The Quantization Handbook for Diffusion

A Deep Dive into Quantization Methods for Diffusion and Flow Matching Inference with Emphasis on Video Generation

This document covers classical PTQ algorithms (GPTQ, AWQ, SmoothQuant), modern rotation-based methods (QuaRot, SpinQuant), SVD-based error correction (SVDQuant, LoftQ, SVD-LLM), FP8 inference, and the unique challenges of quantizing video generation models (HunyuanVideo, Wan2.1, CogVideoX, Mochi-1, LTX-Video, LTX-2).

Version 1 — March 22, 2026

This document is evolving. If you find mistakes, missing references, or suggested fixes, please reach out at `research@moonmath.ai`.

Contents

1 Introduction

The democratization of large generative models has created an uncomfortable tension: the most capable models are often the most computationally expensive to run. A single forward pass through a modern diffusion model like Stable Diffusion XL requires billions of floating-point operations, and that pass needs to repeat dozens or hundreds of times before a final image is produced. Flow matching architectures like Stable Diffusion 3 and Flux have improved sampling efficiency, but the underlying computational burden remains substantial.

Nowhere is this tension more acute than in **video generation**. Models such as HunyuanVideo, Wan2.1, CogVideoX, Mochi-1, and LTX-Video process not just spatial but temporal information, generating coherent motion across dozens of frames. The computational requirements scale harshly: a 5-second clip at 24 fps represents 120 frames of latent content that must remain globally consistent. HunyuanVideo's full-precision transformer alone requires over 60 GB of VRAM. Without quantization, these models are inaccessible to any hardware short of multi-GPU server configurations.

Quantization - the process of reducing the numerical precision of model weights and activations - sits at the centre of making these models practical. Done well, it can cut memory consumption by 50–75 %, meaningfully accelerate inference, and open the door to deployment on consumer hardware without meaningful quality degradation. Done poorly, it introduces artefacts, collapses semantic coherence, causes temporal flickering, or simply fails to deliver the promised speedups.

This document provides a thorough technical treatment of quantization as it applies to AI inference broadly, examines the specific challenges of diffusion and flow matching architectures, and gives particular attention to recent methods - SVD-based quantization, randomized Hadamard transforms, and rotation-based outlier suppression - that have meaningfully pushed the quality frontier.

2 Quantization Fundamentals

2.1 What Quantization Actually Does

At its core, quantization maps a set of floating-point values to a lower-precision discrete representation. The most common form is *uniform affine quantization*:

$$Q(x) = \text{clamp}\Big(\text{round}\Big(\frac{x}{scale} + zero\_point\Big),\, q_{\min},\, q_{\max}\Big) \tag{1}$$

where:

- *scale* is a floating-point scalar determining the step size;
- *zero_point* is an integer offset for asymmetric ranges;
- $q_{\min}, q_{\max}$ define the representable integer range (e.g. $[0, 255]$ for `uint8`, $[-128, 127]$ for `int8`).

Dequantization is the inverse:

$$\hat{x} = \Big(Q(x) - zero\_point\Big) \times scale \tag{2}$$

The quantization error $\varepsilon = x - \hat{x}$ is what we minimize. Moving from FP32 to INT8 reduces memory by 4× but leaves only 256 discrete levels instead of ≈ 16.7 million.

2.2 Floating-Point Formats in Practice

Table 1: Common floating-point and integer formats used in inference.

Format	Bits	Exponent	Mantissa	Notes
FP32	32	8	23	Standard training precision
BF16	16	8	7	Same dynamic range as FP32; lower precision
FP16	16	5	10	Higher precision; smaller dynamic range
FP8 E4M3	8	4	3	Better precision; suited for inference
FP8 E5M2	8	5	2	Better dynamic range; suited for gradients
INT8	8	—	—	Integer; requires explicit scale/zero_point
INT4	4	—	—	Aggressive; high compression
NF4	4	—	—	Non-uniform; optimized for normal distributions

BF16 and FP16 preserve multiplicative semantics, so matrix multiplications work without explicit dequantization passes. Integer formats require quantization/dequantization steps, although hardware accelerators (NVIDIA Tensor Cores) can fuse these efficiently.

2.3 Granularity: Where Scales Are Applied

Per-tensor A single scale for the entire tensor. Maximum compression, minimum accuracy. Rarely sufficient beyond INT8.

Per-channel A separate scale per output channel. Standard for weight quantization - each output neuron can have wildly different activation magnitudes.

Per-group A scale for every G elements (typical group sizes: 32, 64, 128). Critical for 4-bit quantization where per-channel schemes lose too much information.

Per-token Computed dynamically from activations at inference time. Expensive but necessary when activation distributions vary significantly across inputs - a defining characteristic of diffusion inference.

2.4 Post-Training Quantization vs. Quantization-Aware Training

Post-Training Quantization (PTQ) applies quantization to an already-trained model with no retraining required. It breaks into two sub-categories:

- *Static PTQ*: Calibration data is used to collect activation statistics; scales are fixed before deployment.
- *Dynamic PTQ*: Activation scales are computed at runtime from actual inputs - more accurate for highly variable distributions at some runtime overhead.

Quantization-Aware Training (QAT) simulates quantization during training using straight-through estimators (STE):

$$\text{Forward:} \quad Q(x) = \text{round}(x/\mathit{scale}) \times \mathit{scale} \tag{3}$$

$$\text{Backward:} \quad \frac{\partial L}{\partial x} \approx \frac{\partial L}{\partial Q(x)} \quad \text{(straight-through)} \tag{4}$$

QAT consistently outperforms PTQ at aggressive bit-widths (INT4 and below). The cost is significant: access to training data, compute, and ideally the original training pipeline - currently impractical for most practitioners working with video generation models.

3 NVIDIA Tensor Core Throughput and Bitwidth Operations

quantization does not automatically translate into speedup. Whether a lower-bitwidth operation actually runs faster depends entirely on whether the hardware has dedicated execution units for that precision. NVIDIA's Tensor Cores are those units. Understanding their architecture and throughput characteristics is essential for choosing a quantization strategy that delivers real-world acceleration rather than memory savings alone.

3.1 What Is a Tensor Core?

A Tensor Core is a specialized matrix-multiply-accumulate (MMA) unit built directly into each NVIDIA SM (Streaming Multiprocessor). Unlike a CUDA core, which performs a single floating-point multiply-add (FMAD) per clock, a Tensor Core performs a small **matrix fragment multiplication** in a single operation:

$$\mathbf{D} = \mathbf{A} \cdot \mathbf{B} + \mathbf{C} \tag{5}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are small matrix tiles whose shape and precision depend on the Tensor Core generation. The critical insight is that the *accumulator* $\mathbf{C}/\mathbf{D}$ is always kept in a higher precision than the *multiplicands* $\mathbf{A}$ and $\mathbf{B}$. This allows aggressive input quantization without accumulating rounding error across thousands of multiply-add operations.

3.2 Tensor Core Architecture: A Schematic View

Figure 1 shows the internal dataflow of a single Tensor Core MMA operation. Four **warp-level** fragments (each held by a group of 8 threads) feed two input register files. The dot-product array computes partial products across the K-dimension and accumulates them in a higher-precision accumulator register file before writing the result tile.

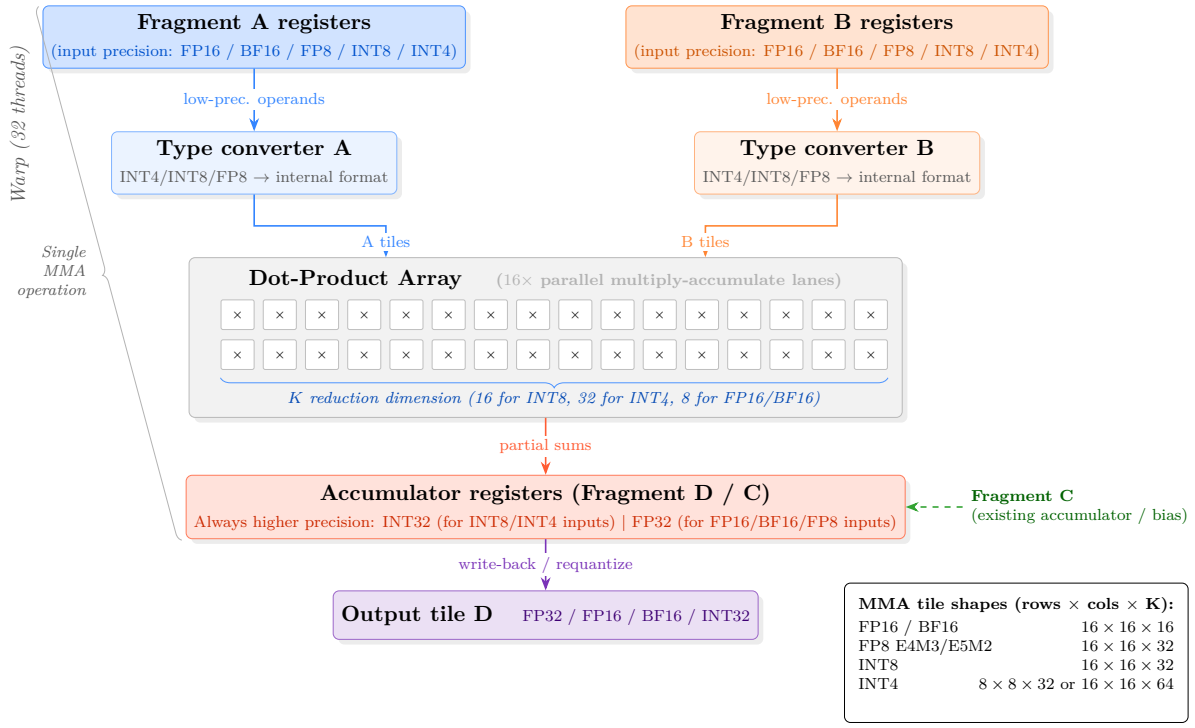

Figure 1: Schematic of a single Tensor Core MMA operation. Fragment registers **A** and **B** hold low-precision input tiles; a type-converter block expands them to the internal arithmetic format before feeding the dot-product array. The dot-product array computes K multiply-accumulate operations in parallel per lane. Results accumulate into a higher-precision Fragment D register. The tile shape (rows × columns × K-depth) varies with the input precision, explaining why k-bit quantizations can process twice as many multiply-adds per instruction as $2k$-bit.

3.3 The MMA Instruction and Tile Shapes

The fundamental unit of Tensor Core computation is the `wmma` / `mma.sync` PTX instruction, which executes one $M \times N \times K$ matrix fragment multiplication across all 32 threads of a warp. Table 2 lists the supported tile shapes by precision and GPU generation.

Table 2: Supported MMA tile shapes by precision and Tensor Core generation. M, N, K refer to the fragment dimensions; a larger K means more multiply-adds are retired per instruction, increasing throughput. Peak TFLOPS/TOPS figures are for a single H100 SXM5 GPU.

Precision	**Arch.**	M	N	K	**Acc. prec.**	**Peak (H100 SXM5)**
FP64	Ampere+	8	8	4	FP64	67 TFLOPS
TF32	Ampere+	16	16	8	FP32	989 TFLOPS (sparse)
FP16	Volta+	16	16	16	FP32 / FP16	1979 TFLOPS (sparse)
BF16	Ampere+	16	16	16	FP32	1979 TFLOPS (sparse)
FP8 E4M3	Hopper	16	16	32	FP32	3958 TFLOPS (sparse)
FP8 E5M2	Hopper	16	16	32	FP32	3958 TFLOPS (sparse)
INT8	Turing+	16	16	32	INT32	3958 TOPS (sparse)
INT4	Turing+	8	8	32	INT32	7915 TOPS (sparse)
INT1 (binary)	Turing+	8	8	128	INT32	–
Mixed-precision modes (Hopper):						
FP8 × FP16	Hopper	16	16	32	FP32	≈3958 TFLOPS
INT8 × INT4	Hopper	16	16	64	INT32	≈5000 TOPS (est.)

Note

Sparse Tensor Cores: Ampere and later architectures support **2:4 structured sparsity** - exactly 2 non-zero values per group of 4 weights. A hardware compressor removes the zero weights before the MMA instruction, halving the effective K dimension and doubling throughput. The peak figures marked "(sparse)" in Table 2 assume this 2:4 sparsity pattern. Dense throughput is exactly half these values. Combining INT4 with 2:4 sparsity gives a theoretical 8× improvement over dense FP16.

3.4 Why Larger K Means Higher Throughput

The Tensor Core dot-product array is physically sized to process a fixed number of multiply-accumulate (MAC) operations per clock cycle per SM. When the operand bitwidth decreases, the silicon that formerly stored one FP16 value can now store two INT8 values or four INT4 values. The hardware exploits this by **doubling or quadrupling the K-depth** of each MMA instruction:

- FP16 MMA: $16 \times 16 \times 16 \rightarrow 16 \times 16 \times 16 = 4096$ MACs per instruction
- INT8 MMA: $16 \times 16 \times 32 \rightarrow 16 \times 16 \times 32 = 8192$ MACs per instruction
- INT4 MMA: $16 \times 16 \times 64 \rightarrow 16 \times 16 \times 64 = 16384$ MACs per instruction

Because clock frequency and SM count are unchanged, the raw throughput in operations per second doubles with each halving of input precision - at least in theory. In practice, the theoretical peak is rarely sustained; see Section 3.7 for a roofline analysis.

3.5 Accumulator Precision: Why It Matters

The accumulator in Eq. 5 is *always* kept at a higher precision than the multiplicands. For INT8 and INT4 inputs, the accumulator is INT32; for all floating-point inputs, it is FP32. This is not optional - it is hardwired.

The implication for quantization is significant: **the numerical range of the accumulator is never the bottleneck**. A 16-element INT8 dot product has a maximum magnitude of $128^2 \times 16 = 262144$, well within INT32's range of $\approx 2.1 \times 10^9$. Overflow in the accumulator is essentially impossible for practical weight and activation magnitudes. quantization error comes from the *input* quantization step, not from the accumulator.

After the MMA instruction completes, the INT32 or FP32 accumulator result is typically:

1. Scaled and requantized back to a lower precision for the next layer (in fused kernels), or
2. Left in FP32 and written to global memory for the next operation.

The choice of output precision after each MMA is a key degree of freedom in kernel design. Keeping accumulators in FP32 throughout a transformer block and only quantizing at layer boundaries ("outer-loop quantization") is often better than requantizing at every layer.

3.6 Mixed-Bitwidth Operations

Hopper (H100) introduced native support for **mixed-input-precision MMA** instructions. Two operands of different bitwidths can be multiplied in a single instruction, with the hardware upconverting the lower-precision operand before the multiply.

FP8 weights × FP16 activations A common deployment pattern for LLM and DiT inference: weights are stored in FP8 (halving memory bandwidth) but activations are kept in FP16 (preserving dynamic range for tokens with unusual distributions). The hardware upconverts FP8 → FP16 inside the Tensor Core before the multiply. Throughput is bounded by the lower-precision operand (FP8), so this mode approaches FP8×FP8 performance. Hopper achieves FP8 2× throughput by having a wider GEMM datapath that processes twice as many FP8 elements per clock. In the same way, mixed precision achieves the same throughput by splitting one input into FP8, and multiplying the FP16 input by both of the FP8 inputs. Note that we only get this throughput if we can reuse the FP16 input, but this is always true for matrix multiplication, where each matrix element is used many times.

INT8 weights × INT4 activations Useful when activation distributions are tightly bounded (e.g., after ReLU) but weight distributions are wider. The INT4 activations are upconverted to INT8 before the multiply; the effective K-depth extends to 64, giving higher throughput than either pure INT8 or pure INT4 for weight-dominated workloads.

INT4 weights × FP16 activations (emulated) Not natively supported in a single instruction on current hardware. Typically implemented as: dequantize INT4 → FP16 (a fast table lookup or shift operation), then execute a standard FP16 MMA. This gives only

memory bandwidth savings, not compute savings. It is the mode used by GPTQ W4A16 and AWQ W4A16 deployments.

Note

Key takeaway for mixed precision: For matrix multiplication, mixed precision throughput is equal to low precision throughput, if architecture supports the low precision natively.

Table 3: Mixed-precision MMA support and effective throughput on H100. "Native" means a single hardware instruction handles both precisions. "Emulated" means the lower-precision operand is upcast in software before a standard MMA instruction.

Weights	Activations	Accum.	Support	Notes
FP8	FP8	FP32	Native (Hopper)	Full compute + memory savings
FP8	FP16	FP32	Native (Hopper)	Activation range preserved; ∼FP8 throughput
FP8	BF16	FP32	Native (Hopper)	As above
INT8	INT8	INT32	Native (Turing+)	Requires SmoothQuant or QuaRot for quality
INT8	INT4	INT32	Native (Hopper)	Useful when activations are bounded
INT4	INT4	INT32	Native (Turing+)	Aggressive; QuaRot/SpinQuant needed
INT4	FP16	FP32	Emulated	Memory savings only; GPTQ/AWQ mode
INT4	BF16	FP32	Emulated	Memory savings only
INT1	INT1	INT32	Native (Turing+)	Binary networks; extremely specialized

3.7 Roofline Analysis for Quantized DiT Inference

The roofline model characterizes whether a kernel is **compute-bound** or **memory-bandwidth-bound**:

$$\text{Attainable performance} = \min\Big(\text{Peak TFLOPS},\ I \times \text{Peak memory bandwidth}\Big) \tag{6}$$

where I (arithmetic intensity, FLOP/byte) is the ratio of compute operations to bytes loaded from memory.

For a linear layer with weight matrix $\mathbf{W} \in \mathbb{R}^{M\times N}$ and batch size B:

$$I = \frac{2BMN}{B(M+N)\cdot b_a + MN \cdot b_w} \tag{7}$$

where b_a is the activation byte-width and b_w is the weight byte-width. Table 4 shows how arithmetic intensity scales with precision and batch size for a typical DiT attention projection ($M = N = 4096$).

Table 4: Arithmetic intensity (FLOP/byte) for a 4096×4096 linear layer at different batch sizes and precision combinations on H100 SXM5 (3.35 TB/s HBM3 bandwidth, 3958 TFLOPS FP8 dense). Values below the roofline ridge point are memory-bandwidth-limited. **Ridge point** for FP8 W&A: $3958 \times 10^{12}/(3.35 \times 10^{12}) \approx 1181$ FLOP/byte.

	Batch size B					
Precision (W/A)	1	4	16	64	256	1024
FP16 / FP16	0.5	2.0	7.9	30	102	293
FP8 / FP8	1.0	4.0	15	57	188	498
INT8 / INT8	1.0	4.0	15	57	188	498
INT4 / FP16	0.9	3.8	15	57	192	540
INT4 / INT8	1.9	7.5	29	107	340	774
INT4 / INT4	2.0	7.9	30	110	350	780
Roofline ridge points (FLOP/byte): FP16: 591 | FP8/INT8: 1181 | INT4: 2362						

Note

Key insight for diffusion inference: Generative inference for image and video models almost always runs at very small batch sizes (1–4 sequences in the denoising loop). At batch size 1, *every* precision combination is deep in the memory-bandwidth-limited regime, often by 100× or more. This means:

- Weight-only quantization (reducing b_w) directly reduces memory traffic and therefore *directly* reduces latency, even without compute gains.
- Activation quantization (reducing b_a) additionally reduces the activation reload cost but has diminishing returns at batch size 1 because activations are a small fraction of total memory traffic for large weight matrices.
- True compute-bound behaviour requires batch sizes of several hundred or more - typical only in high-throughput serving with aggressive batching.

3.8 Tensor Core Generations and Practical Availability

Table 5: Tensor Core generation capabilities by NVIDIA GPU architecture. Consumer GPUs are listed where relevant to the diffusion inference use case.

Architecture	Release	Tensor Core capabilities
Volta (V100)	2017	FP16 × FP16 → FP32. First Tensor Core generation. No INT support.
Turing (T4, RTX 20xx)	2018	+ INT8, INT4, INT1 (binary) MMA. Consumer DiT inference first becomes viable at INT8.
Ampere (A100, RTX 30xx)	2020	+ BF16, TF32, FP64 MMA. Structured 2:4 sparsity for all precisions (doubles effective throughput). A100 provides the first practical FP16 video DiT inference.
Ada Lovelace (RTX 40xx)	2022	Same precision support as Ampere but significantly higher TFLOPS/TOPS per SM. RTX 4090 is the current consumer sweet spot for FP8/INT8 DiT inference. No native FP8 MMA (FP8 is decoded to FP16 on Ada).
Hopper (H100)	2022	+ **Native FP8 E4M3/E5M2 MMA** with FP32 accumulator. Mixed-precision FP8×FP16 and INT8×INT4 native instructions. 4th-gen NVLink. Transformer Engine with per-tensor dynamic FP8 scaling.
Blackwell (B100/B200, RTX 50xx)	2024/2025	+ **Native FP4 MMA** (E2M1 format, ±6 range). FP6 support. 5th-gen Tensor Cores with 2× FP8 throughput vs. Hopper. Micro-tensor scaling (per 32 × 32 block).

3.8.1 Blackwell FP4 and the Micro-Tensor Scaling Innovation

Blackwell's native FP4 (E2M1 format: 2 exponent bits, 1 mantissa bit) is a significant development for future DiT quantization. The representable values in FP4 E2M1 are highly limited: $\{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$ - only 16 values including sign. quantizing modern transformer weights to 16 levels with a naïve per-tensor or even per-group scale would be catastrophically lossy.

Blackwell addresses this with **micro-tensor scaling**: a separate scale factor is maintained for every 32×32 subtile of the weight matrix. These scales are stored in a compressed format and applied in hardware during the MMA instruction - not in software before it. The effective quantization granularity becomes extremely fine (1024 weights per scale), making FP4 viable for the first time without Hadamard preprocessing.

Combined with 2:4 sparsity, Blackwell FP4 offers a theoretical 16× improvement in arithmetic throughput over dense FP16 on the same hardware generation. For video generation models where VRAM and compute are both bottlenecks, this is transformative - though real-world gains for memory-bandwidth-limited single-image/single-video inference will be more modest, as discussed in Section 3.7.

3.9 Practical Throughput vs. Theoretical Peak

It is important to distinguish theoretical peak TOPS/TFLOPS from what inference code actually achieves. Several factors reduce effective utilization:

Memory bandwidth bottleneck As shown in Table 4, small-batch inference is almost always memory-bandwidth-limited. The Tensor Cores are underutilized because the memory system cannot feed them fast enough.

Kernel launch and synchronization overhead Small layers (e.g., linear projections with $d_{\text{model}} < 1024$) have high launch overhead relative to compute time.

Requantization cost W&A quantization requires quantizing activations before each MMA and dequantizing after. If this is not fused into the kernel, it adds memory round-trips.

Layout constraints Tensor Cores require specific matrix layouts (row-major vs. column-major, specific alignment). Data in the wrong layout incurs transposition overhead.

Occupancy limitations Keeping FP32 accumulators in registers reduces the number of active warps per SM, lowering occupancy and potentially reducing latency-hiding ability.

Table 6 gives rough practical multipliers relative to FP16 for common deployment scenarios in diffusion model inference.

Table 6: Practical latency reduction relative to FP16 for DiT / video DiT linear layer inference on H100 SXM5 at batch size 1. Values are approximate empirical estimates; actual results depend on model size, layer shape, and kernel implementation quality.

Precision (W/A)	Mode	Memory savings	Practical latency reduction
FP16 / FP16	Baseline	1.0×	1.0×
BF16 / BF16	Baseline	1.0×	∼1.0×
FP8 / FP16	W-only	2.0×	1.3–1.6×
FP8 / FP8	W&A	2.0×	1.5–2.0×
INT8 / FP16	W-only	2.0×	1.2–1.5×
INT8 / INT8	W&A	2.0×	1.5–2.2×
INT4 / FP16	W-only emul.	4.0×	1.3–1.8×
INT4 / INT8	W&A native	4.0/2.0×	2.0–3.0×
INT4 / INT4	W&A native	4.0×	2.5–3.5×
INT4 / INT4 + 2:4	Sparse W&A	8.0×	3.0–4.5×
FP4 / FP8	Blackwell	8.0×	3.5–5.0× (est.)

The gap between theoretical peak and practical throughput is largest at batch size 1 (memory-bound) and closes significantly at batch sizes of 32 or more (approaching compute-bound). For video generation serving at scale - where multiple users share a GPU and requests can be batched - the higher-precision theoretical peaks become increasingly reachable.

3.10 AMD Matrix Cores: CDNA and RDNA Architectures

AMD's equivalent of NVIDIA's Tensor Cores are called **Matrix Cores**, introduced with the CDNA architecture (MI100) in 2020 and progressively extended through CDNA2 (MI200 series), CDNA3 (MI300 series), and the consumer RDNA3/RDNA4 (RX 7000/9000 series) lines. The underlying operation is identical in spirit to NVIDIA's MMA instruction - a warp-level (in AMD terminology, a **wavefront** of 64 threads) executes a matrix fragment multiply-accumulate:

$$\mathbf{D} = \mathbf{A} \cdot \mathbf{B} + \mathbf{C} \tag{8}$$

exposed through the `MFMA` (Matrix Fused Multiply-Add) instruction on CDNA and `WMMA` (Wave Matrix Multiply-Accumulate) on RDNA. Despite the surface similarity, there are several important architectural and ecosystem differences that affect quantization in practice.

3.10.1 MFMA Instruction and Tile Shapes

AMD CDNA3 (MI300X) Matrix Cores support the tile shapes listed in Table 7. The most important difference from NVIDIA is that AMD's MFMA instructions operate on **wavefronts of 64 threads** rather than NVIDIA's 32-thread warps, which changes the mapping of matrix fragments to registers but does not fundamentally alter throughput scaling with bitwidth.

Table 7: AMD MFMA tile shapes and peak throughput on MI300X (1307 TFLOPS FP16 dense, 2614 TFLOPS FP8 dense, 2614 TOPS INT8 dense). Accumulator precision is always FP32 for floating-point inputs and INT32 for integer inputs.

Precision	**Arch.**	M	N	K	**Peak (MI300X)**
FP64	CDNA2+	4	4	4	163 TFLOPS
FP32	CDNA3	16	16	4	653 TFLOPS
TF32	CDNA3	16	16	8	653 TFLOPS
FP16	CDNA+	16	16	16	1307 TFLOPS
BF16	CDNA2+	16	16	16	1307 TFLOPS
FP8 E4M3	CDNA3	16	16	32	2614 TFLOPS
FP8 E5M2	CDNA3	16	16	32	2614 TFLOPS
INT8	CDNA+	16	16	32	2614 TOPS

The tile shapes and throughput scaling ratios are broadly similar to NVIDIA Hopper with FP8 and INT8 delivering 2× the FP16 throughput. The MI300X also benefits from its **unified memory architecture**: 192 GB of HBM3 shared between the GPU compute dies and the CPU complex, which significantly relaxes the VRAM pressure that drives aggressive quantization in the first place. A HunyuanVideo model that requires INT4 on an H100 (80 GB) can run comfortably in BF16 on an MI300X.

3.10.2 FP8 Implementation Differences

Both NVIDIA Hopper and AMD CDNA3 support FP8 E4M3 and E5M2 formats with a 32-element K-depth MMA instruction. However, the **scaling mechanism** differs in a

practically important way:

NVIDIA (Transformer Engine) Uses a *delayed scaling* strategy: a history of per-tensor activation maxima is maintained across steps, and the scale for the next forward pass is derived from this history. The Transformer Engine API abstracts this behind `fp8_autocast`. Dynamic per-token scaling is also supported via explicit scale tensor arguments to the `cublas` FP8 GEMM API.

AMD (ROCm / hipBLAS) As of ROCm 6.x, FP8 GEMMs are supported through `hipBLASLt` with explicit per-tensor scale arguments passed at call time. There is currently no equivalent of NVIDIA's Transformer Engine delayed-scaling infrastructure. Practitioners must implement their own scaling logic, typically using per-tensor dynamic scaling computed from the absolute maximum of each activation tensor.

The absence of a high-level Transformer Engine equivalent on ROCm is the primary practical friction point for FP8 DiT inference on AMD hardware. Projects such as **vLLM with ROCm support** and **MLC-LLM** have filled part of this gap, but the ecosystem maturity for FP8 image/video diffusion models on AMD lags NVIDIA by roughly one to two tool generations as of early 2026.

3.10.3 FP4 and Sub-8-Bit Quantization on AMD

AMD CDNA4 (MI355X) supports FP4 and FP6 MFMA natively, but the software ecosystem for INT4 quantized DiT inference on ROCm is less mature than on CUDA. Specifically:

- **GPTQ** kernels (Exllama, ExllamaV2) have ROCm ports but typically achieve lower utilization than their CUDA counterparts due to differences in the warp/wavefront size and register file layout.
- **AWQ** has partial ROCm support via AutoAWQ; the GEMM kernels use a software dequantization path rather than native INT4 MFMA in most configurations.
- **bitsandbytes** has an experimental ROCm backend as of version 0.43, but NF4 double-quantization is not yet hardware-accelerated.
- **torchao** INT4/FP8 kernels are CUDA-specific; ROCm support is under active development but not yet production-ready for DiT inference pipelines.

Hadamard rotation methods (QuaRot, SpinQuant) are in principle hardware-agnostic - the rotation preprocessing is a one-time offline operation and the resulting quantized weights are consumed by standard GEMM instructions. However, the fused online Hadamard transform for activations (applied inside the linear layer kernel) requires custom kernel implementations. These exist for CUDA but not yet for ROCm.

3.10.4 RDNA3/RDNA4: Consumer AMD GPUs

AMD's consumer RDNA3 (RX 7900 XTX) and RDNA4 (RX 9070 XT) GPUs expose matrix operations through the `WMMA` instruction set rather than the server-grade `MFMA`. Key differences from CDNA:

- **WMMA tile size**: $16 \times 16 \times 16$ for FP16/BF16, matching NVIDIA's Volta-era Tensor Cores. Smaller tiles mean higher instruction dispatch overhead relative to

compute for the same total matrix size.

- **INT8/INT4**: Supported in RDNA3 via WMMA with reduced throughput multipliers compared to CDNA3. RDNA4 improves INT8 throughput significantly, with reported 2× improvement in dense INT8 over RDNA3 per compute unit.
- **No native FP8 on RDNA3**: FP8 is a CDNA3/RDNA4 addition. RX 7000 series cards must emulate FP8 via FP16, yielding memory savings but no compute throughput gain.
- **FP8 on RDNA4**: The RX 9070 XT (RDNA4) adds native FP8 support, bringing it to parity with Ada Lovelace (RTX 40xx) for FP8 throughput - though Ada also only decodes FP8 to FP16 internally, so neither architecture achieves the full 2× FP8 compute gain that Hopper and CDNA3 provide.

3.10.5 Summary: AMD vs. NVIDIA for quantized DiT Inference

Table 8: Qualitative comparison of AMD and NVIDIA hardware and software ecosystems for quantized diffusion and video generation inference. Ratings reflect the state of the ecosystem as of early 2026. ✓✓ = strong, ✓ = adequate, ∼ = partial / in progress, × = not available.

Capability	NVIDIA (H100/4090)	AMD (MI300X/7900XTX)
FP16 / BF16 inference	✓✓	✓✓
FP8 hardware support	✓✓ (Hopper native)	✓✓ (CDNA3 native)
FP8 software ecosystem	✓✓ (Transformer Engine)	∼ (hipBLASLt, manual scaling)
INT8 W&A inference	✓✓	✓ (less optimized kernels)
INT4 weight-only (GPTQ/AWQ)	✓✓	∼ (partial ROCm ports)
INT4 W&A native MMA	✓✓ (Hopper+)	∼ (CDNA3 hardware, limited SW)
Hadamard rotation (QuaRot)	✓✓ (fused CUDA kernels)	∼ (offline rotation only)
NF4 / bitsandbytes	✓✓	∼ (experimental)
torchao support	✓✓	∼ (in development)
VRAM capacity (flagship)	80 GB (H100 SXM5)	192 GB (MI300X)
Memory bandwidth	3.35 TB/s (H100 SXM5)	5.3 TB/s (MI300X)
Diffusion library support	✓✓ (Diffusers, ComfyUI)	✓ (ROCm Diffusers fork)

The central conclusion for practitioners is that **AMD hardware is competitive or superior on raw matrix throughput and memory capacity**, but the software ecosystem for sub-8-bit and rotation-based quantization lags NVIDIA by a meaningful margin. For BF16 or FP8 weight-only inference of large video models, MI300X is an excellent choice - its 192 GB unified memory pool removes the VRAM constraint that makes quantization necessary on NVIDIA hardware for models like HunyuanVideo. For aggressive INT4 or QuaRot-based quantization targeting maximum throughput, NVIDIA CUDA tooling remains the path of least resistance in early 2026, though the gap is closing as ROCm matures.

4 Classical PTQ Algorithms

4.1 GPTQ: Layer-Wise Second-Order Quantization

GPTQ [1] adapts the Optimal Brain Compression framework to transformer scale. For a linear layer with weight matrix $\mathbf{W}$ it minimizes the layer-wise output error given calibration data $\mathbf{X}$:

$$\arg\min_{\mathbf{Q}} \|\mathbf{W}\mathbf{X} - \mathbf{Q}(\mathbf{W})\mathbf{X}\|_F^2 \tag{9}$$

using a second-order approach based on the Hessian $\mathbf{H} = 2\mathbf{X}\mathbf{X}^\top$. Weights are quantized column by column and residual error is compensated via:

$$\delta W_j = -\frac{W_{q,j} - W_j}{\left[\mathbf{H}_{FF}^{-1}\right]_{jj}} \cdot \mathbf{H}_{Fj}^{-1} \tag{10}$$

GPTQ achieves good 4-bit quantization with per-group scaling (group size 128) and serves as a backbone for many subsequent methods.

4.2 AWQ: Activation-Aware Weight Quantization

AWQ [2] protects salient weight channels by searching for per-channel scaling factors $\mathbf{s}$ that reduce quantization error without changing the mathematical output:

$$Q\Big(\mathbf{W} \cdot \mathrm{diag}(\mathbf{s})^{-1}\Big) \cdot \mathrm{diag}(\mathbf{s}) \cdot \mathbf{x} \;\approx\; \mathbf{W} \cdot \mathbf{x} \tag{11}$$

When $s_j > 1$ for a salient channel, weight values become larger and thus better resolved by quantization. AWQ is typically faster to apply than GPTQ and generalizes well across architectures.

4.3 SmoothQuant: Migrating Quantization Difficulty

SmoothQuant [3] addresses activation outliers by migrating quantization difficulty from activations to weights via a mathematically equivalent transformation:

$$\mathbf{Y} = \Big(\mathbf{X} \cdot \mathrm{diag}(\mathbf{s})^{-1}\Big) \cdot \Big(\mathrm{diag}(\mathbf{s}) \cdot \mathbf{W}\Big) = \tilde{\mathbf{X}} \cdot \tilde{\mathbf{W}} \tag{12}$$

Scale selection balances the migration:

$$s_j = \frac{\max(|X_j|)^\alpha}{\max(|W_j|)^{1-\alpha}}, \quad \alpha = 0.5 \text{ (default)} \tag{13}$$

SmoothQuant enables INT8 W&A quantization that leverages dedicated integer MAC units for genuine throughput gains.

4.4 SpQR: Sparse-Quantized Representation

SpQR [4] handles outlier weights explicitly: most weights are quantized to 3–4 bits, but a small fraction of high-sensitivity weights are stored in FP16 in a sparse format. The sparse high-precision component is small enough not to significantly increase memory while recovering substantial accuracy.

4.5 GGUF Quantization Variants

For local deployment, GGUF k-quants use block quantization with super-block scales (a form of recursive quantization):

- **Q4_K_M**: 4-bit with 6-bit for half the attention layers - good balance.
- **Q5_K_M**: Significantly better quality with modest size increase.
- **Q6_K**: Near-lossless; recommended when memory allows.
- **Q8_0**: Essentially lossless reference format.

5 Modern Quantization Methods: Rotation and Decomposition

5.1 The Outlier Problem, Precisely Stated

Large transformer models develop persistent activation outliers: specific hidden dimensions that consistently produce values an order of magnitude larger than typical channels. In DiT-based image and video models, similar outliers appear with an additional complication - their magnitude varies across timesteps, making static suppression methods less effective.

If one channel reaches magnitude 100 while typical channels are magnitude 1, a per-tensor INT8 scale must accommodate 100, leaving typical channels resolved to only ≈ 2.5 levels out of 128 - catastrophic precision loss.

5.2 QuaRot: Random Hadamard Rotation for Outlier Elimination

QuaRot [6] is one of the most significant recent advances in PTQ. Its core observation: a **random orthogonal rotation** of the weight and activation spaces preserves the mathematical output of a linear layer while spreading outliers across all dimensions, effectively eliminating them.

If $\mathbf{R}$ is a random orthogonal matrix:

$$\mathbf{W}\mathbf{x} = (\mathbf{W}\mathbf{R})(\mathbf{R}^\top \mathbf{x}) = \tilde{\mathbf{W}}\tilde{\mathbf{x}} \tag{14}$$

The computation is identical, but $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$ and $\tilde{\mathbf{x}} = \mathbf{R}^\top \mathbf{x}$ have fundamentally different statistical properties. Outlier energy is distributed across all dimensions, making uniform quantization far more effective.

5.2.1 The Randomized Hadamard Transform

The practical choice is the **randomized Hadamard transform** (RHT) rather than a fully general random orthogonal matrix. The Hadamard matrix $\mathbf{H}_n$ (for n a power of 2) is defined recursively:

$$\mathbf{H}_1 = [1], \qquad \mathbf{H}_{2n} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_n & \mathbf{H}_n \\ \mathbf{H}_n & -\mathbf{H}_n \end{bmatrix} \tag{15}$$

With a random diagonal sign matrix $\mathbf{D}$ (each entry ± 1 with equal probability):

$$\mathbf{R} = \mathbf{H}_n \cdot \mathbf{D} \tag{16}$$

Key properties of the RHT:

1. **Orthogonal**: $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$ - the transform is exactly lossless.
2. **Fast**: Applying it costs $\mathcal{O}(n \log n)$ via the Fast Walsh–Hadamard Transform.
3. **Outlier-spreading**: With overwhelming probability, no single dimension concentrates outlier energy.
4. **Hardware-friendly**: No learned parameters; reproducible given a seed.

QuaRot absorbs the rotation into adjacent weight matrices, so it adds **zero runtime overhead** - it is purely a weight preprocessing step:

$$\text{Original:} \quad \mathbf{y} = \mathbf{W}_2(\mathbf{W}_1 \mathbf{x}) \tag{17}$$

$$\text{Rotated:} \quad \tilde{\mathbf{W}}_1 = \mathbf{W}_1 \mathbf{R}, \quad \tilde{\mathbf{W}}_2 = \mathbf{R}^\top \mathbf{W}_2 \tag{18}$$

$$\mathbf{y} = \tilde{\mathbf{W}}_2(\tilde{\mathbf{W}}_1 \mathbf{x}) \quad \text{(mathematically identical)} \tag{19}$$

QuaRot enables near-lossless 4W8A and makes 4W4A quantization viable for the first time at scale. On LLaMA-2-70B it achieves 4W4A with perplexity degradation of ≈ 1.0 points over FP16, versus ≈ 5.0 for GPTQ alone.

5.3 QuIP#: Incoherence Processing with Lattice Codebooks

QuIP# [9] combines rotation-based incoherence processing with vector quantization using the E_8 lattice - the densest known lattice packing in 8 dimensions:

1. **Incoherence processing**: Apply random Hadamard transforms to both rows and columns of each weight matrix, making entries approximately i.i.d. Gaussian.
2. E_8 **lattice codebooks**: quantize groups of 8 weights jointly to the nearest lattice point, providing ≈ 1.4 dB gain over scalar quantization.
3. **Lookup table decoding**: Groups of 8 weights are decoded together from a compact index, enabling efficient dequantization.

QuIP# achieves **2-bit quantization** with quality that surpasses GPTQ at 4 bits in several benchmarks.

5.4 SpinQuant: Learned Rotation Optimization

SpinQuant [7] learns optimal rotation matrices through differentiable optimization on the Cayley manifold (the manifold of orthogonal matrices):

$$\min_{\mathbf{R} \in O(n)} \mathcal{L}_{\text{quant}}(\text{model}, \mathbf{R}) \tag{20}$$

The Cayley parameterization ensures $\mathbf{R}$ stays exactly orthogonal:

$$\mathbf{R}(\mathbf{A}) = (\mathbf{I} + \mathbf{A})(\mathbf{I} - \mathbf{A})^{-1}, \quad \text{where } \mathbf{A} \text{ is skew-symmetric} \tag{21}$$

SpinQuant outperforms QuaRot at 4-bit because the learned rotation better adapts to the specific outlier structure of each model. The tradeoff is that optimization infrastructure and calibration data are required.

Table 9: Comparison of rotation-based quantization methods.

Method	Rotation	Learned	Runtime overhead	Quality
QuaRot	Random Hadamard	No	Zero	Very good
SpinQuant	Learned orthogonal	Yes	Zero	Excellent
SmoothQuant	Per-channel scale	Yes	Minimal	Good (INT8 only)

5.5 SVDQuant: Low-Rank Decomposition for Error Correction

SVDQuant [10] (and related approaches including LoftQ [11] and SVD-LLM [12]) addresses quantization from an entirely different angle: rather than making weights easier to quantize, it **explicitly models and corrects the quantization error** using a low-rank decomposition.

The key observation: the quantization error matrix $\mathbf{E} = \mathbf{W} - Q(\mathbf{W})$ often has low effective rank. If this error can be represented compactly, it can be corrected without storing the full-precision weights.

5.5.1 Basic Formulation

Given $\mathbf{W} \in \mathbb{R}^{m \times n}$ and its quantized approximation $Q(\mathbf{W})$:

$$\mathbf{E} = \mathbf{W} - Q(\mathbf{W}) \tag{22}$$

$$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\top] = \text{SVD}(\mathbf{E}) \tag{23}$$

$$\mathbf{W}_{\text{corrected}} = Q(\mathbf{W}) + \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^\top \tag{24}$$

Memory analysis for $\mathbf{W} \in \mathbb{R}^{4096 \times 4096}$ with INT4 and rank-32 correction:

- INT4 weights: $4096 \times 4096 \times 0.5$ bytes $= 8.0$ MB
- Rank-32 correction: $(4096 \times 32 + 32 \times 4096) \times 2$ bytes ≈ 0.5 MB
- Total overhead: $\approx 6\%$ memory increase for significant quality improvement.

5.5.2 SVD-LLM: Data-Aware Truncation

SVD-LLM [12] weights the SVD by the activation covariance to preferentially preserve error components that matter most for model outputs:

$$\mathbf{E}_{\text{weighted}} = \mathbf{C}^{1/2}\mathbf{E} \tag{25}$$
$$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\top] = \text{SVD}(\mathbf{E}_{\text{weighted}}) \tag{26}$$
$$\mathbf{W}_{\text{corrected}} = Q(\mathbf{W}) + \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top\mathbf{C}^{-1/2} \tag{27}$$

5.5.3 LoftQ: SVD for LoRA Initialization

LoftQ [11] combines SVD correction with LoRA initialization for fine-tuning scenarios:

```
# LoftQ: alternating quantization and SVD
for _ in range(num_iters):
    residual = W - dequantize(quantize(W - B @ A))
    U, S, Vt = torch.linalg.svd(residual, full_matrices=False)
    A = Vt[:rank, :]
    B = U[:, :rank] * S[:rank]
    W_q = quantize(W - B @ A)
# Final: W ~= dequantize(W_q) + B @ A
```

Listing 1: LoftQ iterative procedure.

5.6 AQLM: Additive Quantization with Learned Codebooks

AQLM [13] uses **additive codebooks** - multiple codebooks whose codewords are summed - to represent weight vectors:

$$\mathbf{w}_{\text{row}} \approx \mathbf{C}_1[i_1] + \mathbf{C}_2[i_2] + \cdots + \mathbf{C}_m[i_m] \tag{28}$$

This is analogous to residual vector quantization (RVQ). AQLM achieves remarkable 2-bit and even lower effective bit-width compression with quality that challenges 4-bit scalar quantization.

5.7 Rotation + SVD: Combined Approaches

A possible novel direction for quantization can be a combined approach. For example, **QuaRot + SVD** applies the Hadamard rotation first, then uses SVD to correct residual quantization error:

1. Apply RHT: $\tilde{\mathbf{W}} = \mathbf{W} \cdot \mathbf{R}$
2. quantize: $\tilde{\mathbf{W}}_q = Q(\tilde{\mathbf{W}})$
3. Compute error: $\mathbf{E} = \tilde{\mathbf{W}} - \tilde{\mathbf{W}}_q$
4. SVD correction: $\mathbf{E} \approx \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top$
5. Final: $\tilde{\mathbf{W}}_{\text{corrected}} = \tilde{\mathbf{W}}_q + \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top$

To our knowledge, there is currently no published work which implements this approach.

6 Quantization for Diffusion Models

6.1 The Iterative Nature of Diffusion Inference

A diffusion model runs the same network dozens to hundreds of times during a single generation. Key implications:

1. **Latency amplification**: quantization-induced quality degradation compounds across steps. An error at step t influences all subsequent denoising steps.
2. **Step-dependent activation distributions**: The input to the network changes dramatically across timesteps - activation statistics are *not stationary* across forward passes, violating a fundamental assumption of static PTQ.
3. **Classifier-free guidance**: Standard inference runs the network twice per step (conditioned and unconditioned). Activation distributions for these two passes differ substantially.

6.2 Timestep-Dependent Activation Statistics

Non-stationarity is arguably the biggest challenge for diffusion model quantization. Solutions include:

Timestep-aware calibration Sample calibration data uniformly across timesteps. Used in Q-Diffusion [14].

Per-timestep scales Separate activation quantization scales for different timestep ranges.

Dynamic activation quantization Compute activation scales on-the-fly from actual activations.

Variance-aware timestep grouping Cluster timesteps by activation statistics and assign one scale per cluster.

Hadamard preprocessing Rotation-based methods eliminate the outliers that make timestep-specific scaling necessary - by spreading outlier energy, the effective dynamic range becomes more uniform across timesteps.

6.3 Attention Mechanism Quantization

Softmax saturation At certain conditioning signals, attention weights approach 0 or 1 - a bimodal distribution difficult to quantize uniformly.

Flash Attention compatibility FlashAttention-2/3 fuses operations in a way that complicates mid-computation quantization. Most deployments quantize input/output projections but keep the attention computation itself in FP16/BF16.

Hadamard in attention QuaRot applies the Hadamard transform within the attention mechanism. For Q and K specifically, the rotation is transparent to attention scores:

$$\mathbf{Q}\mathbf{K}^\top = (\mathbf{Q}\mathbf{R})(\mathbf{K}\mathbf{R})^\top = \mathbf{Q}(\mathbf{R}\mathbf{R}^\top)\mathbf{K}^\top = \mathbf{Q}\mathbf{K}^\top \tag{29}$$

This enables 4-bit quantization of Q and K with no approximation in the attention pattern.

7 Video Generation: The Frontier Challenge

7.1 Video Architectures in Brief

3D U-Net / factored attention (AnimateDiff, early models): Separate spatial and temporal attention. Relatively straightforward to quantize.

Full 3D DiT (HunyuanVideo, Wan2.1, CogVideoX, Mochi-1): Joint space-time attention over all tokens. A 5-second clip at 720p may produce $\approx 115\,200$ spatial-temporal tokens.

Causal video DiT (LTX-Video): Causally masked attention enables streaming generation and changes quantization dynamics.

Video flow matching (most recent models): Extension of image flow matching with 3D positional encodings and temporal attention.

7.2 Temporal Consistency and Quantization

Temporal consistency is the most perceptually sensitive quality dimension for video generation and is disproportionately vulnerable to quantization error.

Sources of temporal inconsistency:

1. **Activation outlier variation across frames**: Different temporal positions may trigger different activation outliers. A static activation scale calibrated on an average will underfit high-outlier frames and overfit low-outlier frames, manifesting as texture flicker.
2. **Attention pattern sensitivity**: quantization error in keys or queries distorts which temporal positions attend to which, breaking long-range temporal dependencies.
3. **Residual stream accumulation**: quantization error injected at layer l affects all subsequent layers and all temporal positions simultaneously.

Mitigation strategies:

Per-frame activation scales Compute separate activation quantization parameters per temporal position during inference.

Temporal-aware calibration Use video clips (not single frames) so that temporal attention patterns are represented.

Rotation preprocessing QuaRot/SpinQuant applied to video DiTs addresses temporal flicker at its root - by eliminating outliers, activation magnitudes become more uniform across temporal positions.

7.3 HunyuanVideo Quantization

HunyuanVideo [16] (13B parameters, > 60 GB VRAM at full precision) has motivated several quantization strategies:

Table 10: HunyuanVideo memory requirements by quantization level.

Precision	VRAM	Hardware
FP16 (original)	≈ 60 GB	Multi-GPU only
BF16 with offload	≈ 35 GB	Single A100
FP8 E4M3	≈ 30 GB	Single A100 or dual 3090
INT8 weight-only	≈ 20 GB	Single 3090/4090
GGUF Q5_K_M	≈ 17 GB	Single 3090/4090
GGUF Q4_K_M	≈ 14 GB	Single 3090/4090
INT4 + SVD	≈ 12 GB	RTX 4080 / 3080 Ti

Community testing has established that:

- The double-stream MM-DiT blocks are more sensitive to quantization than single-stream blocks; keeping double-stream at INT8 while single-stream is INT4 improves quality.
- Temporal attention layers are more sensitive than spatial attention layers. INT4 spatial + INT8 temporal preserves temporal coherence at lower memory cost than uniform INT8.
- The text encoder should not be quantized below INT8 for production use.

7.4 Wan2.1 and CogVideoX

Wan2.1 [20] uses a 3D full-attention flow matching DiT. Community results indicate that GGUF Q5_K_M preserves temporal coherence well for clips under 5 seconds, while Q4_K_M introduces subtle flickering in high-detail regions under sustained motion. Rotation-based preprocessing significantly improves Q4 quality but requires offline preprocessing time.

CogVideoX [17] uses an explicit temporal attention factorization (spatial and temporal attention applied in sequence rather than jointly). This unexpectedly helps quantization: temporal attention layers are a smaller fraction of parameters and can be kept at higher precision at modest memory cost.

7.5 Mochi-1 and LTX-Video

Mochi-1 [18] uses an asymmetric architecture with separate encoders for video and text fused in the transformer. quantization at the fusion points (cross-attention between video and text tokens) is particularly sensitive and benefits strongly from SmoothQuant or AWQ preprocessing.

LTX-Video-1,2 [19, 23] use a causal masking scheme that enables streaming generation. The causal structure means activation statistics for tokens at position t depend on all

previous positions $0, \ldots, t-1$. A calibration set should include sequences of varying length to capture this positional variation. Community results indicate that they use QuaRot-based quantization.

7.6 Practical Calibration for Video Models

A proper calibration dataset for video model quantization should:

1. **Cover the full denoising trajectory**: Sample noise levels from $T = 0$ to $T = T_{\max}$.
2. **Include diverse temporal dynamics**: Fast motion, slow motion, static scenes, and scene cuts.
3. **Match the target generation length**: Calibrate at the same number of frames you will generate.
4. **Include both CFG branches**: Conditioned and unconditioned passes.
5. **Minimum recommended size**: 50–100 video clips, 5+ denoising timestep samples per clip.

8 Measuring Quantization Quality in Video

8.1 Metrics

FID (Fréchet Inception Distance): Distribution-level quality. Necessary but insufficient - can miss temporal artefacts.

FVD (Fréchet Video Distance [21]): Uses I3D features that capture temporal dynamics. Critical for video evaluation.

CLIP Score Text-video alignment. Measures whether semantic content is preserved under quantization.

LPIPS per-frame Pixel-level comparison between full-precision and quantized outputs.

Warp error Measures temporal consistency under estimated optical flow.

DOVER-Mobile Perceptual video quality metric calibrated on human preferences.

EvalCrafter Composite benchmark including motion quality, action correctness, and temporal coherence.

8.2 Human Evaluation Protocol

Blind A/B tests should include separate assessments of:

- Static frame quality
- Motion smoothness
- Temporal consistency (“does anything flicker?”)
- Semantic faithfulness (“does this match the prompt?”)

9 Comprehensive Method Comparison

9.1 Method Summary Table

Note

Quality legend: Excellent (imperceptible), Very Good (< 1% task metric degradation), Good (perceptible but acceptable), Moderate (noticeable degradation), Poor (significant degradation). Throughput gains are estimates and highly hardware/batch-size dependent.

Table 11: Comprehensive quantization method comparison.

Method	Bits (W/A)	Calibration	Runtime overhead	Mem. vs FP16	Throughput	Quality	Video suitability
BF16/FP16	16/16	None	None	1.0×	1.0×	Baseline	Excellent
FP8 E4M3 (W-only)	8/16	None	Minimal	∼0.5×	1.2–1.5×	Near-lossless	Excellent
FP8 E4M3 (W&A)	8/8	Small	None	∼0.5×	1.5–2.0×	Near-lossless	Very Good
INT8 weight-only	8/16	Small	Minimal	∼0.5×	1.2–1.4×	Near-lossless	Very Good
INT8 W&A (SmoothQuant)	8/8	Medium	None	∼0.5×	1.5–2.5×	Very Good	Good
INT8 W&A (QuaRot)	8/8	Medium	Zero	∼0.5×	1.5–2.5×	Excellent	Very Good
GPTQ INT4 (g=128)	4/16	Medium	Dequant	∼0.27×	1.2–1.6×	Good	Moderate
AWQ INT4	4/16	Small	Dequant	∼0.27×	1.2–1.6×	Good	Moderate
GGUF Q4_K_M	∼4.5/16	None	CPU-friendly	∼0.30×	CPU viable	Good	Moderate
GGUF Q5_K_M	∼5.5/16	None	CPU-friendly	∼0.35×	CPU viable	Very Good	Good
INT4 + SVDQuant	4/16+SVD	Medium	SVD matmul	∼0.29×	1.1–1.5×	Very Good	Good
QuaRot INT4 W&A	4/4	Medium	Zero	∼0.27×	1.8–2.5×	Good	Good
SpinQuant INT4 W&A	4/4	Med.+opt.	Zero	∼0.27×	1.8–2.5×	Very Good	Good
QuIP# 2-bit	2/16	Large	Lattice dec.	∼0.14×	1.0–1.2×	Moderate	Poor
NF4 (bitsandbytes)	4/16	None	Dequant	∼0.27×	1.0–1.2×	Good	Moderate
FP8 + QuaRot	8+rot/16	Medium	Zero	∼0.5×	1.5–2.0×	Excellent	Excellent
INT4 + SVD + QuaRot	4+rot/16	Large	SVD matmul	∼0.29×	1.3–1.8×	Very Good	Very Good

9.2 Cost-Quality Tradeoff by Use Case

Table 12: Recommended quantization strategy by deployment scenario.

Use case	Recommended method	VRAM target	Notes
Production datacenter (image)	FP8 W&A + QuaRot	H100	Best throughput/quality
Production datacenter (video)	FP8 W-only or INT8 W&A	A100/H100	Temporal consistency critical
Consumer GPU high-quality (image)	FP8 or INT8 W-only	16–24 GB	RTX 4090 / 3090
Consumer GPU high-quality (video)	INT8 W-only + temporal INT8	24 GB	HunyuanVideo / Wan2.1
Consumer GPU budget (image)	GGUF Q5_K_M or GPTQ INT4	8–12 GB	Acceptable quality
Consumer GPU budget (video)	GGUF Q4_K_M or INT4+SVD	12–16 GB	Some temporal artefacts
LoRA fine-tuning + inference	NF4 + LoftQ	12–24 GB	QLoRA compatible
CPU inference	GGUF Q4_K_M	RAM	No GPU required
Research / ablation	INT4 W&A QuaRot / SpinQuant	16+ GB	Best quality at 4-bit

10 Advanced Topics

10.1 Quantization of LoRA Adapters

QLoRA-style [5] Base model frozen in NF4; adapters in BF16:

$$\mathbf{W}_{\text{effective}} = \text{dequantize}(\mathbf{W}_{\text{NF4}}) + \alpha\,\mathbf{BA} \tag{30}$$

LoftQ SVD-initialized quantized base + LoRA provides a better starting point than arbitrary quantization.

Adapter-aware calibration quantize the base model with the LoRA attached during calibration so scales adapt to the effective weight distribution.

10.2 Speculative Decoding Analogs

Asymmetric timestep quantization Use aggressive quantization (INT4 or FP8) for early denoising steps (structural decisions) and higher precision (INT8 or BF16) for late steps (fine detail). The switching adds minimal overhead and preserves fine detail where it matters.

Consistency model drafting A consistency model generates a rough sample in 1–4 steps; the full model refines for 4–8 steps. The consistency model can be aggressively quantized (INT4) since its output is refined anyway.

10.3 Knowledge Distillation with Quantization

For scenarios where PTQ quality is insufficient, quantization-aware distillation guides a quantized student with a full-precision teacher:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda \cdot \text{KL}\Big(p_{\text{teacher}} \,\|\, p_{\text{student, quantized}}\Big) \tag{31}$$

For video models, intermediate feature matching (attention maps, hidden states) is more stable than output-level distillation.

10.4 On-Device and Edge Deployment

Core ML (Apple Silicon) INT4 and INT8 via `coremltools`. ANE (Apple Neural Engine) well-supports INT8; INT4 is partial.

ONNX Runtime Broad hardware support. quantization via `onnxruntime-tools` with static INT8.

Qualcomm AI Stack INT8 and INT16 on Hexagon DSP. Video models are currently too large for mobile inference but next-generation sub-2B parameter models may change this.

11 Conclusion

quantization for diffusion and flow matching models - and especially for video generation - is a rapidly evolving discipline where the state of the art has shifted dramatically even in the past year. The classical toolkit (GPTQ, AWQ, SmoothQuant) remains valuable and widely deployed, but it has been significantly extended and in some cases superseded by rotation-based methods (QuaRot, SpinQuant) and decomposition-based corrections (SVDQuant, LoftQ, SVD-LLM) that attack the outlier problem more directly.

For practitioners today, the decision hierarchy is roughly:

1. **Start with FP8** if you have H100/4090 hardware. Near-lossless and increasingly well-tooled.
2. **Use INT8 weight-only** if you need broader hardware compatibility. Still near-lossless on most models.
3. **Apply Hadamard preprocessing + INT4** when memory is the hard constraint and you can afford preprocessing time.
4. **Add SVD correction** when INT4 quality is insufficient but memory prevents higher bit-width.
5. **Use GGUF Q5_K_M** for CPU/mixed inference where tooling simplicity matters.

For video specifically: protect temporal attention layers, calibrate on video data (not just images), measure FVD not just FID, and consider asymmetric precision across timestep phases. The temporal consistency dimension is uniquely sensitive to quantization and uniquely important to users.

The trajectory is clear: as hardware support for sub-8-bit formats matures and rotation-based preprocessing becomes standard in tooling, the quality gap between quantized and full-precision video generation will continue to narrow. Models that require 60 GB today will be running with high quality on consumer hardware within two to three years - not through architectural downsizing, but through increasingly principled quantization.

References

[1] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," *arXiv:2210.17323*, 2022.

[2] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han, "AWQ: Activation-aware weight quantization for LLM compression and acceleration," *arXiv:2306.00978*, 2023.

[3] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," *arXiv:2211.10438*, 2022.

[4] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh, "SpQR: A sparse-quantized representation for near-lossless LLM weight compression," *arXiv:2306.03078*, 2023.

[5] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," *arXiv:2305.14314*, 2023.

[6] S. Ashkboos, A. Mohtashami, M. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "QuaRot: Outlier-free 4-bit inference in rotated LLMs," *arXiv:2404.00456*, 2024.

[7] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi, L. Theis, and V. Chandra, "SpinQuant: LLM quantization with learned rotations," *arXiv:2405.16406*, 2024.

[8] J. Chee, F. Cai, V. Kuleshov, and C. De Sa, "QuIP: 2-bit quantization of large language models with guarantees," *arXiv:2307.13304*, 2023.

[9] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa, "QuIP#: Even better LLM quantization with Hadamard incoherence and lattice codebooks," *arXiv:2402.04396*, 2024.

[10] M. Li, Y. Lin, Z. Zhang, T. Cai, X. Li, J. Guo, E. Xie, C. Meng, J Zhu, and S. Han "SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models," *arXiv:2411.05007*, 2024.

[11] Y. Li, Y. Yu, C. Liang, P. He, N. Karampatziakis, C. Chen, and W. Chen, "LoftQ: LoRA-fine-tuning-aware quantization for large language models," *arXiv:2310.08659*, 2023.

[12] X. Wang, Y. Zheng, P. Wan, and Y. Wang, "SVD-LLM: Truncation-aware singular value decomposition for large language model compression," *arXiv:2403.07378*, 2024.

[13] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Borzunov, and D. Alistarh, "Extreme compression of large language models via additive quantization," *arXiv:2401.06118*, 2024.

[14] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Keutzer, S. X. Hu, and K. Keutzer, "Q-Diffusion: Quantizing diffusion models," *arXiv:2302.04304*, 2023.

[15] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-training quantization on diffusion models," in *Proc. IEEE/CVF CVPR*, 2023.

[16] W. Kong *et al.*, "HunyuanVideo: A systematic framework for large video generation models," *arXiv:2412.03603*, 2024.

[17] Z. Yang *et al.*, "CogVideoX: Text-to-video diffusion models with an expert transformer," *arXiv:2408.06072*, 2024.

[18] Genmo Team, "Mochi 1: A family of open video generation models," Technical report, Genmo, 2024.

[19] Lightricks, "LTX-Video: Efficient video generation via latent-space synthesis," Technical report, Lightricks, 2024.

[20] Wan Team, "Wan: Open and advanced large-scale video generative models," Technical report, 2025.

[21] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "FVD: A new metric for video generation," in *ICLR 2019 Workshop*, 2019.

[22] P. Esser *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," *arXiv:2403.03206*, 2024.

[23] Lightricks, "LTX-Video 2: A Distilled, Quantisation-Aware Open Video Generation Model," Technical report, Lightricks, 2025. `https://huggingface.co/Lightricks/LTX-Video-2`