RESEARCH · APPLE METAL · VIDEO DIFFUSION

Fast 3D Convolutions for Local Video Generation on Apple Silicon

From a custom Metal matrix kernel to the video decoder of stable-diffusion.cpp

Exe — Exe AI, developer studio

Measured on Apple M2 Max (38-core GPU, 64 GB unified memory)

34.3×
faster video decode
LTX-2.3 · 384×384
377.07 s → 10.99 s

≥14.5×
faster video decode
LTX-2.3 · 1280×704
>1,360 s → 93.8 s

11,671
GFLOP/s, our own conv3d kernel, best of five padding taps counted · median 11,290

±0 GB
change in peak memory
output visually identical
PSNR 44–45 dB

REFERENCE TASK (BOTH REPORTS)

Video model	LTX-2.3 22B distilled 1.1 (Q8_0)
Text encoder	Gemma 3 12B QAT
Output	384×384 · 121 frames · 25 fps with audio
Sampling	4 Euler steps · seed 42
Baseline	official stable-diffusion.cpp build, master-1000
Rule	one run at a time on the GPU

1 The problem: one operator, three quarters of the time

- Stack: stable-diffusion.cpp (C++ engine) → ggml tensor library → Apple Metal.
- Diffusion transformer: runs well. Video decoder: the bottleneck.

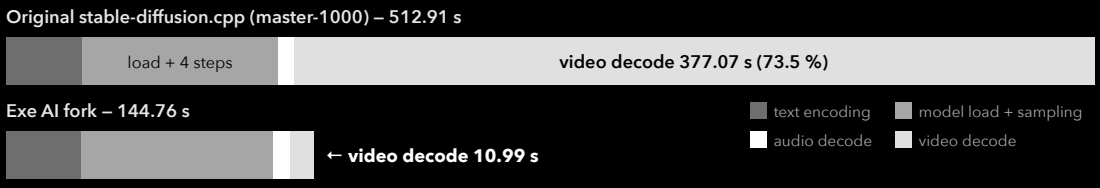


Figure 1. Time per phase on the reference task, same scale. Engine logs; text encoding includes loading Gemma.

- LTX-2.3 decoder: 42 causal 3D convolutions · 3×3×3 kernels · 48 to 4,096 channels · 44 TFLOP per decode.
- In ggml: each convolution is a `GGML_OP_CONV_3D` node, served on Metal by `kernel_conv_3d`.
- That kernel: one thread per output value · serial walk over the receptive field · no matrix instructions.

2 The research path

1. **Profile the pipeline**
   Where the time goes · Section 1
2. **Measure the ceiling of existing tools**
   Apple MPSGraph and MLX `conv3d` · Figure 4
3. **Design our own kernel: `conv3d_mat`**
   Implicit GEMM on 8×8 simdgroup matrices with a shared halo tile · Section 3
4. **Build a complete decoder on it**
   LTX-2.3 VAE decoder in Metal · Section 3
5. **Duel on the real decoder shapes**
   ggml vs. `conv3d_mat` vs. ccv (BSD-3), same data, double-precision reference · Section 4
6. **Precision study**
   Half storage with a float accumulator · Section 5
7. **Integrate into ggml's Metal backend**
   Gated second path with full fallback · Section 6
8. **Remove the assumed grid limit**
   65,535 threadgroups per grid dimension · Section 7

3 Our own kernel: `conv3d_mat`

Implicit GEMM, no im2col

- GEMM per output row (batch, depth, height): M = output positions along the width · N = output channels · K = taps × input channels.
- im2col: would copy the input into a matrix 27 times its size.
- conv3d_mat: computes the GEMM directly from the input volume.

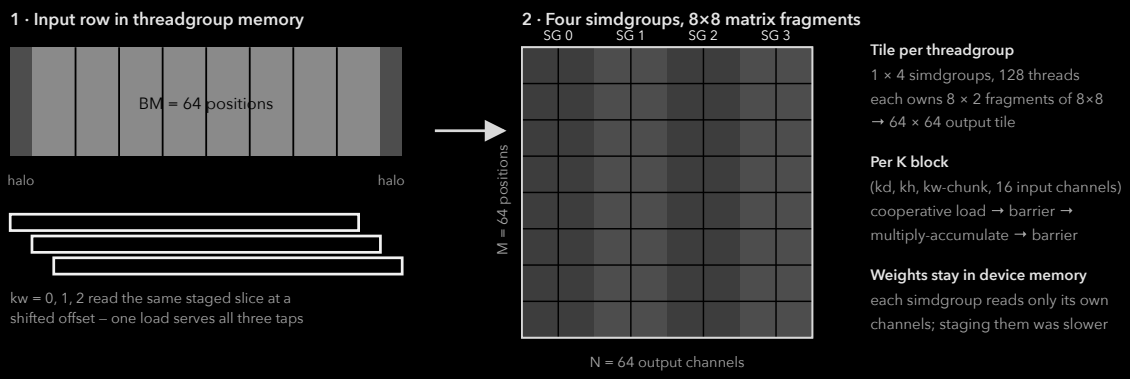


Figure 2. Structure of `conv3d_mat`. One threadgroup: a 64×64 tile (output positions × output channels) of one output row, walking the (kd, kh) rows of the input volume. Padding, ragged channels and ragged edges: zero-filled during the load.

Four design decisions

DECISION	MECHANISM	EFFECT
Halo tile	One input slice of 64 + KW − 1 positions per threadgroup; taps kw = 0, 1, 2 read it at shifted offsets	One device load feeds three multiply-accumulate passes
Weights not staged	Each simdgroup reads only its own output channels, straight from device memory	Staging was slower: more threadgroup memory, fewer resident groups
Unrolled fragment loops	16 accumulator fragments per simdgroup stay in registers	Rolled loop: dynamic accumulator indexing, 5× slower
Float accumulator for half	`conv3d_mat_h`: half storage for input, weights and output; half × half products; float sum	Up to 27 × C_{in} = 27,648 terms per output: too many for a half accumulator

Decoder-style chain

- Workload: 42 layers with the LTX-2.3 decoder's layer sequence and channel counts · latent 15×24×32 · output 120×192×256 (frames × height × width) · 0.365 PFLOP per pass, padding taps included.
- Difference to the real decoder: the three upsampling convolutions run at the upsampled resolution.
- Error of `conv3d_mat_h` (shape 4×64×64, 256→256, CPU reference): for outputs of magnitude ≥ 8 within half a unit in the last place of the half output (0.062 at values around 128; relative ≤ 4.9×10^{-4}); near zero up to 1,057 half-ULP, an absolute 7.9×10^{-6}.

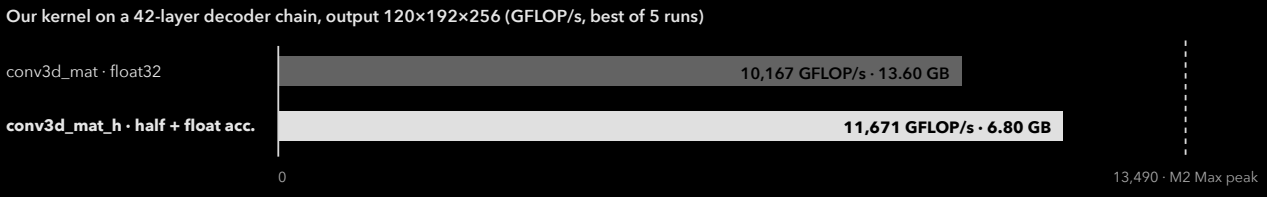


Figure 3. Five runs each: float32 10,132–10,167, half 11,262–11,671 GFLOP/s (median 11,290); full-tap count, padding included. Half storage: +15 % speed, half the memory; accumulator float.

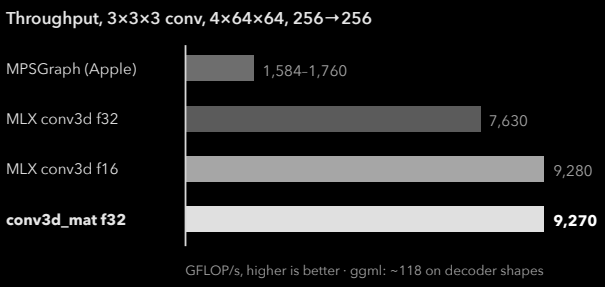


Figure 4. The MLX bars and the `conv3d_mat` bar come from laboratory notes without a raw log; the MPSGraph range was measured on other shapes. Runs with a raw log and a CPU reference in the same run: `conv3d_mat` float32 8,485–8,675 GFLOP/s, maximum absolute deviation ≤ 5×10^{-4}.

A whole decoder on `conv3d_mat`

- LTX-2.3 VAE decoder (and encoder) written directly on Metal.
- All 42 convolutions on `conv3d_mat`, plus normalisation, SiLU and residual paths.

DECODER SHAPE	TIME	MEMORY
384×384, 121 frames	**8.14 s**	4.1 GB
1280×704, 361 frames	119–120 s	41.1 GB

- Standalone, outside the generation pipeline.
- 1280×704×361 (14.4 s at 25 fps): 6,728 GFLOP/s sustained over the whole decode (laboratory notes; no raw log, as for the two rows above).

4 The duel: real decoder shapes, three kernels

Candidates	ggml `kernel_conv_3d` · our `conv3d_mat` · `Conv3DKernel` from ccv by Liu Liu (BSD-3), the library behind the commercial Draw Things app
Data	the ten distinct convolutions of the LTX-2.3 decoder, identical inputs for every kernel
Timing	five runs, first discarded · GPU timestamps · median



Figure 5. Kernel time only (calls × median). Layout conversions, measured separately: +1.3 % for `conv3d_mat`, +1.6 % for ccv.

DECODER SHAPE (D×H×W, $C_{IN}\to C_{OUT}$)	CALLS	GGML	CONV3D_MAT	CCV F32
16×12×12, 128→1024	1	117	1,578	3,015
16×12×12, 1024→1024	4	116	1,632	3,207
16×12×12, 1024→4096	1	116	1,676	3,132
31×24×24, 512→512	4	116	3,283	6,684
31×24×24, 512→4096	1	118	3,312	6,669
61×48×48, 512→512	9	119	6,408	6,547
121×48×48, 256→256	12	119	6,441	6,714
121×48×48, 256→512	1	119	6,446	6,636
121×96×96, 128→128	8	119	6,391	8,762
121×96×96, 128→48 (output layer)	1	119	**4,231**	3,730

- Unit: GFLOP/s per shape.
- 61- and 121-frame layers at 48×48 (22 of 42 calls): `conv3d_mat` within 2–4 % of ccv.
- Output layer: `conv3d_mat` ahead.
- Ahead, ccv: deep low-resolution layers (16×12×12, 31×24×24 — the 64-position tile spans a row of only 12 or 24 positions and computes mostly padding) and 121×96×96, 128→128.

Correctness

CHECK	CONV3D_MAT AND CCV	GGML
Max. absolute error vs. double-precision CPU reference, 3×8×8, 32→32	3.2×10^{-5}	3.8×10^{-5}
Same, 4×12×12, 128→128	1.6×10^{-4}	1.8×10^{-4}
Full-size shapes, the three float32 kernels against each other (sums over up to 27,648 terms in a different order)	within 3×10^{-3}	

5 Precision: half storage needs a float accumulator

CCV HALF PATH	ACCUMULATOR	DEVIATION	SPEED
Original	half	up to 19 in absolute value, ≈ 1 % of the output magnitude	fastest in the duel, 62.7× ggml
Repaired with the `conv3d_mat_h` rule	float	3.4×10^{-5} – 2.1×10^{-4} (float32 level)	+2.5 to +5.9 % vs. float32; −0.8 to +5.1 % including the cast of activations to half (57.5 ms per decode)

- M2 Max: half precision does not double the arithmetic rate; half storage alone does not pay for the extra conversion.

Decision

- Engine kernel: ccv float32 `Conv3DKernel` – the fastest *correct* kernel on the real shapes, 19 % ahead of `conv3d_mat` over the whole decoder, mature, permissively licensed.
- conv3d_mat: same weight layout ([OC][KD][KH][KW][IC]), so it can run behind the same gate at any time.

6 Into the engine: a safe second path in ggml

- stable-diffusion.cpp: untouched. The change lives only in ggml's Metal backend, in front of the existing kernel.
- Outside the gate: the original code runs unchanged.
- A/B switch: `GGML_METAL_CONV3D_CCV=0` turns the new path off.
- Command-line tools: same switches and output as the upstream release – a drop-in replacement.



Figure 6. Four GPU dispatches per convolution; cost of the transposes: Figure 5.

Concurrency	ggml encodes operators concurrently, so a shared scratch buffer would race · three temporary buffers per node through ggml's own buffer-size hook (the mechanism used for flash attention)
Determinism	two runs produce byte-identical video
Gate	stride and dilation 1 · no temporal padding inside the convolution · square odd spatial kernel · batch 1 · input channels divisible by 8 · contiguous tensors · every tensor below 2^{32} elements (32-bit indexing)
Coverage	all ten LTX-2.3 decoder shapes pass the gate
Tests	ggml `test-backend-ops`, Metal vs. CPU backend: **290 of 290** cases pass, new path on and off

7 A finding about Metal: there is no 65,535 grid limit

- First integration: refused any dispatch with more than 65,535 threadgroups in one grid dimension (the limit known from CUDA).
- 1280×704: the largest decoder layer needs 121 × 176 × 10 = 212,960 threadgroups along one axis → the heaviest layers silently fell back to the slow kernel.
- Apple's Metal Feature Set Tables: grid limits only for mesh and object shaders, none for compute dispatches.
- API: `threadgroupsPerGrid` is three 32-bit unsigned integers.

Result

- On hardware: **236,918 threadgroups** along one axis run correctly through the fast path and pass the backend test suite.
- Fix: limit raised to the API type; a separate element-count check guards the kernel's 32-bit indexing.

8 End-to-end results

- Runs: full generation, same models, prompt, seed and settings · peak memory from `/usr/bin/time -l`.

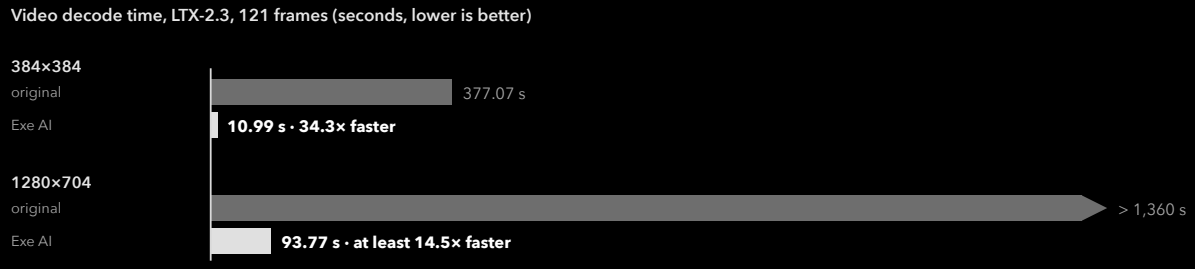


Figure 7. 1280×704: the original kernel was still decoding when the run was stopped after 1,360 s of decode (from the run notes; the stopped log carries no timestamps); the true factor is higher than 14.5×.

	VIDEO DECODE	TOTAL	PER STEP	PEAK MEMORY	PSNR
384×384 · original master-1000	377.07 s	512.91 s	10.64 s	44.6 GB	reference
384×384 · Exe AI	**10.99 s**	**144.76 s**	10.54 s	44.7 GB	44.25 dB
1280×704 · original kernel	> 1,360 s	–	–	43.5 GB[1]	reference
1280×704 · Exe AI	**93.77 s**	**516.53 s**	81–84 s	43.5 GB	45.02 dB

- [1] From a separate complete run at lower priority (`nice 5`).
- 384×384: total 3.5× shorter; time per step and peak memory unchanged.
- 1280×704: the video decode is no longer the dominant cost – the four sampling steps are.

Same picture



original kernel · frame 90 | Exe AI · frame 90 | difference, amplified 10×

Figure 8. 1280×704, frame 90 of 121, shown in greyscale. Maximum per-pixel difference 24 of 255 levels; lowest frame PSNR 43.91 dB. Cause: floating-point summation order over 42 chained convolutions. For scale: the upstream engine's own automatic flash-attention selection moves the same pipeline by 42.79 dB between runs (a 9-frame probe).

9 Where it applies

- Scope: every model in stable-diffusion.cpp whose VAE decoder uses 3D convolutions.
- Measured: LTX-2.3 (42 convolutions per decode) · Wan 2.2 TI2V (303) · Krea 2, an image model with a 3D decoder over a single frame (30).
- Not affected: Qwen-Image — 0 calls, its VAE is two-dimensional.
- Not measured here: LTX-2.5, Wan 2.1, LingBot-Video, HunyuanVideo 1.5, Anima, Ming-Image.

10 Limitations and next steps

OPEN POINT	STATUS
Older chips	Measured on M2 Max only · both kernels need only the Apple7 GPU family (M1 and later) · M1 run pending
Very large videos	Tensors of 2^{32} elements or more (≈ 4K at 121 frames) fall back · splitting along the depth axis would lift this
Encoder and upscaler	RGB input layers, strided convolutions and the LTX latent upscaler outside the gate · not on the critical path of text-to-video
Our own kernel in the engine	Per-shape dispatch of `conv3d_mat` (same weight layout, same gate), first for the output layer · half variant halves activation memory
Against Draw Things (384×384)	Per sampling step on par: 10.54 s vs. 11.50 s · total: 122.58 s (warm file cache; 144.76 s in Section 8) vs. 73.65 s · memory: 44.7 GB vs. 27.2 GB · gap in model loading and text encoding, not in the decoder

Licences and acknowledgements

COMPONENT	LICENCE	ROLE
stable-diffusion.cpp, leejet and contributors	MIT	engine, unchanged
ggml, the ggml authors	MIT	Metal backend, gated path added
ccv / NNC, `lib/nnc/mfa` Conv3DKernel, © Liu Liu	BSD 3-Clause	integrated kernel generator; derived files keep the original notice
`conv3d_mat`, standalone decoder, integration, measurements, grid-limit analysis	Exe AI	own work
GPL-licensed code	—	none used

References

1. leejet et al., *stable-diffusion.cpp*. github.com/leejet/stable-diffusion.cpp
2. *ggml* tensor library. github.com/ggml-org/ggml
3. Liu Liu, *ccv / NNC*, Metal kernels in `lib/nnc/mfa`. github.com/liuliu/ccv
4. Apple, *Metal Feature Set Tables*. developer.apple.com/metal/Metal-Feature-Set-Tables.pdf
5. Apple, *MTLDispatchThreadgroupsIndirectArguments*. developer.apple.com/documentation/metal
6. Apple, *MLX*, `mlx/backend/metal/conv.cpp`. github.com/ml-explore/mlx
7. Lightricks, *LTX-2.3*. huggingface.co/Lightricks/LTX-2.3