



CONFERENCE

—NORTH AMERICA 2026—

One Model Definition, Many Accelerators

Scaling vLLM Across Hardware Without Forks
Richard Zou (Meta) · Thomas Parnell (IBM Research)

The tension

Head models · frontier GPUs

A small set of mission-critical models on rapidly scaling GPU infrastructure (Blackwell, GB300, soon to be Vera Rubin). Needs frontier performance via custom kernels specialized to the hardware.

Tail models · diverse hardware

A long tail of models on older and prosumer GPUs and alternative out-of-tree accelerators (e.g. IBM Spyre). Needs portability and prefers a shared stack.

Can vLLM serve both without forking model code per platform?



Part 1 · Hardware-specific vs Hardware-agnostic

Richard Zou

vLLM at the frontier: “hardware-specific” models

- Previously: one model definition, shared by all backends
- RFC #42770 (May 2025): hardware-specific model definitions
 - `models/deepseek_v4/nvidia/model.py`
 - `models/deepseek_v4/amd/model.py`
 - `models/deepseek_v4/xpu/model.py`
- Why: better performance, faster developer iteration
- Custom model and hardware-specific kernels. No full-model `torch.compile`.
- All new frontier models (~10 total) in recent months use this style

...but that doesn't serve the whole user base

- Significant community interest in alternative hardware
 - More choice is good for the community in the long term
- Supporting many models × many accelerators is hard
 - Sharing parts of the stack across hardware is the simplest, most reliable way
- Goal: embrace hardware-specific and hardware-agnostic model definitions, but set boundaries so neither side slows the other

Proposal: HW-agnostic models in core vLLM

Community-maintained, in-tree. Backends choose: use it, or bring their own hw-specific definition.

```
models/  
  deepseek_v4/  
    nvidia/  
      model.py  
  amd/  
    model.py  
  hw_agnostic/ (new!)  
    model.py
```

RFC requirements

HW agnostic

Native torch ops, plus a custom op abstraction designed for dispatch to different platforms. Testable in core vLLM.

Reasonable baseline perf

Some reasonable baseline performance, but not hyper-optimized. We prefer ease of portability across existing and new backends.

Fullgraph capture

fullgraph
torch.compile-able: TPU, Intel Gaudi, IBM Spyre, CPU benefit heavily; torch.compile is one of PyTorch's levers for hardware heterogeneity.

Isolation

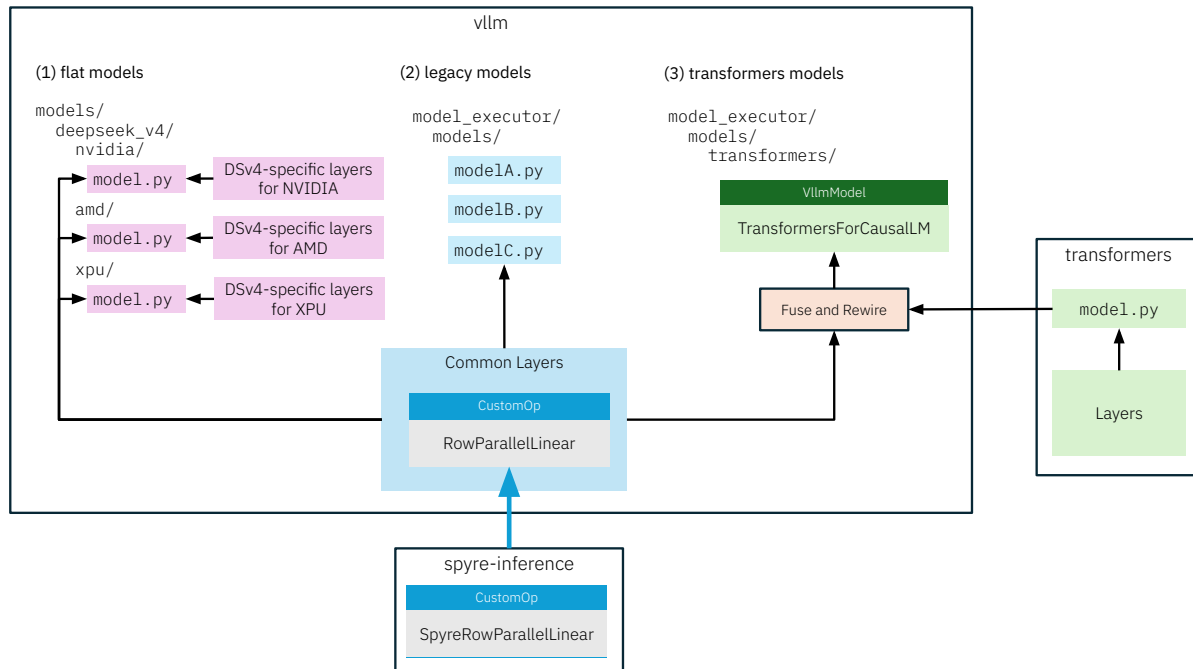
Must not affect hardware-specific (e.g. NVIDIA CUDA) model definitions.



Part 2 · Making it real

Thomas Parnell

Model definitions in vLLM today



Three flavours:

1. New HW-specific aka "flat":
`vllm/models/`
2. Legacy:
`model_executor/models`
3. Transformers backend

All 3 are currently consume common layers from `model_executor/layers`.

These layers are then customized by OOT plugins (e.g., `spyre-inference`).

What do these common layers give us?

1. torch.compile

- Critical for functionality and performance of OOT hardware plugins like IBM Spyre.
- Also important for the vLLM transformers backend, which relies on torch.compile to achieve close-to-native performance.

2. Out-of-tree extensibility

- CustomOp: plugin overrides the forward function
- PluggableLayer: plugin overrides the entire layer
- These mechanisms are very useful for OOT plugins like IBM Spyre that inject custom memory layouts etc.

The problem: flat-model work will break both

- Flat model workstream need to evolve layers in ways that make them incompatible with (a) torch.compile and (b) CustomOp.
- OOT plugins would need to maintain their own models and layers
 - New model: PR to transformers, then vLLM, then potentially every OOT plugin
- Transformers backend regresses on GPU
 - Legacy models are being removed and re-pointed at the transformers backend
- Older and prosumer GPUs left behind
 - A significant share of users still run them
- We argue we should maintain a set of “HW Agnostic” layers that can continue to serve these use-cases.

HW Agnostic layers: 4 design principles

Compilable

Fullgraph
torch.compile-able.
Accelerators that need
compile keep using it.

Extensible

Keep CustomOp and
PluggableLayer so
OOT plugins can
override when
necessary.

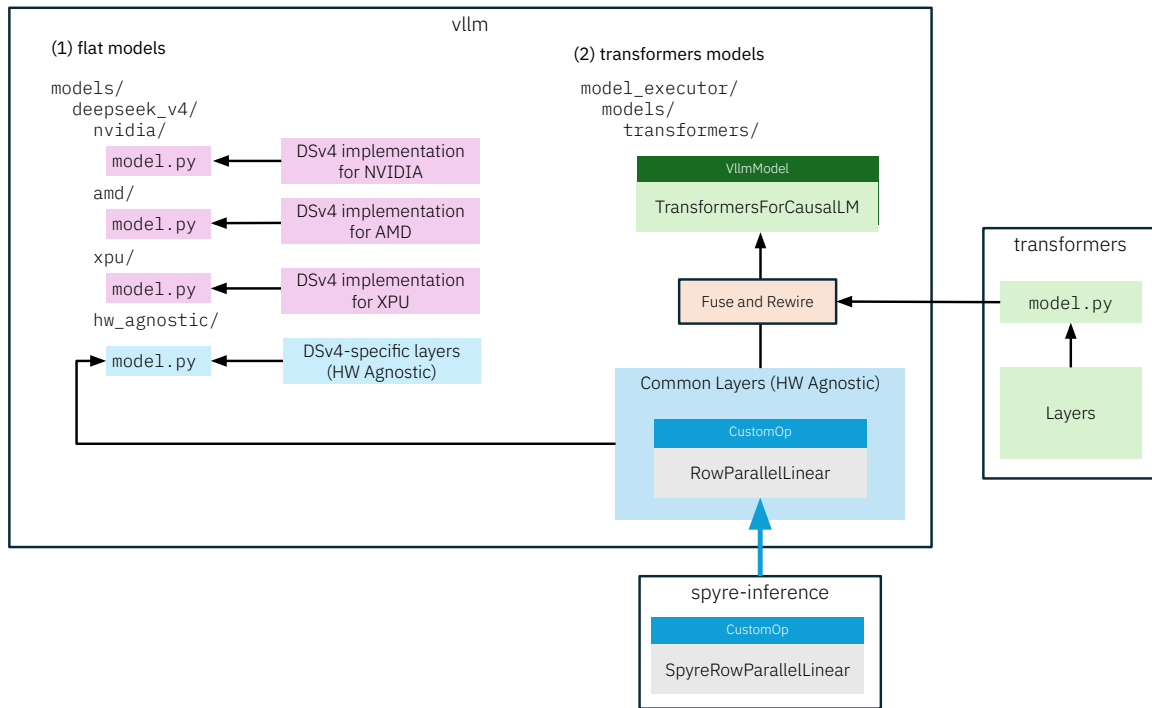
Isolated

Own layers and ops,
separate from the
HW-specific paths.
Both sides move fast.

Portable

Native PyTorch, or
portable DSLs like
Triton and Helion.
Runs anywhere they
do.

Target Architecture



Assumption: legacy model definitions will eventually be removed

Each model is either flat (per-vendor `model.py`) or falls back to the `transformers` backend.

We will offer a HW-agnostic path for both, built on
model executor/hw agnostic.

Available now via transformers backend

The fuse-and-rewire step now targets `model_executor/hw_agnostic` instead of `model_executor/layers`. Landed in main (#49458) for a subset of layers.

```
USE_HW_AGNOSTIC=1 vllm serve google/gemma-4-31B --model-impl=transformers
```

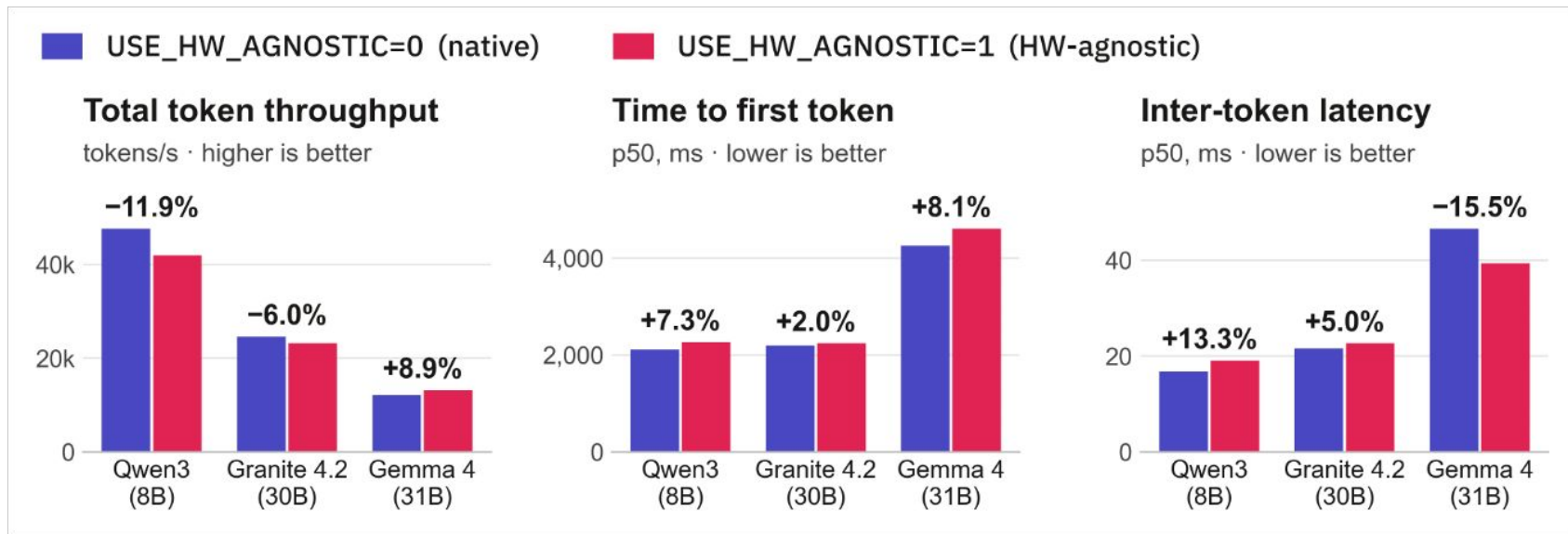
Validated on IBM Spyre: Gemma 4 · Qwen3 · Granite 4.2

Next: HW-agnostic models in Spyre CI, then the default serving path on Spyre

Coming next: HW-agnostic flat models

- A HW-agnostic `model.py` per flat model, built on HW-agnostic layers
- Shared layers live in `model_executor/hw_agnostic`
- Model-specific layers (e.g. `DeepSeekV4FlashMLAAttention`) sit next to `model.py`
 - ...and still follow all four principles
- Example:
 - HW-agnostic DeepSeek V4, PR #45470 (in review)

How does it perform on GPU?



NVIDIA H100, transformers backend: total token throughput within 3.4% of native (geomean, 3 models)

Get involved

- Still work in progress, and feedback is welcome
 - If you are maintaining an OOT vLLM plugin, we'd love to hear from you.
 - RFC: github.com/vllm-project/vllm/issues/44219
 - Slack: #hw-agnostic-models on vLLM Slack
 - Blog: pytorch.org/blog/hardware-agnostic-models-in-vllm
-
- Special thanks to Thomas Ortner (IBM) for implementing much of this work and Harry Mellor (Hugging Face) for his detailed reviews.



Thank you · Questions?