [\bar{d}_{ij}]$ as part of the witness, where each entry is the canonical representative of its corresponding integer modulo p.

If B and C represent fixed matrices—such as learned model weights and associated biases—they may be hardcoded into the circuit. This eliminates the need for the prover to supply $\bar{B}$ and $\bar{C}$ as part of the witness, significantly reducing prover-side workload and simplifying the input interface.

As noted, the circuit's goal—stated in Step (3)—admits an equivalent formulation in terms of congruences modulo p. This equivalence follows from Proposition 4.2, but it is essential that the assumption in (3.1) holds for all i, k, j. Extending the notion of congruence to matrices entrywise, as in Definition 4.1, we may therefore write:

$$AB + C = D \quad \Longleftrightarrow \quad \bar{A}\bar{B} + \bar{C} \equiv \bar{D} \bmod p, \tag{3.17}$$

provided all entries of $AB + C$ and D lie in the same interval of length p.

Thus, the goal of the circuit is to verify that $\bar{A}\bar{B} + \bar{C} \equiv \bar{D} \bmod p$ using a probabilistic check. If the congruence holds, the circuit accepts. If it does not, the circuit rejects with high probability.

(5) See Comment (5) of Subsection 2.2.

(6) In a non-interactive setting, the circuit is responsible for computing the product $\bar{B}\bar{X}$. This ensures soundness: if the prover were allowed to supply the result, they could choose $\bar{B}$ and $\bar{U}$ inconsistently to pass the final check without satisfying the intended matrix product relation. The verifier cannot compute $\bar{B}\bar{X}$ either, since $\bar{B}$ is not assumed to be public.

Although we phrase this step in terms of integer arithmetic—computing $\bar{B}\bar{X}$ and reducing modulo p—this is done purely for notational simplicity. In reality, the circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, and all values should be understood as elements of that field. Throughout this section, we adopt the convention of using least residue representatives from $\{0,1,\ldots,p-1\}$ to make the exposition more accessible, while recognizing that the underlying computations are carried out over the field.

To compute $\bar{B}\bar{X}$, the circuit performs m inner products of length n, requiring mn multiplication gates and $m(n-1)$ addition gates.

(7) In a non-interactive setting, the circuit—not the prover or verifier—must compute the product $\bar{C}\bar{X}$. Allowing the prover to supply the result would compromise soundness, as the prover could fabricate values of $\bar{C}$ that satisfy the check without correctly encoding a matrix product. The verifier cannot compute it either, as $\bar{C}$ is not assumed to be public.

As before, we describe the computation in terms of integer arithmetic—dot products followed by reduction modulo p—to keep the exposition elementary. However, the actual circuit performs these operations over the field $\mathbb{Z}/p\mathbb{Z}$, and values should be interpreted as field elements. We continue to use least residue representatives for clarity, but all arithmetic takes place in the field.

To compute $\bar{C}\bar{X}$, the circuit must evaluate ℓ inner products of length n, requiring ℓn multiplication gates and $\ell(n-1)$ addition gates.

(8) In a non-interactive setting, the circuit—not the prover or verifier—must compute the product $\bar{D}\bar{X}$. Allowing the prover to supply the result would compromise soundness, as the prover could fabricate values of $\bar{D}$ that satisfy the check without correctly encoding a matrix product. The verifier cannot compute it either, since $\bar{D}$ is not assumed to be public.

As with the previous steps, we describe this computation in terms of integer arithmetic for readability, but it is implemented over the field $\mathbb{Z}/p\mathbb{Z}$. All values should be understood as field elements, with least residue representatives used for notational convenience.

To compute $\bar{D}\bar{X}$, the circuit performs ℓ inner products of length n, requiring ℓn multiplication gates and $\ell(n-1)$ addition gates.

(9) Each of the ℓ constraints in (3.15) involves m multiplications and m additions (not counting the subtraction of $\bar{w}_i$).

In what follows, instead of writing $M_1 \equiv M_2 \bmod p$, we will simply write $M_1 = M_2$, with the understanding that equality holds over the field $\mathbb{Z}/p\mathbb{Z}$. Recall from Step (4) and Comment (4) that the goal of the circuit is to verify that $\bar{A}\bar{B}+\bar{C}=\bar{D}$ using a probabilistic check: the circuit accepts if the equality holds, and rejects with high probability—that is, accepts with low probability—if it does not.

The constraints of Step (9) collectively assert that $\bar{A}\bar{U}+\bar{V}=\bar{W}$. Substituting from (3.12), (3.13), and (3.14), we find that $\bar{U}=\bar{B}\bar{X}$, $\bar{V}=\bar{C}\bar{X}$, and $\bar{W}=\bar{D}\bar{X}$, so the constraint becomes

$$\bar{A}\bar{B}\bar{X}+\bar{C}\bar{X}=\bar{D}\bar{X} \tag{3.18}$$

or equivalently,

$$(\bar{A}\bar{B}+\bar{C}-\bar{D})\bar{X}=\mathbf{0}, \tag{3.19}$$

meaning that $\bar{X}$ lies in the null space of the matrix $\bar{A}\bar{B}+\bar{C}-\bar{D}$.

Thus, the constraints (3.15) are equivalent to the constraint (3.19), and we'll refer to the latter formulation below.

Let $\bar{O}_{\ell\times n}$ denote the zero matrix in $(\mathbb{Z}/p\mathbb{Z})^{\ell\times n}$. There are two cases. If the prover is honest and provides the correct $\bar{A},\bar{B},\bar{C},\bar{D}$ to the circuit, then $\bar{A}\bar{B}+\bar{C}-\bar{D}=\bar{O}_{\ell\times n}$ and (3.19) holds trivially.

(10) Continuing from Comment (9), if the prover is dishonest, so that $\bar{A}\bar{B}+\bar{C}-\bar{D}\neq\bar{O}_{\ell\times n}$, it is still possible that the constraint (2.19) holds, resulting in a false positive verification. However, this occurs with probability at most $1/p$, as we now explain.

Given that $\bar{A}\bar{B}+\bar{C}-\bar{D}$ is nonzero, its null space is a proper subspace of $(\mathbb{Z}/p\mathbb{Z})^n$ with dimension at most $n-1$. Therefore, the probability that a uniformly random vector $\bar{X}\in(\mathbb{Z}/p\mathbb{Z})^n$ lies in this null space is at most

$$\frac{\#N(\bar{A}\bar{B}+\bar{C}-\bar{D})}{\#(\mathbb{Z}/p\mathbb{Z})^n}\leqslant\frac{p^{n-1}}{p^n}=\frac{1}{p}. \tag{3.20}$$

(11) If the prover is dishonest, so that $\bar{A}\bar{B}+\bar{C}-\bar{D}\neq\bar{O}_{\ell\times n}$, then each random vector $\bar{X}_r$ lies in the null space of $\bar{A}\bar{B}+\bar{C}-\bar{D}$ with probability at most $1/p$, independently. Therefore, the probability that all s independent tests pass is at most $(1/p)^s$.

4. THE MATHS

Definition 4.1. Let $M = [m_{ij}]$ and $M' = [m'_{ij}]$ be integer matrices of the same dimensions. We write

$$M \equiv M' \bmod p$$

to mean that $m_{ij} \equiv m'_{ij} \bmod p$ for all i, j. We denote by $\bar{M}$ the matrix of least residues modulo p, that is, $\bar{M} = [\bar{m}_{ij}]$, where each $\bar{m}_{ij}$ is the unique integer in $\{0, 1, \ldots, p-1\}$ congruent to $m_{ij} \bmod p$. ■

Proposition 4.2. *Let p be a positive integer (not necessarily a prime), and let integers h, x, y satisfy*

$$h - p \leqslant x, y < h. \tag{4.1}$$

Then $x = y$ if and only if

$$x \equiv y \bmod p. \tag{4.2}$$

Remark 4.3. In Subsection 2.1, Step (2) assumes that both

$$x = \sum_{k=1}^{m} a_{ik} b_{kj} \quad \text{and} \quad y = c_{ij}$$

lie in the same interval of length p:

$$h - p \leqslant x, y < h.$$

Therefore, by Proposition 4.2, we have $x = y$ if and only if

$$x \equiv y \bmod p,$$

which is equivalent to

$$\sum_{k=1}^{m} \bar{a}_{ik} \bar{b}_{kj} \equiv \bar{c}_{ij} \bmod p.$$

(Recall that $\bar{z}$ denotes the least residue of $z \bmod p$.) Extending our notation to matrices as in Definition 4.1, and assuming that (2.1) holds for all i, k, j, we may write

$$AB = C \quad \Longleftrightarrow \quad \bar{A}\bar{B} \equiv \bar{C} \bmod p.$$

■

Proof of Proposition 4.2. If $x = y$ then $x \equiv y \bmod p$. For the converse, note that the congruence (4.2) holds if and only if there is an integer t such that

$$x - y = tp. \tag{4.3}$$

Since h, x, y are integers, the assumption (4.1) implies that x, y lie in the closed interval

$$h - p \leqslant x, y \leqslant h - 1.$$

Subtracting, we obtain

$$-(p-1) = h - p - (h-1) \leqslant x - y \leqslant (h-1) - (h-p) = p - 1.$$

If (4.3) holds, then

$$tp = x - y \in [-(p-1), p-1] \subseteq (-p, p),$$

so $t \in (-1, 1) \cap \mathbb{Z} = \{0\}$. Hence $x - y = 0$. □

Proposition 4.4. *Let V be an n-dimensional vector space over a finite field $\mathbb{F}$ of order q, and let $W \subseteq V$ be a subspace of dimension k. If $x \in V$ is chosen uniformly at random, then*

$$\mathbb{P}(x \in W) = \frac{1}{q^{n-k}}.$$

In particular, if W is the null space of a nonzero matrix in $\mathbb{F}^{\ell \times n}$, then

$$\mathbb{P}(x \in W) \leqslant \frac{1}{q}.$$

Proof. For the first claim, let $\{w_1, \ldots, w_k\}$ be a basis of W. Then each element of W is of the form $a_1 w_1 + \cdots + a_k w_k$ with $a_i \in \mathbb{F}$, so $\#W = q^k$. Since V has dimension n, we have $\#V = q^n$, and therefore

$$\mathbb{P}(x \in W) = \frac{\#W}{\#V} = \frac{q^k}{q^n}.$$

For the second claim, suppose W is the null space of a nonzero matrix $D \in \mathbb{F}^{\ell \times n}$. If $W = V$, then $De_j = 0$ for each standard basis vector $e_j \in \mathbb{F}^n$, where e_j has a 1 in the j-th coordinate and 0 elsewhere. But De_j is simply the j-th column of D, so all columns of D must be zero. Therefore, $D = 0$, contradicting our assumption.

In other words, if $D \neq 0$, then W is a proper subspace of V. Hence, $k \leqslant n-1$, and by the first part, $\mathbb{P}(x \in W) \leqslant 1/q$. □

5. The code

Rust code in this section is designed for implementation within the ExpanderCompilerCollection (ECC) framework [7, 8]. This library provides a specialized interface for constructing arithmetic circuits.

5.1. No bias. For simplicity, we start with the case where there is no bias term: $AB = C$.

Algorithm 5.1 `freivalds_check`: One-sided Monte Carlo verification of $AB = C$. Fails with high probability if $AB \neq C$.

Require: Matrices $A \in \mathbb{F}^{l \times m}$, $B \in \mathbb{F}^{m \times n}$, $C \in \mathbb{F}^{l \times n}$
Ensure: Reject with high probability if $AB \neq C$

Sample random vector $x \in \mathbb{F}^n$
Initialize $Bx \in \mathbb{F}^m$ as zero vector
for $i = 0$ to $m-1$ **do**
    for $j = 0$ to $n-1$ **do**
        $Bx[i] \leftarrow Bx[i] + B[i][j] \cdot x[j]$
    end for
end for
Initialize $ABx \in \mathbb{F}^l$ as zero vector
for $i = 0$ to $l-1$ **do**
    for $j = 0$ to $m-1$ **do**
        $ABx[i] \leftarrow ABx[i] + A[i][j] \cdot Bx[j]$
    end for
end for
Initialize $Cx \in \mathbb{F}^l$ as zero vector
for $i = 0$ to $l-1$ **do**
    for $j = 0$ to $n-1$ **do**
        $Cx[i] \leftarrow Cx[i] + C[i][j] \cdot x[j]$
    end for
end for
for $i = 0$ to $l-1$ **do**
    Assert $ABx[i] = Cx[i]$
end for

```rust
use expander_compiler::frontend::*;

const N_ROWS_A: usize = 4; // l
const N_COLS_A: usize = 4; // m
const N_ROWS_B: usize = 4; // m
const N_COLS_B: usize = 4; // n

declare_circuit!(MatMulCircuit {
    matrix_a: [[Variable; N_COLS_A]; N_ROWS_A],         // (l x m)
    matrix_b: [[Variable; N_COLS_B]; N_ROWS_B],         // (m x n)
    matrix_product_ab: [[Variable; N_COLS_B]; N_ROWS_A] // (l x n), provided by prover
});

impl<C: Config> Define<C> for MatMulCircuit<Variable> {
    fn define<Builder: RootAPI<C>>(&self, api: &mut Builder) {
        let a: Vec<Vec<Variable>> = self.matrix_a.iter().map(|row| row.to_vec()).collect();
        let b: Vec<Vec<Variable>> = self.matrix_b.iter().map(|row| row.to_vec()).collect();
        let c: Vec<Vec<Variable>> = self.matrix_product_ab.iter().map(|row| row.to_vec()).collect
    ();

        // 1. Sample random vector x in F^n
        let x: Vec<Variable> = (0..N_COLS_B).map(|_| api.get_random_value()).collect();
        // let mut x = vec![];
        // for _ in 0..N_COLS_B {
        //     x.push(api.get_random_value());
        // }
        // println!("Random vector x: {:?}", x);

        // 2. Compute u = Bx, which lies in F^m
        let mut bx = vec![api.constant(C::CircuitField::zero()); N_ROWS_B];
        for i in 0..N_ROWS_B {
            for j in 0..N_COLS_B {
                let prod = api.mul(b[i][j], x[j]);
                bx[i] = api.add(bx[i], prod);
            }
        }

        // 3. Compute A(Bx), which lies in F^l
        let mut abx = vec![api.constant(C::CircuitField::zero()); N_ROWS_A];
        for i in 0..N_ROWS_A {
            for j in 0..N_COLS_A {
                let prod = api.mul(a[i][j], bx[j]);
                abx[i] = api.add(abx[i], prod);
            }
        }

        // 4. Compute v = Cx, which lies in F^l
        let mut cx = vec![api.constant(C::CircuitField::zero()); N_ROWS_A];
        for i in 0..N_ROWS_A {
            for j in 0..N_COLS_B {
                let prod = api.mul(c[i][j], x[j]);
                cx[i] = api.add(cx[i], prod);
            }
        }

        // 5. Freivalds check: abx == cx
        for i in 0..N_ROWS_A {
            api.assert_is_equal(abx[i], cx[i]);
        }
    }
}
```

Listing 1: ECC Rust API: One-sided Monte Carlo verification of $AB = C$. Fails with high probability if $AB \neq C$

5.2. Bias. A slight extension allows us to consider a bias term: $AB + C = D$.

Algorithm 5.2 `freivalds_check_with_bias`: One-sided Monte Carlo verification of $AB + C = D$. Fails with high probability if $AB + C \neq D$.

Require: Matrices $A \in \mathbb{F}^{\ell \times m}$, $B \in \mathbb{F}^{m \times n}$, $C, D \in \mathbb{F}^{\ell \times n}$
Ensure: Reject with high probability if $AB + C \neq D$

Sample random vector $x \in \mathbb{F}^n$
Initialize $Bx \in \mathbb{F}^m$ as zero vector
for $i = 0$ to $m - 1$ **do**
    for $j = 0$ to $n - 1$ **do**
        $Bx[i] \leftarrow Bx[i] + B[i][j] \cdot x[j]$
    end for
end for
Initialize $ABx \in \mathbb{F}^\ell$ as zero vector
for $i = 0$ to $\ell - 1$ **do**
    for $j = 0$ to $m - 1$ **do**
        $ABx[i] \leftarrow ABx[i] + A[i][j] \cdot Bx[j]$
    end for
end for
Initialize $Cx \in \mathbb{F}^\ell$ as zero vector
for $i = 0$ to $\ell - 1$ **do**
    for $j = 0$ to $n - 1$ **do**
        $Cx[i] \leftarrow Cx[i] + C[i][j] \cdot x[j]$
    end for
end for
Initialize $Dx \in \mathbb{F}^\ell$ as zero vector
for $i = 0$ to $\ell - 1$ **do**
    for $j = 0$ to $n - 1$ **do**
        $Dx[i] \leftarrow Dx[i] + D[i][j] \cdot x[j]$
    end for
end for
for $i = 0$ to $\ell - 1$ **do**
    Assert $ABx[i] + Cx[i] = Dx[i]$
end for

```rust
use expander_compiler::frontend::*;

const N_ROWS_A: usize = 4; // l
const N_COLS_A: usize = 4; // m
const N_ROWS_B: usize = 4; // m
const N_COLS_B: usize = 4; // n

declare_circuit!(MatMulBiasCircuit {
    matrix_a: [[Variable; N_COLS_A]; N_ROWS_A],          // (l x m)
    matrix_b: [[Variable; N_COLS_B]; N_ROWS_B],          // (m x n)
    matrix_c: [[Variable; N_COLS_B]; N_ROWS_A],          // (l x n), bias
    matrix_d: [[Variable; N_COLS_B]; N_ROWS_A]           // (l x n), claimed output
});

impl<C: Config> Define<C> for MatMulBiasCircuit<Variable> {
    fn define<Builder: RootAPI<C>>(&self, api: &mut Builder) {
        let a: Vec<Vec<Variable>> = self.matrix_a.iter().map(|row| row.to_vec()).collect();
        let b: Vec<Vec<Variable>> = self.matrix_b.iter().map(|row| row.to_vec()).collect();
        let c: Vec<Vec<Variable>> = self.matrix_c.iter().map(|row| row.to_vec()).collect();
        let d: Vec<Vec<Variable>> = self.matrix_d.iter().map(|row| row.to_vec()).collect();

        // 1. Sample random vector x in F^n
        let x: Vec<Variable> = (0..N_COLS_B).map(|_| api.get_random_value()).collect();

        // 2. Compute u = Bx, which lies in F^m
        let mut bx = vec![api.constant(C::CircuitField::zero()); N_ROWS_B];
        for i in 0..N_ROWS_B {
            for j in 0..N_COLS_B {
                let prod = api.mul(b[i][j], x[j]);
                bx[i] = api.add(bx[i], prod);
            }
        }

        // 3. Compute ABx, which lies in F^l
        let mut abx = vec![api.constant(C::CircuitField::zero()); N_ROWS_A];
        for i in 0..N_ROWS_A {
            for j in 0..N_COLS_A {
                let prod = api.mul(a[i][j], bx[j]);
                abx[i] = api.add(abx[i], prod);
            }
        }

        // 4. Compute Cx, which lies in F^l
        let mut cx = vec![api.constant(C::CircuitField::zero()); N_ROWS_A];
        for i in 0..N_ROWS_A {
            for j in 0..N_COLS_B {
                let prod = api.mul(c[i][j], x[j]);
                cx[i] = api.add(cx[i], prod);
            }
        }

        // 5. Compute Dx, which lies in F^l
        let mut dx = vec![api.constant(C::CircuitField::zero()); N_ROWS_A];
        for i in 0..N_ROWS_A {
            for j in 0..N_COLS_B {
                let prod = api.mul(d[i][j], x[j]);
                dx[i] = api.add(dx[i], prod);
            }
        }

        // 6. Freivalds check: ABx + Cx == Dx
        for i in 0..N_ROWS_A {
            let lhs = api.add(abx[i], cx[i]);
            api.assert_is_equal(lhs, dx[i]);
        }
    }
}
```

Listing 2: ECC Rust API: One-sided Monte Carlo verification of $AB + C = D$. Fails with high probability if $AB + C \neq D$

6. EXAMPLES

Example 6.1. We illustrate the process of verifying a matrix multiplication over the finite field $\mathbb{Z}/101\mathbb{Z}$ using the deterministic approach in (a), followed by the probabilistic approach in (b). We assume the balanced residue range $(-50, 50]$ and adopt the least residue convention in all modular representations.

(a) Let

$$A = \begin{bmatrix} 2 & -3 \\ 4 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 5 \\ 2 & 3 \end{bmatrix}.$$

Since each entry of A and B is bounded in absolute value by 5, with equality attained only for one entry, it follows that each entry of AB lies in $(-2 \cdot 5^2, 2 \cdot 5^2) \subseteq (-50, 50]$.

$$\bar{A} = \begin{bmatrix} 2 & 98 \\ 4 & 1 \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} 100 & 5 \\ 2 & 3 \end{bmatrix},$$

and, if the prover is honest,

$$\bar{C} = \begin{bmatrix} 93 & 1 \\ 99 & 23 \end{bmatrix}.$$

To see this, note that

$$\bar{A}\bar{B} \equiv \begin{bmatrix} (2)(100)+(98)(2) & (2)(5)+(98)(3) \\ (4)(100)+(1)(2) & (4)(5)+(1)(3) \end{bmatrix} \equiv \begin{bmatrix} 396 & 304 \\ 402 & 23 \end{bmatrix} \equiv \begin{bmatrix} 93 & 1 \\ 99 & 23 \end{bmatrix} \bmod 101.$$

We now replace the entries of $\bar{C}$ with their balanced residue equivalents in $(-50, 50]$ to obtain

$$C = \begin{bmatrix} -8 & 1 \\ -2 & 23 \end{bmatrix}.$$

Thus, we expect that $AB = C$. Indeed,

$$AB = \begin{bmatrix} (2)(-1)+(-3)(2) & (2)(5)+(-3)(3) \\ (4)(-1)+(1)(2) & (4)(5)+(1)(3) \end{bmatrix} = \begin{bmatrix} -2-6 & 10-9 \\ -4+2 & 20+3 \end{bmatrix} = \begin{bmatrix} -8 & 1 \\ -2 & 23 \end{bmatrix}.$$

(b) As noted in (a), the entries in A and B are such that $AB = C$ if and only if $\bar{A}\bar{B} = \bar{C}$. Suppose the prover is dishonest and claims that $\bar{A}\bar{B} = \bar{D}$, where

$$\bar{D} = \begin{bmatrix} 93 & 99 \\ 1 & 23 \end{bmatrix}.$$

We choose a random vector

$$\bar{X} = \begin{bmatrix} 97 \\ 2 \end{bmatrix} \in \mathbb{Z}/101\mathbb{Z}.$$

We now compute three matrix-vector products to perform Freivalds' check (all arithmetic performed over $\mathbb{Z}/101\mathbb{Z}$):

$$\bar{U} := \bar{B}\bar{X} = \begin{bmatrix} 100 & 5 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 97 \\ 2 \end{bmatrix} = \begin{bmatrix} 100 \cdot 97 + 5 \cdot 2 \\ 2 \cdot 97 + 3 \cdot 2 \end{bmatrix} = \begin{bmatrix} 4+10 \\ 93+6 \end{bmatrix} = \begin{bmatrix} 14 \\ 99 \end{bmatrix}.$$

$$\bar{V} := \bar{D}\bar{X} = \begin{bmatrix} 93 & 99 \\ 1 & 23 \end{bmatrix} \begin{bmatrix} 97 \\ 2 \end{bmatrix} = \begin{bmatrix} 93 \cdot 97 + 99 \cdot 2 \\ 1 \cdot 97 + 23 \cdot 2 \end{bmatrix} = \begin{bmatrix} 32+99 \\ 97+46 \end{bmatrix} = \begin{bmatrix} 30 \\ 42 \end{bmatrix}.$$

$$\bar{A}\bar{U} = \begin{bmatrix} 2 & 98 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 14 \\ 99 \end{bmatrix} = \begin{bmatrix} 2 \cdot 14 + 98 \cdot 99 \\ 4 \cdot 14 + 1 \cdot 99 \end{bmatrix} = \begin{bmatrix} 28+6 \\ 56+99 \end{bmatrix} = \begin{bmatrix} 34 \\ 54 \end{bmatrix}.$$

Since $\bar{A}\bar{U} \neq \bar{V}$, we conclude that the prover's claim $\bar{A}\bar{B} = \bar{D}$ is false, and the verification fails.

In this case, the null space of

$$\bar{A}\bar{B} - \bar{D} = \begin{bmatrix} 0 & 98 \\ 3 & 0 \end{bmatrix}$$

is trivial, consisting only of the zero vector. Therefore, the probability that a random vector $\bar{X}$ lies in this null space is exactly $1/101^2$.

(c) Now for an admittedly contrived example to show that it is possible, albeit unlikely, for a dishonest prover to pass verification. Suppose the prover is dishonest and claims that $\bar{A}\bar{B} = \bar{D}$, where

$$\bar{D} = \begin{bmatrix} 86 & 88 \\ 78 & 82 \end{bmatrix}.$$

We choose a random vector

$$\bar{X} = \begin{bmatrix} 99 \\ 1 \end{bmatrix} \in \mathbb{Z}/101\mathbb{Z}.$$

We now compute three matrix-vector products to perform Freivalds' check (all arithmetic performed over $\mathbb{Z}/101\mathbb{Z}$):

$$\bar{U} := \bar{B}\bar{X} = \begin{bmatrix} 100 & 5 \\ 2 & 3 \end{bmatrix}\begin{bmatrix} 99 \\ 1 \end{bmatrix} = \begin{bmatrix} 100\cdot 99 + 5\cdot 1 \\ 2\cdot 99 + 3\cdot 1 \end{bmatrix} = \begin{bmatrix} 2+5 \\ 97+3 \end{bmatrix} = \begin{bmatrix} 7 \\ 100 \end{bmatrix}.$$

$$\bar{V} := \bar{D}\bar{X} = \begin{bmatrix} 86 & 88 \\ 78 & 82 \end{bmatrix}\begin{bmatrix} 99 \\ 1 \end{bmatrix} = \begin{bmatrix} 86\cdot 99 + 88\cdot 1 \\ 78\cdot 99 + 82\cdot 1 \end{bmatrix} = \begin{bmatrix} 30+88 \\ 46+82 \end{bmatrix} = \begin{bmatrix} 17 \\ 27 \end{bmatrix}.$$

$$\bar{A}\bar{U} = \begin{bmatrix} 2 & 98 \\ 4 & 1 \end{bmatrix}\begin{bmatrix} 7 \\ 100 \end{bmatrix} = \begin{bmatrix} 2\cdot 7 + 98\cdot 100 \\ 4\cdot 7 + 1\cdot 100 \end{bmatrix} = \begin{bmatrix} 14+3 \\ 28+100 \end{bmatrix} = \begin{bmatrix} 17 \\ 27 \end{bmatrix}.$$

Since $\bar{A}\bar{U} = \bar{V}$, the verification passes, in spite of the fact that $\bar{A}\bar{B} \neq \bar{D}$.

In this case, the null space of

$$\bar{A}\bar{B} - \bar{D} = \begin{bmatrix} 7 & 14 \\ 21 & 42 \end{bmatrix}$$

has basis $\{(-2,1)\}$, consisting 101 scalar multiples of this vector. Therefore, the probability that a random vector $\bar{X}$ lies in this null space is exactly $101/101^2 = 1/101$.

The dishonest prover happened to get lucky: the verifier's random choice fell into the null space of the error matrix. Should the verifier repeat the check with another independent random vector, the probability of escaping detection would drop to $1/101^2$. ■

REFERENCES

[1] Rūsiņš Freivalds. "Probabilistic Machines Can Use Less Running Time". In: *Information Processing 77: Proceedings of IFIP Congress 77, Toronto, August 8–12, 1977*. Ed. by Bruce Gilchrist. International Federation for Information Processing (IFIP). Toronto, Canada: North-Holland, 1977, pp. 839–842.

[2] Inference Labs. *Matrix Addition, Hadamard Products, and Matrix Multiplication: Arithmetic Circuit Blueprint*. `https://github.com/inference-labs-inc/zkml-blueprints/blob/main/matmul/matrix_addition_hadamard_product_matrix_multiplication.pdf`. Accessed April 21, 2025.

[3] Inference Labs. *Quantized Matrix Multiplication: Arithmetic Circuit Blueprint*. `https://github.com/inference-labs-inc/zkml-blueprints/blob/main/matmul/quantized_matrix_multiplication.pdf`. Accessed April 21, 2025.

[4] Inference Labs. *Range Check and ReLU: Arithmetic Circuit Blueprint*. `https://github.com/inference-labs-inc/zkml-blueprints/blob/main/core_ops/range_check_and_relu.pdf`. Accessed April 21, 2025.

[5] Inference Labs. *Range Check, Max, Min, and ReLU: Arithmetic Circuit Blueprint*. `https://github.com/inference-labs-inc/zkml-blueprints/blob/main/core_ops/range_check_max_min_relu.pdf`. Accessed April 21, 2025.

[6] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995. DOI: 10.1017/CBO9780511814075. URL: https://doi.org/10.1017/CBO9780511814075.

[7] Polyhedra Network. *ExpanderCompilerCollection: High-Level Circuit Compiler for the Expander Proof System*. `https://github.com/PolyhedraZK/ExpanderCompilerCollection`. Accessed April 21, 2025.

[8] Polyhedra Network. *ExpanderCompilerCollection: Rust Frontend Introduction*. `https://docs.polyhedra.network/expander/rust/intro`. Accessed April 21, 2025.