RESEARCH · APPLE METAL · MEMORY AND SPEED

Memory and Speed Together: Optimising Local Video Generation on Apple Silicon

Winograd convolution inside ggml · text encoder released after use · split GPU command buffers · several agents on one GPU

Exe — Exe AI, developer studio

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

30.2 s
video decode 1280×704, Winograd in ggml, one run; native-layout kernel under the same switches: 46.88 s (1.55×)

2.85×
Winograd against the best direct kernel, 42 decoder calls: 1.58 s against 4.50 s

−14.8 GB
peak memory with te=disk
LTX-2.3 · 384×384
44.6 GB → 29.8 GB

74.9 s → 1.3 s
longest GPU command buffer, 1280×704 sampling step, after splitting

SETUP

Reference task	as in Report 1 (cover)
Test file	`ltx-2.3-22b-distilled-1.1-Q8_0-F16.gguf` (diffusion model 22,516 MB in the log); the pinned `…-1.1-Q8_0.gguf` was no longer present · the tables of Report 1 are comparable only after a re-run
Goal	memory and speed together · ccv (BSD-3) as the speed yardstick · memory levers without code first
Rules	every change in our ggml fork · stable-diffusion.cpp never edited · peak throughput against Apple's 13.49 TFLOPS for the M2 Max

1 Memory without code

RESIDENT UNTIL THE END OF A REFERENCE RUN (ONE RELEASE IN THE LOG)	MB
Gemma 3 12B QAT	11,924
Embedding connectors	2,205
Diffusion model	22,516
Two VAE parts	339 + 777
Released at the end	**37,761**

1.1 `--max-vram` fails on unified memory

- Hypothesis: a VRAM budget evicts Gemma and the connectors after text encoding.
- Observed with `--max-vram 24`: auto-fit puts all parameters in a RAM backend and streams them to the GPU segment by segment (diffusion model staging: 40.19 s); nothing released before the end.

RUN	PEAK FOOTPRINT	GENERATE_VIDEO	SAMPLING	DECODE	VIDEO VS REFERENCE
reference	44.58 GB	148.15 s	92.19 s	11.51 s	–
`--max-vram 24`	**62.98 GB**	184.00 s	129.48 s	11.37 s	PSNR 99.00 dB, identical

- Cause (from log and `src/model_manager.cpp`, about lines 1857–2018; the two copies were not measured separately):
  - eviction only for *reloadable* states: disk residency, or a compute backend different from the parameter backend;
  - only when a capacity check fails, between graph-cut segments;
  - unified memory: RAM copy and staged copy draw on one pool → +18.4 GB, +37 s.

1.2 `--params-backend te=disk` works

- Mechanism: marks the text-encoder module as disk-resident (`diffusion_engine.cpp:323`) → Gemma and the connectors become evictable.
- Log: both released right after text encoding, before the diffusion model is prepared; release at the end 23,632 MB instead of 37,761 MB.
- Side effect: auto-fit off (`diffusion_engine.cpp:916`); buffers and placement identical to the reference.

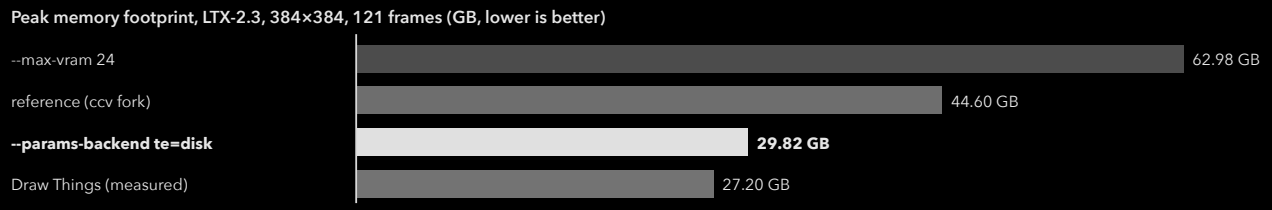


Figure 1. Peak footprint (`/usr/bin/time -l`). Reference: first repeat run (second: 44.64 GB); `--max-vram`: from its own 44.58 GB reference pair; Draw Things: its own measured run.

384×384 RUN	PEAK	GENERATE_VIDEO	STEPS 2 / 3 / 4 (S/IT)	DECODE
first measurement, reference	44.58 GB	148.15 s	10.62 / 10.60 / -	11.51 s
first measurement, te=disk	29.78 GB	131.56 s	11.98 / 12.00 / -	11.46 s
repeat, reference 1	44.60 GB	138.21 s	10.74 / 10.70 / 10.60	11.09 s
repeat, te=disk 1	**29.82 GB**	141.77 s	10.60 / 10.60 / 10.63	10.83 s
repeat, reference 2	44.64 GB	141.42 s	10.61 / 10.61 / 10.63	10.77 s
repeat, te=disk 2	**29.82 GB**	144.13 s	10.61 / 10.61 / 10.61	10.72 s

- Order: alternating reference / te=disk.
- Compute buffers: identical in all six runs (ltxav 466.68 MB, ltx_video_vae 4,381.31 MB).
- Video, reference 1 vs. te=disk 1: PSNR 99.00 dB on every frame – identical pixels.
- Reproduced: the memory gain, in both repeat pairs (−33 %).
- Not reproduced (noise): 11.98 s/it and the 16.6 s shorter total of the first pair – warm file cache (Gemma loaded in 4.4 s, condition graph 14.7 s, against 27 s and 36.7 s).
- Repeats: totals equal within ±3 s (te=disk 0.3–3.6 s slower) → no measurable time cost at 384×384.
- Distance to Draw Things: 2.6 GB.

1.3 At 1280×704

1280×704	PEAK FOOTPRINT	GEMMA + CONNECTORS RELEASED	VAE COMPUTE BUFFER	VIDEO	GENERATE_VIDEO
reference	43.12 GB	at end only	26,766 MB	–	478.44 s

1280×704	PEAK FOOTPRINT	GEMMA + CONNECTORS RELEASED	VAE COMPUTE BUFFER	VIDEO	GENERATE_VIDEO
te=disk	**29.45 GB**	after text encoding	26,766 MB	PSNR 99.00 dB, 121 frames identical	**invalid (Section 1.3)**

- Memory: −13.67 GB (−31.7 %); compute buffers unchanged.
- Time: not valid. A laboratory job held the GPU lock at the same time (incident 3, Section 13). The +58 s (steps 3–4 at 101–106 vs. 81.5–82.2 s/it) count neither as a cost nor as a non-cost.
- Clean pair at this size: only with the native-layout kernel (Section 7).

2 The matrix ceiling

Probe	`metal/mma_ceiling.mm`, register-only loop, no memory traffic
Grid	2,432 threadgroups of four simdgroups · 2,048 iterations · six timed dispatches · one run under the GPU lock
Operation	the 8×8 simdgroup matrix multiply-accumulates of our convolution kernels, on registers filled once

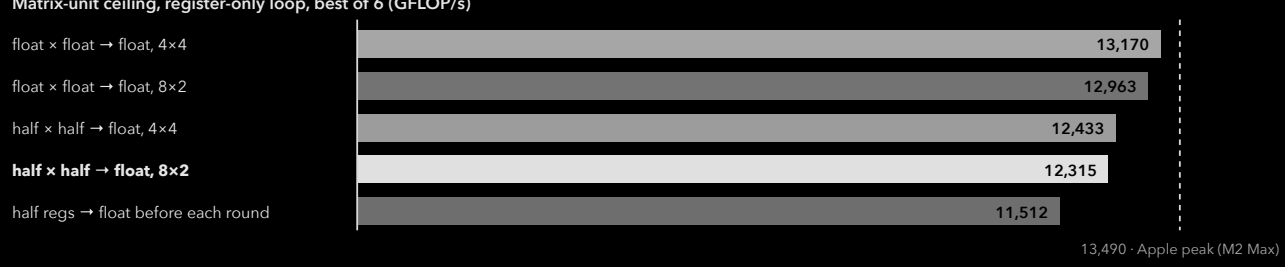


Figure 2. Share of Apple's 13,490 GFLOP/s: float 4×4 97.6 %, float 8×2 96.1 %, half 4×4 92.2 %, half 8×2 91.3 %, half-to-float conversion before each round 85.3 %. Repeat of half 8×2: 12,307.

- Half operands are about 5 % slower than float – for the 8×2 and the 4×4 tile and for eight simdgroups per threadgroup.
- Ceiling, half operands with float accumulation: 12.3–12.4 TFLOP/s; the best chain of Report 1 was at 95 % of it.
- 13,000 GFLOP/s: out of reach with half operands; float operands reach 13.0–13.2 TFLOP/s.
- Rule: float operands into the matrix unit; half only for memory and staging, converted once at staging time.

3 A plane-tiled kernel

- Problem: `conv3d_mat` tiles 64 positions along *one* output row; idle positions at widths 12 / 24 / 48 / 96: 52 / 40 / 16 of 64 and 32 of 128 (two tiles).
- Idea: at stride 1, the (oh, ow) plane of one output frame is a 1D convolution over the sequence of padded rows, row length S = W + 2·pad; tap (kh, kw) reads position f + kh·S + kw.
- Tile: 64 consecutive values of f, across row breaks; the kw − 1 border columns of each row are computed, not stored.
- Waste: 2 of 14 / 26 / 50 / 98 positions.
- Code: new file `metal/conv3d_flat.metal` · `conv3d_flat` (float32) and `conv3d_flat_h` (half storage, float accumulator) · data layout of `conv3d_mat` (channel-last, weights [OC][KD][KH][KW][IC]), which stays unchanged.
- Layout-independent: needs only output positions in sequence with a fixed row pitch – contiguous in the ggml layout, where W is the fastest index.

The ten real shapes, before and after

Bench	`metal/shapes_bench.mm` · the ten distinct LTX-2.3 decoder convolutions (3×3×3, padding 0/1/1, D+2 input frames)
Method	both kernels alternating in *one* process (run `shapes_count1`, set B) · best of three · GPU time per command buffer
Check	before every timing, 600 sampled outputs (borders included) against a double-precision CPU reference
Tile	`conv3d_flat_h` WM1 WN8 FM8 FN2 BKC64, PAD 8, 16-byte staging
Count	full-tap GFLOP/s (Section 6)

#	SHAPE (D×H×W, C_{IN}→C_{OUT})	CALLS	GFLOP / CALL	MAT_H MS	MAT_H GFLOP/S	FLAT_H MS	FLAT_H GFLOP/S	SPEED-UP
1	16×12×12, 128→1024	1	16.3	9.10	1,792	2.35	6,926	3.9×
2	16×12×12, 1024→1024	4	130.5	86.87	1,502	18.87	6,916	4.6×
3	16×12×12, 1024→4096	1	521.8	354.35	1,473	71.21	7,328	5.0×
4	31×24×24, 512→512	4	252.8	78.93	3,203	28.05	9,011	2.8×
5	31×24×24, 512→4096	1	2,022.1	631.72	3,201	231.63	8,730	2.7×
6	61×48×48, 512→512	9	1,989.5	324.45	6,132	217.09	9,165	1.5×
7	121×48×48, 256→256	12	986.6	153.78	6,416	106.91	9,228	1.4×

#	SHAPE (D×H×W, C_{IN}→C_{OUT})	CALLS	GFLOP / CALL	MAT_H MS	MAT_H GFLOP/S	FLAT_H MS	FLAT_H GFLOP/S	SPEED-UP
8	121×48×48, 256→512	1	1,973.2	306.51	6,438	214.04	9,219	1.4×
9	121×96×96, 128→128	8	986.6	149.85	6,584	103.68	9,516	1.4×
10	121×96×96, 128→48 (output layer)	1	370.0	83.28	4,442	**118.79**	**3,115**	**0.7×**
	Sum of 42 calls	42	44,074	**8.012 s**	5,501	**4.892 s**	**9,010**	**1.64×**

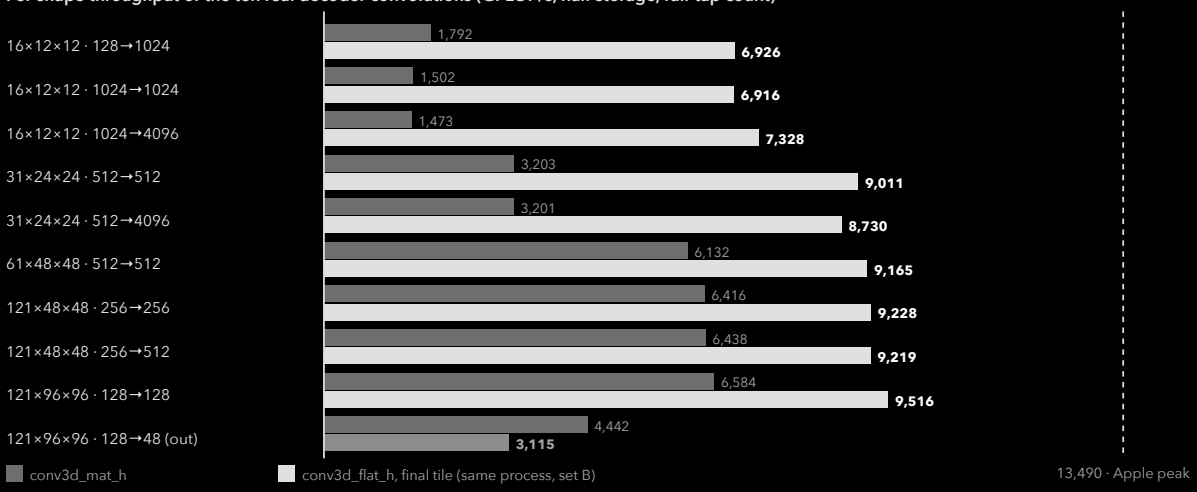


Figure 3. One process (set B). Largest gains on the deep, narrow layers 1–3, where the row tile was mostly padding; the 48- and 96-wide layers gain 1.4–1.5×; the output layer is slower with this tile (box below).

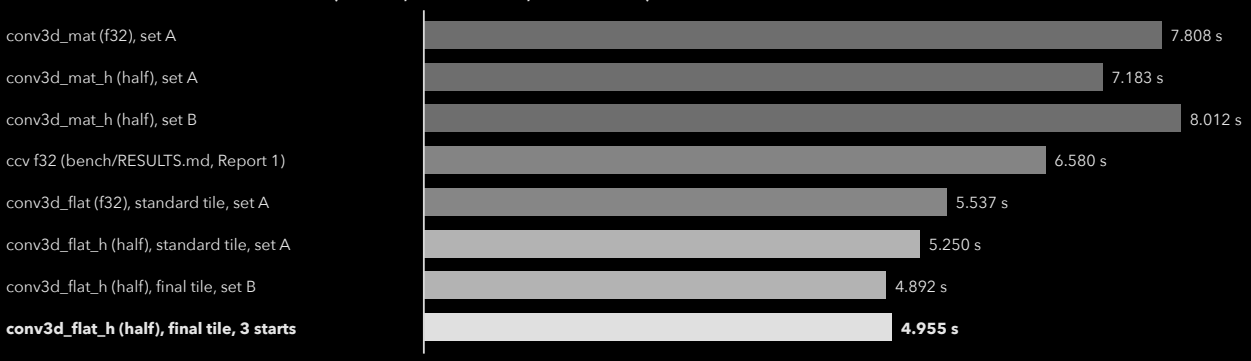


Figure 4. Kernel time over the 42 real calls from several run sets (set in the labels). Reference, ccv: Report 1, `bench/RESULTS.md`, not re-measured. **Bars from different run sets are not comparable**; comparable are the pair of set B and the group of set A. Bottom bar: clean figure (below).

- Clean figure: `conv3d_flat_h` alone, three process starts with three repetitions each: 4.955 / 4.948 / 4.967 s → 8.9 TFLOP/s, 65.9 % of 13,490 (`shapes_abgleich-r1…r3`).
- Drift: same kernel and tile 4.640 s in the sweep set, 4.748 s after a rebuild from the same source; up to 4 % between runs, 7 % against the sweep set; cause not investigated.
- Rule from it: kernels are compared only when they ran in the same process; range for the 42 calls across process starts and rebuilds: 4.58–4.97 s.

> **Output layer (OC = 48)**
> - A separate 48-wide tile for this one layer: per-shape choice in Section 8.
> - Standard tile everywhere: 4.575–4.582 s for the 42 calls (same process, three starts).

Correctness

CHECK	KERNEL	RESULT
Sampled check, first run, 40 (shape, kernel) pairs	float / half	≤ 1.7×10^{-5} / ≤ 9.7×10^{-4}
Sampled check, every sweep variant of this report	final tile	passed; maximum 7.1×10^{-4} – 9.7×10^{-4}

CHECK	KERNEL	RESULT
Full CPU reference (`conv3d_test`), six edge shapes[1]	`conv3d_flat`	correct on 6 of 6 (maximum 4.6×10^{-5})
Same	`conv3d_flat_h`	half-ULP test passed on 3; 3, 21 and 26 ULP on the others at reference values near 10^{-4} (absolute ≤ 0.0156)

- [1] Edge shapes: odd widths · 5×5 kernel with padding 2 · 1×1 kernel · unpadded · ragged channels.
- Cause: truncation of the half output near zero, not accumulation; the earlier half kernel gives identical numbers on two of these shapes (third not run).
- Limit: a criterion of 10^{-3} absolute cannot be met with half output in principle.

4 Where the time goes

- Method: selective removal — variants that deliberately compute wrong results, with one part of the kernel switched off.
- Shapes: 4, 6 and 9 (21 of the 42 calls). The numbers locate cost; they are not results.

VARIANT (21 CALLS)	SUM	GFLOP/S	READING
Before the staging changes (BKC 32)			
standard	3.034 s	8,836	
B fragments from threadgroup instead of device	2.861 s	9,370	device loads of B ≈ 6 %
A fragments always from one address	3.180 s	8,430	slower, code generation worse — no evidence
no device loads in staging (barriers kept)	2.428 s	11,041	**staging ≈ 20 %**
matrix work only	2.183 s	12,283	= the ceiling of Section 2
After (WN8, BKC 64, PAD 8, 16-byte staging)			
standard	2.729 s	9,825	
no device loads in staging	2.670 s	10,042	**staging ≈ 2 %**
B fragments from threadgroup	2.582 s	10,383	device loads of B ≈ 5 %

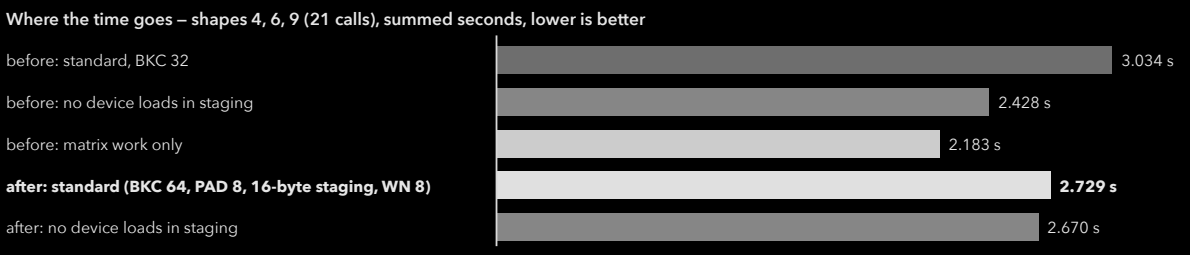


Figure 5. Selective removal on 21 real calls. The pure matrix loop matches the independent ceiling of Section 2.

CHANGE THAT CUT STAGING FROM ≈ 20 % TO ≈ 2 %	MEASURED
16-byte staging unit (one `uint4` of eight halves instead of a `half4`)	−2.6 % at BKC 32
Block of 64 input channels with an 8-element row pad (no bank conflicts)	all ten shapes 4.820 s vs. 4.882 s (BKC 32) and 5.032 s (PAD 16)
Eight simdgroups per threadgroup (BN = 128)	4.696 s, then 4.640 s in a second run, vs. 4.820 s (sweep set)
BKC 32 instead of BKC 16 on the new kernel	−3.5 % (5.251 → 5.065 s); useless for `conv3d_mat`, whose bottleneck was the wasted tile

- Remaining distance to the half ceiling (12.3 vs. 9.8 TFLOP/s in this run): the sequence of fragment loads (ten per sixteen matrix operations) and the barriers.

5 What did not work

- Source: the sweep set (reference tile 4.640 s); compare within a row only.
- Correctness: every variant passed the sampled check unless stated.

IDEA	MEASURED	VERDICT
Prefetch the next block's staging data into registers during the matrix work	3.119 vs. 3.120 s (21 calls)	0.0 %; rejected, source kept as a discarded file
Prefetch the next stage's fragments, two register sets	3.226 vs. 2.729 s	≈ 17 % slower: register pressure

IDEA	MEASURED	VERDICT
Smaller or wider tiles: 32×128, 64×32, 48×64, 32×64 (BKC 16); 64×128 (BKC 32)	5.38–5.90 vs. 5.251 s; 5.144 vs. 5.065 s	all worse; the 128-wide tile collapses the output layer (3,002 GFLOP/s)
Two simdgroup rows (2·2 and 2·4)	6.10 and 6.09 s	worse
32 accumulator fragments (WM1 WN2 FM8 FN4)	6.951 s	register pressure
Fragment shape 4×4 with BKC 64	9.4–9.9 s (≈ 4,500 GFLOP/s)	half the speed; cause not investigated (guess: code size or registers)
Block of 48 channels / 64 channels without pad	6.384 / 5.549 s	96-byte row pitch, bank conflicts; with PAD 8, BKC 64 reaches 8,573 vs. 7,943 GFLOP/s on five shapes
Row padding of 4 or 8 at BKC 32	4.443 / 4.440 vs. 4.456 s (five shapes)	+0.3 %, within noise; PAD 4 with 16-byte staging computes wrong results (PAD must be a multiple of the staging width; caught by the sampled check)
More occupancy (threadgroup memory inflated to limit resident groups)	1 group per core: −29 %; ≥ 3 groups: −2 % at most	latency hidden from about three resident groups per core
FM = 6 with WN 8 / BKC 64	4.787 vs. 4.640 s	worse than FM 8 (both sweep set)

6 Honest counting

MEASURE	COUNTS	SET AGAINST
Full	2 × multiply-accumulates with all 27 taps and the full output — upper bound, used so far	13,490 (Apple)
Useful	only taps that read real input values, no padding	13,490 (Apple)
MMA	8×8 matrix operations the kernel issues, tile-edge padding included; computed from the kernel code, not read from a GPU counter	12,300, measured ceiling (Section 2)

42 DECODER CALLS, ONE PROCESS (SET B)	TIME	FULL (OF 13,490)	USEFUL (OF 13,490)	MMA ISSUED	MMA / CEILING 12,300
`conv3d_mat_h`	8.012 s	5,501 (40.8 %)	5,342 (39.6 %)	8,239	67.0 %
`conv3d_flat_h`	4.892 s	9,010 (66.8 %)	8,750 (64.9 %)	9,675	**78.7 %**

- Unit: GFLOP/s · one process start each · useful / full = 0.971 for both kernels.
- Issued MMA work vs. useful need: `conv3d_flat_h` 1.11×, `conv3d_mat_h` 1.54× (against the full count: 1.07× and 1.50×).
- Per shape, `flat_h`: 75–81 % MMA utilisation on shapes 1–6; shape 10 (48-channel output) 68.9 %, because the 128-wide tile computes 2.72× the needed matrix work.

Decoder chain at latent 15×24×32

- Run: `vae_chain`, one run per kernel · 0.3647 PFLOP full, 0.2270 useful.

CHAIN 15×24×32	GPU TIME	WALL TIME	FULL (WALL)	USEFUL (WALL)	MMA UTILISATION (GPU / WALL)
`conv3d_mat_h`	27.88 s	29.88 s	12,204	7,595	74.2 % / 69.2 %
`conv3d_flat_h`	24.81 s	26.79 s	13,613[1]	8,467 (62.8 %)	**84.4 % / 78.1 %**

- [1] Above the peak of 13,490 because the full count includes padding taps that the kernel does not issue as matrix work (issued: 84.4 % of the 12,300 ceiling).
- Corrected claim: the earlier 95–96 % "of the peak" for `flat_h` in the chain is the full-tap measure; the useful count is in the table.
- Corrected claim: the 65.9 % for the 42 calls (Section 3) is full-tap too; useful: about 64 %.
- Consequence: with the matrix unit this close to its measured ceiling, a direct kernel gets faster only if less is computed (Section 8).
- Caveats: one process start each (`flat_h`: second pass within 0.1 %); chain correctness not checked by `vae_chain`, it rests on the sampled check of the shape bench.

7 A native-layout path in ggml

- Problem: the engine uses the ggml layout `[W,H,D,C·N]`; ccv pays for it with three scratch buffers per convolution node (NDHWC input and output in float32, float32 weights).
- Goal: read and write the ggml layout directly, no NDHWC copies.
- Where: fork branch `exe-native-f16` · `GGML_METAL_CONV3D_KERNEL=exe` · fallback of the Winograd default (Section 9) · sd.cpp sources unchanged.

Tile	plane tile of Section 3: flattened padded (oh, ow) plane, eight simdgroups, 64-channel blocks, 8-element row pad
Data	ggml layout `[C][D][H][W]`
Operands	staged as half through 16-byte threadgroup stores (one `uint4` of eight halves per lane and position)

Device reads	scalar float32: neighbouring channels are not adjacent in this layout, so the 16-byte loads of the lab kernel do not carry over
Accumulator, output	float · weights repacked to half
Coverage	42 of 42 decoder convolutions (42 `[conv3d] exe` log lines per run)
Tests	`test-backend-ops -o CONV_3D`: 290/290 with exe, with ccv and with the path off
Review	by reading; one defect fixed: a pipeline that fails to build now falls back to the plain kernel

LTX-2.3, 121 frames, 4 steps	kernel	VAE compute buffer	decode	generate_video	peak	PSNR vs ccv (mean / min)
384×384	ccv	4,381 MB	10.99 s	145.34 s	44.58 GB	–
	exe	**2,737 MB**	**9.48 s**	144.19 s	44.64 GB	42.09 / 41.33 dB
1280×704	ccv	26,766 MB	55.67 s	479.54 s	43.22 GB	–
	exe	**16,727 MB**	**48.63 s**	488.46 s	44.30 GB	43.15 / 42.19 dB

- Runs: one per cell, each alone on the GPU, one after the other · peak from `/usr/bin/time -l`.
- Decode noise: about ±5 s (the two ccv references at 1280×704 differ by 5 s; at 384×384 the three ccv runs lie within 0.74 s).
- PSNR: two numerical paths of the same pipeline, not ground truth; usual range for half operands.
- Note, the ccv run at 1280×704: coincided with the second watchdog crash (Section 13); it finished and is reported as measured.

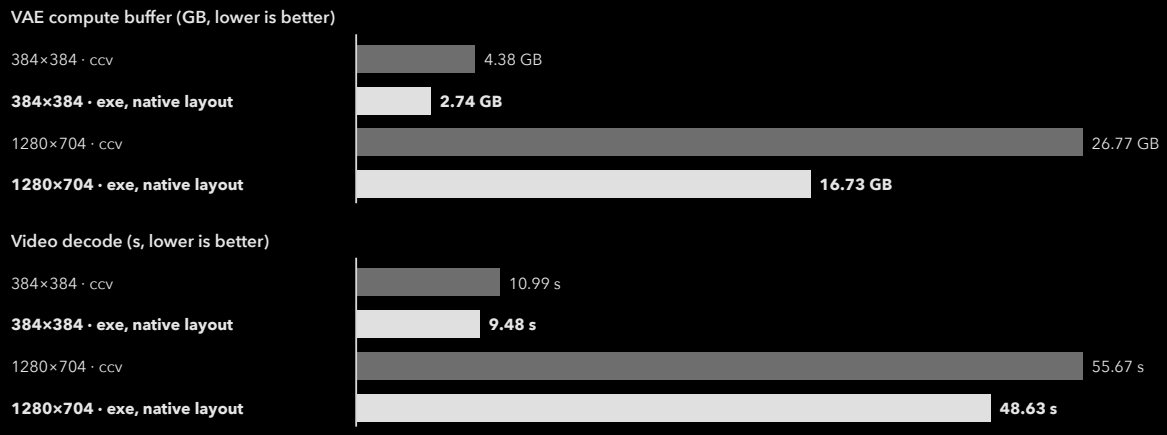


Figure 6. Native-layout kernel against ccv, one run each; separate scales per panel. VAE compute buffer −37.5 % at both sizes, back to its size without ccv.

- Decode: shorter at both sizes; at 1280×704 within noise alone, but below both ccv runs seen at that size (Figure 8).
- Total: equal within noise (+1.9 % at 1280×704, −0.8 % at 384×384); sampling dominates.
- Peak does not fall (table): the model weights, 38.4 GB, dominate it.
- Reading of the logs (not confirmed by a separate run): in the ccv run the model manager released ten diffusion blocks (≈ 9.9 GB) right before the 26.8 GB buffer, because both no longer fitted the 54,783 MiB budget; with the 16.7 GB buffer everything fits and nothing is released. The reading does not account for the measured peak (32.04 GB, Section 11); the explanation is open.
- Lever for the peak: the weights → `te=disk` (Section 1); combination: Section 11.
- Not measured: a micro-benchmark of the 16-byte staging · the tolerance of `test-backend-ops` for the half path · a planned repeat of any cell (near-identical runs exist: Section 14).

8 Winograd

multiplications per (input channel, output channel) pair and T×T output tile	direct 3×3: 9·T²	winograd: (T+2)²
F(2×2, 3×3)	36	16
F(4×4, 3×3)	144	**36 (four times fewer)**

- Depth taps kd = 0 to 2: not transformed; summed, K = 3·IC.
- Four kernels in sequence (`metal/conv3d_wino.metal`, new code):
  - weight transform $U = G \cdot g \cdot G^T$, per layer, counted in the time (shape 3: 6.4 of 28 ms);
  - input transform $V = B^T \cdot d \cdot B$;
  - one GEMM per frequency point, with the body of `conv3d_flat_h`;
  - output transform $A^T \cdot Y \cdot A$.
- Preconditions: H and W divisible by the tile · IC divisible by 64 · OC by the tile width.

42 decoder calls, laboratory	intermediates	calls	time	check against CPU reference
direct `conv3d_flat_h` (`all2`)	–	42	4.638 s	ok

42 DECODER CALLS, LABORATORY	INTERMEDIATES	CALLS	TIME	CHECK AGAINST CPU REFERENCE
direct, best choice per shape (48-wide tile for shape 10)	–	42	4.503–4.510 s	ok
Winograd F(2,3)	half	41	2.218 s	passes, but up to 665 half-ULP maximum error
Winograd F(4,3), Y in half	V, U float; Y half	0	–	**fails**: rms 1.95×10^{-3} relative (shape 1), up to 11,648 half-ULP
Winograd F(4,3), BKC 32	float	41	1.739 s	ok; slower, dropped
Winograd F(4,3), all float	float	41	**1.556–1.605 s**	ok, 1.3–10.4 half-ULP, rms 2.1×10^{-4} (as the direct kernel)
Winograd for all 42 (own 48-wide tile for shape 10)	float	42	**1.580 / 1.580 / 1.582 s**	ok

- Last row: three independent process starts (`wino_final-r1…r3`), spread ≤ 0.2 %.
- Method: direct and Winograd columns alternating in the same process.

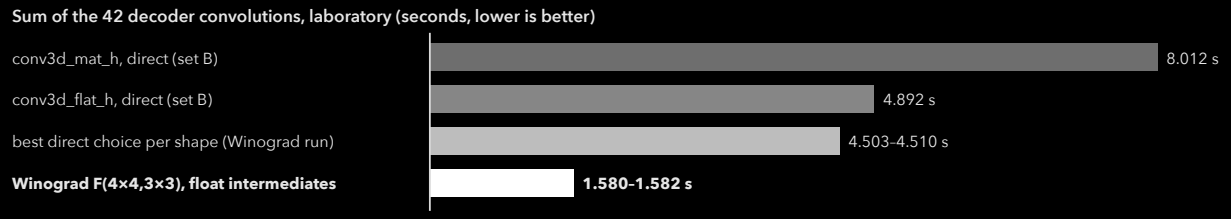


Figure 7. Bars 1-2: same-process pair of Section 3 (set B); bars 3-4: Winograd runs, one process. Absolute times drift between the two groups; read the ratios 8.01 : 4.89 and 4.50 : 1.58.

- "Equivalent GFLOP/s" (27,000–28,000 when 27 taps are counted): only a unit conversion of the time, never stated as a share of the peak.
- Real matrix rate in the GEMMs: 6,600–8,600 GFLOP/s (53–70 % of the MMA ceiling); the gain is four times less matrix work, not a fuller unit.
- Output layer: input and output transforms take 39 % of its time.
- Negative result: a fused Winograd with the transforms in threadgroup memory: 2.52–3.79 s over 41 calls vs. 1.556–1.605 s.

8.1 In the chain

- Mode `vae_chain --real`: real decoder structure (convolution grid = input grid; pixel norm, SiLU and depth-to-space between layers; latent 15×24×32; 0.2196 PFLOP; 42 convolutions) with random weights – a load model with real data dependence, not a decoder.
- Runs: three process starts per column, two passes each.

	FLAT_H	WINOGRAD, BUFFERS PER LAYER	WINOGRAD, BUFFERS POOLED
Wall time	23.59–23.62 s	17.15–17.30 s	**9.76–10.19 s**
GPU time, convolutions	21.51 s	**7.66–7.71 s**	7.66–7.70 s
Longest command buffer	1.002 s	0.330 s	0.329–0.332 s
Memory peak (allocated, incl. 0.81 GB weights)	3.68 GB	16.60 GB	18.41 GB

- Convolution time: 2.80× shorter, as in the shape bench.
- Wall time: 1.37× shorter with 6.5 GB buffers allocated per layer (a ≈ 9 s gap between wall and GPU time; the allocation itself costs 0.01 s; cause not investigated); 2.3–2.4× with the three buffers created once and reused.
- Price: memory 4.5–5× the peak of the direct kernel → scratch cap in ggml (Section 9).
- Deviation over all 266.6 million outputs: Winograd vs. `flat_h` 5.25×10^{-3} (RMS, relative to the RMS of the reference); two direct kernels with a different summation order: 5.20×10^{-3} → chain-level rounding, not specific to Winograd.
- Not shown: the distance of either to a true float chain, which does not exist at this size.

9 Winograd inside ggml

Status	third kernel in branch `exe-native-f16`; default since commit `17a8ad9f`); `GGML_METAL_CONV3D_KERNEL=exe` or `ccv` selects the others
Data	reads ggml's weight bytes (OIDHW) and activations (`[IC][ID][IH][IW]`, float32) directly; writes the output layout directly; all intermediates float
Scratch (U, V, Y)	cut into depth and output-channel pieces within `GGML_METAL_WINO_SCRATCH_MB` (runs below: 512)
Gate	3×3×3 · spatial padding 0 or 1 · IC divisible by 64 · OC by 128 or 48 · otherwise fallback to the native-layout kernel (Section 7)
Edge tiles	`01047b70`: H and W no longer divisible by 4; the input transform reads zeros outside the frame, the output transform writes only pixels inside it
Padding 0	`fc887b41`: needed for Wan (Section 12)

TEST	RESULT
`test-backend-ops -o CONV_3D`	298/298 with wino, exe, ccv, with the path off, and with wino at a 1 MB scratch budget (single-image pieces); eight cases take the Winograd path
Edge-tile build	`CONV_3D` 304/304 · all ops 17,642/17,642
Deviation, float32 weights	below the printed precision (half-operand exe kernel: 7×10^{-8}); the pass threshold of 5×10^{-4} does not separate the kernels – the video PSNR does
Perf cases, LTX shapes	1.97–2.89× faster than the exe kernel (14 shapes, one run each)
42 decoder convolutions (`test-backend-ops perf`)	exe 5.68 s (57.5 % of 13,490, full-tap) · ccv including repacking 7.31 s (44.7 %) · no percentage for Winograd (Section 8)

SD-CLI, LTX-2.3, 121 FRAMES, 4 STEPS, TE=DISK, BUFFERS SPLIT	384×384	1280×704
Video decode, Winograd	**6.03 s**	**30.19 s**
Video decode, exe kernel / ccv fork	9.39 / 10.98 s	46.88–47.12 / 55.67 and 60.67 s
Convolutions on the Winograd path	42 of 42	36 of 42; 6 fall back to exe (latent height 22 is not divisible by 4)
With edge tiles (`01047b70`)	unchanged	**42 of 42**; decode GPU time 27.97 → 27.54 s (wall 31.19 s); PSNR 43.22 dB against the run above
VAE compute buffer (exe: same; ccv)	2,737 MB (ccv 4,381)	16,727 MB (ccv 26,766)
Peak memory	29.77 GB	31.75 GB
generate_video total	137.18 s	440.61 s
Text / sampling / audio	32.2 / 91.3 / 7.6 s	32.4 / 370.2 / 7.6 s
Sampling steps	59.7 / 10.53 / 10.53 / 10.53 s	129.3 / 80.3 / 80.3 / 80.3 s
PSNR of the video	44.46 dB against ccv f32 (exe against ccv: 42.09)	43.46 dB against exe

- Runs: one per column, each alone on the GPU · fork before the Winograd commits (edge-tile row: build `01047b70`) · exe and ccv figures from the runs of Sections 1 and 7 (exe at 1280×704: two runs).
- Wall figures of separate runs depend on the file cache (Section 1).

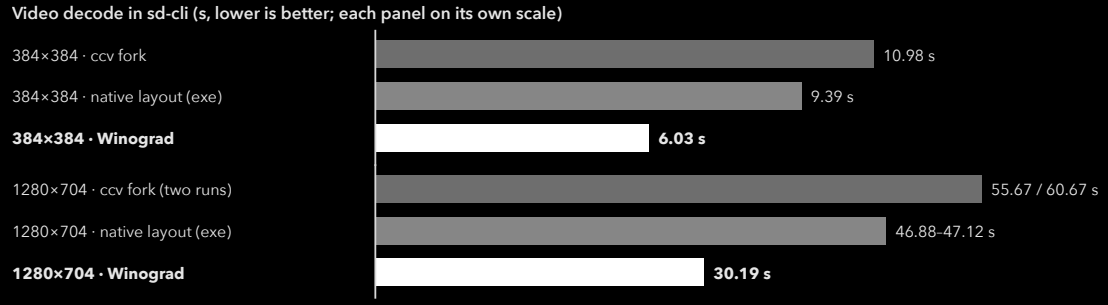


Figure 8. Video decode in sd-cli, one run per bar unless two are named; each panel on its own scale. At 1280×704 Winograd is about 17 s below exe and 25–30 s below ccv, more than the noise of about ±5 s.

- Decode: −36 % against the exe kernel at both sizes, although 6 of 42 convolutions at the large size still ran the slower path before the edge tiles.
- Memory: VAE compute buffer equal to the exe buffer to the megabyte (scratch capped) → the 10 GB saving against ccv stays (Section 7).
- Sampling: unchanged against exe → decode is 7 % of a 1280×704 run.
- PSNR against ccv f32: higher for Winograd than for exe, as expected for float operands; no video compared by eye.
- Edge tiles: six layers move from half operands to float intermediates – hence the PSNR against the earlier Winograd video in the table; two identical runs give 99 dB.
- Not credited to Winograd: the totals (137.18 and 440.61 s vs. 144.15 and 458.68 s for exe under the same switches) – load and text phases differ by file-cache state.
- Peaks: equal to the exe runs under the same switches (29.77 and 31.85 GB) within one run · no repeat of any run.

10 Splitting command buffers

- Trigger: both crashes were WindowServer watchdog session endings, “40 seconds since last successful checkin” (Section 13).

- ggml Metal backend: a whole graph in about two command buffers → a 1280×704 decode, or a single diffusion step, is one buffer of tens of seconds.
- Split: `GGML_METAL_CB_SPLIT=heavy` (default) cuts at 8,000 estimated GFLOP *or* 256 nodes per buffer; `off` and a node count remain selectable.
- Why the node limit: the audio VAE (1D and transposed convolutions, not seen by the work estimate) stayed in one 5.3 s buffer.
- Log: longest buffer per graph.

LONGEST COMMAND BUFFER	384×384 OFF	384×384 HEAVY	1280×704 OFF	1280×704 HEAVY
Diffusion step	9.32 s	< 0.5 s (not listed)	**74.89 s**	**1.28 s** (99 buffers)
Video VAE decode	7.37 s	1.61 s	44.26 s	3.45 s (24 buffers)
Audio VAE	5.32 s	1.17 s	5.33 s	1.17 s
Gemma (text) graph	2.20 s	< 0.5 s	2.20 s	0.39 s

- Runs: one per cell, exe kernel with te=disk · 384×384 heavy column = 256-node variant · logs `…_cb-off.log` and `…_cb-heavy256.log` in `measurements/`.
- Video: pixel-identical (99.00 dB, 121 frames, both sizes).
- Compute: sum of buffer times unchanged (diffusion step at 1280×704: 79,885 vs. 79,916 ms); steps 2–4 unchanged (10.53 and 80.3 s per step).
- Totals: differ by up to 20 s from the load and text phases (Gemma file read in 2.1 s in one run, 26 s in another) — not attributed to the split.
- Tests: whole `test-backend-ops` suite with the default: 0 failures · `CONV_3D` 290/290 with the split on and off (before the Winograd tests).
- Crash prevention: derived, not tried; ccv at 1280×704 with the split not run (off limits after two crashes).
- Commit: `0fe24d10`; review fixed one defect: after a cut, `synchronize()` could wait on the first buffer instead of the last.

11 End-to-end

TASK: LTX-2.3, 121 FRAMES, 4 STEPS	PEAK MEMORY	GENERATE_VIDEO	DECODE	PSNR VS REFERENCE
384×384 · ccv fork, reference (two runs)	44.60 / 44.64 GB	138.21 / 141.42 s	11.09 / 10.77 s	–
384×384 · ccv fork + te=disk (two runs)	**29.82 GB**	141.77 / 144.13 s	10.83 / 10.72 s	99.00 dB
384×384 · native-layout kernel + te=disk	**29.30 GB**	145.72 s	9.71 s	99.00 dB
1280×704 · ccv fork, reference	43.12 GB	478.44 s	60.67 s	–
1280×704 · ccv fork + te=disk	**29.45 GB**	invalid (Section 13)	invalid (Section 13)	99.00 dB
1280×704 · native-layout kernel + te=disk	**32.04 GB**	487.67 s	53.92 s	99.00 dB
384×384 · Winograd + te=disk + split buffers	**29.77 GB**	137.18 s	**6.03 s**	44.46 dB vs ccv f32
1280×704 · Winograd + te=disk + split buffers	**31.75 GB**	440.61 s	**30.19 s**	43.46 dB vs exe

- Draw Things reference values: Report 1, Section 10.
- Winograd rows (Section 9): te=disk and the default buffer split; one run each; PSNR against the ccv f32 video (384×384) and the exe-kernel video (1280×704) → measures the numerical path, not te=disk.
- Native-layout rows with te=disk: PSNR against the same kernel without te=disk (99.00 dB = pixel-identical) · 42 `[conv3d] exe` lines · exit 0 · run after the lock fix (Section 13).

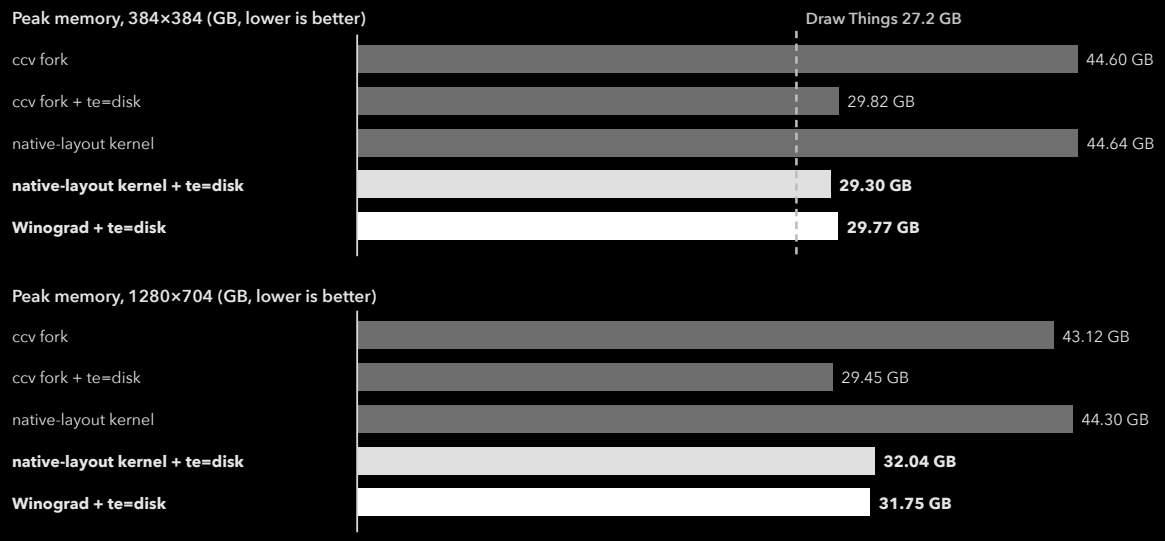


Figure 9. Peak footprint (`/usr/bin/time -l`), one run each except the two ccv rows at 384×384 (first of two shown); one scale for both panels; Winograd rows with the default split. Dashed line: Draw Things, 384×384 only. For ccv at 1280×704: the run of the table (the run of Section 7: 43.22 GB).

Memory

- 1280×704, native layout + te=disk: −11.1 GB (−25.7 %) against the ccv fork; −12.3 GB against the native-layout kernel alone (Section 7).
- 384×384, native layout + te=disk: 2.1 GB above Draw Things, 15.3 GB below the ccv fork; no gain against ccv + te=disk within one run each.
- Open: at 1280×704 the combination sits 2.6 GB *above* ccv + te=disk. Probable reason (logs, untested): ccv's 26.8 GB buffer made the model manager release ten diffusion blocks; with 16.7 GB nothing is released (Section 7).
- Winograd: −11.4 GB against the ccv fork and ≈ 0.3 GB below native layout + te=disk at 1280×704 (within one run); 2.6 GB above Draw Things at 384×384.

Time

- Totals: one run each; the differences to the ccv fork and to the native-layout kernel alone (Section 7) are not assigned to any switch.
- Decode at 1280×704: +5.3 s with te=disk against the run without (Section 7) – one run each, noise about ±5 s, not attributed.
- Not shown: any repeat · a visual comparison (videos as MP4 in `measurements/`, e.g., `…_kern-exe_te-disk.mp4`; pixels compared by PSNR) · nothing here is extrapolated from the kernel benchmarks.

12 A whole run, op by op

Profiler	`GGML_METAL_OP_PROFILE=1` (fork commit `abccda9e`): every op in its own command buffer; GPU time summed per op label at the end of each graph, with estimated GFLOP/s and bytes moved per second
Use	finds the heavy ops; drops the overlap of neighbouring ops, so not used to compare runs
Cost	one diffusion step 10,500 ms with the profile, 10,431 ms without

GRAPH, LTX-2.3 384×384	GPU TIME	LARGEST ITEMS
Diffusion step (17,918 ops)	10,500 ms	MUL_MAT q8_0 6,493 ms (61.8 %, 10.56 TFLOP/s) · **CONT 1,556 ms (14.8 %, 3,582 calls)** · FLASH_ATTN 1,120 ms (10.7 %) · CPY 388 · ADD 352 · MUL 277
Audio VAE (12,747 ops)	6,951 ms	**CONV_TRANSPOSE_1D 2,490 ms (35.8 %, 11 calls)** · IM2COL 1,964 ms (28.3 %) · SCALE 1,103 ms (15.9 %) · CONT 689 · MUL_MAT f16 401
Video VAE (440 ops)	4,285 ms	CONV_3D 2,062 ms (48.1 %) · **CONT 1,565 ms (36.5 %, 178 calls)** · CONCAT 340 · ADD 171
Gemma text encoder	2,412 ms	MUL_MAT q8_0 2,072 ms (85.9 %)

- Logs: `measurements/…_fork-bd04b91-prof_…_op-profile.log` and `…fork-prof2_…_op-profile-bytes.log`, one run each.
- q8_0 matrix products (real shapes of a step, `test-backend-ops perf`, two passes within 0.3 %): 10.60–10.89 TFLOP/s, 79–81 % of 13.49, at 2,304 and at 14,080 tokens.
- Flash attention (32 heads of 128, f16 K/V): 6.53–6.62 TFLOP/s, 48–49 % (13.31 ms at 2,304 tokens, 490.6 ms at 14,080) – the one large op with visible room; not changed.
- Copies (`CONT`, `CPY`): 35–60 GB/s, a tenth of the memory bandwidth (≈ 400 GB/s) – the generic copy kernel derives every destination index with four 64-bit divisions and reads a permuted source across memory.
- Audio VAE, `CONV_TRANSPOSE_1D`: one thread per threadgroup → 31 of 32 SIMD lanes idle.

OP (FORK COMMIT)	BEFORE	AFTER
Same-shape f32/f16 copy into a contiguous destination (`9023480a`)	generic kernel	a row kernel, or, when the source's fastest dimension is not the first, a 32×32 tile through threadgroup memory so that reads and writes are both contiguous
CONV_TRANSPOSE_1D (`ce28fe09`)	1 thread per threadgroup	one thread per output position, 128 per threadgroup

COPY, `TEST-BACKEND-OPS PERF`	BEFORE	AFTER
CONT f32, permute [2,1,0,3], six shapes	34.4–39.9 GB/s	**363–497 GB/s**[2]
CPY f32, permute [0,2,1,3], two shapes	34.7–36.2 GB/s	259–263 GB/s
CPY f32 → f16, contiguous	30.0 GB/s	197 GB/s
CPY f32, transposed (three variants)	30.8–32.7 GB/s	359–360 GB/s
CPY f16, transposed (four variants)	15.3–16.2 GB/s	262 GB/s
CPY bf16 and quantised types		unchanged (not covered by the new path)

- Source: `measurements/tests-…-cpy-fast/perf_{off,on}.log`, one pass each, same build, new path off with `GGML_METAL_CPY_FAST=0`.
- [2] Counted bytes per second; above the memory bandwidth of ≈ 400 GB/s, so part of the traffic is presumably served from the on-chip cache.
- Tests: CPY/CONT/DUP 284/284 · CONV_TRANSPOSE_1D 116/116.

LTX-2.3 384×384, 121 FRAMES, 4 STEPS, TE=DISK	BEFORE	AFTER
One diffusion step	10.53 s	**9.46 s** (−10.2 %)
Audio decode	7.55 s	**4.56 s** (−39.6 %)
Video decode, wall / GPU	6.03 / 4.28 s	**4.66 / 2.93 s**

LTX-2.3 384×384, 121 FRAMES, 4 STEPS, TE=DISK	BEFORE	AFTER
generate_video	137.18 s	**129.04 s**
Peak memory	29.77 GB	29.71 GB
Output	video pixel-identical (99.00 dB), audio bit-identical (same MD5)	

LTX-2.3 1280×704, SAME BUILD, AGAINST THE EDGE-TILE RUN OF SECTION 9	BEFORE	AFTER
One diffusion step	80.30 s	**73.85 s**
Video decode, wall (GPU)	31.19 s (27.54 s)	**20.93 s** (18.14 s)
Audio decode	7.58 s	4.57 s
generate_video	447.08 s	**401.64 s**
Peak memory	32.27 GB	32.14 GB
Output	video pixel-identical (99.00 dB)	

- No arithmetic changed; byte-identical output; step time identical to 0.01 s in all three timed steps.
- Runs: one per column; no repeat.

A second video model: Wan 2.2 TI2V 5B

- Task: 832×480 · 33 frames · 10 steps · its own VAE.
- First run: no Winograd at all — Wan's causal convolution pads its input itself and calls the convolution with padding 0, which the first gate refused → every decoder convolution on the native-layout kernel.

WAN 2.2 TI2V 5B, 832×480×33	CCV	EXE (PADDING 0 REFUSED)	WINOGRAD (PADDING 0 OR 1)
`[conv3d]` lines	303 ccv	303 exe	252 wino, 51 exe
Video decode	51.15 s	43.78 s	**23.99 s**
generate_video	255.06 s	249.85 s	223.15 s
Peak memory	25.33 GB	25.33 GB	25.33 GB
PSNR against ccv (mean / min)	–	44.62 / 43.40 dB	44.56 / 43.21 dB

- Runs: one each, `measurements/models-…/`; the Winograd run also has the fast copies.
- 51 remaining exe nodes: 19 with channel counts outside the gate (input 48, output 12); the other 32 pass the channel test, so by the gate they are not 3×3×3 (kernel sizes are not in the log line).

Loading

READ FROM THE EXTERNAL USB SSD, 384×384 RUN	SIZE	TIME	RATE
Gemma	12.7 GB	26.1 s	0.49 GB/s
Diffusion model	22.8 GB	50.4 s	0.46 GB/s
Same disk with `dd` (4 GiB, 16 MiB blocks, uncached file)	4 GiB	–	**1.00 GB/s**

- Share: about 80 of 137 s are reading weights; the loader uses half of the disk.
- Loader: in stable-diffusion.cpp, not edited here · levers without code: the internal SSD, a server mode that keeps the weights loaded.
- Gap to Draw Things at 384×384: mostly this, not the GPU.

13 Measurement integrity on a shared GPU

- Setup: three automated work processes — measurement, kernel work and memory work — shared one GPU under a coordinating fourth.
- Lock: a bare `mkdir /tmp/exe-gpu.lock` — no owner, so anyone could remove it or mistake it for stale.

INCIDENT	WHAT HAPPENED	CONSEQUENCE
1	Lock left behind, no process holding it	runs blocked; removed after checking every GPU user with `pgrep`
2	WindowServer watchdog ended the user session during the 1280×704 ccv run (`iogpu.wired_limit_mb=59392`; crash report of the window server) · the run finished (exit 0, decode 60.67 s) · the next run aborted with 134, "Unable to reach MTLCompilerService"	all agents and the app gone; files and git intact; cause not proven · rule: no 1280×704 run in parallel to other GPU work
3	A lab job obtained the lock during the 1280×704 te=disk run (a lock had been removed by a process that did not hold it)	time of that run contaminated, not used (Section 1.3)
4	The memory agent took the lock held by the kernel agent; `test-backend-ops perf` and `vae_chain` ran concurrently	numbers of both from that window excluded

INCIDENT	WHAT HAPPENED	CONSEQUENCE
5	Second watchdog (crash report of the window server, "40 seconds since last successful checkin", display off) during the 1280×704 ccv reference run of Section 7, now at `iogpu.wired_limit_mb=55296`	session and all agents ended; files and git intact; `sd-cli` exit 0 · rule: no more ccv runs at 1280×704

A pattern, not a cause

- Both crashes during a ccv decode at 1280×704: incidents 2 and 5, each time while that decode was running.
- No crash: the exe decode of the same size · the ccv decode inside the te=disk run of incident 3.
- Wired limit differed (59,392 and 55,296 MB) → memory pressure alone does not separate the cases.
- Buffer length alone does not either: the unsplit 75 s diffusion steps and the 44 s exe decode buffer did not crash; the split (Section 10) is derived from the 40 s check-in rule, untested against a crash.
- Guess consistent with the pattern: ccv's very long single dispatches at this size leave the window server too long without GPU time. Not proven; no experiment made.

The fix

- Owner: the two lock scripts (`metal/gpu_lock_run.sh`, `measurements/run_testtask_extra.sh`) write `/tmp/exe-gpu.lock/owner` with PID, time and name.
- Removal: a stray `rmdir` fails on the non-empty directory; a lock is cleared only when its numeric owner PID is dead.
- Access: agents take the GPU only through those two scripts, never with `mkdir`, `rmdir` or `rm`.
- Trap: a `pgrep -f test-backend-ops` check in the test script matched the agent's own shell → four false starts.
- Observed afterwards: the lock held by a `test-backend-ops` run; a queued `vae_chain` waited without starting a child.

> **Lesson**
> - A lock for timing work needs an owner and must refuse removal by anyone else.
> - Every number must carry what else was running.
> - Kernel measurements used here were taken outside the incidents, after the lock fix; kernel files written during incidents 3 and 4 are not used.
> - Memory runs: clean for peak footprint in every case; for time everywhere except the ccv te=disk run at 1280×704.

14 Open points and next steps

OPEN POINT	NEXT STEP	SECTION
Repeats of the engine runs	alternating pairs per configuration; every engine figure here is one run · near-identical runs give a first spread: decode 6.21 / 6.03 / 6.24 s (Winograd, 384×384), 30.19 / 29.42 s (Winograd, 1280×704), 9.62 / 9.42 / 9.39 s (exe, 384×384)	7, 9–12
te=disk time effect at 1280×704	alternating pair and an `--auto-fit off` control	1.3, 11
Peak at 1280×704 above ccv + te=disk	test the block-release reading with a run	11
f16 activations throughout the decoder graph	not built · needs a deep ggml-core change: `ggml_conv_3d_direct` returning half; half or mixed-type Metal paths for ADD, MUL, CONCAT (equal types required), RMS_NORM (float32-only pipeline) and the conv input (float32 required) · from reading the source only · **estimate, not measurement:** VAE buffer at 1280×704 16.7 → about 8–10 GB (ideal 8.4 GB) · risk: encoder, audio VAE and other models share the type rules, not checked	–
Float operands with half storage	build; above 12.4 TFLOP/s only with float operands	2
Output layer (OC = 48)	make the per-shape choice the default of the half kernel	3, 8
Winograd variants	fully-half F(4,3) not measured · F(6,3) not tried	8
Wall-to-GPU gap without a buffer pool	find the cause of the ≈ 9 s	8.1
Run-to-run drift of the kernel bench	find the cause	3
Crashes of incidents 2 and 5	prove the cause; test the buffer split against a crash	10, 13
Remaining residency switches	`--max-vram 32` and 28, `--offload-to-cpu`, `te=cpu`, `--auto-fit off` not measured; gain on unified memory doubtful	1
Remaining slow ops	flash attention (49 %) · IM2COL (28 GB/s) and SCALE (11 GB/s) in the audio VAE · CONCAT · copies of bf16 or quantised data · 2D-VAE image models not profiled	12
Classic image models, 2D convolution	outside this report; covered by Report 3	–

Licences and acknowledgements

COMPONENT	ORIGIN
Third-party components (stable-diffusion.cpp, ggml, ccv) and their licences	Report 1
`conv3d_flat`, `conv3d_flat_h`, Winograd kernels, command-buffer split, matrix-ceiling probe, shape bench, memory measurements, analysis	Exe AI, own work

COMPONENT	ORIGIN
Draw Things	measured comparison of the released application's output, time and memory only; no code used
GPL-licensed code	none used

References

1. Exe AI, *Fast 3D Convolutions for Local Video Generation on Apple Silicon*, Technical Report 1 — with the references to stable-diffusion.cpp, ggml, ccv, Metal and LTX-2.3.
2. Apple, M2 Max compute peak of 13.49 TFLOPS (38-core GPU), as published by Apple.
3. Project measurements: `measurements/RESULTS.md` and the run logs in `measurements/`; working notes in `notes/` (residency switches, text-encoder release, buffer split, Winograd in ggml); laboratory raw files in `metal/messungen/` and `shapes_*.tsv`.

EXE AI · DEVELOPER STUDIO