Weave: Execution Strategies for Heterogeneous LLM Inference in the Age of Agents

OpenInfer, Inc. — March 2026

Summary. Autonomous AI agents are replacing interactive chat as the primary driver of inference demand. Agent workloads are structurally different — bimodal latency, growing contexts, sustained volume — and today's serving architectures cannot express these differences. Every major inference engine runs one execution model for all sessions: same algorithm, same hardware, same tradeoffs. The result is GPUs overprovisioned for traffic that doesn't need them, CPUs sitting idle, and pool isolation as the only mechanism for workload diversity.

This is an abstraction problem, not a scheduling problem. What's missing is the ability to route sessions to fundamentally different execution strategies based on what each session actually needs.

We present Weave, a new serving model — implemented in the OpenInfer inference engine — where execution strategy is a first-class concept. Weave classifies sessions at creation time and routes them to qualitatively different strategies: different parallelism schemes, different distribution of computation, different hardware classes. This whitepaper describes the strategy taxonomy, the routing model, and how strategies map to available hardware — including idle server CPUs, heterogeneous GPU fleets, and commodity networking.

1. The Agent Workload Shift

LLM inference serving has evolved through a series of optimizations — continuous batching, chunked prefill, prefill/decode disaggregation — each responding to the same workload: interactive chat where a human waits for every token.

That workload is no longer the only one that matters. Autonomous agents are generating a rapidly growing share of inference traffic. OpenClaw, the open-source agent framework, reached 247,000 GitHub stars within months of its late-2025 launch. Its agents run on configurable heartbeats (30–60 minutes), autonomously executing multi-step workflows while users check in asynchronously via messaging. Gartner projects 40% of enterprise applications will include task-specific AI agents by end of 2026. Major cloud providers report that batch and offline inference already dominates total serving capacity.

Agent workloads differ from chat in three ways that matter for serving infrastructure:

Bimodal latency requirements within a single session. An agent session may run for hours in

background mode — heartbeats, tool calls, autonomous reasoning — where latency is irrelevant and throughput dominates. Then the human checks in via their phone and expects a responsive reply. The same session has two different service-level profiles, separated by time.

Growing, often large, contexts. An agent may start at 2K tokens for a heartbeat check and accumulate 64K+ over a day of multi-step work. A hundred concurrent agents at 32K each generate ~100 GB of aggregate KV cache for a 70B model. That exceeds single-GPU memory.

Sustained, always-on volume. Agent traffic isn't bursty — it's continuous, running overnight, over weekends, and during holidays. Most of it is throughput-tolerant work being served on GPU hardware provisioned for interactive latency.

This is the first time inference workloads require mutually incompatible execution strategies within a single session — low-latency decode when the user is present, high-throughput distributed decode when they're not. No existing serving system can express this.

2. The Problem with Pool Isolation

The current industry approach is to separate interactive and batch traffic into isolated pools, each backed by identical GPU replicas. Together AI offers separate serverless and batch endpoints. Enterprise teams often run two vLLM deployments side by side.

This has obvious limitations:

The interactive pool is oversized. It must be provisioned for peak interactive load, but agent-heavy deployments spend most of their time in background mode. GPUs sit idle during off-peak hours.

The batch pool can't handle urgency. When a user engages with a background agent session, there's no mechanism to elevate priority or shift to a faster serving path. The session is stuck in the batch pool.

Both pools use the same strategy. Identical hardware running identical software. The only difference is routing. This ignores the possibility that different workloads might be better served by fundamentally different approaches.

Idle hardware goes unused. Server CPUs on GPU machines — often high-end Xeons or EPYCs — typically sit at 1–5% utilization. This is capable compute being wasted because the serving system has no strategy that can use it.

These are not configuration problems — they are **architectural limitations**. vLLM, SGLang, and TGI are built around a single execution model: continuous batching on homogeneous GPU replicas. They can tune batch sizes, reorder requests, and manage memory within that model. But they cannot express the idea that one session should run a distributed ring protocol on CPU nodes while another runs speculative decode on a GPU. The abstraction doesn't exist. Pool isolation is the only escape hatch, and it's a blunt instrument.

3. Weave: A New Abstraction Layer for Inference

The history of distributed systems is a history of abstraction. Kubernetes didn't make servers faster — it introduced the abstraction of scheduling workloads across heterogeneous resources based on their requirements. CUDA didn't change the transistors — it gave programmers a way to express parallel computation. Each abstraction made a class of problems tractable that was previously managed by hand.

Inference serving needs a similar abstraction. The mapping from workloads to execution strategies is currently implicit, manual, and static: the operator deploys one engine configuration and all sessions share it. Weave makes this mapping explicit, automatic, and per-session.

Weave is not a scheduling layer on top of existing inference engines — existing engines cannot express strategy heterogeneity. Weave is a new serving model, implemented in the OpenInfer engine, where **execution strategy is a first-class concept**. Each strategy determines *how* the model is served: which parallelism scheme is used, how attention is computed, how KV caches are managed, and on what hardware. Sessions are routed to strategies at creation time based on their SLA profile, context size, and available resources. We call this **strategy-aware inference**.

Weave currently supports four execution strategies for a single model, with more planned:

Execution strategy	Best for	Hardware requirements
Standard prefill	Latency-sensitive prefill	Single node (GPU or multi-GPU)
Pipeline-parallel prefill	Throughput-tolerant batch prefill	Multi-node (any mix of CPU/GPU)
Standard decode	Interactive, all context sizes	Single node (GPU or multi-GPU)
Q-Ring decode	Throughput-tolerant, large/aggregate context	Multi-node ring

Additionally, OpenInfer supports running independent lightweight models on idle CPUs for background agent tasks (heartbeats, monitoring) — a complementary deployment option outside the strategy matrix.

These strategies compose within a dual-axis disaggregation framework. The industry has already accepted that prefill and decode should be disaggregated because they have different compute profiles. Weave extends this to a second axis: sessions with different service-level requirements should run on different strategies. Each session passes through at most two strategies — one for prefill, one for decode — both selected at creation time.

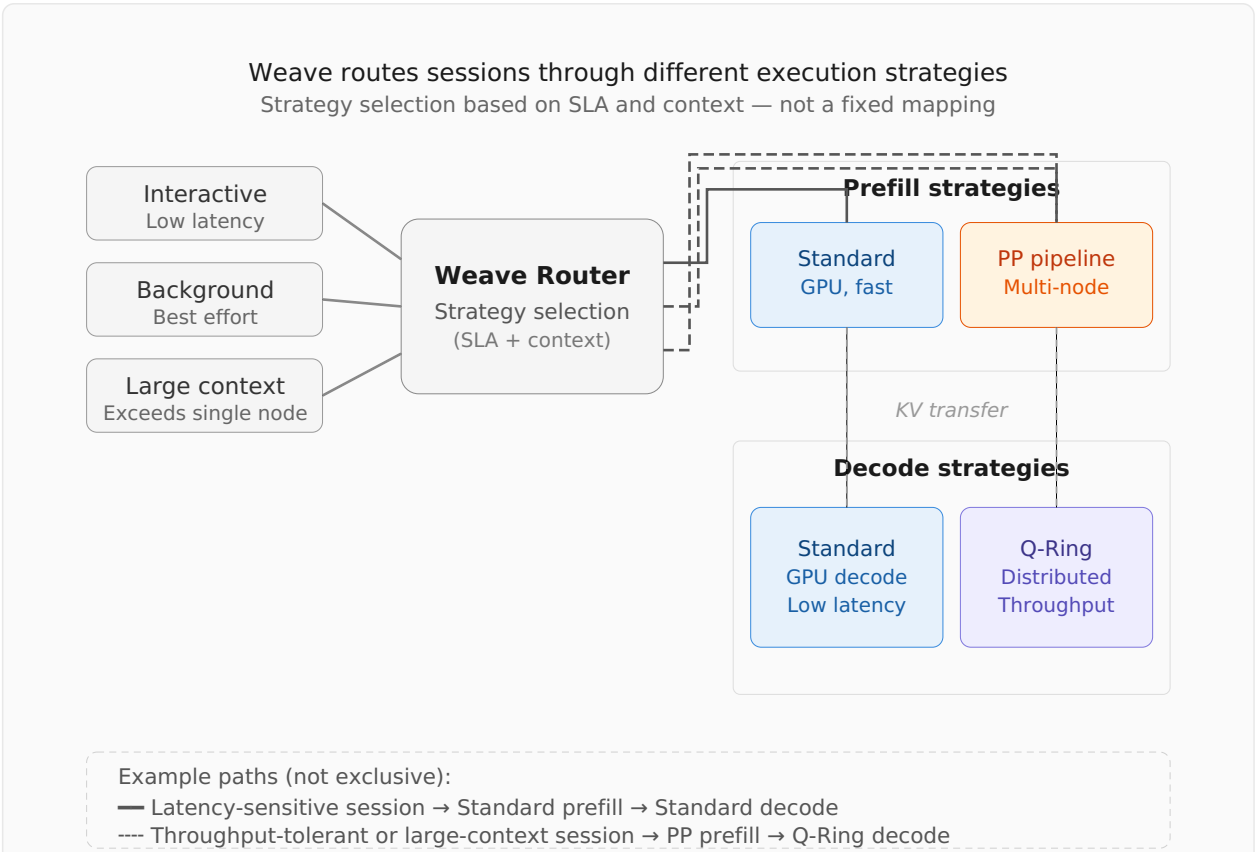

Figure 1. Weave routes sessions to execution strategies based on SLA and context. The mapping is not fixed — the router selects strategies per-session based on requirements and system state.

4. Execution Strategies in Detail

Standard Prefill

The familiar approach: a single GPU (or multi-GPU via tensor parallelism) processes the full input prompt. OpenInfer supports TP across NVLink, PCIe, and mixed-vendor GPU configurations. Even within this strategy, SLA tags are passed to the continuous batcher, which uses them to prioritize latency-sensitive sessions above best-effort work sharing the same hardware.

Pipeline-Parallel Prefill

Model layers are distributed across multiple nodes in a pipeline. A GPU node may hold 30 layers, a CPU node 10, with asymmetric assignment to equalize stage latency. Inter-stage communication is a single hidden-state vector per token per boundary — 16 KB for a 70B model, negligible even at 1 Gbps.

Weave restricts this strategy to prefill and batched workloads, where chunked micro-batch pipelining keeps all stages saturated. This absorbs capacity from diverse hardware — older GPUs, CPU-only

nodes, underutilized servers — that standard serving leaves idle. Each node holds KV only for its owned layers, naturally distributing the memory burden.

Standard Decode

Single-node GPU decode with continuous batching — the workhorse for interactive sessions. This is what vLLM, SGLang, and TGI do today. Within this strategy, OpenInfer adapts behavior to session SLA: at low concurrency, the GPU can run speculative decode natively (draft-verify, EAGLE-style, Medusa-style) to reduce latency further; at higher concurrency, SLA tags drive batch priority so latency-sensitive sessions are served ahead of best-effort work.

Distributed Decode (Q-Ring)

For throughput-tolerant workloads with large aggregate context, Weave can route decode to a distributed ring of nodes — CPU or GPU — that keep KV caches stationary while rotating only the small query vectors during attention. This approach, known as "pass-Q" ring attention, was recently demonstrated for GPU inference by Yang et al. (Meta, MLSys 2025), who showed that rotating queries instead of KV is more efficient during decode when KV >> Q.

We have extended this idea to a different hardware regime: server CPUs on commodity 10 Gbps networks. The key insight is that GQA attention during decode is compute-bound on CPU hardware (arithmetic intensity well above the CPU ridge point), which creates a 5–6× margin between compute and transfer time per head group. This enables head-level pipelining that hides ring communication almost entirely — even on commodity networking. Our implementation also eliminates the All2All step required in Yang et al.'s design by accumulating partial results as they rotate through the ring.

This strategy — which we call Q-Ring — addresses two growing needs: individual sessions whose context exceeds single-node memory, and many concurrent sessions whose aggregate KV exceeds what one node can hold. Both are increasingly common in agent-heavy deployments.

> **A detailed technical paper on Q-Ring** — covering the CPU-optimized algorithm, communication cost analysis, pipelining properties, and throughput scaling — will be published separately.

5. Routing: How Weave Selects Strategies

When a new session arrives, the Weave router selects a prefill strategy and a decode strategy. This decision is made once, at creation time, based on three inputs:

SLA tags (from the API caller): TTFT target, TTLT budget, priority class (interactive, background, batch). These are the primary routing signals. Inference platform operators expose them as API parameters; enterprise deployments configure them per application or agent class.

Context size: prompt length and, optionally, expected maximum context growth. This determines whether the session's KV cache will fit on a single node.

System state: current batch depth and available memory per strategy pool. The router prefers strategies with capacity.

The routing logic is a lightweight decision tree — not a global optimization:

1. **Latency-sensitive** → Standard prefill + Standard decode (with speculative decode enabled at low concurrency).
2. **Throughput-tolerant** → PP prefill (if pipeline can be filled) + Q-Ring decode (if batch depth ≥16 or context exceeds single node).
3. **Context exceeds single-node** → Q-Ring decode regardless of SLA.

Decisions take microseconds. No global state is locked.

Strategy variants and within-strategy optimization. Strategy selection is the coarse decision — it determines the execution path. But within each strategy class, operators can provision multiple pool variants tuned for different operating points. For example, one standard decode pool might have speculative decode enabled for low-latency sessions, while another is configured for maximum concurrency without speculation. Weave's router is aware of these variants and selects based on the session's SLA and the pool's current load. Beyond pool selection, SLA tags flow through to the batch engine within each pool, driving fine-grained behavior: batch priority ordering, chunk scheduling, and preemption thresholds. A latency-sensitive session and a best-effort session on the same hardware get different treatment. Weave isn't "two pools with extra steps" — it's differentiated execution at every level.

Permanent Binding

Once assigned, a session stays in its decode strategy for its lifetime. Cross-strategy migration would require transferring KV caches between fundamentally different storage layouts. This is a deliberate simplification: misclassification is expected to be rare when SLA tags are explicit, and the cost of occasional suboptimal placement is lower than the complexity of a migration mechanism. That said, emerging KV cache compression techniques (e.g., TurboQuant) may make cross-strategy migration practical by reducing the transfer cost, and new architectures like SSM hybrids — with significantly smaller per-layer state — could enable lightweight "interactive boost" promotion and demotion between strategies in future versions.

Bimodal Sessions

When a user engages with a background agent session, the session's priority can be elevated *within* its current strategy pool. The batch engine responds by prioritizing that session's tokens. No migration needed — just a priority boost. If the strategy's baseline speed is insufficient for the interaction (e.g., CPU Q-Ring is too slow for a real-time exchange), the application may reissue on a latency-optimized strategy with a fresh context.

6. Hardware Mapping

Execution strategies have hardware affinities — they work best on certain configurations and require

certain capabilities. Weave maps strategies to available hardware:

Strategy	Hardware affinity	Notes
Standard prefill/decode	Single GPU or multi-GPU (NVLink/PCIe)	Universal default; speculative decode at low concurrency
PP prefill	Multi-node, any mix	Asymmetric: give faster nodes more layers
Q-Ring decode	Multi-node ring (CPU or GPU)	CPU well-suited due to compute-bound attention

A key observation: many enterprises already have the hardware diversity that Weave can exploit. GPU servers with idle CPUs, older GPUs that can't run the latest models alone, CPU-only nodes allocated for other workloads with spare capacity. Weave turns this heterogeneity from a management burden into a performance advantage.

For organizations deploying models exclusively on CPU — viable for models with ≤30B active parameters, including large MoE architectures — Weave's strategy routing still applies. PP prefill across multiple CPU nodes and Q-Ring decode on a CPU ring are both supported.

7. What's Missing and What's Next

The current strategy matrix has gaps we're actively working to fill:

Latency-sensitive large context. When a session needs both low latency *and* a context that doesn't fit on one node, the current answer is multi-GPU TP within a single server (NVLink). For contexts exceeding even multi-GPU capacity, we don't yet have a latency-optimized distributed decode strategy. This is an active research area.

Dynamic strategy adaptation. Currently, strategies are selected at session creation and don't change. A session that starts small and grows large can't migrate to Q-Ring without losing its context. Lightweight KV transfer between strategy layouts is technically feasible but not yet implemented.

Workload-adaptive provisioning. The strategy pools are configured at deployment time. If the agent workload fraction increases, the operator must reconfigure. Automatic rebalancing based on observed traffic patterns is planned.

Learned routing. The current decision tree is hand-tuned. A learned routing policy trained on workload traces could improve classification accuracy, especially for sessions with evolving characteristics.

Further reading:
Inference is no longer a single execution model — it's a routing problem (blog post)
Yang et al. "Context Parallelism for Scalable Million-Token Inference" (MLSys 2025) — foundational pass-Q ring attention for GPU inference
OpenInfer