Running Inference as a Distributed System

Production LLM inference is a distributed systems problem. Everpure supplies the data layer that holds it together.

Enterprises keep treating inference as a serving problem. Teams pick an engine, put a GPU under it, and add replicas when it slows down. At production scale, inference is a distributed systems problem. The teams that run it well operate a platform with a cache-aware gateway, a global key-value (KV) index, disaggregated prefill and decode pools, and a fleet-wide KV cache on shared storage. Everpure™ supplies the data layer of that platform—Pure Key-Value Accelerator (KVA), FlashBlade®, and Portworx®—with an up to 20 times faster time to first token (TTFT) and zero application changes.

The problem: Redundant prefill at fleet scale

The bottleneck in enterprise AI has shifted from data to compute. In 2026, frontier model prices reversed their long decline, with output rates rising two to seven times across major providers. In this environment, GPU supply is constrained, power infrastructure carries 12- to 18-month wait times, and demand doubles every few months. Buying more capacity is no longer a viable strategy. The durable advantage is wasting less of the compute you already own.

The largest source of waste is prefill recomputation. Processing a prompt is an $O(n^2)$ operation in context length: a 100K-token prompt on a 70B parameter model takes 8 to 10 seconds of GPU time before the first output token. When 500 users ask about the same contract, that is 4,000 to 5,000 GPU-seconds spent recomputing identical KV tensors. Agentic workloads are worse: at 5,000 agent calls per day, a typical stack recomputes 55 million tokens of unchanged system prompts, tool definitions, and policy context.

The standard response makes this worse because teams solve inference per node: an engine, a GPU, a local cache, and a round-robin load balancer in front. A per-node KV cache only helps the node that computed it, and a round-robin router sends the follow-up request somewhere else, where the full prefill runs again. Serving one model on one box is a solved problem; vLLM does it well. Serving a fleet is not, because the hard problems live outside the engine: routing, state, and data movement.

The solution: Treat the fleet as one system

A production inference platform is not one product; it is a set of cooperating layers. The cleanest way to see how they work is to follow a request through the system.

Inference-aware routing

A request passes through an inference-aware gateway, such as a load balancer that goes beyond round-robin routing. It classifies the request by service level objective (SLO) class, scores candidate endpoints on prefix-cache hit probability and queue depth, and routes the request to the node where it will complete fastest. To score rather than guess, the gateway consults a **global KV index**. This is a fleet-wide catalog of which prefixes exist, at what tier—high-bandwidth memory (HBM), dynamic random-access memory (DRAM), or flash—and on which nodes. Lookup cost must be trivial relative to the prefill it saves; a metadata query costs microseconds, while the prefill it avoids costs seconds.

The gateway is where cache locality logic lives. An agentic workflow with its KV cache hot in HBM stays pinned to that node, while new work gets redirected to nodes that can hydrate from storage instead. Underneath it all sits the serving layer: a distributed inference framework (for example, llm-d) on an enterprise Kubernetes platform (Red Hat OpenShift), with container-native storage orchestration (Portworx) below that.

Fleet-wide KV cache and tiering

What the index catalogs, the cache layer stores. A **fleet-wide KV cache on shared storage,** like Pure KVA, means any GPU in the fleet can pull any cached prefix. A per-node cache becomes a shared asset instead of a local one. The cache manager promotes and demotes entries across HBM, DRAM, and flash based on access patterns, and a quantization policy multiplies what each tier holds. Using FP8/FP4 weights plus KV cache compression increases the effective capacity and throughput of every tier and shrinks the bytes moved on every transfer, which directly changes the tiering economics.

The transfer layer

The index's answer puts each request on one of three paths. If the prefix is warm in HBM somewhere, the request routes straight to that node and the workflow stays hot. If it exists only in flash, the router picks a node with capacity and the transfer layer pulls the cache in, skipping prefill entirely. If it is genuinely cold, the request goes to a dedicated prefill pool, which computes the KV, streams it to a decode node, and writes it through to the shared flash tier so the whole fleet inherits it.

Two mechanisms make those paths cheap. The first is a **transfer abstraction layer** like the NVIDIA Inference Transfer Library (NIXL). It uses a single API that moves KV blocks and weights over the best available route per transfer—that is, NVIDIA NVLink within a node, Remote Direct Memory Access (RDMA) between nodes, and NVIDIA GPUDirect Storage (GDS) to and from flash—without staging through host memory. The second is **prefill/decode disaggregation**, where there are dedicated compute-heavy nodes for prefill and memory-bandwidth-heavy nodes for decode, each scaled independently because they bottleneck on different resources.

Run it like production software

None of these architecture optimizations survive contact with production without an operational layer. Model lifecycle management keeps a registry, versions every rollout, and streams safetensors over GDS from the same flash platform since cold-start time governs how aggressively you can auto-scale. The autoscaler itself must be GPU-aware, scaling on queue depth, KV utilization, and throughput rather than CPU. It should keep latency-sensitive user traffic and offline batch work in separate queues with independent capacity and drain nodes without throwing away warm caches on scale-down.

Observability closes the loop. TTFT, time per output token (TPOT), throughput, per-tier hit ratios, model floating-point operations per second (FLOPS) and memory bandwidth utilization, and request tracing feed the router's scoring function and the autoscaler. Without this, teams can't tune routing or tiering policy without hit-rate telemetry. On the model side, an evaluation harness gates every model, quantization, or routing policy change before it serves live user traffic. Every cached prefix was computed from someone's actual data, so sharing a cache across tenants requires tenant-scoped namespaces, role-based access control, and encryption in flight and at rest to keep one tenant's content from surfacing in another's requests. And if the platform serves retrieval-augmented generation (RAG) or agents, the retrieval layer (vector database, embedding pipelines, and the corpora behind them) belongs on the same storage platform as the KV and weight tiers.

Stated simply, the gateway decides where, the KV index knows what has already been computed, the transfer layer decides how bytes move, and the storage platform turns local caches into a fleet-wide asset.

Where Everpure fits

Everpure provides the data layer underneath all of this. It is where the state lives when it isn't in HBM, and it makes moving that state cheaper and faster than recomputing it. **Pure KVA** is the fleet-wide caching layer, with persistent KV over Network File System (NFS) and S3, native vLLM and NVIDIA Dynamo integrations, zero application changes, and benchmarks showing up to 20 times faster TTFT with NFS and six times with S3. **FlashBlade//S**™ and **FlashBlade//EXA**™ consolidate the KV tier, weight distribution, and retrieval corpora on one platform. FlashBlade//EXA is validated at 10+ TB/s single-namespace reads while serving tens of thousands of GPUs. **Portworx** provides the container-native storage orchestration under Red Hat OpenShift, plus the portability to run the platform where economics are most favorable and burst to the cloud on demand. Observability comes built in: an OpenTelemetry pipeline reports true hit rates by tier, per-stage latency, and capacity signals—the telemetry the router and autoscaler need.

Business outcomes

Speed and performance

Cache-aware routing plus fleet-wide reuse means cached prefixes skip prefill entirely: a 10-second recomputation becomes a 500ms injection, removing up to 95% of TTFT. The gains grow with context length because the bigger the window, the more recomputation there is to avoid.

Cost efficiency

Repetitive workloads (for example, RAG over the same policy documents, agents with fixed tool definitions) can offload up to 90% of redundant computation, so throughput per GPU rises and the same fleet serves more concurrent requests. Disaggregated pools scale on their actual bottlenecks instead of a single blended metric, and Portworx keeps the cloud-versus-on-premises decision economic rather than architectural.

Reduced complexity and risk

One storage platform holds the KV tier, model weights, and retrieval corpora instead of three systems with three failure modes. Tenant-scoped namespaces and encryption close the cross-tenant side channel that shared caches otherwise open. Pure KVA 1.0 is generally available and fully managed: .deb/.rpm packaging, structured logging, distributed cache expiration, and fleet-wide metrics out of the box, over the NFS and S3 your team already runs.

Critical capabilities

An inference platform built on Everpure provides:

- A fleet-wide KV cache (Pure KVA) with up to 20 times faster TTFT and zero application changes
- One storage platform for KV cache, model weights, and RAG corpora, from FlashBlade//S to FlashBlade//EXA (10+ TB/s)
- Container-native orchestration and hybrid portability with Portworx on Red Hat OpenShift and Kubernetes
- Open ecosystem integrations: vLLM, llm-d, NVIDIA Dynamo, NIXL, and GPUDirect Storage
- The telemetry to run the platform: per-tier hit ratios, TTFT/TPOT, and capacity signals via OpenTelemetry

Learn More About Everpure for AI

Visit Our Website | **800.379.PURE**

©2026 Everpure, the Everpure P Logo, Pure Storage, FlashBlade, FlashBlade//EXA, FlashBlade//S, Portworx, and the marks in the Everpure Trademark List are trademarks or registered trademarks of Everpure, Inc. or its licensed subsidiaries in the U.S. and/or other countries. The Trademark List can be found at everpuredata.com/trademarks. Other names may be trademarks of their respective owners.

ID54260-01-en-09/26