Transformer Inference using MAD vs LUT Kernels

A Comparative Benchmark of MAD and LUT Kernels for Binary and Ternary Dot Products on CPU and Edge Platforms

Mustafa Batu Eren[1]

Supervisor(s): Prof. Qing Wang[1]**, Braden Refalo**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Mustafa Batu Eren
Final project course: CSE3000 Research Project
Thesis committee: Prof. Qing Wang, Braden Refalo, Prof. Julia Olkhovskaia

Abstract

Quantizing Transformer weights to binary or ternary values reduces the inner product to sign manipulation and zero masking, prompting two competing CPU kernel strategies: multiply-add (MAD) and table lookup (LUT). Prior work reports end-to-end speedups but confounds the comparison across data layout, quantization format, and table depth simultaneously. This thesis isolates the trade-off by sweeping LUT depth as a single controlled variable, spanning matrix sizes across the cache hierarchy and attributing results through roofline analysis on an x86 platform with AVX2 and roofline plus hardware-counter analysis on an ARM edge platform with NEON. The LUT advantage proves conditional: binary throughput rises monotonically with depth to 104.4 GOPS, roughly 2.6 times the strongest MAD baseline, while ternary gains are narrower and erode once the table outgrows fast cache or forces a gather. Throughout, instruction throughput, not bandwidth, is the binding limit.

1 Introduction

Transformers have become the dominant architecture for large language models, in large part because their attention mechanism removes the sequential dependency of earlier recurrent models and therefore parallelizes well on modern hardware. That same scalability, however, has driven models to grow rapidly in size, and so has their demand for compute and energy. Running these models efficiently is no longer just an optimization concern but a practical necessity, making it more important than ever to utilize available compute units to their absolute maximum.

Consequently, researchers are seeking cheaper ways to run these models, and quantization offers a highly effective one by representing weights with fewer bits. BitNet [1], a 1-bit Transformer trained from scratch, achieves competitive language modeling performance while substantially reducing memory footprint and energy consumption compared to 8-bit and FP16 baselines. A follow-up, BitNet b1.58 [2], extended this to ternary weights $\{-1, 0, +1\}$, encoding approximately 1.58 bits per parameter, matching full-precision Transformers in perplexity and downstream performance while being significantly more efficient in latency, memory, and throughput. This line of work explicitly calls for kernel-level optimizations tailored to this new compute paradigm.

Recent work such as [3] and [4] has proposed lookup table (LUT)-based kernels for low-bit CPU inference and demonstrated strong speedups over full-precision baselines. However, neither systematically isolates the multiply-add (MAD) vs LUT trade-off across varying vector sizes and cache budgets as a controlled benchmark. This is because their goal was to demonstrate end-to-end inference speedups rather than to attribute them: in each case the LUT and MAD kernels differ simultaneously along several axes, such as data layout, cache behavior, and table depth, so no single factor can be isolated as the source of any observed gap. Optimal kernel selection across the full hardware spectrum, from server-class machines down to edge platforms with constrained cache, narrower SIMD registers, and limited memory bandwidth, remains an open problem.

This project addresses that gap by providing a controlled benchmark of MAD and LUT kernels for binary and ternary dot products on CPU, with an evaluation on edge platforms. The main research question is: to what extent do LUT-based dot-product kernels provide a performance advantage over MAD-based kernels for binary and ternary Transformer inference across different hardware resource constraints? This is broken down into four sub-questions:

1. How do MAD and LUT kernels compare in throughput across varying vector sizes for binary and ternary dot products?
2. What are the memory footprint and cache behavior differences between the two approaches, and how do these change with vector size?
3. What is the compute-memory trade-off profile of each kernel, and at what point does the LUT memory overhead outweigh its compute savings?
4. Are either or both approaches feasible on edge platforms given their cache and SIMD constraints?

The expected outcome is a quantitative trade-off profile that characterizes under which conditions, such as vector size or cache budget, each approach is preferable.

2 Related Work

Extreme weight quantization. Post-training quantization methods such as GPTQ [5] and AWQ [6] compress LLM weights to 4 bits with minimal accuracy loss, but degrade sharply below 3 bits. Quantization-aware training pushes further: BitNet [1] trains binary-weight Transformers from scratch, and BitNet b1.58 [2] extends this to ternary weights $\{-1, 0, +1\}$, matching FP16 baselines in perplexity and downstream accuracy from 3B parameters onward. Crucially, these works change the compute primitive: with binary or ternary weights, the inner-product multiplication degenerates to sign manipulation and zero-masking, and both papers explicitly call for kernels tailored to this regime. They do not, however, provide CPU kernels themselves, which motivates the systems work below.

LUT-based low-bit kernels. Replacing multiply-accumulate with table lookup predates LLMs: DeepGEMM [7] precomputes products of low-bit weights and activations for CNN inference on CPUs. For LLMs, LUT-GEMM [8] pioneered LUT-based weight-only-quantized GEMM on GPUs. Arguing that GPUs lack the fast in-register table access this approach favors, T-MAC [4] brought it to CPUs: it decomposes n-bit weights into n one-bit matrices, performs lookups via the 16-entry byte-shuffle instructions (`pshufb`, `tbl`) at a fixed group size of $g = 4$ bits, and introduces mirror consolidation and table quantization to shrink tables, reporting up to $4\times$ end-to-end speedup over llama.cpp. Bitnet.cpp [3] specializes this to ternary weights with element-wise tables (3^g entries at $g \in \{2, 3\}$ rather than

2^g per bit-plane), reaching 1.67 bits per weight and lossless BitNet b1.58 inference.

MAD-based ternary kernels. The competing line keeps the multiply-add structure. llama.cpp's `TQ1_0`/`TQ2_0` kernels [9] exploit ternary structure within element-wise MAD computation, and Litespark [10] feeds int8-stored ternary weights directly into native dot-product instructions (NEON `SDOT`, AVX-512 VNNI), reporting throughput competitive with Bitnet.cpp's LUT kernels. The literature thus disagrees on when lookup beats multiply-add, and the answer evidently depends on hardware details that existing evaluations do not isolate.

Performance characterization. The roofline model [11] relates achieved throughput to operational intensity and bandwidth ceilings, and per-instruction throughput data from uops.info [12] and the Cortex-A72 optimization guide [13] enable analytical performance prediction at the instruction level. Prior LUT kernel papers report end-to-end token throughput but do not place their kernels in this framework.

Knowledge Gap. T-MAC and Bitnet.cpp do include MAD baselines, but the comparison is confounded: their LUT and MAD kernels differ simultaneously in data layout, quantization format, table depth, and software stack, and results are reported as end-to-end token throughput in which non-GEMM costs participate. Litespark compares against Bitnet.cpp across frameworks, inheriting the same confound. No prior work sweeps the LUT depth g as a controlled variable, holds the surrounding algorithmic structure and benchmarking harness fixed between MAD and LUT families, spans matrix sizes across the cache hierarchy, or attributes the resulting bottlenecks via roofline and hardware-counter analysis. This thesis provides exactly that controlled, kernel-level characterization for binary and ternary dot products on x86 and ARM.

3 Methodology

This work investigates two families of low-precision integer dot-product kernels, namely multiply-accumulate (MAD) kernels and lookup-table (LUT) kernels, across both binary ($W \in \{-1, +1\}$) and ternary ($W \in \{-1, 0, +1\}$) weight quantization. The central operation is matrix-vector multiplication: output$[i] = \langle W[i, :], \mathbf{a} \rangle$ for all output rows i, with activations $\mathbf{a}$ stored as `int8_t`. The central research variable is the *LUT depth* g, defined as the number of consecutive weight elements jointly addressed by a single table lookup. Increasing g reduces the number of memory accesses per dot product but grows the table size exponentially, creating a fundamental compute-versus-memory trade-off that this work characterizes empirically across the host cache hierarchy.

3.1 Kernel Families and Implementation

MAD kernels unpack bit-packed weights at runtime and accumulate products using 256-bit AVX2 SIMD sign-manipulation instructions. Ternary weights are packed at 2 bits per weight (4 weights per byte, encoded as $\{0, 1, 2\} \to \{-1, 0, +1\}$), while binary weights use 1 bit per weight (8 weights per byte, MSB-first). Each MAD kernel processes 32 weights per loop iteration; its operational intensity is derived in Section 4.2. We evaluate two ternary MAD kernels: a plain MAD baseline of our own, and a *MAD BitNet* kernel adapted from bitnet.cpp's `i2_s` GEMV (`ggml_vec_dot_i2_i8_s`) [3], which reuses its stride-packed weight layout and accumulation pattern and adds the sign conversion that bitnet.cpp folds into a separate bias step. The binary MAD kernels are our own extension of the same stride-packed structure, since bitnet.cpp ships no binary kernels.

LUT kernels precompute all K^g partial dot-product values for each group of g weights (where $K = 3$ for ternary, $K = 2$ for binary) into a compact table, replacing runtime multiply-accumulate operations with a single indexed lookup per group. The primary lookup mechanism is `_mm256_shuffle_epi8` (pshufb) [4], which performs a parallel 16-lane byte lookup within each 128-bit half of a 256-bit register. On Skylake microarchitecture, and equivalently on the Comet Lake core of the benchmark platform, `VPSHUFB` achieves a reciprocal throughput of 1 cycle on port p5, where a port is one of the dispatch slots through which the scheduler issues micro-operations to the execution units, sustaining one 256-bit shuffle per clock cycle [12]. This instruction constrains each half-register table to at most 16 entries, and a central design challenge at higher depths is mapping larger weight alphabets onto this hardware limit. By contrast, the vector gather instruction `VPGATHERDD` used in the depth-3 and depth-4 gather variants carries a reciprocal throughput of 5 cycles on the same microarchitecture [12], providing a direct hardware motivation for the shuffle-based decompositions developed at each depth. Table 1 summarises the throughput characteristics of the key AVX2 instructions across the kernel families.

The total LUT memory footprint scales as:

$$\text{Footprint} \approx \left\lceil \frac{n}{g} \right\rceil \cdot K^g \cdot b \quad \text{bytes} \tag{1}$$

where b is the bytes per entry (4 for `int32` gather kernels; 2 for `int16` lo/hi split pshufb kernels). Representative values at $n = 4096$ range from 36 KiB at depth 2 to approximately 324 KiB for the depth-4 ternary gather kernel.

3.2 LUT Depth Progression and Algorithmic Variants

A family of AVX2 LUT variants is developed and benchmarked at depths $g \in \{2, 3, 4\}$, each addressing the over-16-entry problem through a different strategy. Two of these variants reuse established designs: the depth-2 kernel and the mirror-consolidated depth-3 kernel reimplement, respectively, the TL1 and TL2 LUT kernels of bitnet.cpp [3], themselves instances of the table-lookup mpGEMM method introduced by T-MAC [4]. Building on this lineage, we contribute a gather-based depth-3 variant, a depth-4 variant, and binary-weight variants at every depth, none of which bitnet.cpp provides.

Depth 2 (pshufb): One byte encodes two independent weight pairs. Each pair has at most $3^2 = 9$ ternary or $2^2 = 4$ binary combinations, well within the 16-entry pshufb limit. Partial sums are stored in a split `int16` lo/hi byte layout, and two pshufb lookups per vector lane reconstruct the result for

Table 1: Reciprocal throughput and latency of key AVX2 instructions on Intel Skylake microarchitecture [12]. The benchmark platform, an Intel Core i7-10870H, uses the Comet Lake microarchitecture, which shares the same core pipeline as Skylake; these figures therefore apply directly. Lower reciprocal throughput (Rec. TP) indicates higher sustained execution rate.

Instruction	Operation	Rec. TP (cycles)	Latency (cycles)	Port(s)
`VPSHUFB YMM`	Byte shuffle (256-bit)	1.0	1	p5
`VPMADDUBSW YMM`	u8×s8 mul-add (256-bit)	0.5	5	p0+p1
`VPADDD YMM`	Int32 add (256-bit)	0.33	1	p015
`VPGATHERDD YMM`	Int32 gather (256-bit)	5.0	22	p0+p015+8*p23+p5

both pairs simultaneously. The ternary depth-2 kernel follows the LUT mpGEMM scheme of T-MAC [4] as realized in bitnet.cpp's TL1 kernel [3], adapted here to AVX2; the binary depth-2 kernel is our own, as bitnet.cpp defines no binary path.

Depth-3 variants: Two ternary strategies are compared. The *D3a* kernel is our own gather-based design, storing all 27 partial sums as `int32` values and retrieving them with `vpgatherdd`. The *D3b* variant is a faithful reimplementation of bitnet.cpp's TL2 LUT kernel [3], itself an instance of the T-MAC table-lookup method [4], generalized here from bitnet.cpp's machine-generated per-layer tile sizes to arbitrary m and n. It reduces all 27 combinations to 14 canonical entries by observing that negating all three weights simply negates the partial sum, so only the canonical half need be stored to fit a single pshufb table. The remaining 13 sign-mirror combinations are recovered via a branchless conditional two's-complement negate, the exact form of which, together with the lane pre-duplication used to load the consolidated table, is given in Appendix B.

Depth-4 variant: A gather kernel enumerates all $3^4 = 81$ ternary partial sums as `int32` values (approximately 324 KiB at $n = 4096$). Mirror consolidation, which reduces depth-3 entries from 27 to 14, cannot be applied at depth 4: negating all four weights still halves the 81 combinations, but the resulting 41 canonical entries still far exceed the 16-entry limit of `pshufb`. Consequently, no pshufb-based variant is developed at this depth.

All pshufb-based LUT kernels store weights in a column-major 32-row tiled layout for contiguous vectorized access, and periodically flush their `int16` partial sums to `int32` accumulators to prevent overflow (Appendix B). An additional set of scalar template kernels at depths $g \in \{1, \ldots, 4\}$ is included to characterize exponential table growth against execution latency before committing to SIMD implementations. Correctness of all kernel variants is validated against a naive scalar reference using the Google Test framework.

4 Theoretical Analysis

Before evaluating measured performance, we establish the analytical ceilings that govern LUT kernel scalability. We proceed in two stages: an operation-memory trade-off analysis that characterizes when deeper tables help or hurt and derives the resulting cache-tier constraints, and an operational intensity analysis that frames where the compute-versus-memory balance shifts for each kernel variant.

4.1 Operation-Memory Trade-off

A LUT kernel of depth g involves a trade-off between table size and operation count. By grouping g consecutive weights per lookup, the number of lookup operations required for a dot product of length N scales down linearly to N/g.

$$\mathrm{Ops}(g) = \left\lceil \frac{N}{g} \right\rceil .$$

The ceiling is necessary because a partial final group of $N \bmod g$ elements still requires one lookup, with trailing weights zero-padded. At the same time, the number of entries that must be precomputed for each group is K^g, where K is the weight alphabet size, an exponential increase in g. The total LUT memory footprint (Equation 1), denoted $F(g)$ for brevity, is therefore the product of one linearly falling and one exponentially growing factor.

Comparing successive footprints (Appendix C) settles which trend dominates: for any fixed N and K, the footprint $F(g)$ is monotonically non-decreasing in g for $g \geq 1$, strictly so for ternary weights ($K = 3$) and from $g \geq 2$ onward for binary weights ($K = 2$). The single exception is the binary $g = 1 \to g = 2$ step, where the successive-footprint ratio is exactly 1: it costs no additional memory while halving the operation count, making depth-2 binary LUT a Pareto improvement over depth-1. The linear reduction in operation count can therefore never reverse the exponential growth of the table. This defines a hard constraint: a table at depth g fits within a cache of capacity C if and only if $F(g) \leq C$, and the maximum cache-compatible depth is

$$g^*(K, N, b, C) = \max \left\{ g \geq 1 : \left\lceil \frac{N}{g} \right\rceil \cdot K^g \cdot b \leq C \right\}. \quad (2)$$

Evaluating Equation 2 at $N = 4096$, $b = 2$ (pshufb int16 kernels), $C_{\mathrm{L1}} = 32\,\mathrm{KB}$, and $C_{\mathrm{L2}} = 256\,\mathrm{KB}$ gives the depths shown in Table 2. For ternary kernels, $g^*_{\mathrm{L1}} = 1$: even the shallowest non-trivial ternary LUT ($g = 2$, 36 KiB) overflows L1. For binary kernels, $g^*_{\mathrm{L1}} = 4$: depth-4 binary LUT lands exactly on the L1 boundary (32 KiB), and depth-5 would overflow it. These bounds motivate the architectural choices in each kernel family and explain why ternary pshufb variants at $g \geq 2$ operate from L2 by construction.

4.2 Operational Intensity

Operational intensity (OI) [11] measures operational density per unit of memory traffic, defined as

Table 2: LUT memory footprint at $N = 4096$, with $b = 2$ bytes per entry for pshufb-compatible (int16) kernels and $b = 4$ bytes for gather (int32) kernels. Binary: $K = 2$; Ternary: $K = 3$. Cache tier indicates where the working set resides on a typical desktop CPU with 32 KB L1-D and 256 KB L2.

Scheme	**Depth** g	**Groups** $\lceil N/g \rceil$	**Entries** K^g	b **(bytes)**	**Footprint**
Binary	2	2048	4	2	16 KiB (L1)
Binary	3	1366	8	2	21 KiB (L1)
Binary	4	1024	16	2	32 KiB (L1 boundary)
Ternary	2	2048	9	2	36 KiB (L2)
Ternary	3	1366	27	4	144 KiB (L2)
Ternary	3	1366	14	2	37 KiB (L2, mirror-consolidated)
Ternary	4	1024	81	4	324 KiB (L3)

$$\text{OI} = \frac{\text{floating-point or integer operations}}{\text{bytes streamed from memory}} \quad \text{(ops/byte)}.$$

A higher OI indicates that each byte fetched from the memory hierarchy is amortized over more arithmetic, making the kernel more tolerant of bandwidth limitations. We derive OI analytically for the MAD baselines, then describe how LUT kernels structurally alter this quantity.

MAD Kernel Operational Intensity

The AVX2 MAD kernels process 32 weight-activation pairs per loop iteration using `VPMADDUBSW` followed by `VPADDD`. Each weight-activation pair contributes one multiply and one accumulate, so 32 pairs yield 64 integer operations. The memory traffic per iteration consists of the packed activation bytes and the packed weight bytes.

Ternary MAD. Ternary weights are packed at 2 bits per element, so 32 weights occupy $32/4 = 8$ bytes. The 32 corresponding activations are stored as `int8_t` values and occupy 32 bytes. The total streamed bytes per iteration are $8 + 32 = 40$ bytes. The operational intensity is therefore

$$\text{OI}_{\text{tern-MAD}} = \frac{64 \text{ ops}}{40 \text{ bytes}} = 1.6 \text{ ops/byte}. \tag{3}$$

Binary MAD. Binary weights pack at 1 bit per element, so the same 32 weights occupy only $32/8 = 4$ bytes. The operation count and the 32-byte activation stream are unchanged, so the identical accounting gives $\text{OI}_{\text{bin-MAD}} = 64/36 \approx 1.78$ ops/byte, slightly higher than the ternary value only because the denser weight packing shrinks the denominator without touching the numerator.

LUT Kernel Operational Intensity and its Limits

LUT kernels restructure the memory access pattern. Rather than streaming raw weight bytes and performing runtime unpacking and sign manipulation, they stream a precomputed index vector, then use a single `VPSHUFB`/`vqtbl` or `VPGATHERDD` to retrieve the corresponding partial sum. Accounting for the traffic of one such lookup at the tile level (Appendix D) yields a *streaming* operational intensity, the quantity plotted on both rooflines, of

$$\text{OI}_{\text{LUT}} = \frac{64D}{32 + L} \quad \text{(ops/byte)}, \tag{4}$$

where D is the number of activation positions encoded per weight byte, not the LUT depth g (the two coincide only for the byte-aligned ternary depth-3 and depth-4 kernels; the nibble-split variants reach $D = 2g$), and L is the table footprint re-streamed from cache per pair-byte, not the full allocated table. A kernel that reads only part of its table's cache lines, or that gathers its table irregularly, is charged for what actually moves: the mirror-consolidated depth-3 kernel loads only 32 table bytes per pair-byte, but because its constructor keeps duplicate AVX2-compatible halves on the same cache lines, it re-streams the full 64-byte footprint and therefore sits at the same intensity as the direct depth-3 kernel. In the idealized limit where the table is resident and free ($L \to 0$), the K^g partial sums implicit in each entry are charged to a footprint that is g times smaller than the raw weight stream, so intensity would grow as $2D$ with depth. Table 3 evaluates Equation 4 for every LUT kernel on the two rooflines. Once the re-streamed table is counted, this ideal growth does not survive for the ternary kernels: depth-2 is the most intense ternary variant on both platforms, and the depth-3 and depth-4 gather kernels fall back toward or below the ternary MAD baseline (1.6). Intensity rises with depth only for the binary kernels, whose nibble-packing ($D = 6, 8$) shrinks the weight footprint faster than the table grows.

Table 3: Streaming operational intensity (OI, in ops/byte) of the LUT kernels anchoring the AVX2 and NEON rooflines, from Equation 4. D is the activation positions encoded per weight byte; L is the table bytes re-streamed per pair-byte. Binary LUT kernels are plotted on the AVX2 roofline only.

Variant	**Platform**	D	L (B)	OI
LUT D2	AVX2, NEON	4	64	2.67
LUT D3 mirror	AVX2, NEON	3	64	2.00
LUT D3 direct	NEON	3	64	2.00
LUT D3 gather	AVX2	3	108	1.37
LUT D4 gather	NEON	4	128	1.60
LUT D4 gather	AVX2	4	324	0.72
LUT D3 (binary)	AVX2	6	64	4.00
LUT D4 (binary)	AVX2	8	64	5.33

The two depth-4 gather rows differ by platform because the NEON scalar gather fetches only the 32 `int32` entries it indexes (128 bytes per pair-byte), whereas on AVX2

the full 324-byte table is counted as resident traffic for the VPGATHERDD path; in both cases the irregular access pattern, not the nominal depth, sets the intensity.

However, this intensity gain is realized only when the lookup instruction is genuinely single-cycle. When the working set of the table exceeds L1 capacity, or when the hardware provides only a gather instruction rather than a shuffle, the effective cost of each lookup rises sharply. Cache-tier migration is not merely a storage concern; it is the mechanism through which theoretical operation-count reductions collapse into actual slowdowns. Once the working set no longer fits in L1, each lookup incurs a longer-latency access, and the operational savings from grouping g weights together are partially or fully consumed by the increased memory latency. The same penalty governs the gather variants: as established in Section 3 (Table 1), VPGATHERDD's reciprocal throughput is five times that of VPSHUFB, so a kernel that would theoretically benefit from increased operational density at depth 4 may in practice run slower than a depth-2 pshufb kernel. Operational intensity in this case becomes a misleading predictor of performance: the kernel is nominally compute-dense, but the bottleneck has shifted to instruction latency rather than memory bandwidth. The algorithmic design choices discussed in Section 3, in particular the mirror-consolidated D3b variant, are motivated precisely by keeping the working set within pshufb-addressable bounds at each depth, so that the theoretical intensity gains remain achievable on real hardware.

5 Experimental Evaluations

This section describes the benchmarking environment and evaluation protocol, then presents the measured results across both hardware platforms. We evaluate all kernel variants under two regimes, KernelOnly and Full, sweeping matvec dimensions chosen to span the cache hierarchy, and report throughput, memory footprint, cache behavior, and roofline positioning for each variant.

5.1 Setup and Benchmarking Protocol

Execution Regimes

LUT kernels incur a one-time setup cost to construct the partial-sum table from the current activation vector. When the same table is reused across a batch of weight rows this cost is amortized, but when a new table must be built per inference call it can dominate for small matrices. To expose this behavior, each kernel is evaluated under two regimes:

- **KernelOnly:** Only the matvec inference kernel is timed. LUT construction and weight packing are performed once outside the timed benchmark loop. This isolates the pure compute and memory performance of the matrix-vector math, representing the scenario where LUT construction is fully amortized across the M rows of a large matrix, or across a batch of inferences.
- **Full:** LUT construction is included inside the timed loop alongside the inference kernel. Because benchmarking requires looping the operation to get stable timings, this regime forces the LUT to be reconstructed on every iteration. This models the strict single-vector inference case (e.g., autoregressive token generation), capturing the unamortized overhead of table construction that occurs on a per-inference basis.

Hardware Platforms

The primary benchmarking platform is an Intel Core i7-10870H (Comet Lake, 10th generation) desktop-class CPU with AVX2 support. The secondary platform is a Raspberry Pi built around an ARM Cortex-A72 application processor with NEON SIMD, which provides a contrasting architecture with a significantly smaller per-core cache, narrower SIMD registers, and reduced memory bandwidth. The full hardware specifications of both platforms, together with the compiler flags, frequency-pinning, and thread-affinity settings used to isolate the measurements, are given in Appendix A, and every benchmark in this section uses that configuration.

Metrics and Roofline Analysis

Throughput is measured in operations per second and converted to GOPS using the standard convention of 1 MAC = 2 operations. We explicitly note that, because weights are binary ($w \in \{-1, +1\}$) or ternary ($w \in \{-1, 0, +1\}$), the multiplication reduces structurally to sign manipulation and zero-masking rather than true integer multiply; the 1 MAC = 2 ops convention is retained to allow direct comparison with established quantized-inference benchmarks. The memory footprint of each LUT kernel is tracked as the static capacity of the precomputed table as a function of depth g and activation length n. On the ARM edge platform, empirical L1 and L2 data cache miss rates are captured via hardware performance counters using the Linux `perf stat` subsystem. No equivalent counter-based cache measurement is performed on the Intel desktop platform; the cache-tier behavior reported for the AVX2 kernels is instead characterized inferentially, by relating the measured throughput to the analytical table-footprint model (Equation 1, Table 2) and the documented host cache geometry, as detailed in Section 7.2.

Performance boundaries are evaluated using an analytical Roofline Model [11], which plots empirical performance against operational intensity to expose the binding hardware bottleneck, to classify each kernel as memory-bound or compute-bound relative to the hardware ceilings of the target platform. The theoretical peak compute ceiling is derived from the core clock frequency, AVX2 register width, integer execution port constraints, and the throughput of the VPMADDUBSW instruction, which multiplies 32 pairs of unsigned/signed 8-bit integers and accumulates adjacent pairs into 16 16-bit results per instruction. Cache bandwidth ceilings are calculated from the hardware read-port widths and cycle constraints of each cache tier. The operational DRAM bandwidth ceiling is established empirically under quiescent OS conditions using the Intel Memory Latency Checker (MLC). Each kernel is then plotted in the Roofline coordinate space (empirical GOPS against theoretical operational intensity) to identify the binding bottleneck and quantify proximity to the hardware limits.

The benchmark suite sweeps square matvec dimensions $N \in \{1024, 2048, 4096\}$, chosen to span the cache hierarchy. Smaller dimensions fit within per-core L2 (256 KB), while $N = 4096$ pushes data well beyond L2 and into the

shared L3 and DRAM. A separate procedural benchmark sweeps the scalar template kernels at depths $g \in \{1, \ldots, 6\}$ to isolate LUT construction time from kernel execution time. All synthetic weights and activations are generated with `std::mt19937` and fixed seeds to ensure reproducibility.

5.2 AVX2 Throughput Results

Figure 1 reports KernelOnly throughput on the Intel i7-10870H for the AVX2 kernel set, separately for ternary and binary weights. Tables I.1 and I.2 in Appendix I give the corresponding throughput in GOPS for both the KernelOnly and Full regimes. The MAD kernels have no LUT-construction stage, so a single value is reported for them in both tables.

Ternary Kernels

As shown in Figure 1 and Table I.1, LUT D2 is the fastest ternary kernel in the KernelOnly regime across all three matrix sizes, outperforming the plain MAD baseline by roughly $4\times$ and the MAD BitNet kernel by a smaller but consistent margin. LUT D3 Mirror is the second-fastest ternary kernel at $N = 1024$, but its throughput stays roughly flat across sizes while MAD BitNet improves with N, so MAD BitNet draws level at $N = 2048$ and overtakes it at $N = 4096$.

The two gather-based kernels, LUT D3 Gather and LUT D4 Gather, perform similarly to plain MAD across all sizes (Table I.1), despite their substantially higher nominal operational intensity (Section 3). Mirror consolidation alone, which converts the depth-3 kernel from a `vpgatherdd`-based 27-entry `int32` lookup to a two-`pshufb` 14-entry lookup at the same depth, yields a 3.4 to 3.5 times speedup (LUT D3 Gather versus LUT D3 Mirror). This confirms the prediction in Section 3 that the `VPGATHERDD` reciprocal-throughput penalty (Table 1), not the table size itself, is the dominant bottleneck at depth 3.

LUT D4 Gather shows a clearer cache-tier effect than LUT D3 Gather. Its KernelOnly throughput drops by about 16 percent between $N = 1024/N = 2048$ and $N = 4096$ (Table I.1), coinciding with its table footprint crossing from L2 (81 KiB at $N = 1024$) into L3 (324 KiB at $N = 4096$, Table 2). LUT D3 Gather's throughput, by contrast, stays nearly flat even though its own footprint grows from 36 KiB to 144 KiB over the same range, both within L2. Instruction throughput therefore sets the floor for both gather kernels at depth 3, while at depth 4 a larger table adds a further cache-tier penalty on top of that floor.

Binary Kernels

Binary kernels show the opposite depth ordering from ternary: throughput increases monotonically with depth, from LUT D2 to LUT D3 to LUT D4, the fastest kernel measured on AVX2 in the KernelOnly regime (Figure 1, Table I.2). All three LUT variants outperform MAD BitNet by 1.3 to 3.2 times, and plain MAD by an even larger margin. This monotonic improvement is consistent with the binary footprint analysis in Table 2: because $K = 2$ for binary weights, even the depth-4 table (32 KiB) lands at the L1 boundary, so each depth increase keeps reducing the lookup count without paying the L2 or L3 penalty that the ternary depth-3 and depth-4 gather kernels incur.

KernelOnly versus Full Regime

The Full regime, which folds LUT construction into the timed loop, reduces throughput for every LUT kernel relative to KernelOnly (Tables I.1 and I.2). For ternary kernels, the construction overhead weighs most on the kernels that are otherwise fastest, since a fixed table-build cost is a larger fraction of a shorter kernel: LUT D2 loses roughly 7 to 26 percent and LUT D3 Mirror a comparable 6 to 21 percent, whereas the slower gather kernels give up far less, LUT D3 Gather under 4 percent and LUT D4 Gather 3 to 10 percent, the latter despite its 81-entry `int32` table being the most expensive to build. In every case the penalty is largest at $N = 1024$ and is progressively amortized as N grows, falling below 8 percent for all kernels by $N = 4096$.

For binary kernels, the Full regime introduces a larger penalty as depth increases, with LUT D4 suffering a 17 to 44 percent throughput drop due to unamortized table construction costs. Even after this penalty, LUT D4 remains the fastest binary kernel in the Full regime at every matrix size, but its margin over LUT D3 narrows substantially compared to KernelOnly.

For ternary, the Full regime reverses the ranking between the two BitNet-layout kernels. In the amortized KernelOnly regime, the LUT-based BitNet kernel (LUT D3 Mirror) outperforms the MAD-based BitNet kernel at $N = 1024$, draws level at $N = 2048$, and falls behind at $N = 4096$ (Table I.1). In the Full, per-inference regime, MAD BitNet outperforms LUT D3 Mirror at every matrix size, with the gap widening as N increases. The LUT-based BitNet kernel therefore offers an advantage over plain MAD BitNet only for small matrices where LUT construction is amortized across many output rows; for single-vector inference, or for large matrices, plain MAD BitNet is the faster ternary BitNet kernel on this platform.

5.3 Cross-Platform Extension: Edge Platform Evaluation

To characterize how the compute-versus-memory trade-off shifts under strict resource constraints, the kernels are ported to a Raspberry Pi running an ARM Cortex-A processor with NEON SIMD support. The x86_64 AVX2 intrinsics are systematically translated to their ARM NEON equivalents, preserving the same algorithmic structure to isolate architectural effects from algorithmic ones. Building on the throughput profiles and the computational degradation observed in the scalar template variants on x86, the identical benchmark sweeps are then applied on this platform, now augmented with direct `perf stat` cache profiling that was not collected on the x86 host, and the throughput results are compared against the x86_64 baseline to quantify the performance impact of a smaller per-core cache, narrower SIMD registers, and reduced memory bandwidth.

The board used is a Raspberry Pi 4 Model B (Broadcom BCM2711), with four ARM Cortex-A72 cores; its full specifications are listed alongside the desktop host in Appendix A. Compared with the Comet Lake host, this gives roughly an order-of-magnitude smaller per-core cache and a 128-bit NEON register file rather than a 256-bit AVX2 one.

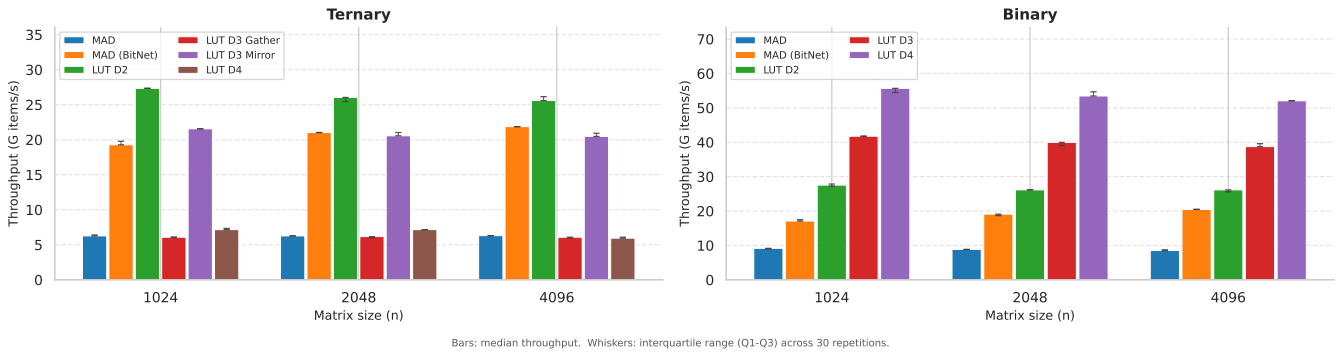

Figure 1: KernelOnly throughput of the AVX2 kernel set across matrix sizes $N \in \{1024, 2048, 4096\}$, for ternary (left) and binary (right) weights.

The NEON analogues of `VPSHUFB` are the table-lookup instructions `vqtbl1q_s8` and `vqtbl2q_s8`, which read a single 16-byte table register and a pair of 16-byte table registers, respectively. The provenance of the NEON LUT kernels differs from their AVX2 counterparts, because bitnet.cpp ships only a TL1 (depth-2) LUT path for ARM and no TL2 kernel. The NEON depth-2 kernel reimplements that TL1 path, the NEON depth-3 mirror kernel is our own port of the TL2 algorithm to NEON with no direct bitnet.cpp counterpart on ARM, and the direct depth-3 kernel (LUT D3 Direct), which uses the two-register `vqtbl2q_s8` lookup that has no AVX2 analog, is an entirely original contribution. As on AVX2, every kernel is run under both the KernelOnly and Full regimes for $N \in \{1024, 2048, 4096\}$, and IPC and cache-miss statistics are collected with `perf stat`.

NEON Throughput Results

Figure 2 reports KernelOnly throughput on the Pi 4B for $N \in \{1024, 2048, 4096\}$, separately for ternary and binary weights, with the underlying per-kernel GOPS for both regimes tabulated in Tables I.3 and I.4 in Appendix I. Converting to GOPS with the same items/s $\times$ 2 convention used for AVX2, the binary kernels reproduce the AVX2 depth ordering: throughput increases monotonically with depth from MAD BitNet through LUT D2 and LUT D3 to LUT D4, the fastest binary kernel at every size and the platform peak at 26.73 GOPS ($N = 1024$, Table I.4). The depth gaps keep the same shape as on AVX2, but absolute throughput is roughly 3.3 to 4.9 times lower than the corresponding AVX2 KernelOnly figures (Table I.2), a larger gap than the 2 times narrower SIMD width and 1.2 times lower clock alone would predict.

The ternary results, by contrast, do not reproduce the AVX2 ordering (Table I.3). On AVX2, LUT D2 is the clear ternary leader, beating its closest competitor (LUT D3 Mirror) by roughly 24 to 27 percent. On NEON, LUT D2 and LUT D3 Direct land within about 2 percent of each other at $N = 1024$ and $N = 2048$, effectively tied, with LUT D2 pulling roughly 5 percent ahead at $N = 4096$, while LUT D3 Mirror, D2's closest competitor on AVX2, falls to fourth place behind even MAD BitNet. LUT D4 is by far the slowest ternary kernel on NEON, at less than half the throughput of MAD BitNet, mirroring its weak showing on AVX2 (Table I.1).

Architectural Efficiency Profile

Figure E.1 in Appendix E reports IPC and L2 cache-miss rate from `perf stat`, KernelOnly, $N = 4096$, for every NEON kernel. The two MAD BitNet kernels stand apart from every LUT variant: they reach the highest IPC (1.52 to 1.57) and the lowest cache-miss rates (0.16 to 0.24 percent) of the entire set, by an order of magnitude over the LUT kernels. This is consistent with their algorithmic structure, sign-manipulation and masking performed entirely on registers loaded from the input vectors, with no precomputed table to read from memory.

Among the LUT kernels, IPC and cache-miss rate both track depth and table-lookup width rather than weight type. The binary kernels cluster tightly (IPC 1.43 to 1.47, cache-miss rate 1.16 to 1.76 percent), with cache-miss rate increasing mildly from D2 to D4 as their tables grow. The ternary kernels spread out more: LUT D2 matches the binary D2 profile exactly (IPC 1.43, miss rate 1.16 percent, since the two kernels share the same table-lookup pattern), while LUT D3 Direct has the lowest IPC of any kernel (1.17) and LUT D4 has by far the highest cache-miss rate of any kernel (2.68 percent, roughly 1.5 to 2.3 times every other LUT kernel). The elevated miss rate is only part of the story, however: at $N = 4096$ LUT D4 also issues about 3.3 billion cache references, roughly 2.7 to 3.5 times more than any other LUT kernel, so the higher miss fraction acts on a much larger volume of table traffic. The two compound into roughly 89 million absolute cache misses, about 4 to 6 times more than any other LUT kernel, and it is this combination, rather than the miss rate alone, that drives LUT D4's collapse to 5.8 to 6.0 GOPS in Figure 2, the weakest result on the platform.

5.4 LUT D2 versus LUT D3 Direct

The closeness of LUT D2 and LUT D3 Direct in Figure 2, and the low IPC of LUT D3 Direct in Figure E.1, point to a result that is the opposite of what the instruction count alone predicts. The two kernels differ only in how each pair-byte is converted into a partial sum: LUT D2 reads a 9-entry table with the single-register `vqtbl1q_s8`, while LUT D3 Direct reads the full $3^3 = 27$-entry table with the two-register

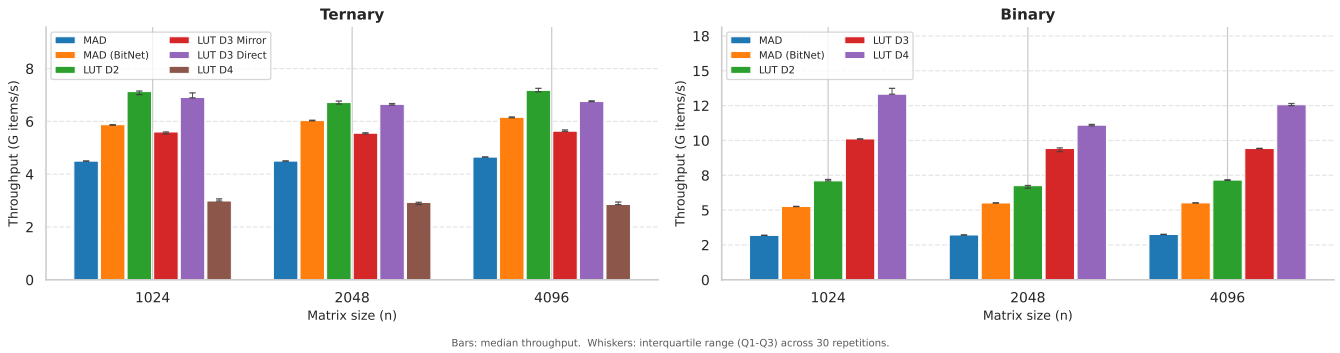

Figure 2: KernelOnly throughput of the NEON kernel set on the Pi 4B (Cortex-A72) across matrix sizes $N \in \{1024, 2048, 4096\}$, for ternary (left) and binary (right) weights.

`vqtbl2q_s8`, which covers more MACs per lookup and so issues fewer lookup instructions for the same number of MACs.

Table 4 shows what the hardware counters say instead. LUT D3 Direct does execute fewer instructions per item, 0.303 versus 0.358, roughly the 15 percent reduction the smaller lookup count predicts. But its IPC drops from 1.43 to 1.17: LUT D3 Direct ends up at 0.259 cycles per item against LUT D2's 0.252 in this perf-instrumented run, so the IPC drop more than cancels the lower instruction count and inverts the naive depth ordering. The uninstrumented timing run (Table I.3) realizes this inversion as a roughly 5 percent throughput deficit at $N = 4096$; both runs rank D2 ahead, and the small difference in margin is run-to-run variance from the perf instrumentation. LUT D3 Mirror is the slowest of the three overall, but for a different reason: its IPC (1.30) actually sits above LUT D3 Direct's, so the slowdown is driven by its higher instruction count per item (0.414), a cost of the extra mirror-consolidation logic rather than slower retirement.

Table 4: NEON KernelOnly mechanism metrics for the ternary depth-2 and depth-3 kernels at $N = 4096$, derived from `perf stat` hardware counters in a separate perf-instrumented run whose absolute throughput differs slightly from the uninstrumented timing run; per-kernel throughput is reported in Table I.3. Here an item denotes a single multiply-accumulate, consistent with the GOPS convention; the lookup figures discussed below are expressed as MACs covered per lookup instruction, since one `vqtbl1q`/`vqtbl2q` lookup serves multiple MACs across a row-tile.

Kernel	IPC	instr/item	cycles/item
LUT D2	1.43	0.358	0.252
LUT D3 Direct	1.17	0.303	0.259
LUT D3 Mirror	1.30	0.414	0.319

The root cause is that `vqtbl2q_s8` is not "one instruction's worth" of throughput on the Cortex-A72: its 2-register, 32-byte table consumes more register-read bandwidth and more NEON pipe issue slots per instruction than the single-register, 16-byte `vqtbl1q_s8`. A dependency-isolating microbenchmark measures the two-register form at roughly three times the per-lookup cost of the one-register form, yet each `vqtbl2q` lookup covers only 50 percent more MACs, so the wider table cannot earn back its per-instruction cost. This is what inverts the naive depth ordering, leaving the kernel with the smaller instruction count as the slower of the two; the per-MAC accounting behind the inversion is derived in Appendix F.

5.5 Roofline Analysis

We place both kernel sets on the Roofline established in Section 3, using the streaming operational intensities of Section 4.2 (Table 3). Figure 3 shows the NEON roofline (ternary kernels, KernelOnly, $N = 4096$); the AVX2 roofline and its per-kernel walkthrough are deferred to Appendix G, and the per-kernel NEON positioning to Appendix H. Three findings carry across both platforms.

First, instruction throughput, not memory bandwidth, is the binding constraint on both platforms. Every kernel falls far below the compute ceiling, reaching 37 percent of it at best on AVX2 and only about 12 percent on NEON, and with intensities spanning just 0.7 to 5.3 ops/byte on AVX2 and 1.6 to 2.67 on NEON, every kernel sits far to the left of its DRAM ridge point. No kernel is DRAM-bandwidth-bound at $N = 4096$ even in principle; whatever limits them lies within the L1/L2 tiers or in instruction issue.

Second, on AVX2 the pshufb redesigns are exactly the kernels that clear the no-ILP, latency-bound ceiling of 28.2 GOPS: LUT D4 (Binary) at 104.4 GOPS and LUT D3 Mirror (Ternary) at 41.3 GOPS replace a gather or a multi-step unpack with one or two single-cycle `pshufb` lookups, while every gather and MAD kernel falls below the line despite spanning a more than two-fold range of operational intensity. This confirms that the Section 3 redesigns convert algorithmic operational-intensity gains into measurable throughput.

Third, on NEON the same redesigns do not transfer: *every* kernel falls below the no-ILP ceiling of 19.2 GOPS, the opposite of the AVX2 result. The clearest evidence that instruction issue and dependency latency bind here, rather than bandwidth, is the spread at fixed operational intensity. Three kernels share OI $= 1.6$ (LUT D4 gather at 5.8 GOPS, MAD at 9.3, MAD BitNet at 12.3) yet differ by more than $2\times$ while moving identical bytes per MAC, and the two depth-3 kernels share OI $= 2.0$ yet differ by 20 percent (direct 13.5 ver-

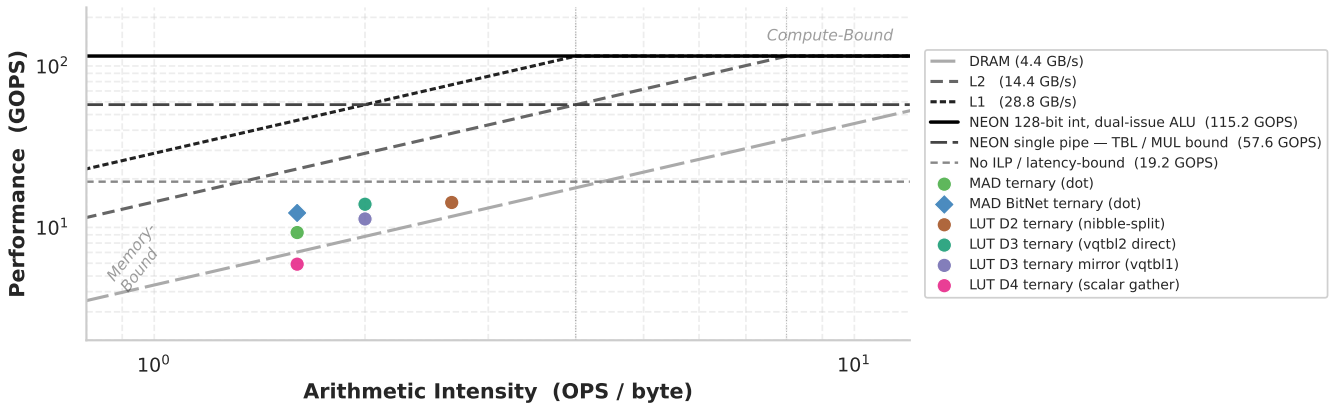

Figure 3: Roofline model for the Broadcom BCM2711 / Cortex-A72 (single core, NEON 128-bit integer) at 1.8 GHz, $N = 4096$, KernelOnly, ternary kernels. Diagonal lines show DRAM, L2, and L1 bandwidth ceilings; horizontal lines show the dual-issue ALU, single-pipe (TBL/MUL), and no-ILP compute ceilings. Every kernel falls below the no-ILP line, and the spread at fixed operational intensity (three kernels at OI $= 1.6$, two at OI $= 2.0$) identifies instruction scheduling, not bandwidth, as the binding constraint.

sus mirror 11.3). A bandwidth-bound regime cannot produce this spread; instruction scheduling on the narrow two-wide ASIMD front end sets the ranking.

6 Conclusion and Future Work

This thesis demonstrates that the performance advantage of LUT-based dot-product kernels over MAD-based kernels for binary and ternary Transformer inference is conditional rather than uniform: it is governed by the weight alphabet, the table depth, the cache tier the table lands in, and the throughput of the host's table-lookup instruction. The clearest win is on binary weights, where on wide-SIMD x86 throughput rises monotonically with depth to 104.4 GOPS at the largest matrix size, roughly 2.6× the strongest MAD baseline, MAD BitNet; the ternary case is more conditional, with the depth-2 kernel ahead of MAD BitNet by a narrower 1.2 to 1.4×. The advantage erodes wherever the table outgrows fast cache, forces a gather instead of a shuffle, or must be rebuilt per inference. These failure modes share a common mechanism. Across the roofline analysis, the binding limit is consistently instruction throughput rather than DRAM bandwidth, so a kernel's standing is set by whether its lookup stays a single-cycle shuffle on an L1-resident table. This is why binary keeps benefiting through depth 4 while ternary gather kernels merely track plain MAD despite higher operational intensity, and it sharpens on the Cortex-A72, where the two-register `vqtbl2q` costs roughly 3× a single-register `vqtbl1q` and inverts the expected depth ordering. Optimal kernel selection is therefore a function of the deployment target, and the trade-off profile established here characterizes which approach to prefer under each constraint.

6.1 Future Work

Several directions follow. The depth sweep stopped at $g = 4$, but the NEON inversion analysis predicts that any kernel relying on `vqtbl2q`, `vqtbl3q`, or `vqtbl4q` should pay an IPC penalty more than proportional to the number of table registers, so measuring deeper kernels directly, alongside table-padding strategies that keep them within the single-register `vqtbl1q` limit, would test whether the trade-off holds at higher depths. Widening the comparison to the native low-bit dot-product instructions, AVX-512 VNNI and NEON `SDOT`, would close the remaining gap by sidestepping the unpack-and-shuffle sequence that bottlenecks the MAD baselines here. Finally, the evaluation is confined to single-core matrix-vector kernels under synthetic inputs; extending it to full GEMM with realistic batch sizes, multi-threaded execution, and an end-to-end inference stack, and broadening the hardware set beyond Comet Lake and the Cortex-A72, would show how far these kernel-level trade-offs translate into token-level throughput and how sensitive the depth orderings are across microarchitectures.

7 Responsible Research

7.1 Ethical Considerations

This work is a pure systems and hardware benchmarking study. No human subjects are involved, no personal data is collected or processed, and no user studies or surveys are conducted. The research does not involve training new machine learning models, and therefore does not generate additional carbon emissions beyond the benchmarking runs themselves, which are short-lived microbenchmarks on a single workstation.

The broader motivation of this work is to reduce the computational cost of large language model inference. Running LLMs at full precision on server-class hardware carries substantial energy and memory costs, and scaling these systems has raised legitimate environmental concerns [14]. By enabling competitive inference quality at 1-bit and 1.58-bit precision on commodity and edge hardware, this line of research directly addresses that concern, potentially reducing the energy per inference significantly and making capable models accessible on devices with far lower power budgets.

A secondary consideration is the dual-use nature of making AI more efficient. Lowering the barrier to running LLMs

on resource-constrained hardware democratizes access, but it also removes some of the infrastructure friction that currently limits misuse. We note, however, that the kernels benchmarked here operate exclusively at the integer dot-product level and do not extend or improve model capabilities in any way. The work evaluates existing quantization schemes; it does not contribute new model architectures, training procedures, or datasets.

7.2 Reproducibility

Reproducibility is a central concern in performance research, where measurements are sensitive to compiler flags, OS scheduling, and hardware frequency scaling. The experimental setup and benchmarking protocol described in Section 5.1 address these concerns directly through fixed random seeds, CPU frequency pinning, thread affinity, and empirically measured bandwidth ceilings. The operational intensity values used in the Roofline analysis are derived analytically from documented instruction counts rather than measured indirectly (Equation 4 and Table 3), making them independently verifiable from the kernel source.

One measurement-scope limitation should be made explicit. Direct hardware-counter cache profiling was carried out only on the ARM edge platform, via `perf stat`; no equivalent counter-based cache measurement was collected on the x86_64 Comet Lake host. The cache-tier behavior reported for the AVX2 kernels in Section 5.2 is therefore inferential: it is derived from the analytical table-footprint model (Equation 1, Table 2) evaluated against the documented host cache geometry and read in conjunction with the measured AVX2 throughput, rather than from measured miss rates. Statements about which cache tier an AVX2 table occupies, and about cache-tier transitions affecting throughput, should accordingly be read as model-based predictions consistent with the observed throughput, not as direct cache-counter measurements. The NEON cache-miss and IPC figures (Figure E.1, Table 4) are the only directly measured cache statistics in this work.

All kernel implementations and benchmark code are publicly available at https://github.com/batuEren/fast-qdot, allowing results to be reproduced directly from source.

The primary limitation on reproducibility is that the compiled binaries are ISA-specific: the AVX2 build targets AVX2 explicitly (`/arch:AVX2` under MSVC) while the NEON build uses `-march=native` under GCC, so each binary produces different code on hardware that lacks the corresponding SIMD extensions. The x86_64 results are specific to the Intel Comet Lake microarchitecture, and results on other microarchitectures may differ due to differences in execution port counts, shuffle unit availability, and cache geometry. The ARM NEON results similarly depend on the specific Raspberry Pi board revision used.

7.3 Usage of LLMs

Large language models were used as a development and writing aid during this work. On the implementation side, we used them to generate explanatory comments for the kernels, to assist with debugging, and to suggest performance improvements. On the writing side, they were used to correct grammar and spelling and to help with the construction of LaTeX tables and with improving general readability. All LLM output was reviewed and verified before being included in the code or the paper.

References

[1] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models. *CoRR*, abs/2310.11453, 2023.

[2] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *CoRR*, abs/2402.17764, 2024.

[3] Jinheng Wang, Hansong Zhou, Ting Cao, Shijie Cao, Shuming Ma, Furu Wei, et al. Bitnet.cpp: Efficient edge inference for ternary LLMs. *CoRR*, abs/2502.11880, 2025. Code: https://github.com/microsoft/BitNet.

[4] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. T-MAC: CPU renaissance via table lookup for low-bit LLM deployment on edge. In *Twentieth European Conference on Computer Systems (EuroSys '25)*, Rotterdam, Netherlands, 2025.

[5] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[6] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

[7] Darshan C. Ganji, Saad Ashfaq, Ehsan Saboori, Sudhakar Sah, Saptarshi Mitra, MohammadHossein AskariHemmat, Alexander Hoffman, Ahmed Hassanien, and Mathieu Léonardon. DeepGEMM: Accelerated ultra low-precision inference on CPU architectures using lookup tables. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2023.

[8] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. *CoRR*, abs/2206.09557, 2024.

[9] Georgi Gerganov. llama.cpp: LLM inference in C/C++. https://github.com/ggerganov/llama.cpp, 2023.

[10] Nii Osae Osae Dade, Tony Morri, Moinul Hossain Rahat, and Sayandip Pal. Litespark inference on consumer

CPUs: Custom SIMD kernels for ternary neural networks. *CoRR*, abs/2605.06485, 2026.

[11] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[12] Andreas Abel and Jan Reineke. uops.info: Characterizing latency, throughput, and port usage of instructions on Intel microarchitectures. https://uops.info/table.html, 2019. Accessed: June 2026.

[13] Arm Limited. Cortex-a72 software optimization guide. Technical Report UAN0016A, Arm Limited, 2015. Application Note.

[14] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, Florence, Italy, 2019.

A Hardware and Environment Configuration

This appendix collects the full hardware specifications and the build and measurement-isolation settings used throughout the experimental evaluation. The main text identifies the two platforms only by microarchitecture, a Comet Lake desktop CPU and a Cortex-A72 Raspberry Pi, and the granular figures required to reproduce the measurements are gathered here. Table A.1 lists the per-platform hardware parameters.

Build configuration. The two platforms use different toolchains. The AVX2 kernels are compiled with MSVC (Microsoft Visual C++) at the `/O2` optimization level, with `/arch:AVX2` applied to the SIMD translation units to enable AVX2 code generation. The NEON kernels are compiled on the Raspberry Pi with GCC using `-O3 -march=native`. Both configurations apply aggressive optimization and full exploitation of the host ISA. Dead-code elimination of the benchmarked computation is prevented by the Google Benchmark `DoNotOptimize` barrier rather than by the optimization flags.

Measurement isolation. For high-fidelity timing, the clock frequency of the target core is pinned to a fixed, non-boosted value, which eliminates turbo-boost variance from the measurements; on the desktop this is achieved by disabling turbo boost, and on the Pi by setting the CPU governor to `performance`. The benchmark thread is bound to a single physical core through thread-affinity masks, preventing OS-induced context switches, thread migration, and the cache pollution they cause between measurement intervals.

B AVX2 LUT Kernel Implementation Details

This appendix collects the low-level implementation details of the AVX2 LUT kernels deferred from Section 3.2.

Mirror-consolidated sign recovery. In the depth-3 *D3b* kernel, the 13 sign-mirror combinations omitted from the 14-entry canonical table are reconstructed from their stored counterparts by a branchless conditional two's-complement negate,

$$v_{\text{out}} = (v + \text{mask}) \oplus \text{mask}, \quad \text{mask} \in \{0\texttt{x0000}, 0\texttt{xFFFF}\}, \tag{5}$$

where the mask is all-zeros for an entry already in canonical sign, leaving v unchanged, and all-ones (`0xFFFF`) when the sign must be flipped, in which case the expression evaluates to $-v$ in two's-complement arithmetic. Casting the sign decision as a mask rather than a branch keeps the negation off the critical path and avoids a data-dependent branch in the inner loop.

Lane pre-duplication. Each 16-byte LUT is pre-duplicated across both 128-bit lanes of a 32-byte block, so that a single 256-bit load replaces the otherwise required 128-bit load and lane-insert sequence when the table is broadcast into a `YMM` register.

Overflow flushing. All pshufb-based LUT kernels accumulate partial sums in `int16` lanes and flush them to `int32` accumulators every 32 iterations to prevent overflow. The 32-iteration bound is verified analytically for each variant from the maximum per-iteration partial-sum magnitude, guaranteeing that no `int16` lane overflows between flushes.

C Footprint Ratio Derivation

This appendix gives the successive-footprint ratio behind the monotonicity result of Section 4.1, deferred from the main text.

To determine which trend dominates, we examine the ratio of successive footprints. Approximating $\lceil N/g \rceil \approx N/g$, which is tight for the large values of N used in practice ($N = 4096$ throughout this work), we obtain

$$\frac{F(g+1)}{F(g)} \approx K \cdot \frac{g}{g+1}. \tag{6}$$

This ratio exceeds 1 whenever $K > (g+1)/g$, that is, whenever $K \geq 2$ and $g \geq 1/(K-1)$. For ternary weights ($K = 3$), the ratio equals $3g/(g+1) \geq 3/2 > 1$ for all integer $g \geq 1$, so the footprint is strictly increasing from the shallowest depth onward. For binary weights ($K = 2$), the ratio is $2g/(g+1)$, which equals exactly 1 at $g = 1$. The $g = 1 \to g = 2$ step therefore costs no additional memory while halving the operation count, making depth-2 binary LUT a Pareto improvement over depth-1. For all $g \geq 2$ the binary ratio also exceeds 1, and the footprint grows monotonically thereafter.

D LUT Streaming Operational Intensity Derivation

This appendix gives the tile-level traffic accounting behind the streaming operational intensity of Equation 4, deferred from Section 4.2.

Each LUT kernel processes weights in 32-row tiles, one packed weight byte per row, where each byte encodes D activation positions. Here D is the number of activation positions resolved per weight byte, not the LUT depth g: the two coincide only for the byte-aligned ternary depth-3 and

Table A.1: Hardware specifications of the two benchmarking platforms.

	Desktop	**Edge**
CPU	Intel Core i7-10870H	Broadcom BCM2711
Microarchitecture	Comet Lake (10th gen)	ARM Cortex-A72
Base clock	2.20 GHz	1.8 GHz
SIMD ISA	AVX2 (256-bit)	NEON (128-bit)
L1 data	32 KB/core	32 KB/core
L1 instruction	32 KB/core	48 KB/core
L2	256 KB/core	1 MB shared (4 cores)
L3	16 MB shared	—

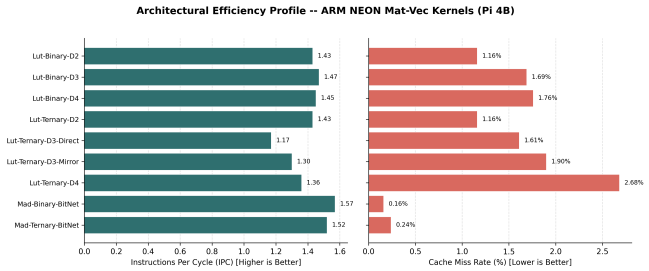

Figure E.1: IPC (left) and L2 cache-miss rate (right) for all NEON kernels on the Pi 4B, KernelOnly, $N = 4096$.

depth-4 kernels, whereas the nibble-split variants pack two depth-g groups into a byte and reach $D = 2g$ (LUT D2 at $D = 4$, and the binary depth-3 and depth-4 kernels at $D = 6, 8$). For one pair-byte position, a tile iteration loads the 32 weight bytes (one per row) together with the L table bytes that the lookup re-streams for that position, and produces $32D$ multiply-accumulates, that is $64D$ operations. Dividing these $64D$ operations by the $32 + L$ streamed bytes yields Equation 4.

E NEON Architectural Efficiency Profile

Figure E.1 gives the per-kernel IPC and L2 cache-miss rate on the Pi 4B (Cortex-A72), measured with `perf stat` in the KernelOnly regime at $N = 4096$, underlying the architectural efficiency discussion in the main text.

F Cortex-A72 Table-Lookup Microbenchmark

This appendix details the dependency-isolating microbenchmark used to measure the latency and reciprocal throughput of `vqtbl1q_s8` and `vqtbl2q_s8` on the Cortex-A72, in support of the LUT D2 versus LUT D3 Direct inversion analysis in the main text. The Cortex-A72 Software Optimization Guide [13] documents execution latencies for these instructions but lists no throughput value for either; Table F.1 reproduces the relevant entries. Because throughput, not latency, governs a kernel that issues many independent lookups, the guide alone cannot settle whether the wider lookup drives the inversion, and this microbenchmark fills in the missing column.

Table F.1: Cortex-A72 ASIMD table-lookup instruction timings for the AArch64 `TBL/TBX` forms, ASIMD table lookup, Q-form: 3×N + 3, from the Cortex-A72 Software Optimization Guide [13].

Instruction	Latency (cycles)	Throughput	Pipelines
vqtbl1q	6	-	F0/F1
vqtbl2q	9	-	F0/F1

Both variants of the microbenchmark share a single kernel shape, a fixed unrolled run of table lookups inside the Google Benchmark timing loop, and differ only in the dependency structure between operations. The latency variant threads one accumulator through the chain, so that each lookup takes the previous lookup's result vector as its index input (`acc = tbl(table, acc)`). Operation $i{+}1$ cannot begin until operation i has produced its result, so the measured time per operation is the execution latency of the instruction plus the forwarding the recurrence requires. The table register is held constant and kept off the dependency chain, so what the chain measures is index-to-result latency, exactly the figure the guide reports. The throughput variant instead advances eight independent accumulators, each its own feedback chain. Operations on different chains carry no data dependency, so the core can issue from one chain while another is still in flight; once enough chains are live to cover the latency, the rate is no longer limited by any single recurrence and saturates at reciprocal throughput, the rate at which the issue ports and register-read bandwidth retire independent lookups.

The choice of eight chains is checked against the data rather than assumed. With N chains and per-operation latency L, the chain-limited ceiling is N/L lookups per cycle, which gives $8/6.27 = 1.28$ for `vqtbl1q` and $8/9.45 = 0.85$ for `vqtbl2q`. The achieved rates, 0.66 and 0.22 lookups per cycle, sit well below both ceilings, confirming that the throughput measurement is genuinely throughput-bound and not a latency-limited number in disguise. Wall time is converted to cycles using the pinned 1.8 GHz clock (governor set to performance, single-core affinity), and the counters report cycles-per-lookup and lookups-per-cycle as iteration-invariant rates, each computed as total cycles over total lookups independent of the iteration count.

Table F.2 reports the result. The measured latencies, 6.27

cycles for `vqtbl1q` and 9.45 for `vqtbl2q`, reproduce the guide's documented 6 and 9 to within about five percent. This agreement shows that the harness is measuring the right instructions under the right conditions, which is what licenses trusting the throughput numbers it produces for the same instructions, the column the guide leaves blank.

Table F.2: Measured Cortex-A72 timings for the two NEON table-lookup instructions, from the dependency-isolating microbenchmark (1.8 GHz, single core, performance governor). Latency is measured with a single serial accumulator chain and throughput with eight independent chains; reciprocal throughput is the inverse of the lookups-per-cycle rate. The optimization guide (Table F.1) lists no throughput value for either instruction.

Instruction	Latency (cycles)	Recip. throughput (cycles/lookup)
vqtbl1q	6.27	1.51
vqtbl2q	9.45	4.64

The measured latencies land just above the guide's 6 and 9 cycles, an excess of roughly 0.27 and 0.45 cycles. Two ordinary effects account for this, and neither undermines the result. First, the inner loop carries a counter increment, a compare, and a back-edge branch per operation; these run on the integer pipes concurrently with the NEON chain and are largely hidden, but the benchmark state-loop bookkeeping that wraps each batch is amortized over the unrolled operations rather than eliminated, leaving a small constant per operation. Second, the guide latency is the raw execution latency of the instruction in isolation, whereas a back-to-back dependent recurrence also pays the forwarding cost from one lookup's writeback to the next lookup's index-read port, a bypass the single-instruction figure does not include.

With the throughput column filled in, the main-text claim can be made exact. The measured reciprocal throughputs are 1.51 cycles per lookup for `vqtbl1q` and 4.64 for `vqtbl2q`, a ratio of 3.07: the two-register form is roughly three times, not twice, as expensive per lookup as the one-register form. The argument rests on this ratio rather than on the absolute figures: the 1.51 cycles per lookup measured for `vqtbl1q` is itself somewhat above the rate an ideal dual-issue single-micro-op lookup would achieve, so a roughly equal additive overhead may sit on both instructions, but such an overhead cancels in the ratio, and if anything removing it would only widen the gap.

Feeding the measured ratio into a per-MAC accounting produces the figures quoted in the main text. Each `vqtbl1q` lookup in LUT D2 covers 16 MACs at a reciprocal throughput of 1.51 cycles, i.e. $1.51/16 \approx 0.094$ cycles per MAC; each `vqtbl2q` lookup in LUT D3 Direct covers 24 MACs at 4.64 cycles, i.e. $4.64/24 \approx 0.193$ cycles per MAC. Expressed in TBL1-equivalent cycles per MAC, LUT D2 costs $1.51/16 \approx 0.094$ and LUT D3 Direct costs $(3.07 \times 1.51)/24 \approx 0.193$, about double. Although LUT D3 Direct's lookups cover 50 percent more MACs per instruction, the lookup portion of its work therefore costs about twice as many cycles per MAC: the roughly $3\times$ per-instruction penalty nets out to about $2\times$ per MAC because each LUT D3 Direct lookup does more work. LUT D2's eight small, mutually independent single-register lookups also expose more instruction-level parallelism than LUT D3 Direct's four wide, register-read-heavy ones, the same asymmetry seen from the scheduler's side, which shows up directly as the IPC gap in Table 4.

This $2\times$ per-MAC penalty is confined to the lookup portion of the kernel. Both kernels share the same outer structure, 32-row tiles, eight `int32x4_t` accumulators, and `int16x8_t` running sums flushed via `vmovl_s16`/`vmovl_high_s16`; the accumulation, the int16-to-int32 widening through `vmovl`, and the weight loads are therefore identical across LUT D2 and LUT D3 Direct, so the blowup applies to only one component of the work; spread across the whole kernel it shows up only as the small cycles-per-item gap in Table 4 (0.252 versus 0.259, about 2.5 percent in this perf-instrumented run), not the near-$2\times$ the lookup arithmetic alone would imply.

G AVX2 Roofline Analysis

This appendix gives the per-kernel AVX2 roofline positioning summarized in the main-text Roofline Analysis (Section 5.5). Figure G.1 places the MAD baselines and the depth-2, depth-3, and depth-4 LUT kernels (KernelOnly, $N = 4096$) on the AVX2 roofline established in Section 3, using the operational intensities derived in Section 4.2 (Table 3). The depth-2 kernels, which achieve the highest ternary throughput in Figure 1, are included here as well.

All ten plotted kernels fall well below the AVX2 compute ceiling of 281.6 GOPS, and below the bandwidth ceiling implied by their own operational intensity. LUT D3 Mirror (Ternary), at 41.3 GOPS, sits almost exactly at the L1 ridge point but reaches only about 15 percent of the compute ceiling the L1 roofline permits there, and LUT D3 (Binary), at 77.6 GOPS, sits at the analogous L2 ridge point at about 28 percent of its ceiling (Figure G.1). The DRAM ridge point lies well to the right of every measured kernel (maximum OI $= 5.3$), so none of these kernels can be DRAM-bandwidth-bound at $N = 4096$, even in principle; whatever limits them lies within the L1/L2 tiers or in instruction issue.

Four kernels sit at or below the no-ILP, latency-bound ceiling of 28.2 GOPS: MAD Ternary (12.7 GOPS), MAD Binary (17.2 GOPS), LUT D3 Gather Ternary (12.0 GOPS), and LUT D4 Gather Ternary (12.0 GOPS) all fall below this line, despite spanning a more than two-fold range of operational intensity (0.7 to 1.8 ops/byte). Because this line represents the throughput attainable if every element required one full-latency, non-pipelined multiply-accumulate, kernels below it indicate a per-element instruction sequence that is even more costly than that single-instruction baseline. For the gather kernels this is consistent with `VPGATHERDD`'s 5-cycle reciprocal throughput (Table 1); for the MAD kernels it reflects the multi-instruction unpack, shuffle, and sign-mapping sequence that precedes each `VPMADDUBSW`.

The LUT kernels that clear the no-ILP line by the largest margin, LUT D2 (Ternary and Binary) at 51.3 and 52.0 GOPS, LUT D3 Mirror (Ternary) at 41.3 GOPS, and LUT D3 and LUT D4 (Binary) at 77.6 and 104.4 GOPS, are exactly the ones whose inner loop replaces a gather or a multi-step unpack with one or two single-cycle `pshufb`

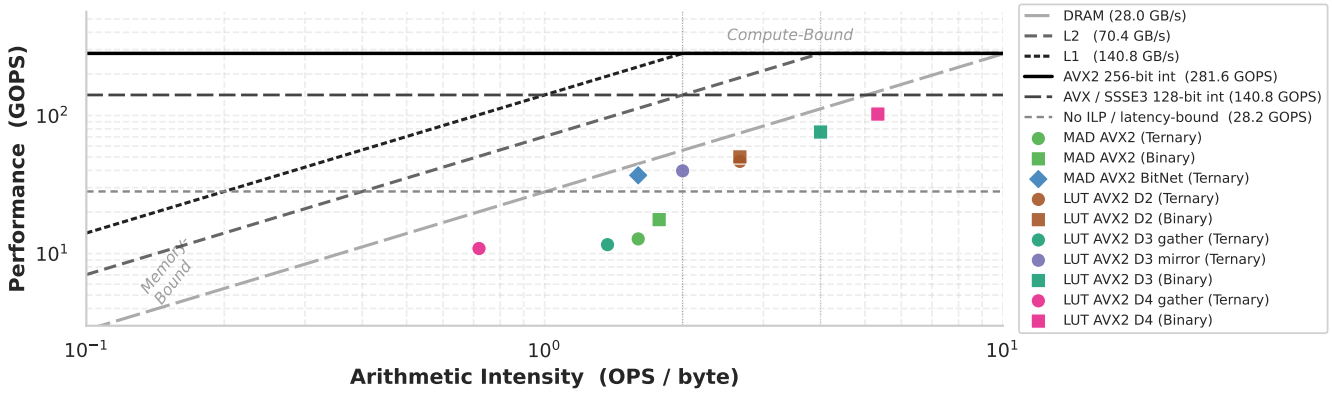

Figure G.1: Roofline model for the Intel i7-10870H (single core, AVX2 integer), $N = 4096$, KernelOnly. Diagonal lines show DRAM, L2, and L1 bandwidth ceilings; horizontal lines show the AVX2 and no-ILP compute ceilings.

lookups; the only MAD kernel to clear the line is the BitNet variant (43.7 GOPS, Ternary). This confirms that the lookup-table redesigns described in Section 3 succeed in converting algorithmic operational-intensity gains into measurable throughput, even though none reach the bandwidth- or compute-bound regions of the roofline. LUT D4 (Binary), the fastest kernel measured on this platform at 104.4 GOPS, still operates at only 37 percent of the AVX2 compute ceiling and below the DRAM bandwidth line projected to its own intensity ($28.0 \times 5.3 \approx 149$ GOPS), confirming that its bottleneck is the four-`pshufb`-plus-interleave sequence per pair-byte rather than any memory-bandwidth tier. LUT D4 Gather (Ternary) is the worst-positioned kernel on the chart: at OI $= 0.7$, even the DRAM bandwidth line permits only about 20 GOPS (28.0×0.7), yet the kernel achieves just 12.0 GOPS, around 60 percent of even that modest ceiling.

H NEON Roofline Positioning

This appendix gives the per-kernel NEON roofline detail behind the main-text Roofline Analysis (Section 5.5). Figure 3 places the ternary MAD baselines and the depth-2, depth-3 (direct and mirror), and depth-4 LUT kernels (KernelOnly, $N = 4096$) on the NEON roofline for the Raspberry Pi 4B (Cortex-A72, single core), using the streaming operational intensities derived in Section 4.2 (Table 3; OPS/byte $= 2\times$ the MAC/byte values). Unlike the AVX2 figure, depth-2 is included here, it is the throughput leader on this platform, and only the ternary variants are plotted.

All six kernels fall far below the NEON dual-issue compute ceiling of 115.2 GOPS, the fastest reaching only about 12 percent of it. Their plotted intensities span just 1.6 to 2.67 OPS/byte, far to the left of the DRAM ridge point at $115.2/4.4 = 26.2$ OPS/byte, so all three bandwidth lines remain in their sloped, diagonal regime across the figure. The DRAM diagonal is the lowest of the three and therefore the tier the kernels sit closest to, yet five of the six lie on or above the DRAM line projected to their intensity (LUT D2 at 14.4 GOPS versus $4.4 \times 2.67 = 11.7$; MAD BitNet at 12.3 versus $4.4 \times 1.6 = 7.0$). This is the expected signature of cache residency rather than a paradox: the plotted intensity is a *streaming* figure that counts the activation LUT restreamed from L2, so at $N = 4096$ the 64 KiB activation LUT (the depth-2 kernel's four 16-byte `vqtbl1` tables per pair-byte, $n/4 = 1024$ pair-bytes $\times$ 64 B, the L of Table 3) is served from the 1 MiB L2 while only the 4 MiB weight matrix (4096^2 weights at 2 bits) touches DRAM. Measured against the L2 diagonal that actually serves this traffic, every kernel sits far below; single-core DRAM bandwidth binds none of them.

The defining feature of this platform is that *every* kernel falls below the no-ILP, latency-bound ceiling of 19.2 GOPS, the opposite of the AVX2 result, where the lookup-table redesigns cleared the no-ILP line by a wide margin. The Cortex-A72 combines a narrow two-wide ASIMD issue, a single shuffle/TBL pipe, and a multi-cycle dependent-accumulate latency. Together these prevent the per-element unpack, table-lookup, and `vzip` reconstruction sequence from being overlapped tightly enough to retire even one full-latency multiply-accumulate per element. The eight-accumulator scheduling in the LUT kernels recovers some of this, but not enough to cross the line. The best kernel, LUT D2 nibble-split at 14.4 GOPS, reaches only about 75 percent of the no-ILP ceiling and 25 percent of the single-pipe (57.6 GOPS) ceiling. The clearest evidence that the binding constraint is instruction issue and dependency latency rather than bandwidth is the spread at fixed operational intensity: three kernels share OI $= 1.6$ (LUT D4 gather at 5.8 GOPS, MAD at 9.3, and MAD BitNet at 12.3) yet differ by more than $2\times$ despite moving identical bytes per MAC, and the two depth-3 kernels share OI $= 2.0$ yet differ by 20 percent (direct 13.5 versus mirror 11.3). In every case the ranking is set by instruction scheduling, not memory traffic. MAD BitNet's stride-32 layout feeds sequential `vld1q` loads into a 16-`vmlal` / 4-accumulator schedule that covers the dependent-accumulate latency. The plain MAD kernel's `vld4` de-interleave instead leaves stalls, and the gather kernel is dominated by SIMD↔GPR traffic.

Two effects that improved AVX2 throughput fail to transfer to the A72. First, the mirror consolidation, the fastest ternary kernel on AVX2, is here *slower* than both LUT D2

and the direct depth-3 kernel it is meant to improve on. The two depth-3 kernels share the same streaming intensity (2.0 OPS/byte), since the mirror's 16-byte `vqtbl1` tables are a load-port saving only while the constructor still re-streams an identically sized LUT, so the roofline cannot account for the mirror's 20 percent deficit (11.3 versus 13.5 GOPS); as the counters in Table 4 show, it is the higher instruction count of the mirror's branchless conditional-negate sequence, not its memory traffic, that the narrow ASIMD front end cannot hide. Second, the absence of a NEON gather instruction makes LUT D4 the worst-positioned kernel on the chart, exactly as `VPGATHERDD` did on AVX2: at OI $= 1.6$ it achieves just 5.8 GOPS, below even the DRAM line (7.0 GOPS) at that intensity, because each of its 32 per-pair-byte lookups is a scalar `vgetq_lane`→GPR→load round-trip whose cross-domain transfers dominate the kernel.

I Full Results

This appendix collects the complete per-kernel throughput tables in GOPS underlying the throughput figures in the main text, reported for both the KernelOnly and Full regimes on each hardware platform. Each entry is the mean over 30 benchmark repetitions, accompanied by the sample standard deviation (σ) and the interquartile range (IQR) across those repetitions, the same dispersion measures drawn as whiskers in the main-text throughput figures.

Tables I.1 and I.2 give the per-kernel AVX2 throughput in GOPS underlying Figure 1, for both the KernelOnly and Full regimes.

Tables I.3 and I.4 give the corresponding per-kernel NEON throughput in GOPS on the Raspberry Pi 4B (Cortex-A72) underlying Figure 2, again for both the KernelOnly and Full regimes.

Table I.1: AVX2 ternary kernel throughput in GOPS (items/s × 2), reported as the mean over 30 benchmark repetitions together with the sample standard deviation (σ) and interquartile range (IQR). MAD kernels have no LUT-construction stage, so a single KernelOnly value is reported.

		$N = 1024$			$N = 2048$			$N = 4096$		
Kernel	**Regime**	Mean	σ	IQR	Mean	σ	IQR	Mean	σ	IQR
MAD	KernelOnly	12.59	0.32	0.27	12.57	0.14	0.00	12.66	0.16	0.00
MAD BitNet	KernelOnly	38.79	0.62	1.01	41.99	0.56	0.00	43.74	0.69	0.00
LUT D2	KernelOnly	54.83	0.55	0.00	51.50	0.69	1.21	51.33	0.80	1.11
	Full	40.72	0.40	0.00	44.42	0.77	0.97	47.60	0.85	1.17
LUT D3 Gather	KernelOnly	12.23	0.30	0.00	12.24	0.26	0.31	12.04	0.15	0.27
	Full	11.79	0.16	0.25	12.01	0.17	0.00	12.01	0.19	0.27
LUT D3 Mirror	KernelOnly	43.24	0.66	0.00	41.59	0.73	0.93	41.33	0.60	0.89
	Full	34.35	0.45	0.79	37.28	0.71	0.60	38.76	0.53	0.93
LUT D4 Gather	KernelOnly	14.43	0.21	0.35	14.31	0.14	0.00	12.04	0.19	0.27
	Full	12.99	0.15	0.00	13.63	0.16	0.00	11.70	0.20	0.27

Table I.2: AVX2 binary kernel throughput in GOPS (items/s × 2), reported as the mean over 30 benchmark repetitions together with the sample standard deviation (σ) and interquartile range (IQR). MAD kernels have no LUT-construction stage, so a single KernelOnly value is reported.

		$N = 1024$			$N = 2048$			$N = 4096$		
Kernel	**Regime**	Mean	σ	IQR	Mean	σ	IQR	Mean	σ	IQR
MAD	KernelOnly	18.10	0.26	0.38	17.55	0.19	0.00	17.24	0.24	0.37
MAD BitNet	KernelOnly	34.26	0.75	0.79	37.96	0.44	0.83	40.92	0.59	0.00
LUT D2	KernelOnly	55.20	0.83	1.21	52.15	0.77	0.00	51.96	0.89	1.11
	Full	43.27	0.45	0.00	46.30	0.70	1.06	49.15	1.01	1.23
LUT D3	KernelOnly	83.22	1.08	0.00	79.25	1.23	1.82	77.64	1.36	1.72
	Full	55.60	0.79	0.00	63.62	0.93	1.33	69.92	1.25	1.55
LUT D4	KernelOnly	110.40	1.84	2.42	107.63	1.13	2.43	104.45	1.77	0.00
	Full	61.76	0.74	1.32	77.70	1.46	1.74	87.24	1.27	0.00

Table I.3: NEON (Pi 4B, Cortex-A72) ternary kernel throughput in GOPS (items/s × 2), reported as the mean over 30 benchmark repetitions together with the sample standard deviation (σ) and interquartile range (IQR). MAD kernels have no LUT-construction stage, so a single KernelOnly value is reported.

		$N = 1024$			$N = 2048$			$N = 4096$		
Kernel	**Regime**	Mean	σ	IQR	Mean	σ	IQR	Mean	σ	IQR
MAD	KernelOnly	8.99	0.01	0.01	8.99	0.01	0.01	9.30	0.00	0.00
MAD BitNet	KernelOnly	11.73	0.02	0.04	12.06	0.04	0.05	12.31	0.03	0.07
LUT D2	KernelOnly	14.18	0.15	0.27	13.42	0.11	0.22	14.37	0.13	0.27
	Full	13.17	0.20	0.43	13.12	0.11	0.15	14.02	0.04	0.06
LUT D3 Direct	KernelOnly	13.90	0.24	0.38	13.32	0.11	0.11	13.54	0.08	0.06
	Full	12.95	0.15	0.10	12.84	0.12	0.23	13.28	0.08	0.09
LUT D3 Mirror	KernelOnly	11.15	0.10	0.15	11.09	0.07	0.09	11.30	0.07	0.11
	Full	10.32	0.18	0.29	10.66	0.16	0.19	11.18	0.07	0.08
LUT D4	KernelOnly	6.04	0.09	0.16	5.80	0.09	0.17	5.78	0.10	0.19
	Full	5.90	0.06	0.13	5.66	0.09	0.18	5.78	0.10	0.20

Table I.4: NEON (Pi 4B, Cortex-A72) binary kernel throughput in GOPS (items/s $\times$ 2), reported as the mean over 30 benchmark repetitions together with the sample standard deviation (σ) and interquartile range (IQR). MAD kernels have no LUT-construction stage, so a single KernelOnly value is reported.

		$N = 1024$			$N = 2048$			$N = 4096$		
Kernel	**Regime**	Mean	σ	IQR	Mean	σ	IQR	Mean	σ	IQR
MAD	KernelOnly	6.38	0.00	0.01	6.43	0.00	0.01	6.51	0.01	0.01
MAD BitNet	KernelOnly	10.52	0.01	0.01	11.02	0.01	0.01	11.04	0.01	0.01
LUT D2	KernelOnly	14.27	0.12	0.22	13.39	0.21	0.38	14.29	0.10	0.10
	Full	13.83	0.17	0.33	13.27	0.13	0.25	14.13	0.03	0.04
LUT D3	KernelOnly	20.22	0.07	0.03	18.68	0.30	0.54	18.84	0.04	0.07
	Full	18.87	0.28	0.11	17.92	0.17	0.22	18.66	0.05	0.06
LUT D4	KernelOnly	26.73	0.83	0.85	22.17	0.21	0.20	25.12	0.20	0.30
	Full	20.63	0.60	1.24	19.98	0.15	0.13	23.51	0.23	0.23