



CONFERENCE

—NORTH AMERICA 2026—

Primus Tuning

Hybrid Projection and Agentic Search for Distributed Training

Anshu Raina & Peyman Razaghi

THE PROBLEM

Every tuning knob moves several bottlenecks at once

ONE RUN = ONE COUPLED CONFIGURATION

TP · PP · VPP · EP · CP · DP

micro-batch · sequence length · grad accumulation · recompute · FSDP · overlap

Each choice shifts the same four constraints

MEMORY

fit ↔ OOM

COMPUTE

GPU utilization

NETWORK

collective
overhead

PIPELINE

bubble fraction

- ▶ **Irregular:** throughput is not smooth across knobs, so local tweaks and interpolation mislead.
- ▶ **Near-optimal sits at the cliff edge:** one small change flips fit → OOM, or fast → slow.
- ▶ **Every bad trial is expensive:** queue time and cluster hours before useful feedback.

MIXTRAL 8×22B · 64 GPUs

~160

TP × PP × EP × CP layouts on 64 GPUs

- × 8 micro-batch sizes (1–8)
- × 3 recompute: none / selective / full
- × 2 MoE backends: All-to-All / DeepEP
- × 5 pipeline schedules (1F1B ... ILP)

~10⁴

topologically valid configurations, before checking memory or throughput

What is **Primus**?

- ▶ **AMD's open-source training framework** for large models on AMD Instinct GPUs.
- ▶ **One config system, several backends:** Megatron-LM, TorchTitan, JAX MaxText, Megatron-Bridge.
- ▶ **Full lifecycle:** pretraining, SFT / LoRA post-training, and RL.
- ▶ **Ready-to-run models:** Llama 3 / 4, DeepSeek V3 / V4, Qwen3, Mixtral, GPT-OSS, and more.
- ▶ **One CLI everywhere:** primus-cli runs locally, in containers, or on Slurm; shipped as a pip wheel and rocm/primus Docker images.

Primus-SaFE

Platform: cluster management, fault tolerance, scheduling

Projection + Tuning Agent

Primus-LM

Training framework: Megatron-LM | TorchTitan | MaxText, unified CLI

Primus-Turbo

Kernels: attention, GEMM, grouped GEMM, collectives (AITER, CK, hipBLASLt, Triton)

github.com/AMD-AGI/Primus

Why existing tools don't close the loop

Approach	Examples	Limitation for planning at MoE scale
Analytical / roofline	Calculon, Galvatron	Misses clock throttling, kernel/framework-stack drift, grouped-GEMM & MoE routing effects
Parallelism auto-search	FlexFlow, Alpa	Requires profiling each candidate on target hardware
Trace / simulation	ASTRA-sim, Echo, Lumos	Needs target-hardware traces or heavy calibration; don't scale to sweeps
Config autotuning	DeepSpeed Autotune	Grid-search ZeRO/MBS; requires full scale

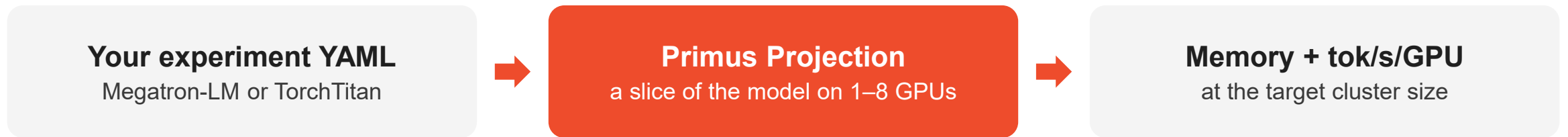
Our insight: benchmark a small slice of the job on the real stack; model the rest analytically.

Primus Projection

Benchmark a slice, project the whole cluster.

Built on the PyTorch training stack

- ▶ **Same experiment YAML:** Megatron-LM or TorchTitan, no separate model description.
- ▶ **Real kernels:** time representative dense / MoE layers with HIP events; read peak allocator memory.
- ▶ **One PyTorch harness:** handles both backends and their tensor-layout differences.
- ▶ **TorchTitan-native:** maps TorchTitan model and parallelism settings directly.
- ▶ **Coverage:** FSDP2, activation checkpointing, FP8 / BF16, and 1F1B / interleaved / zero-bubble schedules.



Results use Megatron-LM on AMD Instinct MI300X, MI325X, and MI355X.

At MoE scale, activations dominate GPU memory

FIT · MIXTRAL 8x22B PP4 / EP8, PER GPU

~2.5×

activations vs. model state

Model state

79 GB

Activations

198 GB

ONE MoE LAYER vs. ATTENTION

~30×

activation memory per layer

Attention

0.51 GB

MoE MLP

16.19 GB

FULL RECOMPUTE

~72×

less stored per MoE layer, for ~33% more compute

No recompute

18.2 GB

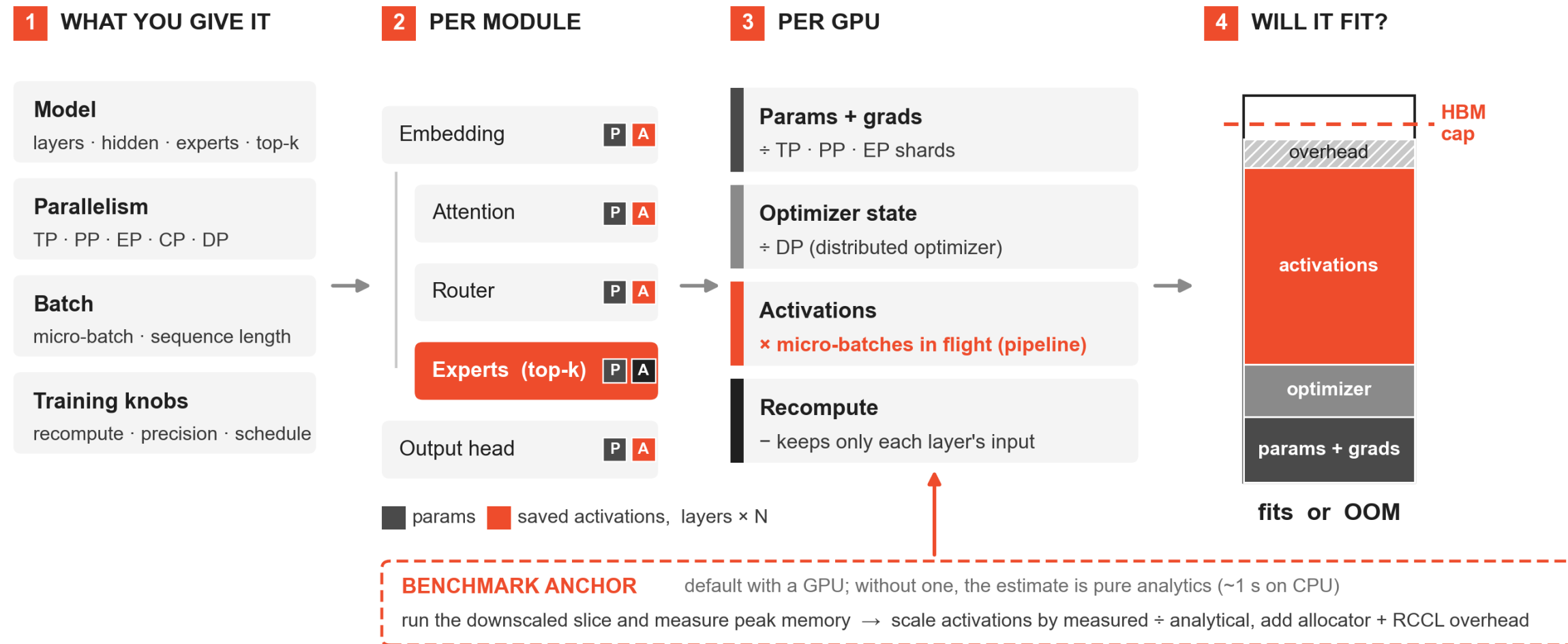
Full recompute

0.25 GB

Middle and right: large-MoE example from the Primus Projection blog (H 8192, top-k 36, MBS 4, seq 16K, CP 4, BF16).

Get memory wrong and you OOM or waste GPUs. So the agent searches recompute, MBS, and FP8 activations.

Memory model: from the module tree to one GPU



Measure a slice on real GPUs, model the rest

TARGET



Mixtral 8x22B
TP1 · PP4 · EP8 · 32 GPUs

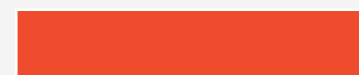
Downscale

fit 8 GPUs:
PP 4→1

MEASURE

1 node · TP1 · PP1 · EP8

Dense layer



fwd + bwd timed

MoE layer

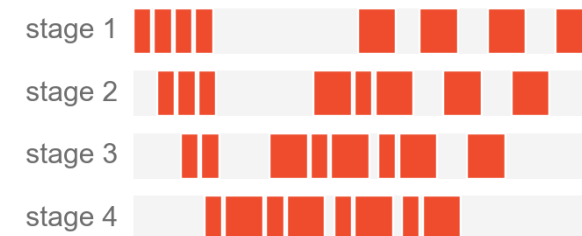


compute | in-node All-to-All

real kernels on the real stack

Upscale

RESTORE THE TARGET



replay the PP schedule,
add P2P and DP AllReduce

→ **iteration time, tok/s/GPU**

1 → 8 nodes: -1.4% vs. measured

Downscale Heuristic shrink to the GPUs you have, in order of how exactly each dimension can be restored · CP is never reduced



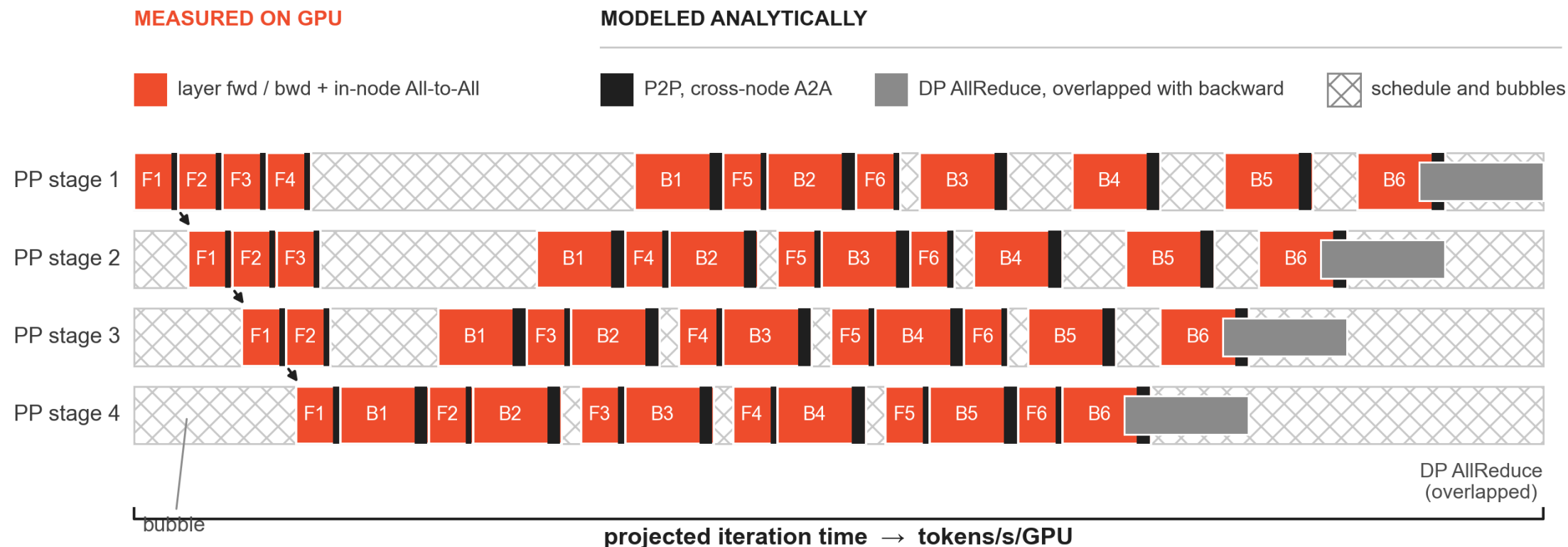
MIXTRAL TARGET (TP1 · PP4 · EP8), BY BUDGET

8 GPUs → TP1 · PP1 · EP8 (this example)

4 GPUs → TP1 · PP1 · EP4

1 GPU → TP1 · PP1 · EP1

What we measure, and what we model

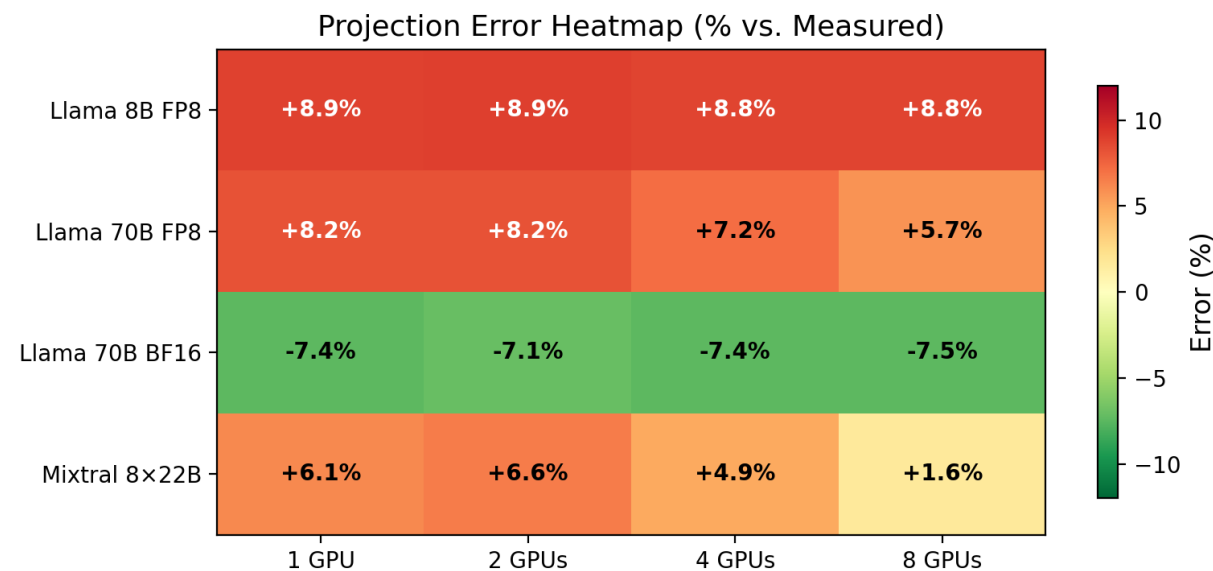


Only the layers run on GPUs, and the modeled parts hold up

Collectives alone (preflight, MI325X, 2 GPUs—4 nodes): 4.2% avg error · Mixtral from 1 GPU, schedule + all comm modeled: within 6.1%

All 16 cells within ~10% of measured

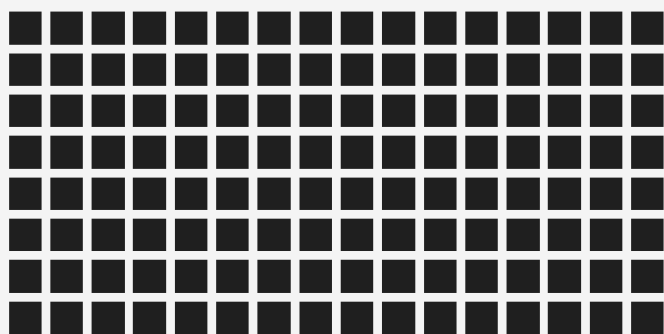
- ▶ **Spans 2 model families** (dense Llama 3.1, MoE Mixtral 8x22B)
- ▶ **2 precision modes** (FP8, BF16)
- ▶ **2 GPU generations** (MI325X gfx942, MI355X gfx950)
- ▶ **FSDP2 on Llama 70B** (full shard, FP8 and BF16); Llama 8B runs pure DP
- ▶ **Pipeline + experts on Mixtral** (PP=4, VPP=2 interleaved, EP=8 All-to-All)



Fidelity holds up to 1,024 GPUs

DeepSeek V3 671B

MLA attention · 3 dense + 58 MoE layers, unseen in development



128 nodes · 1,024 GPUs

Projected

2,808

Measured

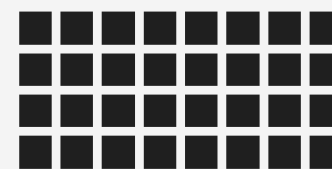
2,946

4.7%

error

Qwen3-235B-A22B

MoE · 235B total, 22B active



32 nodes · 256 GPUs

Projected

4,860

Measured

4,952

1.9%

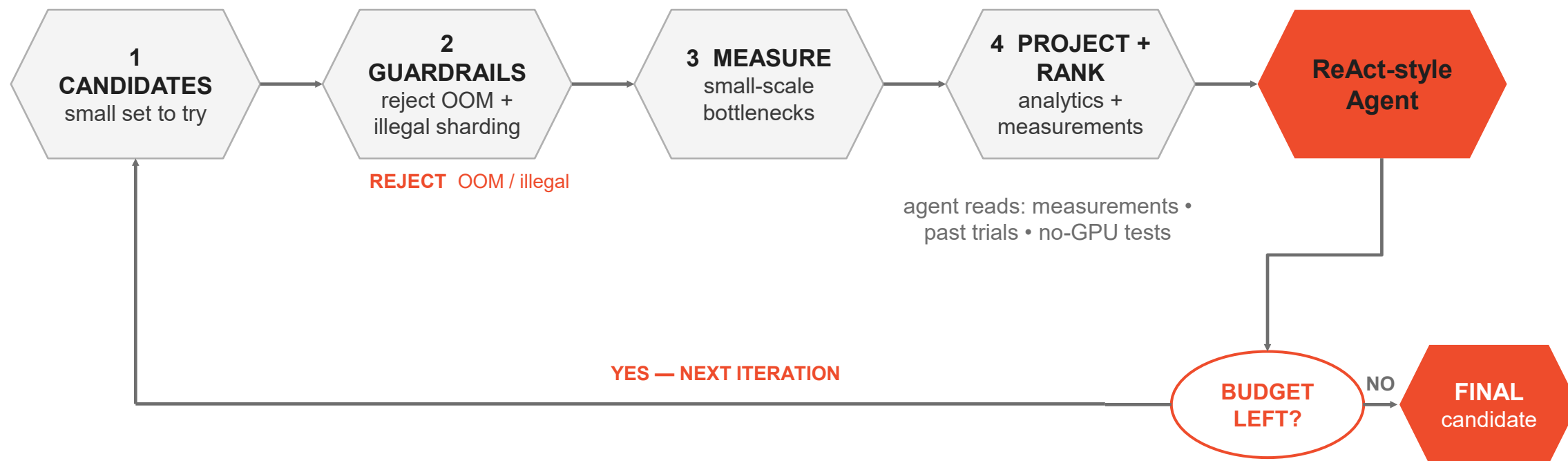
error

FP8, tokens/s/GPU. 1 square = 1 node of 8 GPUs. Two independent large-scale checks, not a small-scale artifact.

Primus Tuning Agent

Agentic reasoning searching the nonlinear space guided by the projection oracle

Closed loop search using cheap projections and agentic reasoning



Each cycle learns; expensive cluster runs wait until search converges.

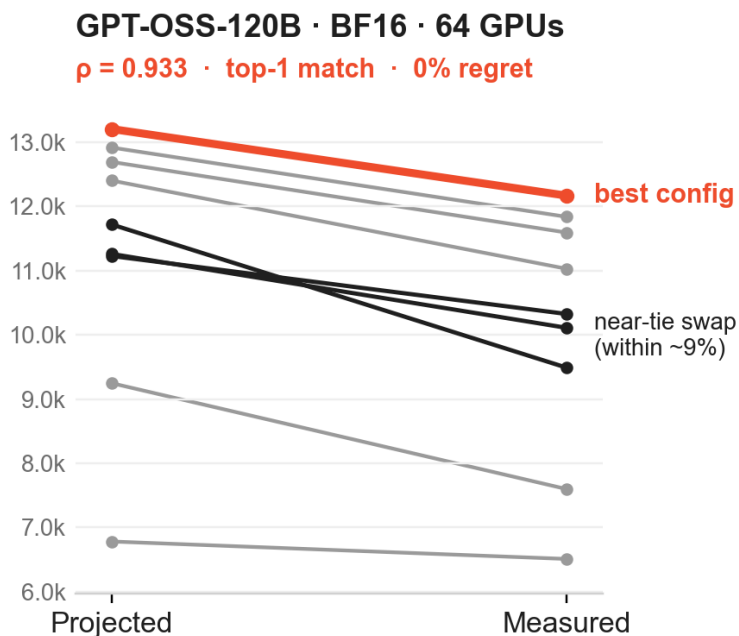
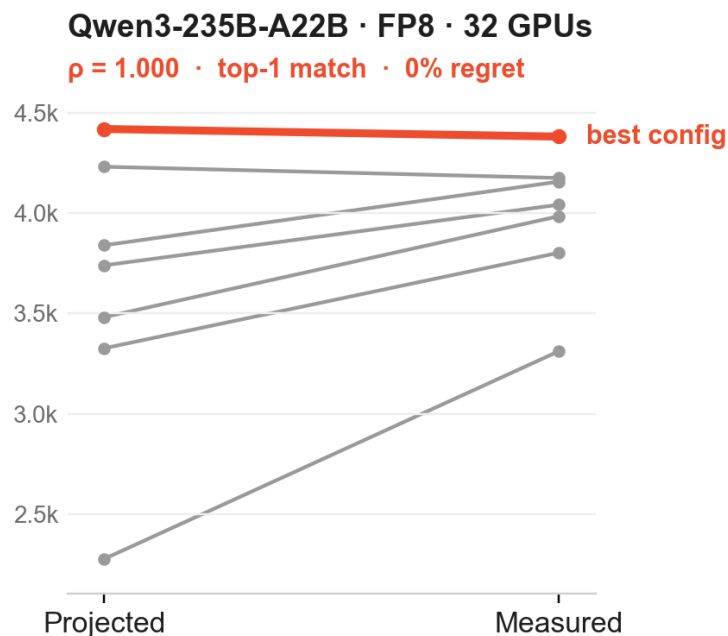
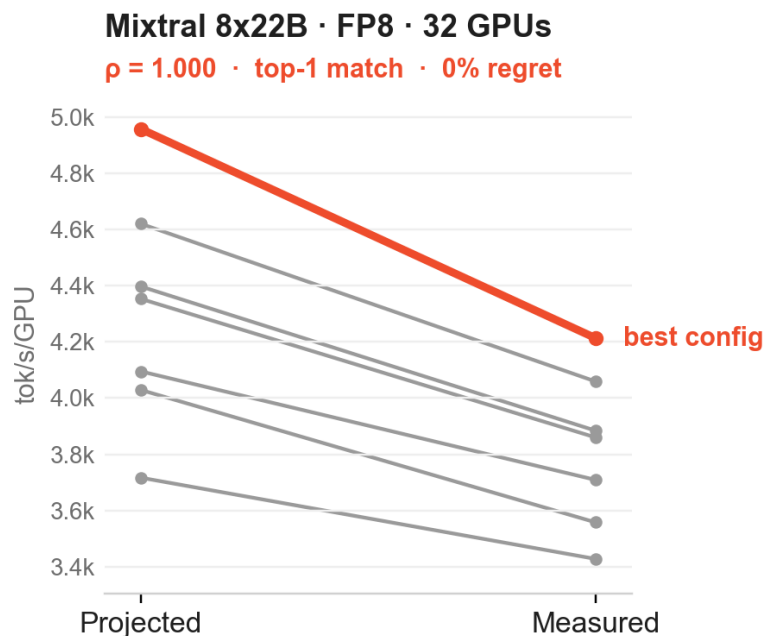
Seed planner + LLM agent, scored by the projection

- ▶ **Cheap trials:** each candidate is scored by a sub-node benchmark (1 GPU up to one node) instead of a multi-node launch; a memory pre-filter rejects OOM configs for free.
- ▶ **Phase 1, seed planner:** 12 single-axis sweeps — recompute, MoE backend (All-to-All vs. DeepEP), pipeline schedule and VPP, micro-batch and distributed optimizer.
- ▶ **Phase 2, LLM loop (DSPy, ReAct-style):** reads the trial ledger, proposes cross-axis combinations as JSON, and calls the projection as a tool. Up to 25 trials × 3 rounds.
- ▶ **Output:** the highest-throughput memory-legal config, as a ready-to-run Primus YAML.

```
python -m primus.agents.tuning_agent \  
  --workload examples/megatron/configs/MI355X/mixtral_8x22B_v0.1-BF16-pretrain.yaml \  
  --target-cluster examples/megatron/guides/tuning_agent/target_cluster_mi355x_4nodes.yaml \  
  --out-dir tuning_runs/mixtral-22b      # add --seed-only to skip the LLM stage
```

Search needs the right ranking, not perfect absolute numbers

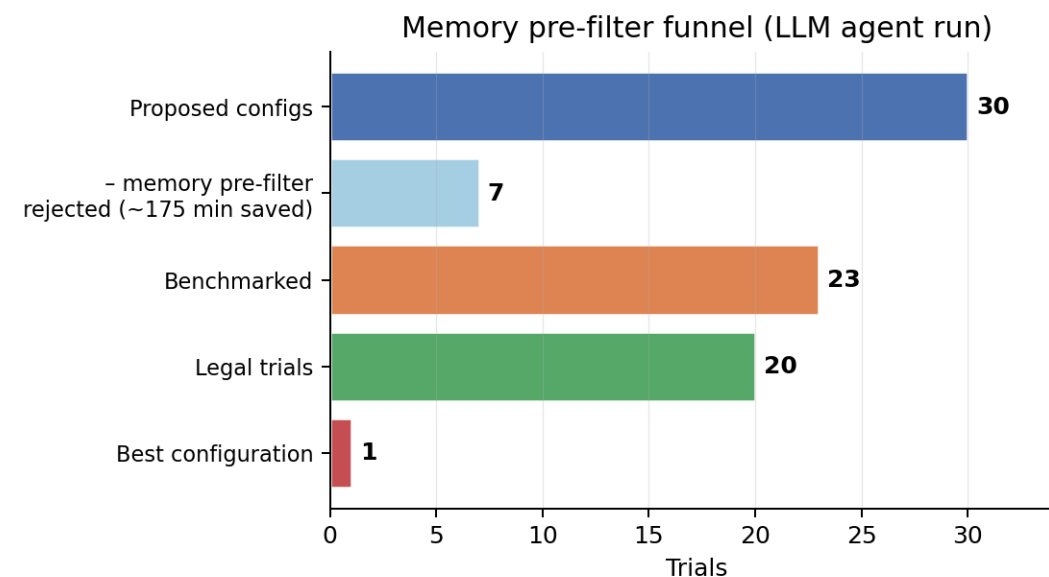
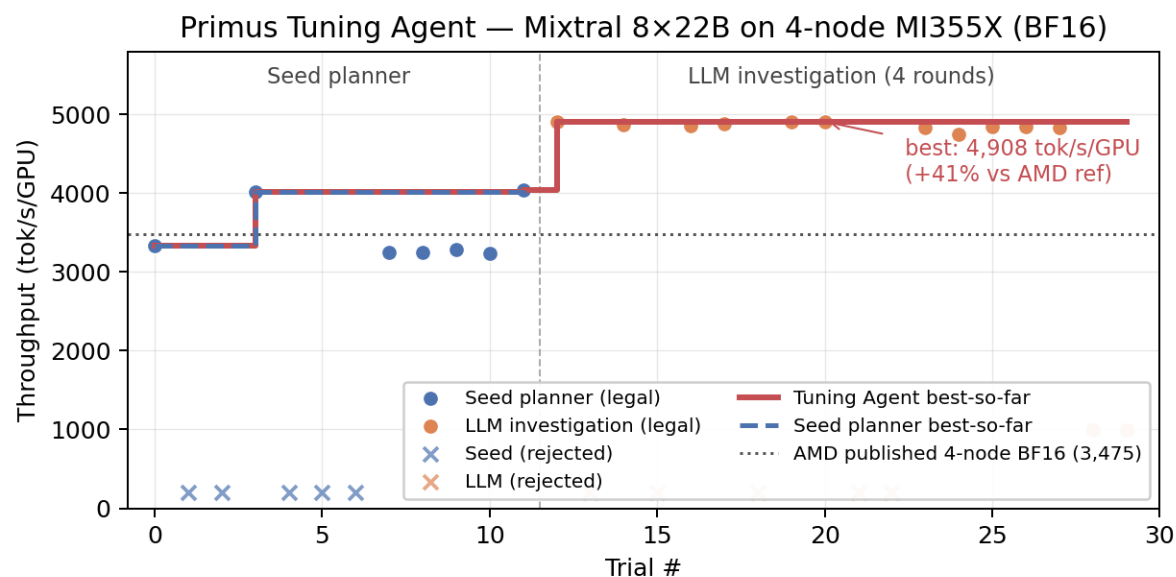
- The agent's pick is only as good as the measured ordering. Each line is one config, projected vs. measured.



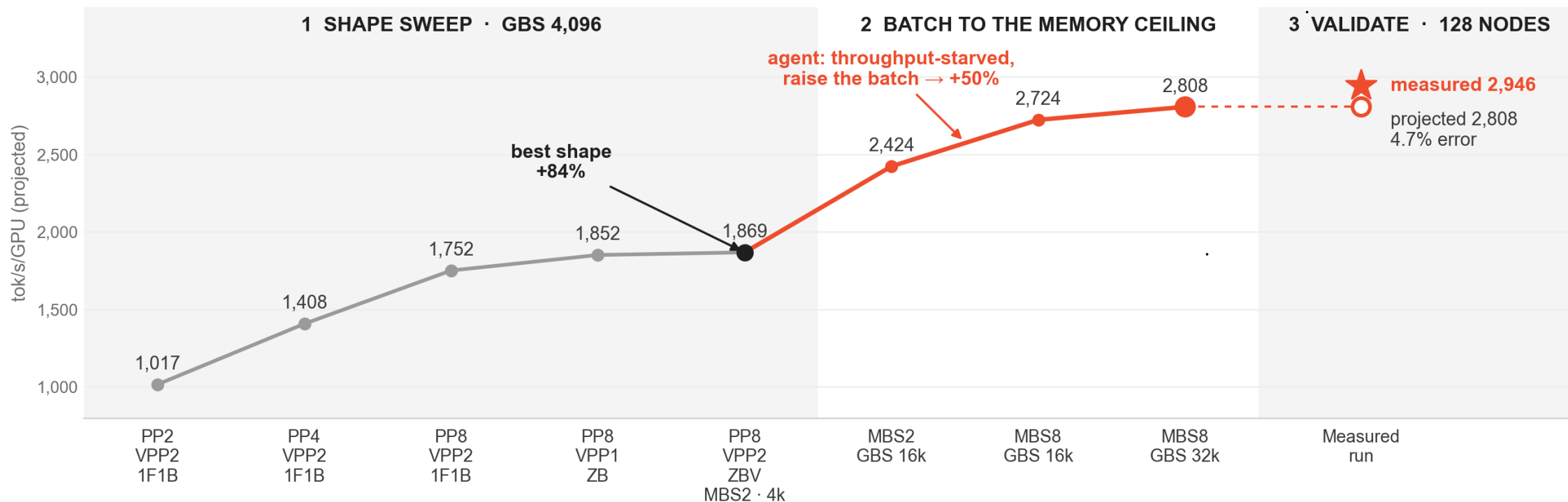
Offsets tilt the lines; they don't reorder them. The measured best is the projected best in all three sweeps.

Mixtral 8x22B — +27% over the reference in ~30 minutes of GPU

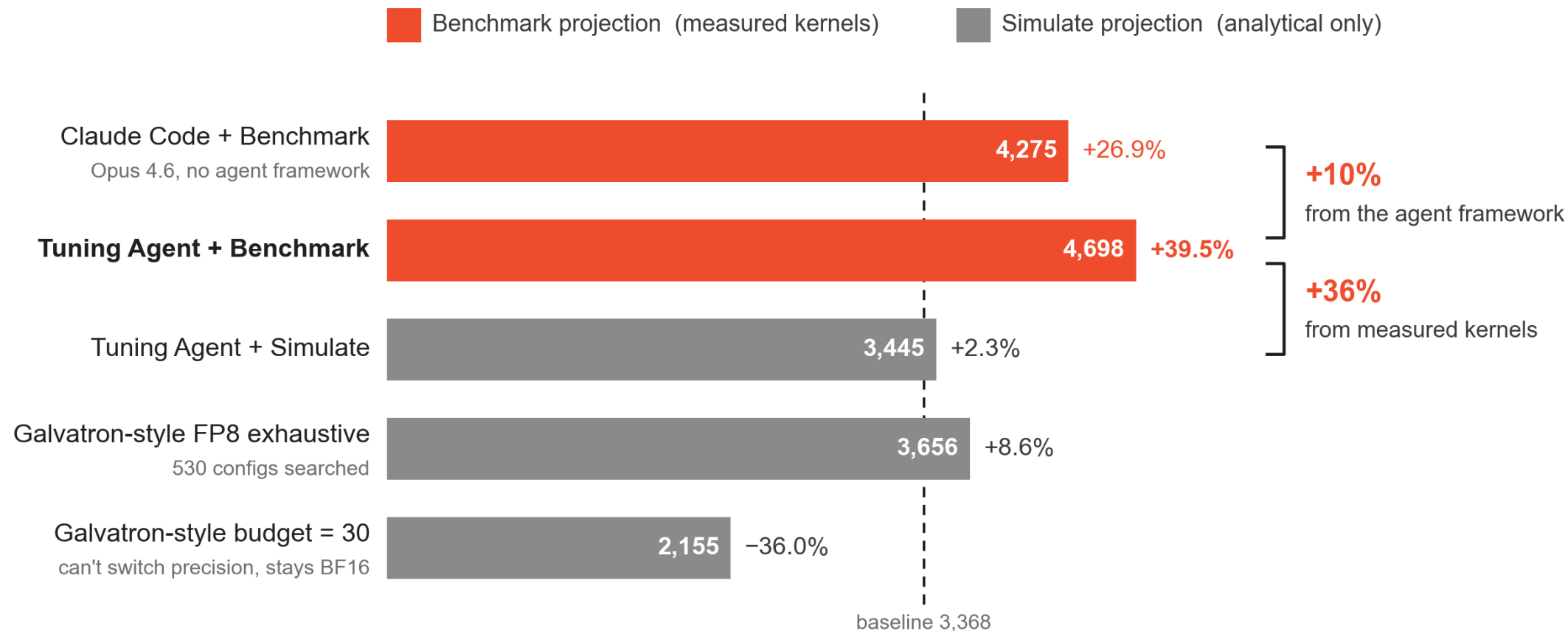
- ▶ **Baseline:** AMD's 4-node BF16 recipe, 3,475 tok/s/GPU.
- ▶ **Seed planner:** 4,036 (+16%) from single-axis sweeps.
- ▶ **LLM:** MBS=1 + no recompute + DeepEP → 4,402 (+27%).
- ▶ **Cheap:** 7 of 30 trials rejected before GPU time; ~30 min.



Inside The Agent's Head: DeepSeek V3 671B, 1,024 GPU



What drives the win: measured kernels + agent



Mixtral 8x22B, 4-node MI355X. Projected tokens/s/GPU of the best config each method finds.

Plan small, train at scale

PROJECTION

~10%

error vs. measured

Dense and MoE, FP8 and BF16, up to 1,024 GPUs, from benchmarks on 1 GPU up to one node.

RANKING

0%

selection regret

Projected #1 config was the measured #1 on all three MoE sweeps.

TUNING AGENT

+27%

over AMD's Mixtral 8x22B recipe

Found in 29 min on one node, and +10% over Claude Code given the same projection.

GPU COST

~40×

fewer GPU-hours to tune

~4 GPU-hours on one node vs. ~160 for trial and error on the 4-node target.

Benchmark a slice → Project the cluster → Let the agent search

Spend GPU-hours on training, not on trial and error

MIXTRAL 8x22B · FINDING THE +27% CONFIG · 30 CANDIDATES

Trial and error on the 4-node target

~160 GPU-hours

Primus Tuning Agent

~4 GPU-hours · 29 min on one 8-GPU node

~40× fewer GPU hours

for the same answer, and the cluster stays free for training.

~1 min per trial

each of the 23 benchmarked candidates ran on one 8-GPU node (median 62 s)

0 GPU-hours

spent on the 7 of 30 configs the memory pre-filter rejected as OOM

GPUs are costly and scarce. Search on the node you have; launch the cluster once.

Target: conservative ~10 min per 4-node launch, no queue time. Primus: from the run ledger, 29 min on one 8-GPU MI355X node (~1 min per benchmark trial; OOM configs rejected on CPU).

Tune on a slice, launch the cluster once

1

Tuning doesn't have to burn the cluster

Whole Mixtral search: ~4 GPU-hours on one node vs. ~160 of trial and error on the 4-node target.

~40×

fewer GPU-hours

2

Reduce → measure → restore works

Benchmark a slice of the job on the real stack, model the rest analytically.

~10%

error up to 1,024 GPUs

3

Benchmarking real kernels pays off

Same agent, measured vs. simulated scoring: a much better final config.

+36%

vs. simulate-only

4

At MoE scale, memory modeling is critical

Activations become the largest share of memory; OOM configs are rejected before any GPU time.

7 of 30

rejected, 0 GPU-hours

5

A purpose-built agent beats a general coding agent

Same projection, Tuning Agent vs. Claude Code: the planner and search loop matter.

+10%

vs. Claude Code

Try Primus Projection and the Tuning Agent

READ

Beyond the Trial-and-Error Loop (NeurIPS 2026)

arxiv.org/abs/2610.19423

RUN

Projection and Tuning Agent ship in Primus

github.com/AMD-AGI/Primus

LEARN · ROCm BLOGS · rocm.blogs.amd.com

Primus Projection .../primus-projection

Primus Tuning Agent .../primus-tuning-agent

CONTRIBUTE

New models and hardware profiles welcome

Open an issue or PR on GitHub

QUICK START

```
# Install the CLI
pip install primus \
  --extra-index-url \
  https://amd-agi.github.io/Primus/simple/
```

```
# Or use the training image
docker pull rocm/primus:v26.8
```

```
# Dry-run the agent on a CPU-only host
python -m primus.agents.tuning_agent \
  ... --dry-run --seed-only
```



Scan for the code

github.com/AMD-AGI/Primus



CONFERENCE

—NORTH AMERICA 2026—

Thank you! Questions?



Anshu Raina
LinkedIn



Peyman Razaghi
X · @peyman_razaghi