QUANTIZED MATRIX MULTIPLICATION I

1. THE PROCESS

1.1. Steps in the process. Each step in the process has a corresponding explanatory note in Subsection 1.2 that provides additional context and details.

1. The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is an n-bit prime:
$$2^{n-1} < p < 2^n.$$
The modulus p and its bitlength n are *public circuit constants*.

2. Let $\alpha > 1$ be an integer scaling factor. Consider integer matrices $A = [a_{ik}]$ and $B = [b_{kj}]$ of dimensions $\ell \times m$ and $m \times n$, respectively. Let $Q = [q_{ij}]$ be an $\ell \times n$ integer matrix.
We aim to verify that for all i, j, there exists an integer r_{ij} such that
$$\sum_{k=1}^{m} a_{ik} b_{kj} = \alpha q_{ij} + r_{ij}, \quad \text{where} \quad 0 \leqslant r_{ij} < \alpha. \tag{1.1}$$
This expresses q_{ij} as the integer quotient of the (i, j) entry of AB when divided by α.
The scaling factor α and matrix dimensions ℓ, m, n are *public circuit constants*. Since many proof systems do not support negative integers, we work with the least residues modulo p. Instead of taking A, B, and Q as inputs to the circuit, we take their least-residue representations $A' = [a'_{ik}]$, $B' = [b'_{kj}]$, and $Q' = [q'_{ij}]$. The matrices A' and Q' are *private* circuit inputs, while B' may be *public*, depending on the context (e.g., if B represents fixed neural network weights).
Rather than providing Q' as an explicit input, we compute it within the circuit in an unconstrained environment. This reduces memory overhead during compilation.
To verify correctness, we impose constraints that ensure A', B', and Q' satisfy a relationship that implies the desired mathematical relationship between A, B, and Q. While computing Q' in an unconstrained environment ensures correctness, these constraints are necessary to guarantee that Q' corresponds to the expected quantized product within the constrained execution of the circuit.

3. Assume that for all i, k, j,
$$-\frac{p}{2} < a_{ik}, b_{kj}, q_{ij} \leqslant \frac{p}{2}. \tag{1.2}$$
This assumption must be justified by *off-circuit preprocessing*.

4. Let U be an integer such that, for all i, k, j,
$$-(\alpha U + 1) \leqslant a_{ik}, b_{kj} \leqslant \alpha U + 1. \tag{1.3}$$
From (1.1) and (1.3), any valid q_{ij} must satisfy
$$\alpha |q_{ij}| \leqslant m(\alpha U + 1)^2 + (\alpha - 1). \tag{1.4}$$
Assume that there exists an integer $\nu \leqslant n - 1$ such that
$$m(\alpha U + 1)^2 + (\alpha - 1) \leqslant 2^{\nu - 1}\alpha < \frac{p}{2}. \tag{1.5}$$
Choose the smallest such ν. The left inequality ensures that the range check in Step 8 does not exclude any valid q_{ij}.
This step is performed *off-circuit* to find suitable U and ν. Once chosen, U and ν become *public circuit constants* for the subsequent on-circuit verification.

5. Since many frameworks do not support signed integers, even in unconstrained environments, convert each a_{ik} and b_{kj} to its least-residue modulo p, denoted a'_{ik} and b'_{kj}, respectively.

   This conversion is handled as *off-circuit preprocessing*. The circuit will then receive a'_{ik} and b'_{kj} as inputs.

6. For each pair of indices i, j, impose the following constraint in the circuit:

$$d_{ij}^{\sharp} \equiv 2^{\nu-1}\alpha + \sum_{k=1}^{m} a'_{ik} b'_{kj} \bmod p. \tag{1.6}$$

   Because all circuit variables are naturally taken modulo p, we interpret each $d_{ij}^{\sharp}$ as its least-residue representative in the interval $[0, p)$.

7. In an *unconstrained* environment, use the division algorithm to determine the unique pair of integers $q_{ij}^{\sharp}$ and r_{ij} such that

$$d_{ij}^{\sharp} = \alpha q_{ij}^{\sharp} + r_{ij} \quad \text{with} \quad 0 \leqslant r_{ij} < \alpha. \tag{1.7}$$

   If (1.3) and (1.5) hold and $d_{ij}^{\sharp}$ is computed correctly—which, in view of the constraint (1.6), we may assume is the case—then in fact

$$d_{ij}^{\sharp} = 2^{\nu-1}\alpha + \sum_{k=1}^{m} a_{ik} b_{kj},$$

   which lies in $[0, 2^{\nu}\alpha)$. It follows that if $q_{ij}^{\sharp}$ and r_{ij} are computed correctly, then $q_{ij}^{\sharp}$ will fall within $[0, 2^{\nu})$.

   Once fed into the circuit, $q_{ij}^{\sharp}$ and r_{ij} are each represented as least residues modulo p (i.e. in the range $[0, p)$). In Step 8, we perform in-circuit range checks to confirm $q_{ij}^{\sharp} < 2^{\nu}$ and $r_{ij} < \alpha$.

8. Perform a range check on each $q_{ij}^{\sharp}$ to verify that it satisfies

$$0 \leqslant q_{ij}^{\sharp} < 2^{\nu}. \tag{1.8}$$

   One approach is as follows:

   - Extract the ν least significant bits $\beta_{\nu-1}, \ldots, \beta_0$ from the binary representation of $q_{ij}^{\sharp}$.
   - Impose the constraints

$$\beta_i(\beta_i - 1) \equiv 0 \bmod p \quad \text{and} \quad q_{ij}^{\sharp} \equiv 2^{\nu-1}\beta_{\nu-1} + \cdots + 2^0\beta_0 \bmod p.$$

   Similarly, perform a range check on each r_{ij} to verify that it satisfies

$$0 \leqslant r_{ij} < \alpha. \tag{1.9}$$

9. If $q_{ij}^{\sharp}$ and r_{ij} have been computed correctly, we must have

$$q_{ij}^{\sharp} = q_{ij} + 2^{\nu-1},$$

   where q_{ij} is the integer quotient from (1.1). Let q'_{ij} be the (circuit) variable representing the least residue of $q_{ij}^{\sharp} - 2^{\nu-1}$ modulo p. Impose the constraint

$$q_{ij}^{\sharp} \equiv q'_{ij} + 2^{\nu-1} \bmod p. \tag{1.10}$$

   Together with the range checks (1.8) and (1.9), as well as the other assumptions ((1.2), (1.3), (1.5), etc.), this ensures that q'_{ij} indeed corresponds to the least-residue representation of q_{ij}, where q_{ij} is the integer satisfying (1.1).

1.2. Commentary on each step.

1. Typically, $n \approx 256$.

2. In practice, common choices for the scaling factor α include 2^{16}, 2^{21}, and 2^{32}. A power of two simplifies range checking [Step 8]. The value of m varies and may be around 28, 256, 1568, and so on.

   While not directly visible to the circuit, the integer matrices A and B are quantized versions of real-valued matrices $X = [x_{ik}]$ and $Y = [y_{kj}]$ at scale α, meaning:

   $$a_{ik} = \lfloor \alpha x_{ik} \rfloor, \quad \text{and} \quad b_{kj} = \lfloor \alpha y_{kj} \rfloor.$$

   We refer to Q as the *quantized product* of XY at scale α. It provides a coarser approximation to αXY than $\lfloor \alpha XY \rfloor$ because the discretization happens before multiplication.

   While not strictly required, matrices X and Y are often normalized to the range $[-1,1]$ with mean zero. In general, we assume that all entries of X and Y lie in $[-U,U]$ for some integer U. Ideally, their values are not too small, and their product $Z = [z_{ij}] = XY$, given by

   $$z_{ij} = \sum_{k=1}^{m} x_{ik} y_{kj},$$

   also consists of values that are not too small. However, cancellation effects could still result in small z_{ij} even if $x_{ik} y_{kj}$ are bounded away from zero.

   If $|z_{ij}|$ is not too small, the relative error in approximating z_{ij} by q_{ij}/α is acceptable. However, this error can be large if $|z_{ij}|$ happens to be very small. See Proposition 2.2 and Remark 2.3 for details.

   Naturally, the error introduced by quantization impacts the performance of a neural network.

   Since the circuit does not have access to the original values in X and Y, it is impossible to verify whether an individual matrix has been correctly quantized. For example, with $\alpha = 2^4$, both 0.04 and 0.05 would be mapped to 0 if we take the floor of scaled values, meaning the circuit cannot distinguish whether the original value was 0.04 or 0.05 (or anything else in $[0, 1/16)$).

   However, this is irrelevant to our verification: the neural network and inputs, as implemented in the circuit, *are* the quantized versions of an underlying network and dataset. If certain weights are public, we can quantize them ourselves.

   Thus, the circuit accepts as inputs the scaling factor α and the least-residue representations A', B', and Q' (the superscript indicates that the values have been reduced modulo p). In a typical neural network setting, where A represents the layer inputs and B represents the weights, A' and Q' are kept private while α and—if applicable—B' are public. The matrix dimensions are also public and treated as fixed circuit constants. The objective is to verify a specific relationship among A', B', and Q', which in turn implies the desired relationship between A, B, and Q.

   As mentioned, Q' is computed in an unconstrained environment to reduce memory usage during circuit compilation.

3. No range check is needed to verify (1.2), as it reflects a fundamental assumption: all inputs to the circuit are elements of a complete set of residues modulo p (balanced residues in this case).

   In fact, no range check can verify this—since the circuit operates over the integers modulo p, it cannot determine whether a given input lies in a particular interval of length p.

   However, in practice, there must be some off-circuit reasoning or preprocessing that ensures this assumption is valid. If the system represents all values as 64-bit signed integers (i.e., `i64`), then they naturally lie within the range $[-2^{63}, 2^{63})$, which typically falls within $[-p/2, p/2)$ with a large margin. While this does not directly enforce (1.2) as an arithmetic circuit constraint, it provides a practical justification for assuming that all inputs lie within the balanced residue range.

4. As noted in Comment 2, there exists and integer U such that for all i,k,j,

   $$-U \leqslant x_{ik}, y_{kj} \leqslant U. \tag{1.11}$$

   Since a_{ik} and b_{kj} are quantized versions of x_{ik} and y_{kj}, obtained by scaling by α and rounding, the bound (1.3) follows. Thus, the (i,j) entry of AB satisfies

   $$\left| \sum_{k=1}^{m} a_{ik} b_{kj} \right| \leqslant \sum_{k=1}^{m} |a_{ik}||b_{kj}| \leqslant \sum_{k=1}^{m} (\alpha U + 1)(\alpha U + 1) = m(\alpha U + 1)^2. \tag{1.12}$$

   From (1.1), a correct q_{ij} satisfies

   $$\alpha |q_{ij}| \leqslant \left| \sum_{k=1}^{m} a_{ik} b_{kj} \right| + |r_{ij}| \leqslant m(\alpha U + 1)^2 + (\alpha - 1).$$

To ensure that the range check in Step 8 does not exclude any valid q_{ij}, we require that (1.5) holds.

This condition imposes bounds on m, α, and U, relative to p, which are easily satisfied in practice when p is of order 2^{256}. For example, the following parameter choices satisfy (1.5):

$$U = 1, \quad m \leqslant 256, \quad \alpha = 2^{21}, \quad \nu = 10, \quad p > 2^{32}.$$

One might think that (1.5) makes (1.2) redundant. However, this is not necessarily the case: for example, row i of A may be orthogonal to column j of B, meaning

$$\sum_{k=1}^{m} a_{ik} b_{kj} = 0,$$

even if individual entries of A and B are arbitrarily large.

- The requirement that $\nu \leqslant n-1$ is crucial for Step 8 (see Comment 8).
- We can replace $2^{\nu-1}$ with a more general integer V, but using a power of 2 simplifies range checking [Step 8].
- We treat α and ν as circuit constants, so no explicit constraints are needed to ensure that (1.5) holds.
- We do not impose constraints to enforce (1.3) and (1.4). That is, we do not strictly require these bounds to hold. The only essential requirement is that (1.1) holds for all i, j. It is possible that for certain A, B, and Q, (1.1) is satisfied even if (1.3) or (1.4) do not hold. Such cases will still pass verification.

  The constraints (1.8), (1.9), and (1.10) alone do not imply (1.1), which is why we assume (1.3) and (1.4). These assumptions are consistent with the setup described in Comment 2.

5. Converting integers to least residues modulo p is done in off-circuit preprocessing. Assuming $-p/2 \leqslant x < p/2$ (see (1.2)) and letting x' denote the least residue of x modulo p, we have

$$x' = \begin{cases} x & \text{if } x \geqslant 0 \\ p + x & \text{if } x < 0 \end{cases}$$

   Conversely, assuming $0 \leqslant x < p$ and letting x' denote the balanced residue of x modulo p, we have

$$x = \begin{cases} x' & \text{if } x < p/2 \\ x' - p & \text{if } x \geqslant p/2 \end{cases}$$

6. This step is straightforward.

7. Since $0 \leqslant d_{ij}^{\sharp} < p$ by definition ($d_{ij}^{\sharp}$ is a least residue modulo p), and since α is positive, the quotient $q_{ij}^{\sharp}$ is necessarily nonnegative. If all relevant calculations are performed correctly, then

$$d_{ij}^{\sharp} \equiv 2^{\nu-1}\alpha + \sum_{k=1}^{m} a_{ik} b_{kj} \bmod p. \tag{1.13}$$

   (Since $a'_{ik} \equiv a_{ik} \bmod p$ and $b'_{kj} \equiv b_{kj} \bmod p$, we may replace $a'_{ik} b'_{kj}$ by $a_{ik} b_{kj}$ in (1.6).)

   By (1.12), which follows from (1.3), and the left inequality in (1.5),

$$\left| \sum_{k=1}^{m} a_{ik} b_{kj} \right| \leqslant m(\alpha U + 1)^2 \leqslant 2^{\nu-1}\alpha - (\alpha - 1) < 2^{\nu-1}\alpha,$$

   because $\alpha > 1$. Therefore,

$$0 \leqslant 2^{\nu-1}\alpha + \sum_{k=1}^{m} a_{ik} b_{kj} < 2^{\nu}\alpha. \tag{1.14}$$

   The right inequality in (1.5) ensures that $2^{\nu}\alpha < p$. Since both sides of (1.13) are in $[0, p)$, we conclude

$$d_{ij}^{\sharp} = 2^{\nu-1}\alpha + \sum_{k=1}^{m} a_{ik} b_{kj}. \tag{1.15}$$

   By (1.14) and (1.15), if (1.7) holds, then

$$0 \leqslant q_{ij}^{\sharp} < 2^{\nu}. \tag{1.16}$$

   While (1.15) and (1.16) hold if all calculations are correct, verifying this correctness is precisely the role of the circuit. However, we assume that $q_{ij}^{\sharp}$ and r_{ij} are within $[0, p)$ as least residues modulo p. This assumption is valid because all circuit variables are defined as least residues modulo p, ensuring that any value assigned to them by the prover is automatically reduced to this range.

8. Our companion document on range checks provides a detailed discussion. The constraint $\nu \leqslant n-1$ (from Step 4) is crucial to prevent aliasing. If $\nu = n$, the extracted ν least significant bits $\beta_{\nu-1}, \ldots, \beta_0$ could correspond to either $q_{ij}^\sharp$ or $q_{ij}^\sharp + p$, provided the latter is still less than 2^n.

   The assumption that $q_{ij}^\sharp$ and r_{ij} fall within $[0, p)$ [see the end of Step 7 and Comment 7] is essential. If $q_{ij}^\sharp$ exceeds the expected range $[0, 2^\nu)$ but remains within $[0, p)$, then its least residue modulo p falls in $[2^\nu, p)$. In this case, $q_{ij}^\sharp$ is not fully determined by its ν least significant bits, leading to range check rejection. However, if values outside the least residue range were allowed, this reasoning could fail. For instance, if $q_{ij}^\sharp = p$, then its least residue modulo p is 0, which is fully determined by its ν least significant bits. Consequently, the range check would erroneously accept $q_{ij}^\sharp$ as valid despite being out of range.

   The range check on r_{ij} is necessary to prevent a dishonest prover from manipulating the decomposition

$$2^{\nu-1}\alpha + \sum_{k=1}^{m} a_{ik} b_{kj} \overset{(1.7)}{\underset{(1.15)}{=}} \alpha q_{ij}^\sharp + r_{ij} \tag{1.17}$$

   by choosing an incorrect $q_{ij}^\sharp$ and compensating with an out-of-range r_{ij}. Without this check, a prover could arbitrarily shift $q_{ij}^\sharp$ by multiples of α while maintaining a valid sum, leading to false positive verification. The constraint $0 \leqslant r_{ij} < \alpha$ ensures that $q_{ij}^\sharp$ is uniquely determined.

9. Subtracting $2^{\nu-1}\alpha$ from both sides of (1.17) yields

$$\sum_{k=1}^{m} a_{ik} b_{kj} = \alpha(q_{ij}^\sharp - 2^{\nu-1}) + r_{ij}. \tag{1.18}$$

   Comparing (1.1) with (1.18), and noting that the quotient and remainder from the division algorithm are uniquely determined when $0 \leqslant r_{ij} < \alpha$ (which has been verified), it follows that

$$q_{ij} = q_{ij}^\sharp - 2^{\nu-1}.$$

   The guarantee provided by (1.8) that $q_{ij}^\sharp$ falls within $[0, 2^\nu)$ is equivalent to

$$-2^{\nu-1} \leqslant q_{ij} < 2^{\nu-1}.$$

   Thus, the range check on $q_{ij}^\sharp$ in Step 8 does not erroneously reject any valid q_{ij} (see (1.4) and the left inequality in (1.5)).

   However, we must still impose a circuit constraint to verify the relationship between $q_{ij}^\sharp$ and q_{ij}. Since we work with least residues, we compute the least residue q'_{ij} of $q_{ij} = q_{ij}^\sharp - 2^{\nu-1}$. The constraint (1.10) is mathematically equivalent to

$$q_{ij}^\sharp \equiv q_{ij} + 2^{\nu-1} \bmod p.$$

   See Proposition 2.1 for a proof that constraints (1.8), (1.9), and (1.10), along with the underlying assumptions, ensure that q'_{ij} is indeed the least residue of the integer q_{ij} (defined in (1.1)) modulo p.

Remark 1.1. The definitions of $q_{ij}^\sharp$, $d^\sharp$, a'_{ik}, b'_{kj}, q'_{ij}, etc., serve to disambiguate different representations of key quantities at various stages of computation and verification. Mathematically, it is straightforward to work with integer values, but in circuit frameworks, all computations must be performed modulo p, even in unconstrained environments. This necessitates working with least residues to ensure that modular reductions, range checks, and arithmetic constraints remain valid.

In particular, these definitions prevent aliasing issues in range checks, maintain consistency in modular arithmetic, and separate computational steps from verification steps. Since circuits do not inherently track how values were originally defined, all conversions (e.g., mapping integers to their least residues) must be handled in preprocessing before verification occurs. ■

2. The Maths

2.1. Proof of correctness under constraints.

Proposition 2.1. *Let p be a positive integer (not necessarily prime). Suppose $\alpha > 1$, m, U, and ν are positive integers satisfying*

$$m(\alpha U+1)^2+(\alpha-1)\leqslant 2^{\nu-1}\alpha<\frac{p}{2}. \tag{2.1}$$

For fixed indices i and j, let a_{ik}, b_{kj} (for $k=1,\dots,m$) be integers with

$$|a_{ik}|,|b_{kj}|\leqslant \alpha U+1. \tag{2.2}$$

Denote by a'_{ik} and b'_{kj} their least residues modulo p. Define

$$d_{ij}^{\sharp}\equiv 2^{\nu-1}\alpha+\sum_{k=1}^{m}a'_{ik}b'_{kj},\quad 0\leqslant d_{ij}^{\sharp}<p.$$

Let $q_{ij}^{\sharp}$, r_{ij} be the unique integers satisfying

$$d_{ij}^{\sharp}=\alpha q_{ij}^{\sharp}+r_{ij},\quad 0\leqslant r_{ij}<\alpha.$$

Choose any $q'_{ij}\in[0,p)$, and define q_{ij} to be the integer in $[-p/2,p/2)$ such that

$$q_{ij}\equiv q'_{ij} \bmod p. \tag{2.3}$$

If

$$q_{ij}^{\sharp}\equiv q'_{ij}+2^{\nu-1} \bmod p \quad \textit{and} \quad 0\leqslant q_{ij}^{\sharp}<2^{\nu}, \tag{2.4}$$

then

$$\sum_{k=1}^{m}a_{ik}b_{kj}=\alpha q_{ij}+r_{ij}.$$

Proof. Since $q_{ij}\equiv q'_{ij} \bmod p$ and $q_{ij}^{\sharp}\equiv q'_{ij}+2^{\nu-1} \bmod p$, we have

$$\alpha(q_{ij}+2^{\nu-1})+r_{ij}\equiv\alpha(q'_{ij}+2^{\nu-1})+r_{ij}\equiv\alpha q_{ij}^{\sharp}+r_{ij} \bmod p. \tag{2.5}$$

Since $\alpha q_{ij}^{\sharp}+r_{ij}=d_{ij}^{\sharp}$, since $d_{ij}^{\sharp}$ is the least residue of $2^{\nu-1}\alpha+\sum_{k=1}^{m}a'_{ik}b'_{kj}$ modulo p, and since a'_{ik} and b'_{kj} are the least residues of a_{ik} and b_{kj} modulo p, we have

$$\alpha q_{ij}^{\sharp}+r_{ij}\equiv d_{ij}^{\sharp}\equiv 2^{\nu-1}\alpha+\sum_{k=1}^{m}a'_{ik}b'_{kj}\equiv 2^{\nu-1}\alpha+\sum_{k=1}^{m}a_{ik}b_{kj} \bmod p. \tag{2.6}$$

Combining (2.5) with (2.6), we see that

$$\alpha(q_{ij}+2^{\nu-1})+r_{ij}\equiv 2^{\nu-1}\alpha+\sum_{k=1}^{m}a_{ik}b_{kj} \bmod p.$$

Expanding the left-hand side and canceling the term $2^{\nu-1}\alpha$ from both sides, this becomes

$$\alpha q_{ij}+r_{ij}\equiv\sum_{k=1}^{m}a_{ik}b_{kj} \bmod p.$$

Thus, there exists an integer t such that

$$\alpha q_{ij}+r_{ij}=tp+\sum_{k=1}^{m}a_{ik}b_{kj}.$$

We claim $t=0$. Suppose not; then $|t|\geqslant 1$. We split into two cases:

- Case $t \geqslant 1$. Then

$$\begin{aligned} \alpha q_{ij} + r_{ij} &\geqslant p + \sum_{k=1}^{m} a_{ik} b_{kj} \\ &\overset{(2.2)}{\geqslant} p - m(\alpha U + 1)^2 \\ &\overset{(2.1)}{>} p - \left(\frac{p}{2} - (\alpha - 1)\right). \end{aligned} \tag{2.7}$$

To see the last inequality, note that from (2.1) we have

$$m(\alpha U + 1)^2 + (\alpha - 1) < \frac{p}{2} \implies m(\alpha U + 1)^2 < \frac{p}{2} - (\alpha - 1) \implies -m(\alpha U + 1)^2 > -\left(\frac{p}{2} - (\alpha - 1)\right).$$

Now, since $r_{ij} \leqslant \alpha - 1$, we have

$$p - \left(\frac{p}{2} - (\alpha - 1)\right) = \frac{p}{2} + (\alpha - 1) \geqslant \frac{p}{2} + r_{ij}.$$

Putting this into (2.7) and canceling the r_{ij} term that now appears on both sides, we obtain

$$\alpha q_{ij} > \frac{p}{2} \overset{(2.1)}{>} 2^{\nu - 1}\alpha.$$

Since α is positive, this implies that $q_{ij} > 2^{\nu-1}$, and hence $q_{ij} + 2^{\nu-1} > 2^{\nu}$. As $q_{ij} \in [-p/2, p/2)$, we have

$$2^{\nu} < q_{ij} < \frac{p}{2}.$$

On the other hand,

$$q_{ij} + 2^{\nu-1} \equiv q'_{ij} + 2^{\nu-1} \equiv q^{\sharp}_{ij} \bmod p,$$

and $0 \leqslant q^{\sharp}_{ij} < 2^{\nu} < p$ by (2.4) and the upper bound in (2.1). Two integers that lie in $[0, p)$ and are equivalent mod p must be equal as integers. Thus,

$$q_{ij} + 2^{\nu-1} = q^{\sharp}_{ij}.$$

This is absurd, since $q_{ij} + 2^{\nu-1} > 2^{\nu}$ while $q^{\sharp}_{ij} < 2^{\nu}$.

- Case $t \leqslant -1$. Then

$$\begin{aligned} \alpha q_{ij} + r_{ij} &\leqslant -p + \sum_{k=1}^{m} a_{ik} b_{kj} \\ &\overset{(2.2)}{\leqslant} -p + m(\alpha U + 1)^2 \\ &\overset{(2.1)}{<} -p + 2^{\nu-1}\alpha - (\alpha - 1). \end{aligned} \tag{2.8}$$

Now, since $r_{ij} \geqslant 0$, we see from (2.8) that

$$\alpha q_{ij} < -p + 2^{\nu-1}\alpha - (\alpha - 1) \overset{(2.1)}{<} -2^{\nu}\alpha + 2^{\nu-1}\alpha - (\alpha - 1) = -2^{\nu-1}\alpha - (\alpha - 1).$$

Since α is positive, this implies that $q_{ij} < -2^{\nu-1}$, and hence $q_{ij} + 2^{\nu-1} < 0$. As $q_{ij} \in [-p/2, p/2)$, we have

$$-\frac{p}{2} \leqslant q_{ij} < -2^{\nu-1}.$$

On the other hand,

$$q_{ij} \equiv q'_{ij} \equiv q^{\sharp}_{ij} - 2^{\nu-1} \bmod p,$$

and $-2^{\nu-1} \leqslant q^{\sharp}_{ij} - 2^{\nu-1} < 2^{\nu-1} < p/2$ by (2.4) and the upper bound in (2.1). Two integers that lie in $[-p/2, p/2)$ and are equivalent mod p must be equal as integers. Thus,

$$q_{ij} = q^{\sharp}_{ij} - 2^{\nu-1}.$$

This is absurd, since $q_{ij} < -2^{\nu-1}$ while $q^{\sharp}_{ij} - 2^{\nu-1} \geqslant -2^{\nu-1}$.

□

2.2. Fixed-point multiplication. Suppose we have real numbers x, y, and z, and we fix a positive integer scaling factor α. In fixed-point arithmetic we represent these numbers by integers that approximate the scaled real values. For example, we choose integers

$$a, b, c \in \mathbb{Z}$$

that approximate αx, αy, and αz, respectively; that is,

$$a \approx \alpha x, \quad b \approx \alpha y, \quad c \approx \alpha z.$$

Then a, b, and c form the fixed-point representations of x, y, and z (with scaling α). One way to quantify the error is to introduce discrepancy terms:

$$\delta_x = \alpha x - a, \quad \delta_y = \alpha y - b, \quad \delta_z = \alpha z - c.$$

In many quantization schemes these errors are bounded by a constant. For example, if we use the floor function we have

$$0 \leqslant \delta_x, \delta_y, \delta_z < 1.$$

A key observation is that if

$$a \approx \alpha x \quad \text{and} \quad b \approx \alpha y,$$

then their product satisfies

$$ab \approx (\alpha x)(\alpha y) = \alpha^2 xy.$$

Likewise, since

$$c \approx \alpha z,$$

we have

$$\alpha c \approx \alpha^2 z.$$

Thus, if the fixed-point multiplication is consistent, then

$$\frac{ab}{\alpha} \approx c,$$

which in turn implies

$$xy \approx z.$$

This is the basic idea behind fixed-point multiplication: one computes the product ab (which is at scale α^2) and then "rescales" it by dividing by α to obtain an approximation for the fixed-point representation of xy.

Below are three standard examples of how one might define the fixed-point representations:

1. *Using the floor function:* Define
$$a = \lfloor \alpha x \rfloor, \quad b = \lfloor \alpha y \rfloor, \quad c = \lfloor \alpha z \rfloor.$$
Then the discrepancies are given by
$$\delta_x = \alpha x - \lfloor \alpha x \rfloor, \quad \delta_y = \alpha y - \lfloor \alpha y \rfloor, \quad \delta_z = \alpha z - \lfloor \alpha z \rfloor,$$
with
$$0 \leqslant \delta_x, \delta_y, \delta_z < 1.$$

2. *Using the ceiling function:* Define
$$a = \lceil \alpha x \rceil, \quad b = \lceil \alpha y \rceil, \quad c = \lceil \alpha z \rceil.$$
In this case we may write
$$a = \alpha x + \delta'_x, \quad b = \alpha y + \delta'_y, \quad c = \alpha z + \delta'_z,$$
where
$$-1 < \delta'_x, \delta'_y, \delta'_z \leqslant 0.$$

3. *Using rounding to the nearest integer:* Define the fixed-point representations of x, y, z as follows:
$$a = \mathrm{round}(\alpha x), \quad b = \mathrm{round}(\alpha y), \quad c = \mathrm{round}(\alpha z).$$
Then, setting
$$\delta''_x = \alpha x - a, \quad \delta''_y = \alpha y - b, \quad \delta''_z = \alpha z - c,$$
we have
$$-\frac{1}{2} \leqslant \delta''_x, \delta''_y, \delta''_z \leqslant \frac{1}{2}.$$

Proposition 2.2. *Let α be a positive integer, and let $X = [x_{ik}]$ and $Y = [y_{kj}]$ be real matrices of dimensions $\ell \times m$ and $m \times n$, respectively. Define*

$$Z = [z_{ij}] = XY.$$

Let $A = [a_{ik}]$, $B = [b_{kj}]$, and $C = [c_{ij}]$ be integer matrices of the same dimensions as X, Y, and Z, respectively. Assume that there exist real matrices $\Delta_X = [\delta_{x,ik}]$, $\Delta_Y = [\delta_{y,kj}]$, and $\Delta_Z = [\delta_{z,ij}]$, whose entries all lie in $[-1,1]$, such that

$$A = \alpha X - \Delta_X, \quad B = \alpha Y - \Delta_Y, \quad C = \alpha Z - \Delta_Z.$$

Consider the product $AB = [(AB)_{ij}]$. For each i, j, apply the division algorithm with divisor α to express

$$(AB)_{ij} = \alpha q_{ij} + r_{ij}, \quad 0 \leqslant r_{ij} < \alpha.$$

Thus, the matrix product can be rewritten as

$$AB = \alpha Q + R,$$

where $Q = [q_{ij}]$ and $R = [r_{ij}]$ are the quotient and remainder matrices, respectively. Then,

$$C = Q + E,$$

where $E = [\varepsilon_{ij}]$ is an error matrix such that, for all i, j,

$$|\varepsilon_{ij}| \leqslant 2 + \frac{m-1}{\alpha} + \sum_{k=1}^{m} \left(|x_{ik}| + |y_{kj}| \right).$$

Remark 2.3. In a typical scenario, we have[1] $m, x_{ik}, y_{kj}, z_{ij} \asymp 1$, and α is large. Thus,

$$\varepsilon_{ij} \ll \frac{m}{\alpha} + m \ll 1, \quad c_{ij} = \alpha z_{ij} - \delta_{z,ij} \gg \alpha, \quad \text{and} \quad q_{ij} = c_{ij} - \varepsilon_{ij} = c_{ij} \left(1 + O\left(\frac{1}{\alpha} \right) \right).$$

Furthermore,

$$q_{ij} = c_{ij} - \varepsilon_{ij} = \alpha z_{ij} - \delta_{z,ij} - \varepsilon_{ij} = \alpha z_{ij} + O(1) = \alpha z_{ij} \left(1 + O\left(\frac{1}{\alpha} \right) \right).$$

This implies that

$$z_{ij} = \frac{q_{ij}}{\alpha} \left(1 + O\left(\frac{1}{\alpha} \right) \right).$$

In some cases, cancellation effects may be such that the sum

$$z_{ij} = \sum_{k=1}^{m} x_{ik} y_{kj}$$

is significantly smaller than 1—in extreme cases, it could be as small as $O(1/\alpha)$. (For instance, if row i of X is orthogonal to column j of Y, then $z_{ij} = 0$.) However, such occurrences are expected to be rare. In practical applications such as quantization in neural networks, the network is typically robust enough to maintain performance even when some entries of the product matrix are unusually small. ■

Proof of Proposition 2.2. For each fixed pair of indices $i \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, n\}$, the (i, j)-entry of the true matrix product is given by

$$z_{ij} = \sum_{k=1}^{m} x_{ik} y_{kj}.$$

By the definition of C, we have

$$c_{ij} = \alpha z_{ij} - \delta_{z,ij} \quad \Longrightarrow \quad \alpha^2 z_{ij} = \alpha (c_{ij} + \delta_{z,ij}), \tag{2.9}$$

where $\delta_{z,ij} \in [-1, 1]$. Meanwhile, in fixed-point arithmetic, we compute

$$(AB)_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}.$$

Since

$$a_{ik} = \alpha x_{ik} - \delta_{x,ik} \quad \text{and} \quad b_{kj} = \alpha y_{kj} - \delta_{y,kj},$$

[1] The notation $f \asymp g$ denotes that $f \ll g$ and $g \ll f$, i.e., there exist absolute positive constants c_1 and c_2 such that $c_1 |f| < |g| < c_2 |f|$.

it follows that

$$a_{ik}b_{kj} = (\alpha x_{ik} - \delta_{x,ik})(\alpha y_{kj} - \delta_{y,kj}) = \alpha^2 x_{ik}y_{kj} - \alpha\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) + \delta_{x,ik}\delta_{y,kj}.$$

Summing over all k, we obtain

$$\begin{aligned}
(AB)_{ij} &= \sum_{k=1}^{m} a_{ik}b_{kj} \\
&= \sum_{k=1}^{m} \left\{\alpha^2 x_{ik}y_{kj} - \alpha\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) + \delta_{x,ik}\delta_{y,kj}\right\} \\
&= \alpha^2 \sum_{k=1}^{m} x_{ik}y_{kj} - \alpha \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) + \sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj} \\
&= \alpha^2 z_{ij} - \alpha \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) + \sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj} \\
&\overset{(2.9)}{=} \alpha\left(c_{ij} + \delta_{z,ij}\right) - \alpha \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) + \sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj} \\
&= \alpha c_{ij} + \alpha\left\{\delta_{z,ij} - \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right)\right\} + \sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj}.
\end{aligned} \tag{2.10}$$

On the other hand, we have integers q_{ij} and r_{ij}, $0 \leqslant r_{ij} < \alpha$, such that

$$(AB)_{ij} = \alpha q_{ij} + r_{ij}.$$

Substituting this into (2.10), we obtain

$$\alpha q_{ij} + r_{ij} = \alpha c_{ij} + \alpha\left\{\delta_{z,ij} - \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right)\right\} + \sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj}.$$

Rearranging and dividing by α gives

$$c_{ij} = q_{ij} + \frac{r_{ij}}{\alpha} - \delta_{z,ij} + \sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right) - \frac{1}{\alpha}\sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj}.$$

Since $\delta_{x,ik}, \delta_{y,kj}, \delta_{z,ij} \in [-1,1]$ and $0 \leqslant r_{ij} < \alpha$, we bound the error terms:

$$\left|\frac{r_{ij}}{\alpha} - \delta_{z,ij} - \frac{1}{\alpha}\sum_{k=1}^{m}\delta_{x,ik}\delta_{y,kj}\right| \leqslant \frac{\alpha-1}{\alpha} + 1 + \frac{m}{\alpha} = 2 + \frac{m-1}{\alpha}$$

and

$$\left|\sum_{k=1}^{m}\left(x_{ik}\delta_{y,kj} + y_{kj}\delta_{x,ik}\right)\right| \leqslant \sum_{k=1}^{m}\left(|x_{ik}| + |y_{kj}|\right).$$

Combining these, we conclude that

$$c_{ij} = q_{ij} + \varepsilon_{ij},$$

where

$$|\varepsilon_{ij}| \leqslant 2 + \frac{m-1}{\alpha} + \sum_{k=1}^{m}\left(|x_{ik}| + |y_{kj}|\right).$$

□

Example 2.4. Let

$$X = \begin{bmatrix} 1.11 & -3.23 \\ 4.0 & 2.56 \end{bmatrix} \quad \text{and} \quad Y = \begin{bmatrix} -2.12 & 0.9 \\ 3.324 & -1.15 \end{bmatrix}.$$

We simulate the computation of the product $Z = XY$ in fixed-point arithmetic using a scaling factor of $\alpha = 10$.

Each entry of X and Y is converted to fixed-point form by scaling it by $\alpha = 10$ and taking the integer part (floor). This yields the fixed-point matrices A and B:

$$A = \lfloor \alpha X \rfloor = \begin{bmatrix} 11 & -33 \\ 40 & 25 \end{bmatrix} \quad \text{and} \quad B = \lfloor \alpha Y \rfloor = \begin{bmatrix} -22 & 9 \\ 33 & -12 \end{bmatrix}.$$

Our goal is to approximate

$$C = \lfloor \alpha Z \rfloor, \quad \text{where} \quad Z = XY.$$

However, computing Z exactly and then applying the fixed-point transformation would defeat the purpose of fixed-point arithmetic. Instead, we must work exclusively with A and B.

The product of the fixed-point matrices is computed as

$$AB = \begin{bmatrix} 11 & -32 \\ 40 & 25 \end{bmatrix} \begin{bmatrix} -22 & 9 \\ 33 & -11 \end{bmatrix} = \begin{bmatrix} -1331 & 495 \\ -55 & 60 \end{bmatrix}.$$

We now apply the division algorithm with divisor $\alpha = 10$ to each entry of AB, yielding a matrix of quotients Q and a matrix of remainders R:

$$AB = \alpha Q + R = 10 \begin{bmatrix} -134 & 49 \\ -6 & 6 \end{bmatrix} + \begin{bmatrix} 9 & 5 \\ 5 & 0 \end{bmatrix}.$$

Thus,

$$\alpha^{-1} AB = Q + \alpha^{-1} R = \begin{bmatrix} -134 & 49 \\ -6 & 6 \end{bmatrix} + \alpha^{-1} R,$$

where the entries of $\alpha^{-1} R$ all lie in $[0, 1)$. We approximate the fixed-point representation C of Z as

$$C \approx Q = \begin{bmatrix} -134 & 49 \\ -6 & 6 \end{bmatrix}.$$

For comparison, we compute the exact product $Z = XY$ using standard arithmetic:

$$Z = \begin{bmatrix} -13.08972 & 4.7135 \\ 0.02944 & 0.656 \end{bmatrix}.$$

Applying the fixed-point transformation,

$$C = \begin{bmatrix} \lfloor -130.8972 \rfloor & \lfloor 47.135 \rfloor \\ \lfloor 0.2944 \rfloor & \lfloor 6.56 \rfloor \end{bmatrix} = \begin{bmatrix} -131 & 47 \\ 0 & 7 \end{bmatrix}.$$

The error in approximating C by Q is

$$E = C - Q = \begin{bmatrix} 3 & -2 \\ 6 & 1 \end{bmatrix}.$$

Finally, the error in approximating Z by $\alpha^{-1} Q$ is

$$Z - \alpha^{-1} Q = \begin{bmatrix} 0.31028 & -0.1865 \\ 0.62944 & 0.056 \end{bmatrix}.$$

■

3. The Code

The pseudocode and Rust code that follow are designed for implementation within the *ExpanderCompilerCollection* (ECC) framework [1]. This library provides a specialized interface for constructing and verifying arithmetic circuits.

```rust
fn div_unconstrained<C: Config, Builder: RootAPI<C>>(
    api: &mut Builder,
    d: Variable,
    alpha: u32,
) -> (Variable, Variable) {
    // Since d is nonnegative (a least residue mod p), we can directly compute
    // the quotient and remainder using the unconstrained division and modulo APIs.
    let q = api.unconstrained_int_div(d, alpha);
    let r = api.unconstrained_mod(d, alpha);
    (q, r)
}
```

Listing 1: Simplified unconstrained division algorithm using the ECC Rust API

Algorithm 3.1 `verify_quantized_product`: verify quantized matrix multiplication for one row of A' and one column of B'

Require: A row $a' = [a'_1, \ldots, a'_m]$ and B column $b' = [b'_1, \ldots, b'_m]$ (least-residue representations modulo p)
Require: Scaling factor $\alpha > 1$ (with $\alpha = 2^\eta$), and parameters ν and η
Ensure: Impose the constraint $q^\sharp \equiv q' + 2^{\nu-1} \bmod p$, where q' is the least residue of $q^\sharp - 2^{\nu-1}$

`shift` $\leftarrow 2^{\nu-1}$
`shifted_alpha` $\leftarrow$ `shift` $\times \alpha$
`sum` $\leftarrow 0$
for $k \leftarrow 1$ to m **do**
    `sum` $\leftarrow$ `sum` $+ a'_k \times b'_k$
end for
$d^\sharp \leftarrow$ `shifted_alpha` $+$ `sum` ▷ $d^\sharp \equiv 2^{\nu-1}\alpha + \sum_{k=1}^{m} a'_k b'_k \bmod p$
$(q^\sharp, r) \leftarrow$ `div_unconstrained`$(d^\sharp, \alpha)$ ▷ Compute $q^\sharp, r$ with $d^\sharp = \alpha q^\sharp + r$, $0 \leqslant r < \alpha$
Impose constraint: $d^\sharp \equiv \alpha q^\sharp + r \bmod p$
`range_check`$(q^\sharp, \nu)$ ▷ Enforce $0 \leqslant q^\sharp < 2^\nu$
`range_check`(r, η) ▷ Enforce $0 \leqslant r < 2^\eta$
$q' \leftarrow$ `least_residue`$(q^\sharp -$ `shift`$)$
Impose constraint: $q^\sharp \equiv q' +$ `shift` $\bmod p$

```rust
// Verify the quantized product for one row of A' and one column of B'.
// This function imposes constraints to verify that the following holds:
//   Let d_sharp = 2^(nu-1) * alpha + sum_k (a'_k * b'_k),
//   and let the division algorithm yield
//       d_sharp = alpha * q^sharp + r,   with 0 <= r < alpha,
//   where alpha = 2^(eta). Then, after range checking q^sharp (in [0, 2^(nu)))
//   and r (in [0, 2^(eta))), we compute q' as the least residue of
//       q^sharp - 2^(nu-1)
//   modulo p, and finally impose
//       d_sharp = alpha*q^sharp + r (mod p)
//   and
//       q^sharp = q' + 2^(nu-1) (mod p).
//
// Here, all inputs (A', B', etc.) are least-residue representations modulo p,
// and functions such as `range_check` and `least_residue` are assumed to be defined externally.
fn verify_quantized_product<C: Config, Builder: RootAPI<C>>(
    api: &mut Builder,
    a_row: &[Variable],  // a row of A' (least-residue representation)
    b_col: &[Variable],  // a column of B' (least-residue representation)
    alpha: u32,          // scaling factor, with alpha = 2^(eta)
    nu: usize,           // parameter for range checking q^sharp ([0, 2^(nu)))
    eta: usize,          // alpha = 2^(eta), hence r in [0, 2^(eta))
) {
    // Ensure nu > 0 (optional check)
    // Compute shift = 2^(nu - 1) using the left-shift operator.
    let one = api.constant(1);
    let shift_amount = api.constant((nu - 1) as u32);
    let shift = api.unconstrained_shift_l(one, shift_amount);

    // Compute shifted_alpha = shift * alpha.
    let alpha_var = api.constant(alpha);
    let shifted_alpha = api.unconstrained_mul(shift, alpha_var);

    // Compute the inner product: sum_k (a_row[k] * b_col[k]).
    let mut sum = api.constant(0);
    for (&a, &b) in a_row.iter().zip(b_col.iter()) {
        let prod = api.unconstrained_mul(a, b);
        sum = api.unconstrained_add(sum, prod);
    }

    // Compute d_sharp = shifted_alpha + sum.
    let d_sharp = api.unconstrained_add(shifted_alpha, sum);

    // Perform division: d_sharp = alpha * q^sharp + r, with 0 <= r < alpha.
    let (q_sharp, r) = div_unconstrained(api, d_sharp, alpha);

    // Impose the constraint: d_sharp = alpha * q^sharp + r (mod p).
    let rhs = api.unconstrained_add(api.unconstrained_mul(alpha_var, q_sharp), r);
    api.assert_is_equal(d_sharp, rhs);

    // Range check q^sharp: ensure it lies in [0, 2^(nu)).
    range_check(api, q_sharp, nu);
```

```
    // Range check r: since alpha = 2^(eta), enforce 0 <= r < 2^(eta).
    range_check(api, r, eta);

    // Compute q' as the least residue of (q_sharp - shift) modulo p.
    let diff = api.unconstrained_sub(q_sharp, shift);
    let q_prime = least_residue(api, diff);

    // Impose the final constraint: q_sharp = q' + shift (mod p).
    let sum_for_q_sharp = api.unconstrained_add(q_prime, shift);
    api.assert_is_equal(q_sharp, sum_for_q_sharp);
}
```

Listing 2: Quantized matrix multiplication verification using the ECC Rust API

4. Example

Let

$$p = 521, \quad \alpha = 2^3, \quad U = 1, \quad m = 2, \quad \nu = 6.$$

Note that

$$m(\alpha U + 1)^2 + (\alpha - 1) = 169, \quad 2^{\nu-1}\alpha = 256, \quad \frac{p}{2} = 260.5,$$

and so

$$m(\alpha U + 1)^2 + (\alpha - 1) \leqslant 2^{\nu-1}\alpha < \frac{p}{2}.$$

Consider

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

Note that each entry of A and B lies in

$$[-(\alpha U + 1), \alpha U + 1] = [-9, 9].$$

Taking least residues modulo $p = 521$ of each entry yields

$$A' = \begin{bmatrix} 2 & 518 \\ 520 & 4 \end{bmatrix}, \quad B' = \begin{bmatrix} 520 & 2 \\ 3 & 519 \end{bmatrix}.$$

We compute $A'B'$, add $2^{\nu-1}\alpha = 256$ to each entry, and take least residues modulo $p = 521$. Then, we divide each entry of the resulting matrix by $\alpha = 2^3$, obtaining quotients $Q^\sharp$ and remainders R:

$$\begin{aligned} D^\sharp &= \text{least residue modulo } p = 521 \text{ of } \begin{bmatrix} 2 & 518 \\ 520 & 4 \end{bmatrix}\begin{bmatrix} 520 & 2 \\ 3 & 519 \end{bmatrix} + \begin{bmatrix} 256 & 256 \\ 256 & 256 \end{bmatrix} \\ &= \begin{bmatrix} 245 & 266 \\ 269 & 246 \end{bmatrix} \\ &= 8\begin{bmatrix} 30 & 33 \\ 33 & 30 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 5 & 6 \end{bmatrix} \\ &= \alpha Q^\sharp + R. \end{aligned}$$

We impose a constraint to verify this:

$$\begin{bmatrix} 245 & 266 \\ 269 & 246 \end{bmatrix} \equiv 8\begin{bmatrix} 30 & 33 \\ 33 & 30 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 5 & 6 \end{bmatrix} \bmod p.$$

With a range check, we verify that each entry of $Q^\sharp$ lies in $[0, 2^\nu) = [0, 64)$, and that each entry in R lies in $[0, \alpha) = [0, 7)$.

We subtract $2^{\nu-1} = 32$ from each entry of $Q^\sharp$ and take least residues modulo $p = 521$ to obtain

$$Q' = \text{least residue modulo } p = 521 \text{ of } \begin{bmatrix} 30 & 33 \\ 33 & 30 \end{bmatrix} - \begin{bmatrix} 32 & 32 \\ 32 & 32 \end{bmatrix} = \begin{bmatrix} 519 & 1 \\ 1 & 519 \end{bmatrix}.$$

We impose a constraint to verify that $Q^\sharp \equiv Q' + [2^{\nu-1}] \bmod p$:

$$\begin{bmatrix} 30 & 33 \\ 33 & 30 \end{bmatrix} \equiv \begin{bmatrix} 519 & 1 \\ 1 & 519 \end{bmatrix} + \begin{bmatrix} 32 & 32 \\ 32 & 32 \end{bmatrix} \bmod 521.$$

Assuming that each entry of Q' is the least residue modulo $p = 521$ of some integer in $[-p/2, p/2) = [-260.5, 260.5)$, we conclude that Q' represents

$$Q = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}.$$

For illustrative purposes, we check this calculation using regular off-circuit integer arithmetic:

$$AB = \begin{bmatrix} 2 & -3 \\ -1 & 4 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} -11 & 10 \\ 13 & -10 \end{bmatrix} = 8 \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 5 & 6 \end{bmatrix} = \alpha Q + R.$$

REFERENCES

[1] Polyhedra Network. *ExpanderCompilerCollection*. GitHub repository. Accessed January 28, 2025.