

Model Intervention as a Service:

Read and edit the internals of a frontier model while an engine keeps serving.

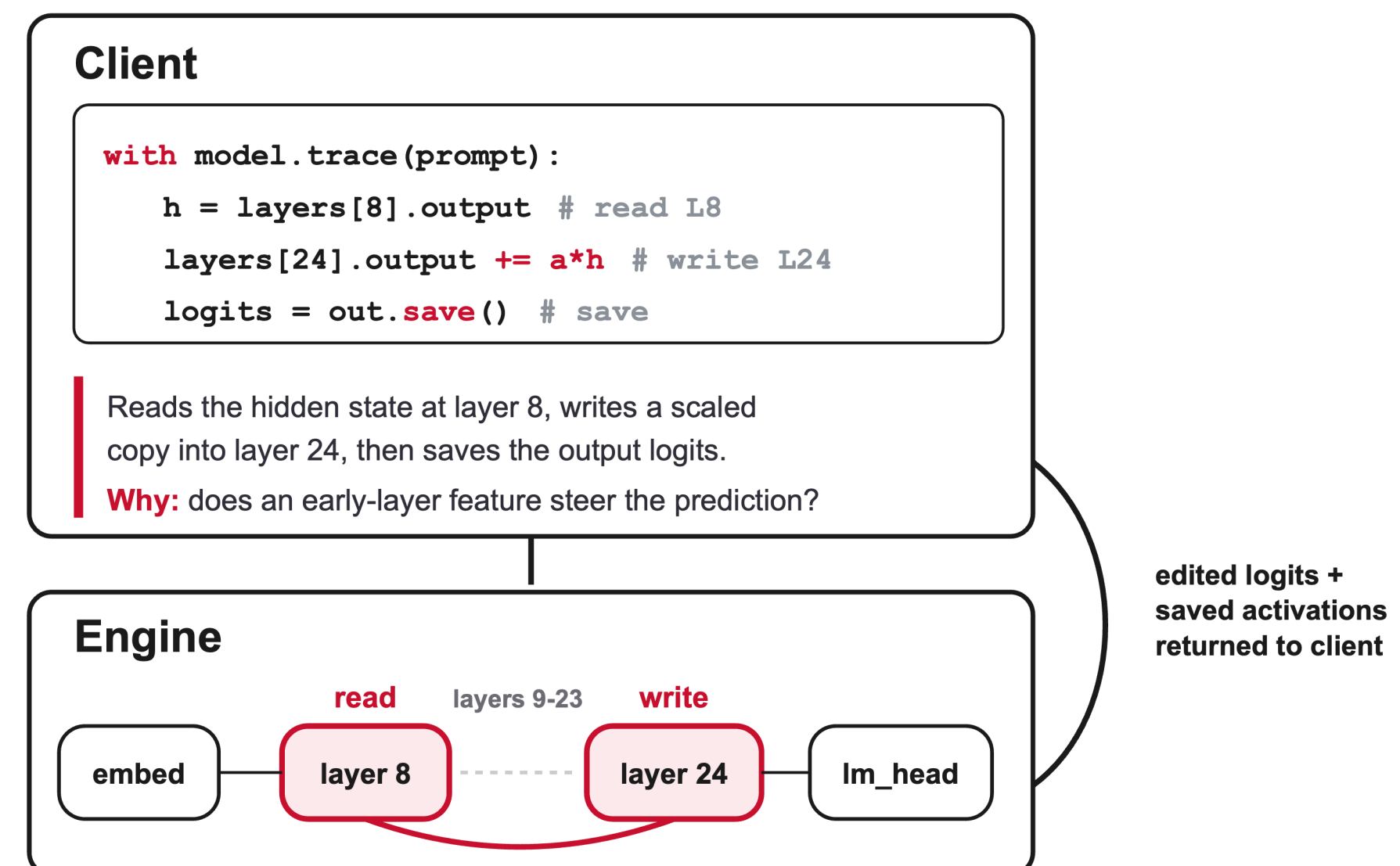
01 The Problem

Interpretability is now production practice: steering behavior, probing activations, feature-level debugging, comparing checkpoints. All need one thing: **programmable read/write access to internal state, mid-inference, at frontier scale.**

No serving stack offers it. Every option falls short:

- **Serving engines** transform execution for throughput: batching, sharding, fused kernels, graph capture.
- **Interpretability tools** assume one process, one module tree, one visible forward call.
- **Fixed intervention APIs** cover static workloads only, not programmable access.

Model Intervention as a Service: an intervention layer over production engines, exposing a stable logical view of internals (activations, logits, generation steps, cross-prompt state) while the engine keeps serving, making them inspectable and programmable.



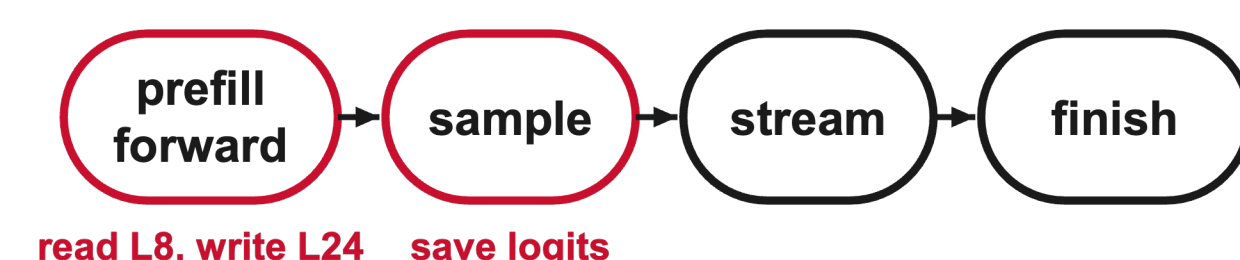
02 The Core Challenges

Take the running example: **read h at layer 8, write a*h at layer 24.** Preserving it while the engine keeps serving breaks in three places:

1 Scheduler-driven lifecycle

vLLM runs your request as scheduler steps, not one pass you can wrap:
(batched with other requests)

the scheduler fires phases in turn:

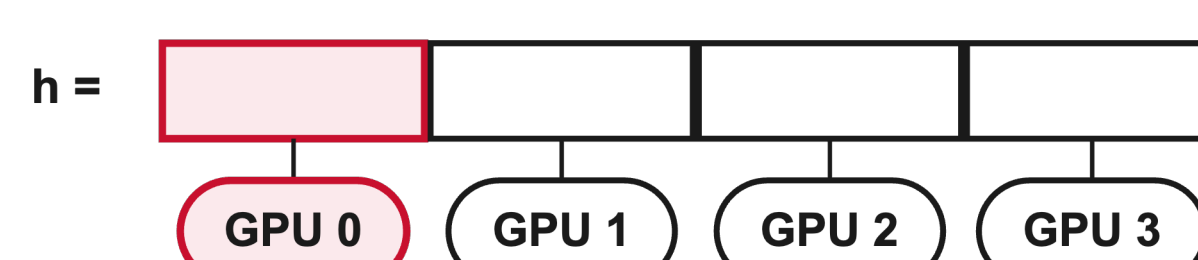


Read and write happen inside these steps, one at a time.

There is no single pass to hook.

2 Partitioned values

Tensor parallel: one h, split across GPUs



Pipeline parallel: layers on workers



No GPU or worker owns the full h.

3 Optimizations hide state

Continuous batching



Your tokens are a concatenated slice, not their own tensor.

Fused kernels / CUDA graphs



Prefix caching

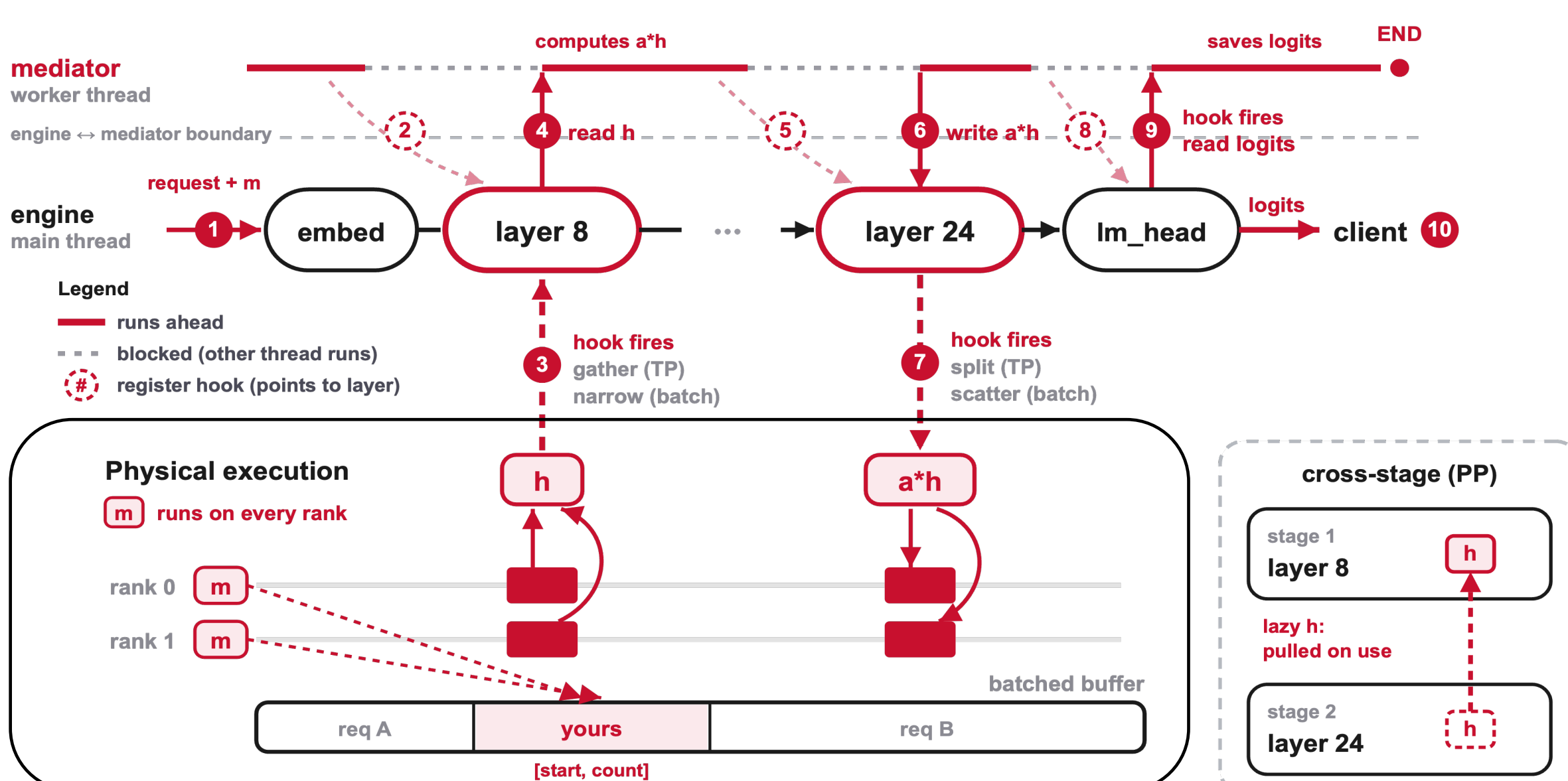


Layer 8 is skipped on the cached prefix, so h isn't produced X

Each optimization changes whether layers[8].output exists to hook.

03 Our Design: NNsight

Mediator lifecycle: hooks + engine scheduling



Intervention programs (mediators) attach to inference requests; their logical reads and writes bind to vLLM's existing execution path.

Engine-driven. Intervention code rides vLLM's normal scheduling and re-enters at phase boundaries: model execution, logits, sampling, result collection.

Distributed. Reconstructs tensor-parallel values for user code and scatters edits back to ranks. Under pipeline parallelism the same mediator runs on every rank: layers missing on a rank resolve to lazy stubs, and the real tensor is pulled from its owning rank only when dereferenced.

Optimized. Preserves continuous batching by translating between request-level logical tokens and the engine's flat physical buffers. Prefix cache and CUDA graph are disabled right now.

