DualView: Exact Phase-Specialized Weight Representations for Quantized LLM Inference

A mathematical and systems analysis of Q7 decode compactness with Q8-class prefill execution on AMD unified-memory GPUs

Ciru Inference Lab

Technical Whitepaper · 27 July 2026

Abstract

Quantized large-language-model inference is commonly described as a choice of one weight precision for an entire request. That framing hides a useful asymmetry. Autoregressive decode repeatedly streams a model for one or a few tokens and therefore benefits strongly from compact weights, whereas prefill exposes wide matrix products whose performance depends on regular hardware lanes, kernel maturity, and instruction count. We present **DualView**, a runtime representation technique that keeps one authoritative FPX7 quantized tensor for storage and decode while materializing an exact Q8_0 execution view for selected prefill matrix multiplications. The view is not a second quantization: every 7-bit two's-complement code is sign-extended into an 8-bit lane and its FP16 group scale is copied bit-for-bit. Hence the dequantized weight is invariant, $d'_g q'_i = d_g q_i$, and any quality difference between the two routes indicates an implementation defect rather than representation error. On released Ornith-1.0-35B weights, a controlled three-load ablation increases PP4096 from 713.421 to 1202.550 tokens/s (+68.56%) while TG256 changes by +0.43%. On a released dense Qwen3.5-9B control, DualView improves prefill by 37.59–40.38% over canonical Q7 across 128–2048 prompt tokens, remains within ±2.93% of native Q8 prefill, and preserves Q7-like generation. We formalize the transform, derive the prefill/decode asymmetry, describe a public ROCmFPX implementation, evaluate quality and memory costs, and report failed alternatives. DualView's central contribution is a separation of *model approximation* from *hardware representation*: one quantized function can have multiple exact execution views, selected by phase.

- 1 Introduction
- 2 Problem formulation
  - 2.1 Quantization and representation are different operations
  - 2.2 Desired properties
- 3 FPX7 as the authoritative model
  - 3.1 Code domain and reconstruction
  - 3.2 Packed macroblock
- 4 The exact Q8 execution view
  - 4.1 Byte-level transform
  - 4.2 Exactness theorem
  - 4.3 What exactness does and does not imply
  - 4.4 Persistent storage arithmetic
- 5 Why phase specialization matters

- 5.1 A simple arithmetic-intensity model
- 5.2 Why Q8 can be faster than packed Q7
- 5.3 Exact layout specialization as an I/O optimization
- 6 Runtime design and implementation
  - 6.1 View lifecycle
  - 6.2 Tensor eligibility
  - 6.3 Expert-bank flattening and strides
  - 6.4 Prefill/decode routing
- 7 Experimental methodology
  - 7.1 Research questions
  - 7.2 Platform and software
  - 7.3 Models and conditions
  - 7.4 Metrics and repetition
  - 7.5 Claim-strength protocol
- 8 Results
  - 8.1 RQ1: representation and implementation exactness
  - 8.2 RQ2: controlled same-weight phase ablation
  - 8.3 RQ3: dense-model replication and native-Q8 control
  - 8.4 Kernel mechanism
  - 8.5 RQ4: comparison with released Ornith quantizations
  - 8.6 Long-context end-to-end panel
  - 8.7 Quality behavior
  - 8.8 RQ5: memory cost and selective materialization
- 9 Negative results and design lessons
  - 9.1 Direct specialized kernels
  - 9.2 Split and panel layouts
  - 9.3 Weight error is not graph quality
  - 9.4 Bitwidth is not a performance ordering
- 10 Limitations and threats to validity
  - 10.1 Hardware and software scope
  - 10.2 Memory capacity
  - 10.3 Statistical scope
  - 10.4 Teacher quality
  - 10.5 Dynamic execution
  - 10.6 Model and workload generality
- 11 Related work
- 12 Conclusion
- 13 Appendix A — Reference conversion algorithm
- 14 Appendix B — Reproducibility checklist
- 15 Appendix C — Orthogonal MTP extension
- 16 Appendix D — Evidence ledger summary
- References

1 Introduction

The dominant intuition for quantized inference is simple: fewer weight bits mean less storage, less memory traffic, and therefore more speed. That intuition is directionally useful for single-token generation, but it is incomplete for modern GPU prefill. A compact sub-byte representation can reduce bytes read while simultaneously imposing irregular unpacking, sign reconstruction, scale handling, or a kernel path that cannot use the device's best integer matrix instructions. Conversely, an eight-bit representation can be larger yet execute faster because it is already arranged in the native lanes expected by a mature matrix kernel.

This distinction matters because an LLM request contains at least two computational regimes:

1. **Prefill** evaluates many prompt tokens in parallel. Its linear layers are wide matrix–matrix products, and each loaded weight can be reused across many activation rows.
2. **Autoregressive decode** evaluates one or a few new tokens at a time. Its linear layers are matrix–vector or very narrow matrix products, so the full active model is repeatedly streamed with little weight reuse.

The Roofline model relates attainable throughput to arithmetic intensity—the amount of useful work per transferred byte (Williams et al. 2009). Work on on-device quantized LLMs similarly observes that generation is strongly memory-bandwidth-bound, while the context phase has substantially higher

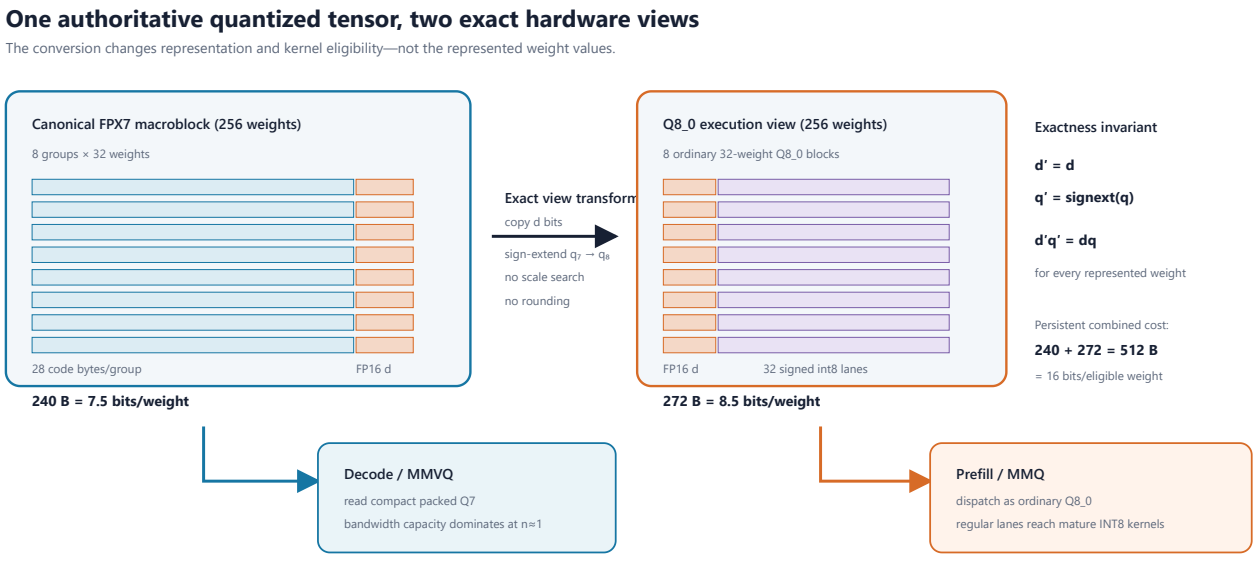
arithmetic intensity (Lin et al. 2024). These observations suggest that the representation best suited to decode need not be the representation best suited to prefill.

DualView operationalizes that idea without introducing a second approximate model. It stores one authoritative groupwise Q7 tensor. At load time, the runtime can create an auxiliary Q8_0-compatible *execution view* by copying each group scale and sign-extending each 7-bit signed integer into an 8-bit lane. Both representations evaluate the same quantized numbers. The packed view is used for bandwidth-sensitive decode; the regular view is used for eligible prefill matrix multiplication.

This paper makes four contributions:

- **An exact representation transform.** We define the FPX7 layout, give the byte-level Q7→Q8 view algorithm, and prove equality of every represented weight. “Exact” is explicitly relative to the authoritative Q7 target, not to its BF16 source.
- **A phase-specialized runtime design.** We describe view materialization, tensor eligibility, stride preservation, cache lifecycle, and dispatch into existing Q8_0 matrix kernels while retaining Q7 decode and gather paths.
- **Controlled and replicated evaluation.** A same-weight Ornith-1.0-35B ablation isolates a 68.56% prefill gain with a 0.43% decode change. A dense Qwen3.5-9B sweep reproduces the mechanism over four prompt widths and includes native Q8, served time-to-first-token, profiler, quality, and memory controls.
- **A negative-results and limits account.** We quantify the persistent-view memory cost, compare released public quantizations only, distinguish exact views from mixed-precision “quality islands,” and document attractive alternatives that failed at full-model scale.

The public implementation is released in the ROCmFPX research runtime at tag `dualview-ornith-35b-v1` (Ciru Inference Lab 2026). Ornith-1.0-35B is a released Mixture-of-Experts (MoE) model derived from the Qwen3.5 family (DeepReinforce Team 2026). All model comparisons in this paper use released artifacts; no private or unnamed model is used as a baseline.



DualView retains a canonical packed FPX7 tensor for decode and constructs an exact Q8_0-compatible execution view for prefill. Copying the scale and sign-extending the integer code preserves every represented value.

2 Problem formulation

2.1 Quantization and representation are different operations

Let a group of real-valued source weights be $w_g = (w_{g,0}, \ldots, w_{g,31})$. A symmetric groupwise quantizer selects a scale d_g and signed integer codes $q_{g,i}$ to approximate the source:

$$\hat{w}_{g,i} = d_g q_{g,i}, \qquad q_{g,i} \in \{-64, \ldots, 63\}. \tag{1}$$

The *quantization operation* maps w_g to (d_g, q_g). It is lossy in general because $\hat{w}_{g,i} \neq w_{g,i}$. The choice of scale, rounding rule, clipping, and any calibration objective determines that approximation error. GPTQ, AWQ, and SmoothQuant improve different aspects of this mapping or the surrounding activation problem (Frantar et al. 2022; Lin et al. 2024; Xiao et al. 2023).

A *representation operation* maps already-quantized (d_g, q_g) into a different byte layout while preserving the mathematical pair. DualView is in the second category. It does not revisit the source weights, search a new scale, or round a value again. This distinction is the central reason that its prefill route cannot introduce new quantization error.

For clarity, we use three different equality notions:

- **Source equality:** $\hat{W} = W_{\mathrm{BF16}}$. This generally does not hold after quantization.
- **Quantized-function equality:** two stored layouts dequantize to the same $\hat{W}$. This is the DualView guarantee.
- **Program equality:** two complete runtime executions produce the same outputs. This additionally requires correct indexing, accumulation, kernel semantics, and deterministic surrounding operations.

The mathematical transform proves quantized-function equality. Exhaustive unit tests and saved-logit comparisons test the implementation conditions needed for program equality.

2.2 Desired properties

An effective phase-specialized view must satisfy five constraints.

Numerical invariance. The view must not be an independently quantized copy. Otherwise a prefill/decode boundary could change the model function, invalidate cached states, or make quality depend on request shape.

Native kernel compatibility. The auxiliary layout should match an existing well-optimized data type and dispatch path. A novel layout that merely moves unpacking elsewhere can be slower than the original.

Stable tensor semantics. Ordinary 2-D weights, three-dimensional expert banks, row strides, channel strides, and gather operations must retain their logical indexing.

Lifecycle safety. A view must be generated from the current canonical payload, invalidated when that payload changes, and never outlive its owning allocation.

Selective materialization. The runtime should be able to exclude tensors whose view offers little benefit or whose memory cost is undesirable.

3 FPX7 as the authoritative model

3.1 Code domain and reconstruction

FPX7 uses signed two's-complement integer codes with seven meaningful bits. For code bit pattern $b_6 b_5 \ldots b_0$, the integer interpretation is

$$q = \begin{cases} \sum_{j=0}^{6} b_j 2^j, & b_6 = 0, \\ \sum_{j=0}^{6} b_j 2^j - 2^7, & b_6 = 1, \end{cases} \tag{2}$$

so $q \in [-64, 63]$. One FP16 scale d_g is shared by 32 adjacent weights, and reconstruction is Equation (1). The scale is stored as its two raw FP16 bytes. FP16 scale storage matters to the proof: the exact view copies those bytes rather than converting the scale through FP32 or recomputing it.

The current quantizer uses a bounded scale search rather than naïve max-absolute scaling. On a 32,768,000-weight study, the selected search reduced Q7 reconstruction mean-squared error from 2.2575×10^{-8} to 1.7693×10^{-8}, a 21.63% improvement. Q8 remained lower at 5.5728×10^{-9}, so the result does not imply that Q7 is intrinsically more accurate than Q8. It establishes only that the authoritative Q7 target is better constructed than a naïve scale rule (Evidence E21).

This quantizer detail is deliberately separated from DualView. Any method that produces the same Q7 (d_g, q_g) can use the exact execution view.

3.2 Packed macroblock

Eight groups form one 256-weight FPX7 macroblock. Each group contains:

- $32 \times 7 = 224$ code bits $= 28$ bytes; and
- one FP16 scale $= 2$ bytes.

The macroblock therefore occupies

$$B_{Q7} = 8(28 + 2) = 240 \text{ bytes}, \qquad \text{bpw}_{Q7} = \frac{8B_{Q7}}{256} = 7.5. \tag{3}$$

Packing gives decode its principal benefit: relative to a Q8_0 block, fewer weight bytes must cross the memory hierarchy for each generated token. The cost is that integer lanes are not byte-aligned. A consumer must locate bits that may cross byte boundaries, reconstruct the signed code, and pair it with the group scale.

4 The exact Q8 execution view

4.1 Byte-level transform

For each 32-weight Q7 group, the view constructor performs:

1. Read the two scale bytes and copy them unchanged into the Q8_0 scale field.

2. For each $i \in [0, 31]$, extract the seven-bit code $u_i \in [0, 127]$.
3. Sign-extend it:

$$q_i' = \begin{cases} u_i, & u_i < 64, \\ u_i - 128, & u_i \geq 64. \end{cases} \tag{4}$$

4. Store q_i' in one signed int8 lane.

The boundary cases make the transform concrete:

Raw 7-bit field	Q7 integer	Stored int8 byte	int8 integer
`0b0000000`	0	`0x00`	0
`0b0111111`	63	`0x3F`	63
`0b1000000`	−64	`0xC0`	−64
`0b1111111`	−1	`0xFF`	−1

The high int8 bit is therefore a replicated sign bit, not a newly estimated information bit. Q8_0 has room for values outside the Q7 domain, but the exact view never creates them.

The resulting group is an ordinary `block_q8_0`: a two-byte FP16 scale followed by 32 signed int8 codes. Eight groups occupy

$$B_{\text{view}} = 8(2 + 32) = 272 \text{ bytes}, \qquad \text{bpw}_{\text{view}} = 8.5. \tag{5}$$

The 0.5-bit overhead beyond “eight bits per weight” is the amortized FP16 scale: $16/32 = 0.5$ bits per weight.

4.2 Exactness theorem

Theorem 1 (DualView value invariance). Let an FPX7 group represent $\hat{w}_i = dq_i$, where d is the stored FP16 scale and $q_i \in [-64, 63]$ is the signed value of a seven-bit two’s-complement code. Construct a Q8_0 view with $d' = d$ bit-for-bit and $q_i' = \text{signext}_{7\to 8}(q_i)$. Then the view represents the same value for every element:

$$\hat{w}_i' = d'q_i' = dq_i = \hat{w}_i. \tag{6}$$

Proof. A signed integer in $[-64, 63]$ is exactly representable in int8. Two’s-complement sign extension preserves its integer value, so $q_i' = q_i$ as integers. Copying the two scale bytes gives $d' = d$ as an FP16 value. Substitution yields Equation (6). No rounding or floating-point arithmetic is performed by the transform. □

The proof is stronger than an empirical “close logits” result: it covers the complete code domain. The public unit test nevertheless exhaustively enumerates all 128 raw seven-bit patterns and verifies the mapping, because bit extraction, block offsets, and expert flattening can violate an otherwise correct theorem if implemented incorrectly (Evidence E02).

4.3 What exactness does and does not imply

Theorem 1 implies that choosing the Q7 or Q8 representation for a multiplication does not change the stored quantized matrix. It does **not** imply:

- equality to the BF16 source weights;
- bit-identical accumulation across arbitrary kernels, because different reduction orders or activation quantizers can round differently;
- identical stochastic samples when tiny floating-point differences cross a sampling boundary; or
- that separately stored Q8 "quality island" tensors are equivalent to Q7.

In the tested DualView path, saved final logits for a 4,054-token sequence and all 3,973,120 FP32 output values from a 16-prompt panel were bit-identical between canonical and viewed routes. Saved perplexity traces were also identical (Evidence E12). Those tests validate the current program; the theorem explains why equality is the expected invariant.

4.4 Persistent storage arithmetic

When both representations are resident, an eligible 256-weight region consumes

$$B_{\text{persistent}} = B_{Q7} + B_{\text{view}} = 240 + 272 = 512 \text{ bytes}, \tag{7}$$

or exactly 16 bits per eligible weight before allocator alignment and unrelated tensor types. The number is initially counterintuitive: a "Q7 model with a view" can occupy more live accelerator memory than a native Q8 model. DualView is therefore not a universal memory-compression technique. It is a **performance specialization for machines with spare capacity**, especially large unified-memory systems where a high-quality model fits but its packed prefill path leaves compute performance unused.

Equation (7) also creates a clean design-space problem. If only a fraction ρ of Q7 weights receive persistent views, the view payload is approximately

$$B_{\text{extra}}(\rho) = \rho N_{Q7} \frac{8.5}{8}, \tag{8}$$

where N_{Q7} is the number of eligible Q7 weights. Tensor alignment, metadata, F32/Q8 tensors, and backend allocations add fixed or piecewise terms. Section 8 measures the resulting memory–prefill frontier.

5 Why phase specialization matters

5.1 A simple arithmetic-intensity model

Consider a linear layer

$$Y = X\hat{W}^{T}, \qquad X \in R^{n \times k}, \quad \hat{W} \in R^{m \times k}. \tag{9}$$

For a first-order argument, let b_{eff} be effective bits per weight, including amortized group metadata but excluding allocator alignment. Ignore activation/output traffic and cache effects. The layer performs approximately $2mnk$ arithmetic operations and reads $b_{\text{eff}}mk/8$ weight bytes. The arithmetic intensity attributable to weight traffic is

$$I_W(n, b_{\text{eff}}) \approx \frac{2mnk}{b_{\text{eff}}mk/8} = \frac{16n}{b_{\text{eff}}} \quad \text{operations/byte.} \tag{10}$$

During decode, $n \approx 1$. Equation (10) gives roughly 2.13 operations/byte for 7.5-bpw FPX7 and 1.88 operations/byte for 8.5-bpw Q8_0 including their amortized scales. Both are low-intensity regimes. Compactness directly increases the maximum weight bandwidth available to useful work, provided unpacking does not dominate.

During prefill, n can be hundreds or thousands. The same weights are reused across many rows, so their one-time byte cost is amortized. Activation traffic, output traffic, tiling, occupancy, and compute instructions become increasingly important. The representation with the fewest stored bits need not provide the fastest kernel.

This derivation is intentionally not a claim that prefill is always compute-bound or decode always bandwidth-bound. Attention, KV-cache traffic, MoE routing, batch shape, cache residency, and sequence length change the balance. It states a more limited and robust proposition: increasing n raises weight reuse linearly, weakening the value of sub-byte storage while increasing the value of regular matrix execution.

5.2 Why Q8 can be faster than packed Q7

On the tested RDNA3.5 `gfx1151` path (Advanced Micro Devices, Inc. 2024), Q8_0 is an established landing format for staged integer matrix multiplication. Codes already occupy signed byte lanes. The kernel can load regular blocks, stage them predictably, and use the backend's mature W8A8/MMQ implementation.

Packed Q7 must reconstruct four conceptual objects:

1. the group scale;
2. a possibly cross-byte seven-bit field;
3. the sign of that field; and
4. the byte or register lane consumed by the dot-product path.

Those operations are not "free because the model is smaller." They consume integer instructions, registers, address calculations, and potentially extra data rearrangement. A custom Q7 matrix kernel can still win for favorable shapes, but bit count alone does not determine the outcome. In the isolated Ornith expert geometry, substituting the exact Q8 representation lowers gate/up latency from 4271.03 to 1843.43 µs (2.32×) and down-projection latency from 4355.82 to 2015.37 µs (2.16×). A separate Instella-shaped probe shows a 2.18× gate/up ratio (Evidence E07–E08).

These operator probes establish a mechanism, not a model-level speedup. Router launches, attention, non-viewed tensors, synchronization, and memory allocation all intervene in a complete request. The controlled end-to-end ablation in Section 7 is the causal model result.

5.3 Exact layout specialization as an I/O optimization

DualView belongs to a broader class of I/O-aware optimizations: keep the mathematical operation fixed while changing how data traverses the hierarchy. FlashAttention, for example, preserves exact attention while reducing high-level memory traffic through tiling (Dao et al. 2022). DualView operates at a different layer and with a different trade-off: it spends persistent capacity to avoid repeated unpacking and reach an efficient integer landing format.

The design principle can be stated independently of Q7 and AMD:

> If a compact canonical representation and a hardware-native execution representation are related by an exact transform, and if request phases have different reuse regimes, it can be profitable to retain both and select by phase.

Porting this principle to another device requires a native second format, a profitable kernel path, and lifecycle-safe view management. It does not imply that the current ROCm implementation already runs on every backend.

6 Runtime design and implementation

6.1 View lifecycle

The public ROCmFPX implementation extends the `llama.cpp`/GGML accelerator backend (ggml.org contributors 2026). It does not alter the GGUF on-disk payload. Model loading proceeds as follows:

1. Allocate the canonical device tensor.
2. If the tensor is Q7, contiguous, on an enabled `gfx1151` HIP device, and its semantic name passes the allowlist, compute the aligned tail size for a Q8 view.
3. Allocate one owning region containing the canonical payload followed by the aligned auxiliary region.
4. During full tensor upload, construct the view from the complete host Q7 payload and transfer it to the tail.
5. Record the view pointer, logical dimensions, effective Q8 row/channel strides, and a generation identifier.
6. On tensor mutation or owner destruction, invalidate the record so a stale view cannot be dispatched.

The view is built once per loaded tensor, not before every prompt and not at the prefill/decode boundary. There is no “compress, run, unload, decompress” cycle. Both byte layouts remain available, and dispatch selects one.

The generation identifier is a correctness device rather than a performance feature. A pointer to an old tail allocation could otherwise survive a tensor reload and silently combine new metadata with stale weights. Public tests cover allocation, invalidation, exact code conversion, and expert-bank flattening (Evidence E04).

6.2 Tensor eligibility

The public route is intentionally conservative. It requires:

- HIP execution on `gfx1151`;
- explicit runtime opt-in;
- a contiguous Q7 weight tensor;
- a matrix operation supported by the staged Q8 MMQ path; and
- a semantic tensor name in the attention/feed-forward allowlist.

The output projection and token embedding can be excluded. `GET_ROWS` and token-generation MMVQ continue to read canonical Q7. Qwen-style attention projections, feed-forward projections, and three-dimensional MoE expert banks are eligible.

Conservative eligibility is important because a transform can be numerically exact but operationally wrong. A tensor's data type does not reveal whether its dimensions describe a two-dimensional matrix, an expert bank, an embedding table, or a gather-oriented layout. The view route must preserve the consumer's logical indexing and kernel contract.

6.3 Expert-bank flattening and strides

An MoE expert bank is commonly represented as $W \in R^{E \times m \times k}$, where E is the number of experts. The Q8 MMQ path can treat this as E adjacent matrices, but only if three stride relationships are preserved. Let $S_j^{(7)}$ be a runtime stride measured in FPX7 macroblocks for dimension $j \in \{\mathrm{row}, \mathrm{channel}, \mathrm{sample}\}$. One FPX7 macroblock contains eight 32-weight groups, and the view represents those groups as eight ordinary Q8_0 blocks. Therefore the block-count stride presented to the Q8 kernel is

$$S_j^{(8)} = 8S_j^{(7)}, \qquad j \in \{\mathrm{row}, \mathrm{channel}, \mathrm{sample}\}. \tag{11}$$

In byte units, the same logical span is $240S_j^{(7)}$ in canonical FPX7 and $34S_j^{(8)} = 272S_j^{(7)}$ in the view. The public MMQ implementation operates in block-count units, so it multiplies each Q7 block stride by the exact integer factor eight rather than applying a floating byte-size ratio.

The runtime then rewrites only the execution descriptor:

- weight pointer → Q8 view base;
- weight type → `Q8_0`;
- row/channel/sample block strides → view strides;
- staged MMQ policy → Q8-compatible path.

The canonical tensor metadata remains Q7. Decode and any non-view consumer therefore cannot accidentally infer that the stored model has changed type.

6.4 Prefill/decode routing

Let $R(o, n)$ be the execution representation selected for operation o and activation-row count n:

$$R(o,n) = \begin{cases} \mathrm{Q8View}, & o = \mathrm{MMQ} \wedge n \geq n_{\min} \wedge \mathrm{eligible}(W), \\ \mathrm{Q7Canonical}, & \mathrm{otherwise}. \end{cases} \tag{12}$$

The published release uses the existing prefill MMQ boundary and conservative shape gates rather than claiming a universal optimal n_{min}. This is a deliberate systems choice. Earlier experiments found that isolated Q8 WMMA kernels could be 2.49–6.24× faster on selected shapes while a broad complete model route regressed prefill by roughly 34%. A narrowly safe direct route recovered only about 0.4%. Kernel profitability is shape- and graph-dependent; the view should enter a mature path whose complete-model behavior is measured, not a path chosen from a single microbenchmark (Evidence E20).

7 Experimental methodology

7.1 Research questions

The evaluation asks five questions:

1. **RQ1 — Exactness:** Does the implementation preserve every Q7-represented value and complete-model outputs?
2. **RQ2 — Causality:** When model weights and protocol are fixed, how much prefill improvement comes from the exact execution view, and what happens to decode?
3. **RQ3 — Portability:** Does the prefill/decode behavior reproduce on a released dense model with a different matrix topology?
4. **RQ4 — Practical frontier:** How does a quality-oriented DualView artifact compare with released Q6/Q8-family artifacts, including long context?
5. **RQ5 — Cost and limits:** How much memory does persistence consume, and which alternative layouts or routing ideas fail?

7.2 Platform and software

Experiments ran on the Ciru `gfx1151` unified-memory system under Linux using a pinned TheRock ROCm/HIP 7.15.0 development toolchain (`7.15.0a20260718`, AMD Clang 23.0.0git) and a ROCmFPX branch built for `gfx1151`. The branch lineage was frozen and publicly released at commit `b515125810cd233936e52620cd73055a12825e71`.

The paper intentionally scopes performance claims to this platform and software revision. The exact representation theorem is architecture-neutral; the measured profitability is not.

7.3 Models and conditions

Ornith-1.0-35B. The principal MoE target is the released DeepReinforce Ornith-1.0-35B model (DeepReinforce Team 2026). The clean causal ablation uses the same pure FPX7 artifact in both conditions. The later quality-max artifact contains 362 Q7 tensors, 70 deliberately retained Q8 “quality island” tensors, and 301 F32 tensors, for 7.533 effective bits per weight and a 32,638,875,232-byte file. Every promoted Q8 tensor is byte-identical to its released Q8 donor, and every retained tensor is byte-identical to the Q7 base (Evidence E18).

Qwen3.5-9B dense control. A released dense Qwen-family model provides a different topology and cleaner BF16/Q8/Q7 quality triangle. Its Q7 file is 8,408,409,056 bytes, 11.746% smaller than the 9,527,502,048-byte Q8 artifact. DualView modes exclude or include the output tensor according to the named condition.

No private model is used. Instella appears only in a public-shape operator probe because the tested runtime could not serve it end-to-end.

7.4 Metrics and repetition

- **PP**N **:** prompt-processing throughput for N prompt tokens.
- **TG**N **:** autoregressive generation throughput for N generated tokens.
- **TTFP:** wall time from request start to first generated token.
- **GTT:** live GPU translation-table allocation reported by the ROCm runtime; on this unified-memory system it is used as a live accelerator-memory measure.
- **Perplexity/KLD:** token-distribution diagnostics under frozen token streams.
- **HellaSwag:** commonsense completion accuracy (Zellers et al. 2019).
- **Code benchmarks:** HumanEval/EvalPlus and BigCodeBench are reported only with their exact adapter/sampling protocol because they execute generated programs and are sensitive to prompt and termination behavior (Chen et al. 2021; Liu et al. 2023; Zhuo et al. 2025).

The primary Ornith same-weight comparison uses three clean model loads and three repetitions per load. Qwen prompt-width results are clean paired model runs. Selective-view memory experiments have one repetition and are labeled diagnostic rather than inferential. Throughput is reported to sufficient precision to reproduce calculated ratios, but that precision should not be read as an uncertainty bound.

7.5 Claim-strength protocol

Evidence is classified before interpretation:

- a byte-layout consequence is a **proof**;
- a same-weight, same-runtime route change is a **controlled causal result**;
- a second-model or multi-width result is a **replication**;
- a microbenchmark, profiler counter, or one-shot selective-view run is a **diagnostic**;
- interrupted teacher studies and protocol-corrected code scores are **unresolved**.

The complete claim-to-evidence map is provided in Appendix E and the companion `EVIDENCE.md`. This protocol prevents a visually compelling diagnostic from silently becoming a general claim.

8 Results

8.1 RQ1: representation and implementation exactness

The transform is exact by Theorem 1. The public test suite enumerates every seven-bit code and validates expert flattening. Complete-model validation adds two empirical checks:

- the final raw FP32 logits after a 4,054-token sequence are bit-identical between the canonical and viewed routes; and
- a 16-prompt all-output run contains 3,973,120 bit-identical FP32 values, with identical saved perplexity traces.

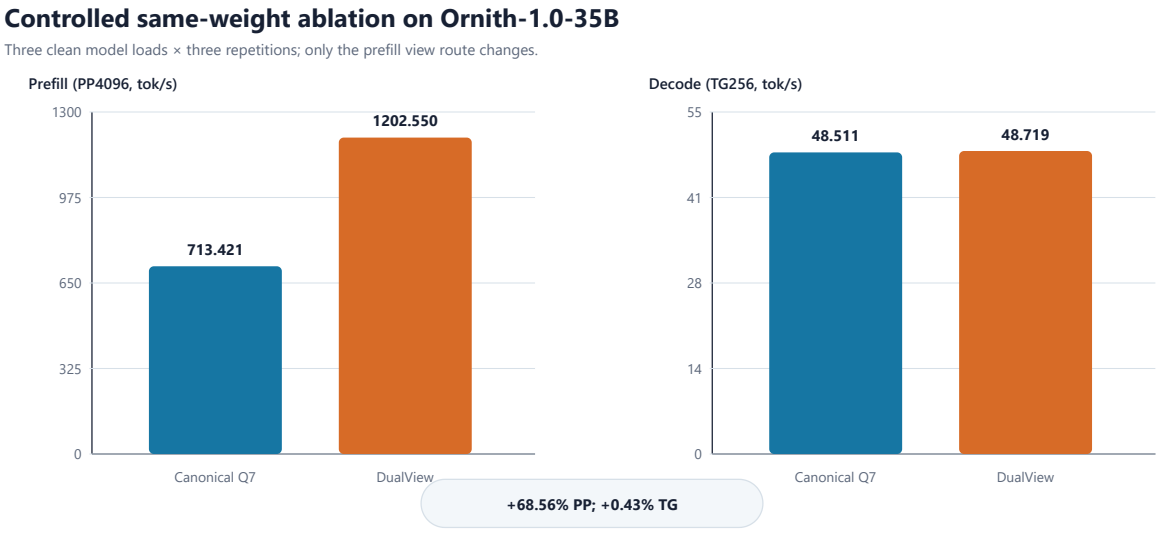
The mathematical and empirical statements play different roles. Exhaustive code enumeration tests the conversion implementation; large-output equality tests indexing, strides, routing, and surrounding kernel behavior. Neither establishes source-BF16 equality, which depends on the original Q7 quantization.

8.2 RQ2: controlled same-weight phase ablation

Table 1 and Figure 2 report the clean causal result. Both conditions load the same pure FPX7 Ornith artifact. The treatment changes only whether eligible prefill MMQ operations may use the exact Q8 view.

Table 1. Three clean loads × three repetitions per condition. The result isolates representation routing; the weights themselves are unchanged (Evidence E06).

Condition	PP4096 (tok/s)	TG256 (tok/s)	Δ prefill	Δ decode
Canonical packed FPX7	713.421	48.511	—	—
Q8 view prefill + Q7 decode	**1202.550**	**48.719**	**+68.56%**	**+0.43%**



The exact view increases Ornith PP4096 by 68.56% while TG256 remains effectively unchanged.

The asymmetry is the desired result. If DualView were merely "a faster Q8 model," generation should move toward native Q8 behavior. Instead, generation continues through canonical packed Q7 and changes by only 0.43%, while prefill captures the Q8-compatible kernel path.

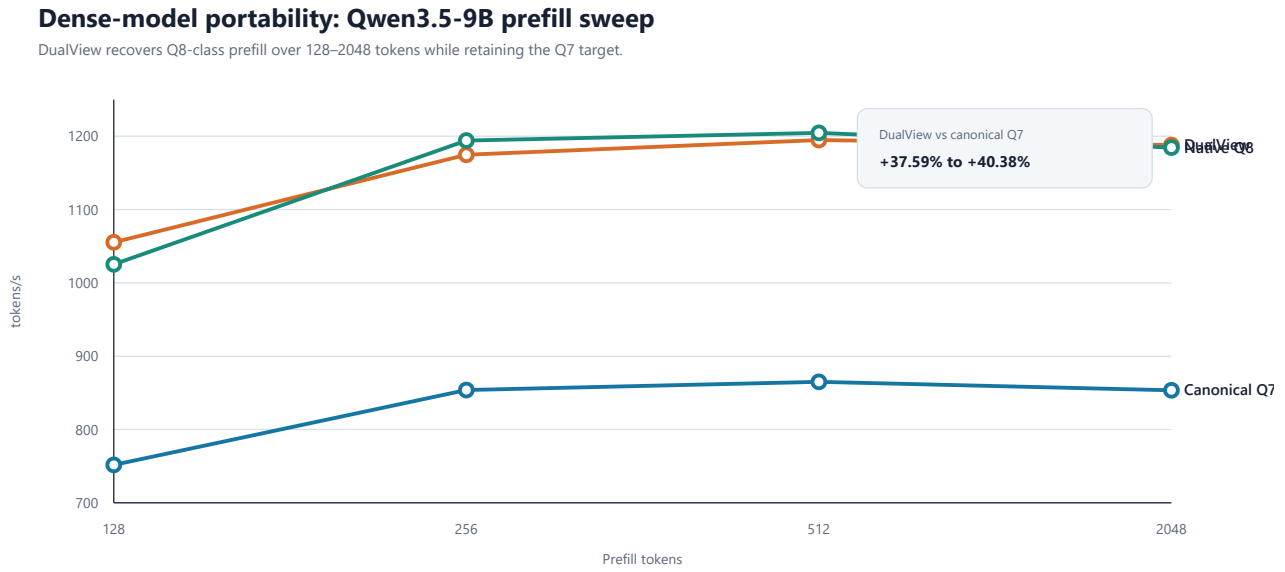
The 68.56% model-level gain is smaller than the 2.16–2.32× isolated expert latency ratios because not all request time is eligible expert MMQ. Attention, routing, non-viewed projections, synchronization, and other operations dilute the operator gain. This gap is expected under Amdahl's law and is evidence against extrapolating a microbenchmark directly to full-model throughput.

8.3 RQ3: dense-model replication and native-Q8 control

Table 2 sweeps four prompt widths on Qwen3.5-9B.

Table 2. Qwen3.5-9B prompt-processing throughput in tokens/s. DualView recovers a stable 37.59–40.38% over canonical Q7 and remains within ±2.93% of native Q8 (Evidence E09).

Prompt tokens	Canonical Q7	DualView	Native Q8	DualView vs Q7	DualView vs Q8
128	751.728	**1055.270**	1025.210	**+40.379%**	+2.932%
256	853.873	1174.830	**1194.130**	**+37.588%**	−1.616%
512	864.976	1194.910	**1204.550**	**+38.144%**	−0.800%
2048	853.501	**1188.000**	1184.440	**+39.191%**	+0.301%



Across four prompt widths, DualView remains near the native-Q8 prefill curve while preserving the authoritative Q7 target.

At TG128, canonical Q7, DualView, and native Q8 measure 26.8419, 26.7566, and 23.9415 tokens/s respectively. DualView changes by −0.318% versus canonical Q7 and remains 11.758% faster than native Q8. The combined PP2048+TG128 metric is 302.436, 332.411, and 305.594 tokens/s respectively, placing DualView 9.911% above Q7 and 8.775% above Q8 under that mixed workload.

A served 1,846-token prompt produces the same practical pattern:

Table 3. Served Qwen control. DualView lowers TTFP by 27.91% and raises prefill by 38.89% relative to canonical Q7 while changing generation by −0.48% (Evidence E10).

Condition	TTFP (ms)	Prefill (tok/s)	Generation (tok/s)
Canonical Q7	2273.820	813.768	**26.551**
DualView (`no-output`)	**1639.249**	**1130.223**	26.423
Native Q8	1662.997	1113.830	23.528

The output-tensor-excluded mode is significant. It shows that materialization can be selective without losing the principal prefill benefit, which becomes important when live memory is the constraint.

8.4 Kernel mechanism

On a matched Qwen kernel grid (3072, 8, 1), sampled kernel time falls from 972,851 ns on canonical Q7 to 640,381 ns with DualView, a 34.18% reduction. Profiler counters show directionally fewer total instructions (−20.90%), VALU instructions (−19.31%), GL1C busy samples (−61.94%), and GL2C busy samples (−51.15%), while dual-VALU issue and LDS-conflict samples rise modestly (Evidence E11).

The kernel-time result agrees with the end-to-end sweep. The individual counters are not treated as exact event counts: the available profiler samples, and some events are sensitive to collection mode. Their role is explanatory. The view removes sub-byte reconstruction work and changes cache/instruction behavior in the direction expected from a byte-lane-native path.

8.5 RQ4: comparison with released Ornith quantizations

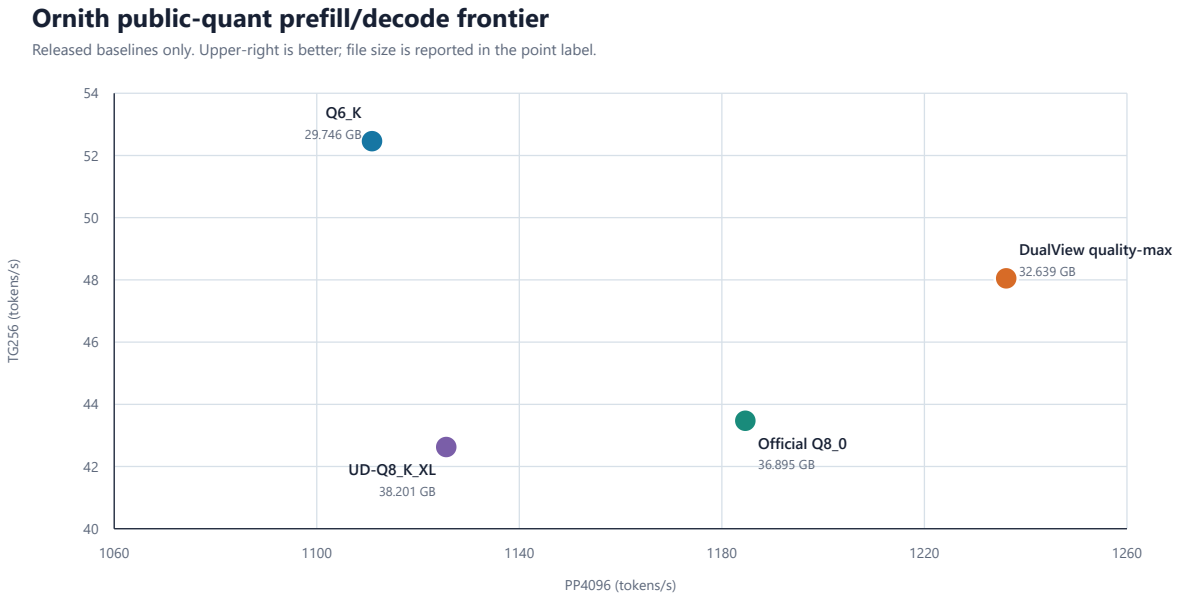
The quality-max artifact combines exact DualView execution with a separately chosen mixed-precision target. The distinction is essential:

- **DualView** changes the hardware representation of an existing Q7 tensor and is exactly value-preserving.
- **Quality islands** store selected tensors as authoritative Q8 instead of Q7 and therefore change the quantized model approximation.

The quality-max artifact retains Q8 for 30 Mamba `ssm_out` tensors and all Q/K/V/O tensors in ten full-attention blocks. All other eligible Q7 tensors can still receive exact prefill views. This yields PP4096 1236.156 and TG256 48.049. Figure 4 positions that result against released public controls.

Table 4. Released Ornith artifacts only. The panel shows a frontier rather than a universal winner: Q6_K has the highest decode, while DualView has the highest prefill and is 10.53% faster than official Q8_0 in decode (Evidence E14).

Released artifact	File (GB)	PP4096	TG256
DualView quality-max	32.639	**1236.156**	48.049
Official Q8_0	36.895	1184.626	43.470
Q6_K	29.746	1110.903	**52.458**
UD-Q8_K_XL	38.201	1125.562	42.628



DualView occupies the high-prefill region while retaining substantially more decode throughput than the released Q8-family controls. Q6_K remains the decode leader.

Relative to official Q8_0, DualView is 4.35% faster in PP4096 and 10.53% faster in TG256. Relative to Q6_K, it is 11.28% faster in prefill and 9.18% slower in decode. The correct interpretation is therefore workload-specific:

- long prompts or frequent cache misses favor the DualView point;
- very long generation with short prompts can favor the compact Q6 control;
- Q8-family targets give up decode bandwidth without consistently gaining prefill on this hardware.

That final observation is important. Q8 is not “the prefill baseline” merely because it has more bits. It is a useful native landing format, but complete artifact topology, tensor recipe, kernel choice, and file/runtime overhead all remain relevant.

8.6 Long-context end-to-end panel

The 64K-context target-only panel combines attention/KV overhead with linear layers and is less dominated by the PP4096 micro-regime.

Table 5. Same 64K prompt and runtime protocol, without MTP. The missing DualView GTT cell was not captured in the public-panel summary and is not backfilled from an incompatible run.

Released artifact	PP	TG	Wall time (s)	Live GTT (GB)
UD-Q5_K_M	**667.44**	36.49	**105.28**	29.514
DualView quality-max	652.01	**37.05**	107.50	—
UD-Q6_K_XL	647.18	36.42	108.37	34.901
UD-Q8_K_XL	619.20	33.78	113.50	41.256

At 64K context, DualView is within 2.1% of Q5_K_M wall time, slightly faster than Q6_K_XL, and 5.3% faster than Q8_K_XL.

DualView remains within 2.1% of the Q5 wall time, is 0.8% faster than Q6, and is 5.3% faster than Q8 in this panel. Its 37.05-token/s generation rate is the highest of the four, though differences among Q5/Q6/DualView are small. The result does not imply that DualView is always fastest at long context; it shows that the view's prefill benefit remains practically relevant after attention and KV work are included.

8.7 Quality behavior

8.7.1 Exact-view quality

For a fixed authoritative Q7 model, view quality is not a statistical question: Theorem 1 establishes equality of represented weights. The Qwen raw-logit tests validate the implementation. Qwen quality sweeps then check that the quantized target itself remains reasonable:

Table 6. All 290 non-overlapping 1,024-token WikiText-2 chunks were used. Intervals overlap. The table supports "no observed loss," not "Q7 is better than BF16" (Evidence E13; WikiText provenance (Merity et al. 2016)).

Qwen target	HellaSwag-1000 correct	WikiText-2 perplexity (mean ± SE)
BF16	770	7.6633 ± 0.05117
Q8	770	7.6779 ± 0.05132
Q7 / DualView	769	7.6452 ± 0.05095

BF16 and Q7 agree on 995 of 1,000 HellaSwag items. The one-point aggregate difference is smaller than what this subset can resolve reliably.

8.7.2 Mixed-precision target quality

For Ornith quality-max, promotion is graph-sensitive. A saved released-Q8 probability stream is used as an engineering reference:

Table 7. The retained topology contains full-attention Q/K/V/O and Mamba `ssm_out` islands. KLD is relative to released Q8 and is not a BF16-teacher score (Evidence E17).

Target topology	Q8-reference KLD	Relative to Q7 baseline	Perplexity	Same top-1
Q7 baseline	0.011560	—	8.475987	95.041%
Retained islands	**0.009796**	**−15.26%**	**8.461721**	**95.621%**

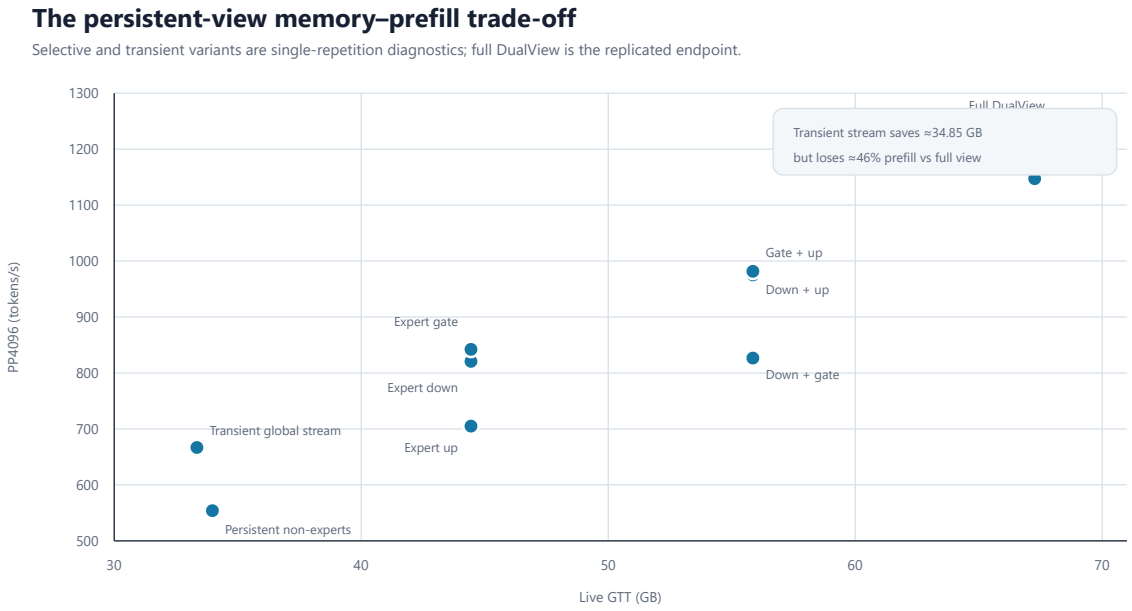
Promoting additional gate/up/shared-expert tensors often worsened KLD, while a core+down recipe improved KLD further but failed the speed target. Precision is not monotone after composition: a locally lower-error tensor can alter router scores, expert selection, activation scale, or downstream cancellation. This is why DualView's exact transform is conceptually cleaner than independent phase-specific quantization, and why quality-island selection requires full-graph evaluation.

The released quality-max artifact also records 41/148 = 27.70% pass@1 on BigCodeBench-Hard-Instruct with thinking disabled, normal `stop` termination for all 148 generations, and zero reasoning-channel leakage (Evidence E23). That result characterizes the model artifact; it does not measure the DualView transform. A HumanEval+ figure previously discussed in development is excluded from headline results because adapter-induced nontermination required adjudication and a complete independent raw-prompt rerun was not yet available (Evidence E24).

8.8 RQ5: memory cost and selective materialization

On Qwen3.5-9B, canonical Q7 uses 8,287,236,096 live GTT bytes. A view-without-output condition uses 15,634,485,248 bytes, an additional 7,347,249,152 bytes. Full view residency uses 16,716,615,680 bytes, while native Q8 uses 9,275,920,384 bytes. The measured values agree with Equation (7): persistence trades substantial capacity for execution speed.

The Ornith quality-max full-view endpoint uses 68.202 GB of live GTT. Figure 5 maps diagnostic selective and transient variants.



Persistent materialization produces a memory–prefill frontier. A transient whole-model stream recovers capacity but gives up much of the prefill benefit; expert-only persistence captures most, but not all, of the full endpoint.

The global transient stream reduces live GTT to 33.350 GB—approximately 34.85 GB below full DualView—but PP4096 falls from 1236.156 to 666.864 tokens/s, about 46%. Its generation remains Q7-like at 49.360 tokens/s. Materializing all expert views reaches 1146.984 PP at 67.263 GB; adding the roughly 0.94 GB of non-expert views reaches 1236.156 PP at 68.202 GB.

Because the selective points have one repetition, they establish a design frontier rather than precise rankings among similarly placed configurations. Two robust conclusions survive that limitation:

1. repeated host conversion or transient staging can erase the kernel advantage;
2. the final non-expert fraction is small in memory but materially important to complete-model prefill.

9 Negative results and design lessons

Academic system descriptions are distorted when only the surviving design is reported. The DualView path emerged from alternatives whose local advantages did not compose.

9.1 Direct specialized kernels

A direct Q8 WMMA route produced 2.49–6.24× isolated gains on selected matrix shapes. Routing it broadly regressed complete-model prefill by roughly 34%; restricting it to safe shapes recovered only about 0.4%. The failure demonstrates that instruction-level promise is insufficient. Launch topology, tail shapes, staging, occupancy, and routing overhead must be evaluated over the real graph.

A direct routed-MoE Q8 WMMA experiment likewise increased latency by 3.08% on Ornith shapes and 21.93% on Instella shapes. A compact task-list variant was 14.8% slower for Ornith and 0.8% slower for Instella. Avoiding apparently redundant work did not compensate for changed execution geometry.

9.2 Split and panel layouts

A split representation storing scales and codes in separate arrays was 11–51% slower across tested kernels. It improved conceptual regularity while adding address streams and damaging locality.

An exclusive “panel16” prefill layout gave attractive isolated prefill behavior, but decode regressed by 17–38% and the cache consumed 4.21875 GiB. It was superseded by a non-exclusive exact view: retain the decode-optimal canonical format and add the prefill landing format rather than forcing one layout to serve both phases.

9.3 Weight error is not graph quality

A Q8 scale modification reduced tensor-level squared error by 17.69% yet worsened full-graph logit/perplexity behavior. MoE expert probes also found locally improved approximation with changed router or downstream behavior. These failures argue against optimizing only $\|W - \hat{W}\|_F^2$. A layer’s relevant error is filtered by its activation distribution and then composed through nonlinearities, residual paths, and routing. AWQ’s activation-aware motivation is consistent with this observation (Lin et al. 2024), though the present work does not adopt AWQ as its Q7 construction method.

9.4 Bitwidth is not a performance ordering

Routed prefill probes show Q8 fastest at one small activation width, a four-bit path roughly 11% faster at another, and Q8/Q4 close at a larger width. The correct independent variables are representation, kernel, and shape—not a single “bits” scalar. DualView is useful precisely because it treats storage bitwidth and execution landing format as separable choices.

10 Limitations and threats to validity

10.1 Hardware and software scope

The implementation is validated on AMD `gfx1151` with a pinned ROCm development toolchain. The theorem is portable; the performance result depends on this device’s memory system, integer kernels, compiler, and backend implementation. CUDA, Vulkan, other RDNA generations, and discrete HBM devices require new qualification. The current public code deliberately hard-gates its tested path.

10.2 Memory capacity

Persistent views can approach 16 bits per eligible Q7 weight. DualView is most appropriate when capacity is available but prefill execution is the bottleneck. It may be a poor choice on a device that barely fits the canonical model or on a decode-dominated service with excellent packed-Q7 kernels.

10.3 Statistical scope

The primary same-weight result has clean-load replication but not a large machine population. The Qwen sweep reproduces the effect on a second released model, yet both run on one physical platform. Selective-view points are single-repetition diagnostics. Small behavioral panels are not used to claim statistical model rankings.

10.4 Teacher quality

An absolute BF16-relative Ornith quality study remains incomplete. A BF16-faithful F16 proxy audit found that 2,564 of 1,769,040 inspected values (0.145%) differ because of underflow/subnormal representation, with maximum absolute difference 2.98×10^{-8}. The intended full source-relative logit/KLD run was interrupted. Consequently, released-Q8-relative KLD is reported only as an engineering topology diagnostic, never as proof of BF16-relative quality.

10.5 Dynamic execution

Exact weight equality does not force bit-identical end-to-end sampling across all future kernels. Accumulation order, fused activation quantization, router tie boundaries, and stochastic sampling can magnify tiny arithmetic differences. Each new execution kernel therefore needs output-level validation even when its weight view is exact.

10.6 Model and workload generality

The dense Qwen control and MoE Ornith result cover two useful regimes, not all architectures. Vision encoders, multimodal projectors, convolutional blocks, very small matrices, and batch-heavy servers may have different phase boundaries. The view mechanism should be evaluated with their actual shapes.

11 Related work

Post-training quantization methods primarily improve how a high-precision model is approximated at lower precision. GPTQ uses approximate second-order information for one-shot weight quantization (Frantar et al. 2022). AWQ identifies activation-salient channels and searches protective scaling (Lin et al. 2024). SmoothQuant migrates activation difficulty into weights to enable efficient W8A8 execution (Xiao et al. 2023). DualView is complementary: after a Q7 target has been chosen, it changes the exact hardware representation used by selected phases.

Mixed-precision quantization and tensor recipes choose different *authoritative precisions* for different layers. The Ornith quality islands are an example, and their non-monotone topology sweep illustrates why full-graph calibration is needed. DualView differs because an eligible Q7 tensor remains authoritative in both phases; its Q8 view is numerically the same tensor.

I/O-aware exact algorithms such as FlashAttention show that substantial speed can come from changing data movement without approximating the mathematical operator (Dao et al. 2022). DualView applies that systems philosophy to quantized weight representation, accepting persistent capacity overhead in exchange for a native prefill landing format.

Speculative decoding accelerates generation by proposing multiple tokens and verifying them with the target model while preserving the target distribution under the appropriate sampling construction (Leviathan et al. 2023). DualView is orthogonal: it accelerates the target's prefill representation and retains compact ordinary decode. An experimental multi-token-prediction (MTP) extension is summarized in Appendix C but is not part of DualView's definition or primary evidence.

12 Conclusion

DualView starts from a narrow mathematical fact with useful systems consequences. A seven-bit signed code can be sign-extended into an eight-bit lane without changing its integer value, and an FP16 scale can be copied bit-for-bit. Therefore a packed FPX7 tensor and its Q8_0 execution view represent exactly the same quantized weights:

$$d'_g q'_i = d_g q_i.$$

That equality allows storage/decode compactness and prefill kernel compatibility to be optimized separately. The controlled Ornith ablation—+68.56% PP4096 with +0.43% TG256—and the dense Qwen sweep—+37.59–40.38% prefill versus Q7, within ±2.93% of Q8 prefill, with Q7-like decode—support the mechanism on two released architectures. Public Ornith comparisons show why it matters in practice: DualView can occupy a useful prefill/decode frontier rather than accepting either canonical Q7 prefill or native-Q8 decode as an unavoidable cost.

The price is explicit. Persistent views consume substantial live memory, up to 16 bits per eligible Q7 weight before other tensors and alignment. The method is therefore a capacity-for-throughput trade, not a free compression win. Its best use case is a high-memory system whose model fits comfortably, whose decode benefits from compact weights, and whose prefill can exploit a more regular exact representation.

The broader lesson is that **model precision and execution representation are not the same design axis**. Treating them separately creates new optimization space without requiring a second approximate model.

13 Appendix A — Reference conversion algorithm

The following pseudocode expresses the transform independently of the public implementation:

```
function q7_group_to_q8_view(q7_bytes[28], scale_bytes[2]):
    out.scale_bytes = scale_bytes          // exact byte copy

    bit_cursor = 0
    for i in 0..31:
        u = extract_7_bits(q7_bytes, bit_cursor)  // 0..127
        bit_cursor += 7

        if (u & 0x40) != 0:
            q = u - 128                    // -64..-1
        else:
            q = u                          // 0..63

        out.qs[i] = int8(q)

    return out                              // 34-byte Q8_0 block
```

The extractor must handle fields that cross byte boundaries. A branchless implementation may use an arithmetic or mask-based sign extension, but its integer result must be Equation (4). The transform must not:

- decode d to FP32 and re-encode it;
- select a new scale;
- multiply or round the weight;
- saturate to a narrower interval than int8; or
- reorder codes without changing the accompanying stride metadata.

14 Appendix B — Reproducibility checklist

Public code

- Repository: `https://github.com/ciru-ai/ROCmFPX`
- Tag: `dualview-ornith-35b-v1`
- Commit: `b515125810cd233936e52620cd73055a12825e71`
- Core files: `ggml/rocmfpx/q7-q8-view.h`, `ggml/src/ggml-cuda/q7-q8-view-cache.cuh`, `ggml/src/ggml-cuda/mmq.cu`, `ggml/src/ggml-cuda/ggml-cuda.cu`, `tests/test-q7-q8-view.cpp`.

Required controls

1. Verify the model artifact hash before each condition.
2. Record runtime commit, compiler/toolchain, GPU target, environment switches, context, batch/ubatch, KV type, and any speculative-decoding setting.
3. Use clean model loads for the causal route comparison.

4. Capture both PP and TG; prefill-only reporting cannot show whether the canonical decode advantage was retained.
5. Include native Q8 and canonical Q7 controls where memory permits.
6. Compare saved logits or all-output tensors, not generated text alone.
7. Report live memory after view construction.
8. Label operator probes, profiler counters, and one-shot selective-view points as diagnostics.

Artifact integrity

The quality-max release includes SHA-256 checksums and a tensor-level donor ledger. A reproducible mixed-precision build must validate not only tensor type counts but byte equality to each intended donor.

15 Appendix C — Orthogonal MTP extension

Multi-token prediction/speculative decoding targets the serial decode bottleneck and is conceptually separate from DualView. In an experimental Ornith integration, a target-only approximately 1K-token prompt measures 1112.85 PP and 46.71 TG, while MTP depth 6 measures 964.85 PP and 84.40 TG, a 80.7% decode increase but lower prefill. At 64K context, target-only is 652.01 PP / 37.05 TG and MTP depth 7 is 563.32 / 57.82.

The benefit depends strongly on draft acceptance and output length. At 64K prompt plus 256 output tokens, MTP is 12.89% slower in wall time because its prefill overhead is not amortized. Estimated break-even output length ranges from roughly 1,072 tokens on repetitive code to 11,259 on prose; these estimates were not independently validated.

A state-reset defect was also found during this work: pending hidden state was not cleared on slot reuse. The public validation covers uncached position-zero starts, while cached nonzero-prefix behavior remained unresolved at the freeze. For those reasons MTP is not combined with the primary DualView claim.

16 Appendix D — Evidence ledger summary

Claim	Strength	Primary evidence
Q7→Q8 view preserves every represented weight	Proof	E01–E02
Persistent eligible-weight cost is 16 bpw	Proof	E03
Upload-time construction and safe invalidation	Public implementation	E04–E05
Ornith +68.56% PP, +0.43% TG	Controlled causal	E06
Qwen +37.59–40.38% PP vs Q7	Replicated sweep	E09
Qwen Q7/view output equality	Proof + implementation validation	E12
Qwen quality shows no observed regression	Replicated quality panel	E13
Ornith public frontier and 64K behavior	Released-artifact comparison	E14–E15
Selective memory–speed curve	Single-repetition diagnostic	E16
Precision topology is non-monotone	Engineering diagnostic	E17, E19
Absolute BF16-relative Ornith quality	Unresolved; not claimed	E22
Adjudicated HumanEval+ number	Unresolved; excluded	E24

The full register gives exact local and Ciru source paths in `whitepaper/EVIDENCE.md`.

References

Advanced Micro Devices, Inc. 2024. *RDNA 3.5 Instruction Set Architecture: Reference Guide*. AMD. https://docs.amd.com/api/khub/documents/UVVZM22UN7tMUeiW_4ShTQ/content.

Chen, Mark, Jerry Tworek, Heewoo Jun, et al. 2021. “Evaluating Large Language Models Trained on Code.” *arXiv Preprint arXiv:2107.03374*, ahead of print. https://doi.org/10.48550/arXiv.2107.03374.

Ciru Inference Lab. 2026. *ROCmFPX: Quantized LLM Inference Research Runtime*. V. dualview-ornith-35b-v1. Released. https://github.com/ciru-ai/ROCmFPX.

Dao, Tri, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.” *Advances in Neural Information Processing Systems* 35. https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract.html.

DeepReinforce Team. 2026. *Ornith-1.0-35B: Agentic Coding, Open to All*. https://huggingface.co/deepreinforce-ai/Ornith-1.0-35B.

Frantar, Elias, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. “GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers.” *arXiv Preprint arXiv:2210.17323*, ahead of print. https://doi.org/10.48550/arXiv.2210.17323.

ggml.org contributors. 2026. *Llama.cpp*. Released. https://github.com/ggml-org/llama.cpp.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. 2023. “Fast Inference from Transformers via Speculative Decoding.” *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 202: 19274–86. https://proceedings.mlr.press/v202/leviathan23a.html.

Lin, Ji, Jiaming Tang, Haotian Tang, et al. 2024. “AWQ: Activation-Aware Weight Quantization for on-Device LLM Compression and Acceleration.” *Proceedings of Machine Learning and Systems* 6. https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Liu, Jiawei, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation.” *arXiv Preprint arXiv:2305.01210*, ahead of print. https://doi.org/10.48550/arXiv.2305.01210.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher. 2016. “Pointer Sentinel Mixture Models.” *arXiv Preprint arXiv:1609.07843*, ahead of print. https://doi.org/10.48550/arXiv.1609.07843.

Williams, Samuel, Andrew Waterman, and David Patterson. 2009. “Roofline: An Insightful Visual Performance Model for Multicore Architectures.” *Communications of the ACM* 52 (4): 65–76. https://doi.org/10.1145/1498765.1498785.

Xiao, Guangxuan, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. “SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models.” *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 202: 38087–99. https://proceedings.mlr.press/v202/xiao23c.html.

Zellers, Rowan, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. “HellaSwag: Can a Machine Really Finish Your Sentence?” *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4791–800. https://doi.org/10.18653/v1/P19-1472.

Zhuo, Terry Yue, Minh Chien Vu, Jenny Chim, et al. 2025. “BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions.” *International Conference on Learning Representations*. https://openreview.net/forum?id=YrycTjllL0.