



PerfModel

A Validation-Driven Performance Model
for Triton Kernels

Xiaohu Guo, Lixun Zhang . AMD . PyTorch Conference 2026

What Triton Autotune Measurement Can and Cannot Tell You

Autotune answers one question completely - which of these candidates ran fastest here. The model exists to answer the others

The config space, per shape

```
BM x BN x BK
num_warps . num_stages
mfmaNonKDim (16x16x32, 32x32x16,
              scaled 16x16x128 for MXFP)
kpack . GROUP_SIZE_M . waves_per_eu

~10^4 candidates. Tile is one axis of eight.
```

How Triton answers it today: runtime autotune

The first time a shape is seen, JIT-compile every candidate, benchmark each on the actual GPU, keep the fastest, and cache the answer keyed by shape.

This works. In production, with diverse GEMM shapes, the cost is paid again for every shape that has not been seen before.

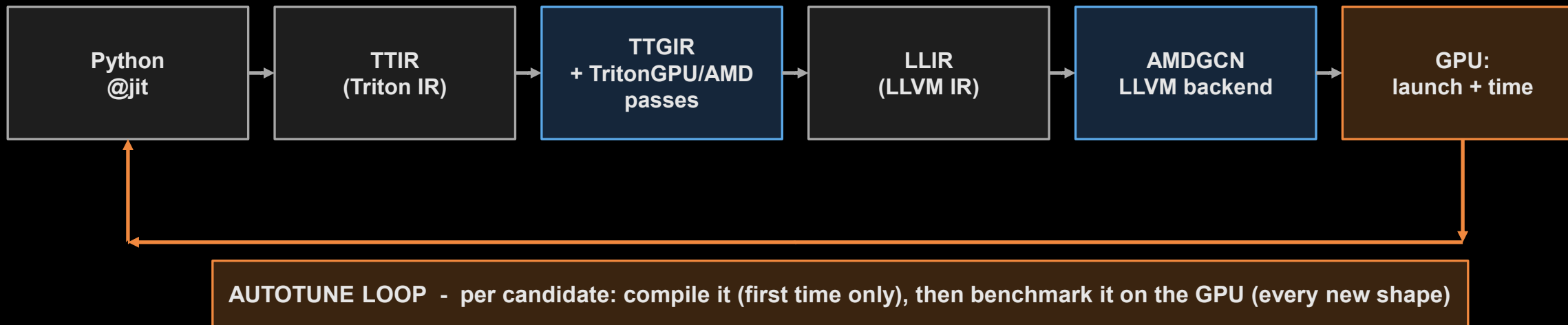
Question	What measurement gives you
Which candidate is fastest on THIS shape?	A definitive answer - this is what autotune is for, and it is exact
Which will be fastest on a shape not yet seen?	A new measurement. The space is large, so practical tuning samples it
Why is that one fastest?	Not carried in the signal: the result is a time, not a reason
What changes if the kernel changes?	A new measurement, because the previous result described the old kernel

None of this is a shortcoming of autotune. Measurement is simply the only signal available at that point in the pipeline. A cost model is available earlier, and can answer the other three.

Where Triton Decides Today

A shape you have not seen before re-runs the whole benchmark sweep - even though the kernels it needs are already compiled and cached

config = (BM, BN, BK, num_warps, num_stages, mfmNonKDim, kpack, ...) feeds every one of these stages



Compiled kernels - cached per CONFIG

Key = every constexpr VALUE (BLOCK_M/N/K) + num_warps, num_stages, dtypes and alignment hints. So each candidate config is its own binary: N configs means N compilations. M, N and K enter only as alignment hints, never exact values.

You pay this ONCE per config. `jit.py: compute_cache_key()`

Tuning results - cached per SHAPE

Key = ['M','N','K'] - the exact shape values. A new shape misses, and re-benchmarks all N candidates on the GPU. Those N binaries already exist and are reused, so the repeat cost is pure measurement, with zero compilation.

You pay this AGAIN for every new shape. `autotuner.py`

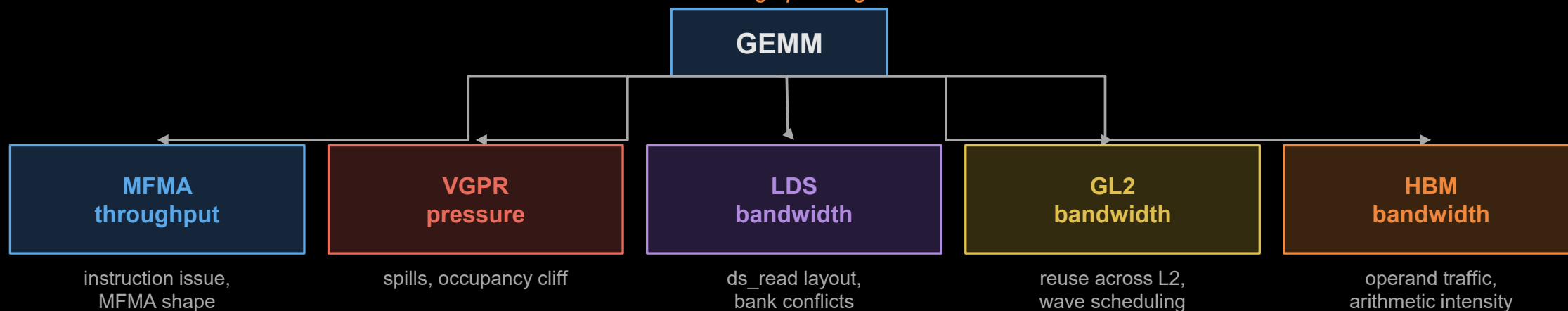
The cost that repeats in production is measurement, not compilation - because measurement is the only ranking signal available at this point in the pipeline.

Why Config Choice Is Worth Modelling At All

Configs doing identical arithmetic are not interchangeable - and which resource runs out first is a property of the config

```
time = FLOPs / peak_FLOP_per_second      # if arithmetic were all that mattered
```

It predicts the same time for the same arithmetic. Measured, the best and worst config on one shape differ by up to 2x on dense fp16 - and by 5-7x once a config spills registers.



achieved performance = whichever of these saturates first

**Which one saturates is a property of the CONFIG -
computable before anything is compiled or run.**

Autotune MEASURES. PerfModel PREDICTS.

An analytical cost model that ranks candidates before JIT compilation - three roles, only the first replaces autotune

1 . REPLACE

Analytic top-1 pick,
with no GPU launch.

Production / JIT path -
the first call already
uses the ranked config.

*the only role that
replaces autotune*

2 . PRUNE

Rank the space by
predicted TFLOP/s, hand
the top-K to the tuner.

Plugs into Triton's
existing `prune_configs_by`
hook - no fork needed.

*a faster autotune,
not a new capability*

3 . GUIDE

Answer pass-level
questions autotune
cannot reach: MFMA tile,
LDS layout, schedule.

Inside the compiler.

*autotune sees whole
configs, not passes*

Autotune can only tell you THAT a config is fast. The model tells you WHICH and WHY.

The "why" is the part that compounds: a predicted bottleneck - roofline, occupancy, LDS pressure - is transferable insight that lets a regression be diagnosed analytically instead of re-benchmarked.

The Model Mirrors the Compiler's Decisions

We do not parse IR for features. Each cost function re-implements one pass's decision logic, so the estimate reflects what the backend will actually emit

PerfModel function	mirrors which MLIR pass / decision	contributes to the cost formula
selectMfmaNonKDim()	AccelerateAMDMatmul - MFMA instruction pick	mDim . nDim . kDim -> compute cycles
deriveKWidth()	AccelerateAMDMatmul - kBase x kPack	kWidth -> A/B fragment regs + ds_read width
estimateNumBuffers()	LowerLoops::initSchedule - pipelining	numStages -> buffer count (pipeline depth)
estimateVgpr()	backend register allocation	acc tile + frags x unroll x numBuf -> occupancy
estimateLdsBytes()	LowerLoops - LDS alloc + bank padding	LDS bytes -> occupancy / ldsExceeded
estimatePerf()	the whole lowered + scheduled loop	roofline max(compute, mem, lds) -> TFLOP/s

...which combine into one predicted number per candidate

```

TFLOPS  = 2.M.N.K / (effTile x numWaves x occPenalty / clockHz)
effTile = max(compute, memory, lds) + (1 - overlap) . min(compute, memory)
compute = numMFMA . tputCyc / activeSIMDs ,  numMFMA = (BM/mDim)(BN/nDim)(BK/kDim).(K/BK)
memory  = tileBytesAB . [ (1 - l2hit)/dramBw + l2hit/l2Bw ]
lds      = ldsBytesPerCTA / ldsBwPerCU        numWaves = ceil( ceil(M/BM).ceil(N/BN) / numCUs )

```

Two operating modes, one formula: OFFLINE derives every pass decision analytically before any pass runs; IN-COMPILER reads back what the passes actually produced. The ~210-line MLIR bridge only supplies inputs.

Roofline and Occupancy Are the Same Question

Is the MFMA pipeline busy, or is HBM busy? Both are Little's law - enough work in flight to hide latency

One governing quantity

$$\text{bytes_in_flight} = \text{CTAs} \times \text{effectivePipelineDepth} \times \text{input_tile_bytes}$$
$$\text{effectivePipelineDepth} = \min(\text{numStages} - 1, \text{numKIter})$$

*Every knob that used to tie - BK, num_warps, num_stages, occupancy - appears in this one expression.
input_tile_bytes counts both operands at their real element size, including MX scale metadata - so this holds for bf16 and for MXFP4 alike.*

1 . Is the MFMA pipeline busy? (compute-bound)

hiding = wavesPerSimd x pipelineDepth
inflatedCmp = computeCyc x (1 + c / hiding) (c fitted)

Resident waves per SIMD AND prefetched k-iters both hide MFMA latency, so ONE term now ranks num_warps (W8 > W4, +23%) and num_stages (ns3 > ns2, +23%) that the old min(nW,4), ns-independent model tied.

2 . Is HBM busy? (memory-bound)

outstanding = ctasPerCU x depth x dramBytesPerKIter
dramBwEff = min(peakBW, outstanding / hbmLatency)

The occupancy penalty IS HBM saturation, not CTA count - one CTA can already saturate HBM. Block-K then falls out for free: deep-K wants big BK, shallow-K small BK. Non-monotonic, and no fudge factor is needed to produce it.

The result: the physics got UNIFIED. We removed constants - the kGran fudge, stallAmp - instead of adding them.

Dense path only. The MX / a8w4 path still uses 1/occupancy here and adds its own fill-steady compute and register model - see the MXFP4 section.

How the Model Gets Built: Predict, Validate, Correct

Every number in this talk came out of this loop. A discrepancy is not noise to be tolerated - it is the next model change

1 . PREDICT

rank_configs() emits, for every candidate and before any JIT:

- . predicted TFLOP/s
- . the predicted BOTTLENECK

The second output is what makes the loop possible - a wrong number with a named cause is debuggable.

2 . VALIDATE

Against three independent truths:

- . exhaustive tuning sweep is our top-1 the real winner?
- . generated AMDGCN are the VGPR / ds_read / unroll assumptions what was emitted?
- . rocprofv3 counters is the absolute time right?

3 . CORRECT

Triage the discrepancy first:

- . FUNDAMENTAL (silicon-rigid) -> goes INTO the model
- . COMPILER-RESIDUE (version-dependent) -> stays in the harness, never the model

Then gate on the 13-category A/B sweep. Net-negative is reverted.

re-predict the whole suite - a fix that helps one shape and hurts a category is not a fix

Why step 2 needs three sources, not one

- . One kernel, one config, timed three ways - up to 2x apart
- . Tuning cache = launch-amortised inner loop
- . Wall clock, shared GPU = launch overhead + co-tenants
- . Device time per dispatch, GPU verified idle = our ground truth
- . Protocol, not tool: device time . interleaved A/B . IQR-filtered . idle GPU
- . Fix the protocol and a result becomes reproducible by anyone who repeats it

Five Physics Insights, One Governing Quantity

Each insight replaced an empirical correction with a term that generalises. The tell: a constant that is not constant is a missing term

Insight	What it replaced	Regime fixed
1 . Clean-wave MFMA efficiency	one efficiency factor keyed on the smaller tile dimension	MEDIUM 75->98%, LARGE 83->100%
2 . Occupancy = HBM saturation	occupancy penalty as the reciprocal of VGPR occupancy	LARGE_N 71->100%, tiny-M LARGE_NK
3 . MFMA latency-hiding stall	compute / min(nW,4), ns-independent W4 = W8, ns2 = ns3 tied	compute-bound deep-K: W8/ns3 +23%
4 . Block-K = DRAM saturation	a block-K granularity fudge whose constant was never constant	LARGE_MK 83->100%, LARGE_NK 94->97%
5 . Tiny-K masked block	K/BK compute under-charge + ad-hoc depth caps	skinny 93->95->100%

One physical quantity threads through all five

$\text{effectivePipelineDepth} = \min(\text{numStages} - 1, \text{numKIter}) \rightarrow$ DRAM saturation, roofline overlap and the MFMA stall all read the same value.
This is the difference between a model and a curve fit: the insights generalise because they are the same quantity seen from three directions.

Fewer constants is the point. Each insight replaced a fitted factor with a term derived from the hardware - and accuracy improved.

Each insight is traced to the specific ranking error that produced it.

Lesson 1 - Occupancy Is Set by the Binding Resource

Registers and shared memory each cap occupancy. A ranking model has to take the lower of the two, or configs differing only in shared memory look identical

Requirement: two configs can match on register use and still differ in achievable occupancy.

one turn of the slide-8 loop

BK = 128

VGPR usage	identical
Waves from VGPR	identical
LDS at numStages=2	2.25x larger
CTAs / CU	1
Waves / SIMD	1
-> fewer waves to hide MFMA latency	

BK = 64

VGPR usage	identical
Waves from VGPR	identical
LDS at numStages=2	baseline
CTAs / CU	2
Waves / SIMD	2
-> enough waves to keep MFMA fed	

Identical registers, identical wave count from registers. Anything keyed on registers alone cannot tell these two configurations apart.

The standard occupancy rule, applied to the ranking

$$\text{effectiveWavesPerSimd} = \min(\text{wavesFromVgpr}, \text{wavesFromLds}) / \text{numSimdPerCU}$$

Shared-memory capacity, not register pressure, is what sets occupancy here. Taking the lower of the two limits is textbook - the point is that a config-ranking model has to do it, because this is exactly where two otherwise-identical configurations diverge.

Lesson 2 - Cache Reuse Follows Launch Order, Not Tile Size

Workgroup swizzle is a standard Triton technique. The model has to account for it, because it decides what stays resident in L2

Several L2 partitions, not one



one turn of the slide-8 loop

Triton already lets a kernel choose the order workgroups are issued in - GROUP_SIZE_M grouping, and swizzles on top of it. That order decides which A rows and B columns are still resident when the next tile asks for them.

OLD - a closed-form hit rate

$I2HitRate = f(\text{per-tile bytes})$

Per-tile bytes only, so BK cancels out entirely.
Two configs with the same per-tile footprint get the same predicted hit rate - and BK could never move the ranking.

NEW - hit rate from the swizzle

Walk the workgroup order the swizzle produces and track what each partition would still be holding.

BK and tile shape now both move the hit rate, because both change what the next tile finds there.

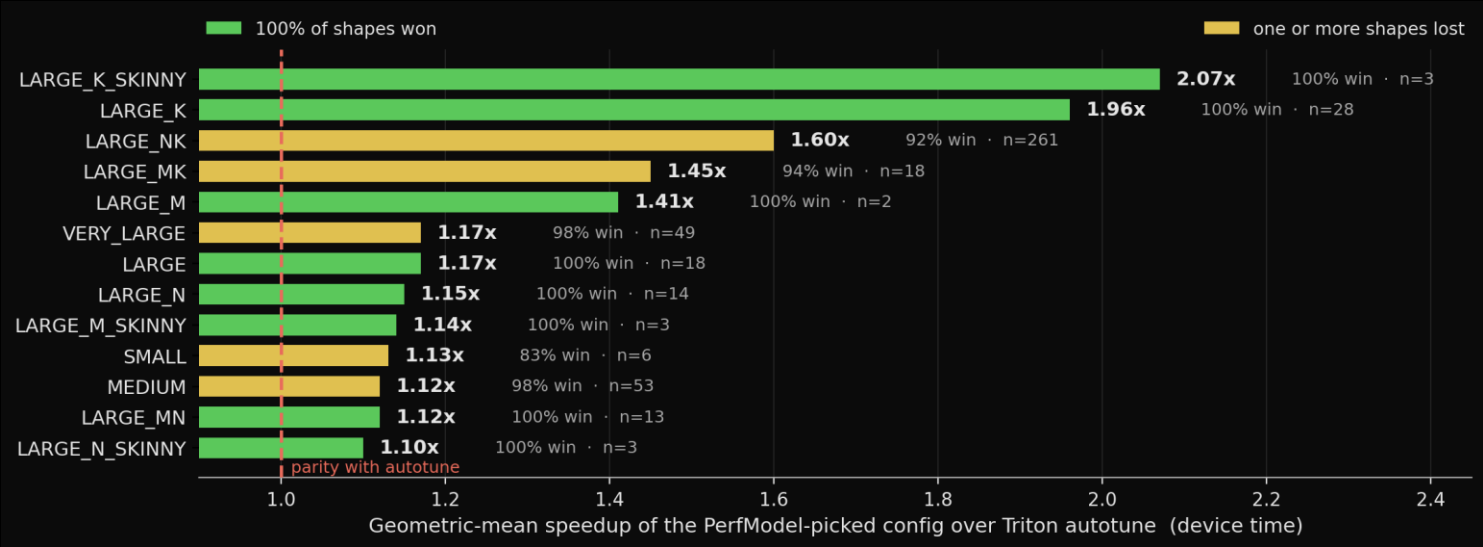
The general lesson: a cache model has to see the scheduler, because that is the axis that moves the answer.

Side effect: a hit rate we could trust is what let the model use an L2 bandwidth roof at all. Before this it was forced to zero.

Results: 471 Shapes, 95% Win, Geomean 1.45x

PerfModel config-pick vs Triton autotune, device time on a verified-idle MI355X (gfx950)

The 471 shapes are not synthetic. They are the GEMMs that DeepSeek R1 and V4, Kimi K3, Llama 3 and Llama 4 Maverick, Qwen3 235B-A22B and GPT-OSS 120B actually issue - taken at TP1, TP4 and TP8, with M swept from decode to prefill, then grouped into 13 shape categories.



10 of 13 categories at 100% . no configuration in the suite regressed . pick_gemm_config is a sub-ms to ~10 ms CPU call, against autotune's seconds to minutes

Precision	Workload	Result
fp16 (a16w16)	dense GEMM suite, 471 shapes	95% win, geomean 1.45x
fp8 x mxfp4 (a8w4)	MoE case study, gpt-oss-120B W4A8	offline top-1 rank 0.959 . end to end 0.951 vs the tuned table

Measured July 2026 on one AMD Instinct MI355X (gfx950); PerfModel branch on a Triton 3.8.0 fork; fp16; standard Triton matmul; device time, interleaved A/B, IQR-filtered, on a GPU verified idle - wall-clock timing inflates short kernels, so the protocol is part of the result. Results may vary with system configuration, software versions, driver and workload.

MXFP4 Was Not a Reparameterisation

gpt-oss-120B W4A8 grouped GEMM: four changes to the physics, each gated on data type so the dense path is unaffected

Change	Mechanism	Effect on a8w4 top-1 rank
fp4 MFMA peaks + fractional MX bytes	10.1 PF fp4 roof; narrowest-operand rate; fp4 = 0.5 B/elem + E8M0 scale per 32	gives the MX path a correct roofline at all
VGPR: stop scaling live fragments by num_stages	over-counted registers caused false invalidation near the VGPR cliff	removes false invalidations near the register cliff
num_stages tie-break for MX-scaled operands	shallow-K ns2 vs ns3 tie broken toward the measured winner	matches the measured winner on shallow-K shapes
Origami fill / steady tile latency, dtype-gated	(nS-1) K-iterations fill serially; only steady iterations overlap; raw compute roof	top-1 rank 0.959

The register cliff is the story: on this path the fp4 tiles that lose are the ones that spill, where a dense BF16 tile of the same shape does not. Modelling register pressure is load-bearing here, not an optimisation.

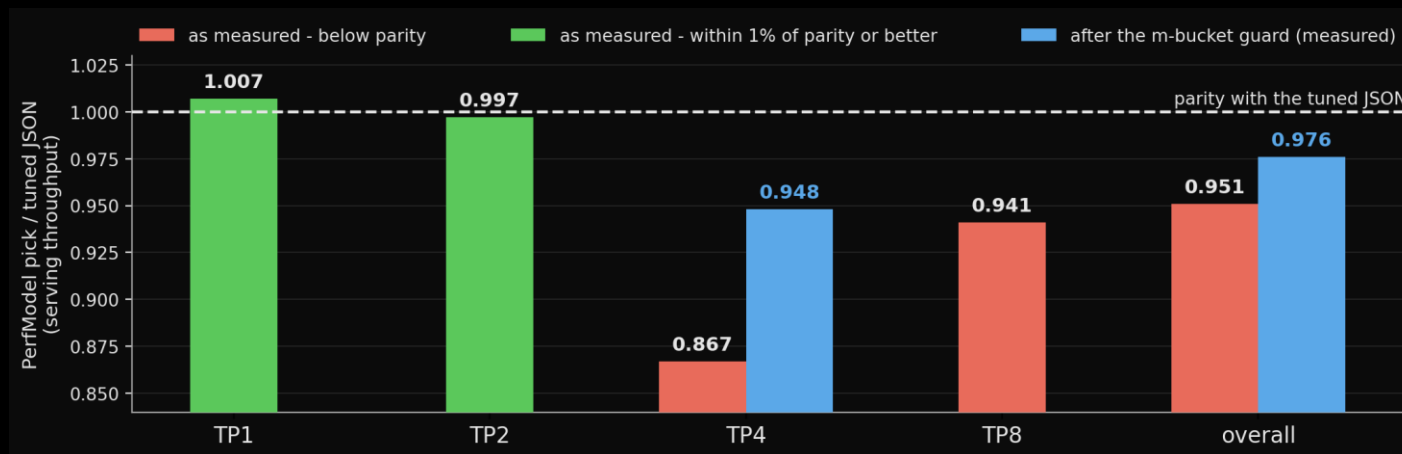
Every one of these is gated on hasMxScales(), so the dense 471-shape result was unchanged across every validation dataset we ran.

Ground truth: an exhaustive winner sweep over 28 tuned shapes, with the measured best config for each.

Measured May 2026 on one AMD Instinct MI355X (gfx950); gpt-oss-120B W4A8 (MXFP4 weights, FP8 activations); ranks are top-1 agreement against the measured winner over 28 tuned shapes. Results may vary with system configuration, software versions, driver and workload.

Where the Model Does Not Yet Match a Hand-Tuned Table

A different test: MXFP4 instead of fp16, configs picked live in the serving loop instead of offline, and scored on end-to-end throughput against a hand-tuned dispatch table instead of against autotune



TP1 is above parity and TP2 within 0.3% of it. The shortfall is entirely at TP4 and TP8.

Three things left to close, all specific: a candidate grid that does not yet reach every tuned winner, one tile shape the model over-rates, and too coarse an M fed in as input. The loop named them; none needed guesswork.

The blue bars are the third cause fixed. An m-bucket guard lifts TP4 from 0.867 to 0.948 and the whole surface from 0.951 to 0.976 - with no change to the model itself.

**A synthetic uniform-routing microbenchmark and end-to-end serving disagreed on this path.
We validate config selection end to end now. Offline prediction is not validation.**

Measured July 2026 on one AMD Instinct MI355X (gfx950); gpt-oss-120B W4A8 (MXFP4 weights, FP8 activations); vLLM with AITER kernels on Triton 3.6.0; 56-cell serving surface; ratio is the PerfModel-selected config against the tuned dispatch table on the same node and harness. Results may vary with system configuration, software versions, driver and workload.

What We Would Tell Ourselves on Day One

And where this goes next

Lessons

Unify the physics; do not accumulate fudges.

A fudge whose constant is not constant is a missing term.

Measure the right thing or chase phantoms.

Device time, not wall clock. Better measurement UNCOVERED two real losses.

Gate every change on the full A/B sweep.

Net-negative on a reliable category means revert, not explain away.

The model only works if you feed the misrankings back.

No insight here was designed top-down. Every one came from a regression.

Offline prediction is not validation.

The 0.959 offline rank still lost end to end. Validate where it ships.

What's next

Close the a8w4 serving gap, in order: grid coverage, then bm128 nw8 ranking, then histogram-aware M.

Port the MX occupancy term from 1/occupancy onto the Little's-law saturation model; add mxfp8 and int8 a8w8.

Extend to attention kernels - different parallelisation, different memory pattern.

Track prediction accuracy continuously, so a model change that helps one regime and hurts another is caught by the sweep rather than by a user.

The through-line: a cost model is only as good as the loop that corrects it - and the loop only works if the ground truth is honest.



Questions?

Xiaohu Guo, Lixun Zhang . AMD . PyTorch Conference 2026

Copyright and disclaimer

(c) 2026 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, AMD Instinct, AMD CDNA, ROCm and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and other jurisdictions.

PyTorch is a trademark of The Linux Foundation. Triton, Gluon and LLVM are projects of their respective maintainers and communities. Model names referenced in this presentation - including DeepSeek, Kimi, Llama, Qwen and GPT-OSS - are the property of their respective owners. Other product, service and company names appearing here are for identification purposes only and may be trademarks of their respective owners.

The information in this presentation is provided for informational purposes only and may contain technical inaccuracies, omissions and typographical errors. Performance figures are measurements of the specific configurations described alongside them; results may vary with system configuration, software versions, driver and workload. AMD assumes no obligation to update this information.