Solving the One and the Many with LSF, Symphony, GPFS, and RHEL AI: A Dynamic Compute Platform for NVIDIA AI Factories

Technical Paper
April 1, 2026

Kevin D. Johnson, MBA, MAcc, MSGTD, MAT

DOI: 10.5281/zenodo.19890422

***Abstract*—The AI factory is one infrastructure serving many workloads, yet the workload management layer remains architecturally fragmented. Training and inference differ in temporal profile, resource consumption, failure semantics, and scaling model, yet current architectures force both into a single platform. This paper proposes a multi-ontology architecture in which IBM Spectrum LSF manages batch training workloads, IBM Spectrum Symphony makes available inference services, and IBM Storage Scale (GPFS) serves as the unified coordination substrate. LSF and Symphony descend from Songnian Zhou's 1987 doctoral research on dynamic load balancing, share the same dynamic compute platform and its Load Information Manager (LIM/ELIM) infrastructure, and both implement formally definable resource ontologies. LSF's batch compute ontology provides topology-aware GPU placement and distributed training coordination proven at Oak Ridge's Summit supercomputer. Symphony's service compute ontology provides 38 per-instance ELIM inference metrics, semantic routing, GPFS-based KV cache sharing across Granite or other model tiers and sub-millisecond metric freshness. GPFS connects the two compute domains through extended attributes and Information Lifecycle Management tiering, providing so much more than its typical designation as a distributed file system. RHEL AI and its vLLM inference runtime provide the model serving layer within the Symphony compute domain, with support for over 70 model architectures across NVIDIA, AMD and Intel accelerators. Five demonstrations on commodity hardware and across multiple IBM Cloud regions validate the architecture, including multi-model vLLM inference, neuromorphic routing at 622 microseconds, cross-model KV cache transfer with 8.2x latency improvement, 47-second model handoff and an Obfuscation-as-a-Service pipeline. Within the context of NVIDIA's acquisitions of Run:ai and SchedMD, the architecture provides a vendor-independent alternative grounded in four decades of production deployment at many of the largest organizations, enterprises, and research facilities in the world.**

I. Introduction

The AI factory is one infrastructure serving many workloads. NVIDIA's reference architecture defines a hardware and software stack optimized for training foundation models and serving inference at scale, with DGX SuperPOD configurations scaling from 4 to 256 GPUs under unified networking and storage [1]. The compute infrastructure is available. The storage infrastructure is certified. The networking fabric is deployed. The workload management layer, however, remains architecturally fragmented because current approaches force many workloads into one platform, and the resulting mismatch introduces inefficiencies the hardware stack was designed to eliminate.

Training and inference are fundamentally different workloads. Training is batch-oriented, long-running, checkpoint-driven and measured in hours to weeks. Training workloads consume entire GPU allocations for sustained periods, require deterministic reproducibility across runs and produce model artifacts whose size ranges from gigabytes to terabytes. Inference is service-oriented, latency-sensitive, request-driven and measured in milliseconds to seconds. Inference workloads serve individual requests against loaded model weights, require sub-second response times under variable load and consume GPU memory proportional to model size rather than to training batch dimensions. The two workloads share GPU hardware but share almost nothing else in their scheduling requirements, temporal profiles, failure semantics or resource consumption patterns.

Current AI factory architectures collapse many workloads into one management domain. NVIDIA's reference architecture positions Slurm for HPC-oriented deployments and Run:ai for Kubernetes-oriented deployments, with each scheduler managing both training and inference workloads within a single orchestration domain [1], [2]. The approach inherits the homogenization problem identified in prior work on heterogeneous orchestration [3]. A batch scheduler optimized for long-running training jobs cannot simultaneously optimize for sub-second inference dispatch without architectural compromise. A Kubernetes orchestrator optimized for container lifecycle management cannot express the fine-grained, per-model resource metrics inference workloads demand without custom operators restoring the distinctions the container abstraction erased.

The present paper resolves the problem by assigning each workload its own ontology within one unified infrastructure. IBM Spectrum LSF manages training workloads, IBM Spectrum Symphony manages inference workloads and IBM Stor-

age Scale (GPFS) provides the data and coordination ontology connecting the two compute domains. The architecture exploits the complementary strengths of three platforms sharing a common operational philosophy. LSF is the established leader in large-scale GPU training orchestration, managing the workloads at Oak Ridge National Laboratory's Summit supercomputer and serving as a certified platform for NVIDIA DGX environments [4], [5]. Symphony is a service-oriented dynamic compute platform whose SOAM framework and ELIM metric reporting are architecturally suited to the latency-sensitive, request-driven characteristics of LLM inference [6], [7]. RHEL AI and its vLLM inference runtime provide the model serving layer within the Symphony compute domain, supporting over 70 model architectures across NVIDIA, AMD and Intel accelerators [18], [20]. Both compute platforms descend from Songnian Zhou's 1987 doctoral research on dynamic load balancing in heterogeneous distributed systems at UC Berkeley [8]. Both operate through formally definable resource ontologies, consuming per-resource metrics through descendants of Zhou's load index. Both are proven at production scale in financial services and national laboratory environments processing billions of tasks daily.

GPFS constitutes the third ontology in the multi-ontology architecture. GPFS is not merely distributed storage but a typed and governed model of the data domain with its own ontological structure. Extended attributes (xattrs) provide typed metadata with atomic cluster-wide visibility, enabling model readiness signaling, provenance tracking and training metadata without external coordination services. Information Lifecycle Management (ILM) policies govern data placement across storage tiers based on access patterns and data classification, automatically promoting hot artifacts to NVMe and demoting cold artifacts to capacity pools. Content-Aware Storage, generally available since March 2025, extends the ontology with semantic vectorization of data as it changes. GPFS ACLs enforce domain-separated access governance between training and inference data paths. Quorum-based consistency guarantees ensure no partial artifact is visible across compute domains. Model checkpoints produced by LSF-managed training jobs are written to GPFS. Symphony-managed inference services read model weights from the same GPFS filesystem. Intermediate artifacts, including training logs, evaluation metrics, KV cache state and inference telemetry, traverse the storage fabric without requiring a message broker, a model store or an intermediate serialization layer. GPFS over RDMA on the NVIDIA-certified storage fabric provides the throughput required for both training checkpoint I/O and inference model loading, with IBM Storage Scale System 6000 delivering up to 40 GBps read throughput per node in DGX SuperPOD configurations [9]. The filesystem is not a passive substrate but an active participant in the architecture, carrying typed metadata, enforcing governance policies and coordinating between compute domains through its own ontological primitives.

The contributions of the paper are fourfold. First, the paper presents a multi-ontology architecture separating training orchestration, inference orchestration and data coordination within the same AI factory, with LSF managing batch training, Symphony managing service-oriented inference and GPFS providing a typed and governed data ontology connecting the two compute domains. Second, the paper demonstrates GPFS as an active ontological participant rather than passive storage, carrying typed metadata, enforcing governance policies and coordinating between compute domains through xattrs, ILM, Content-Aware Storage and quorum-based consistency. Third, the paper provides working demonstrations of the architecture on commodity hardware, including vLLM inference serving under Symphony SOAM with GPFS-based KV cache sharing across Granite or other model tiers and semantic routing for intelligent workload placement across model sizes. Fourth, the paper situates the architecture within the context of NVIDIA's recent acquisitions of Run:ai and SchedMD, arguing that the consolidation of workload management under a single GPU vendor creates both competitive risk and architectural opportunity for multi-ontology compute and data platforms with established production deployments.

II. The Training-Inference Gap

A. Two Workload Paradigms

Training and inference occupy opposite ends of the workload spectrum along every workload management dimension [24], [25]. Understanding the gap between them is prerequisite to understanding why a single platform cannot optimize for both simultaneously.

Temporal profile. A training job runs for hours, days or weeks. The job occupies a fixed GPU allocation for its entire duration. Checkpointing occurs at intervals determined by the training configuration, typically every few thousand steps, producing multi-gigabyte artifacts written to persistent storage. The platform's role is to allocate resources at job submission, manage preemption and fairness across competing training jobs and handle failure recovery through checkpoint restart. A training platform optimized for the workload prioritizes throughput over latency, aggregate GPU utilization over individual request response time and deterministic resource allocation over dynamic rebalancing.

Inference profile. An inference request arrives, consumes model-specific GPU resources for milliseconds to seconds and returns a result. The model weights are pre-loaded into GPU memory. The platform's role is to route the request to an appropriate model instance based on current load, model availability and request characteristics. An inference platform optimized for the workload prioritizes latency over throughput, per-request response time over aggregate utilization and dynamic routing over static allocation. The platform must respond to load changes in real time, scaling model instances up or down based on demand and routing requests to the least-loaded instance serving the appropriate model tier.

Resource consumption. Training consumes GPU compute (FLOPs), GPU memory (for activations, gradients and optimizer state), network bandwidth (for gradient synchronization in distributed training) and storage I/O (for checkpointing and data loading). The resource profile is predictable and sustained.

Inference consumes GPU memory (for model weights and KV cache) and GPU compute (for forward pass execution) in patterns determined by request arrival rate and sequence length. The resource profile is variable and bursty. A training job's resource needs are known at submission; an inference service's resource needs fluctuate with traffic.

Failure semantics. Training failure means a job crashes and must restart from the last checkpoint. The recovery model is temporal, rewinding to a prior state. Inference failure means a request is dropped or times out. The recovery model is spatial, routing subsequent requests to a surviving instance. Training failure tolerance is measured in minutes of lost computation. Inference failure tolerance is measured in milliseconds of added latency. A platform designed for training checkpoint restart and one designed for inference instance failover implement fundamentally different failure recovery strategies.

Scaling model. Training scales by adding GPU nodes to the distributed training cluster, increasing data parallelism or model parallelism. The scaling decision is made at job submission and remains fixed for the job's duration. Inference scales by adding or removing model instances based on demand. The scaling decision is continuous and reactive. Training scaling is horizontal expansion of a single job. Inference scaling is horizontal replication of a stateless (or KV-cache-stateful) service.

B. Why Single-Platform Approaches Compromise

Slurm remains a batch scheduler. Slurm 25.11 has added the Step Manager for scalable step management, slurmrestd for REST-based job submission, TRES for multi-dimensional resource accounting, and OpenMetrics telemetry export [10]. These are meaningful improvements to the batch paradigm. The fundamental unit of work remains a job submitted to a queue, allocated a set of nodes, and executed until completion or failure. Slurm's GRES mechanism and the newer TRES framework extend the node model with counted and tracked accelerator resources, enabling GPU-aware scheduling for training workloads. TRES adds priority weighting and billing across resource categories but remains a static declaration system. The model works well for training because training workloads match the batch paradigm.

Inference workloads do not fit the batch model. Production inference systems run persistent model servers (vLLM, TensorRT-LLM, Triton) as long-running processes that accept requests via HTTP or gRPC. Slurm can manage the lifecycle of the model server process but has no visibility into the requests the server handles, the models the server loads, the GPU memory the KV cache consumes, or the latency each request experiences. Slurm has no native support for inference serving, persistent services, request routing, per-model metrics, semantic routing, or KV cache awareness. Slurm has no mechanism equivalent to ELIM through which a running service instance can dynamically report custom operational metrics back to the resource broker in real time. SchedMD acknowledged this limitation by developing Slinky, an operator that runs Slurm inside Kubernetes so that Kubernetes can handle inference and persistent service workloads while Slurm handles batch training. The Slinky architecture validates the dual-domain thesis of the present paper: even Slurm's own developers recognize that batch and service workloads require different platforms.

Run:ai (v2.24, January 2026) addresses the Kubernetes side of the same problem [2]. Run:ai provides GPU-aware scheduling within Kubernetes, including fractional GPU allocation with runtime memory enforcement, GPU pooling, NIM-native autoscaling, and workload-aware scheduling policies. NVIDIA open-sourced the underlying KAI Scheduler under Apache 2.0 in April 2025, making the scheduling engine available independently of the commercial platform. Run:ai captures the service-oriented nature of inference better than Slurm's batch model but inherits Kubernetes' fundamental limitation: resource requests are expressed as scalar quantities (CPU cores, memory bytes, GPU count) rather than as typed metrics capturing the operational state of the resource. Run:ai does not perform intelligent request routing between multiple models based on per-model operational metrics. Run:ai is a workload placer, not a workload router. For inference routing, NVIDIA points to Dynamo 1.0 (production release March 2026) and llm-d at the framework level, separate systems operating above Run:ai in the stack. The Kubernetes scheduler framework (since v1.19) and the custom metrics API do enable multi-signal scoring plugins, and projects like KEDA provide event-driven autoscaling from external metric sources. These extensions restore some of the capability the container abstraction erased. The distinction is between restoring capabilities through middleware layered atop a flat resource model and expressing them natively within a typed resource ontology where metric semantics, governance hierarchies, and service lifecycles are first-class constructs.

Run:ai's limitations extend beyond the Kubernetes resource model. Run:ai supports only NVIDIA GPUs, excluding AMD, Intel, NPU, FPGA, QPU, and mainframe resources from its compute domain entirely. Run:ai requires Kubernetes infrastructure as a foundation, preventing management of bare-metal HPC clusters or non-containerized workloads. The open-sourced KAI Scheduler carries active defects including GPU over-allocation causing infinite retry loops (GitHub Issue #848, January 2026), gang-scheduled jobs failing to schedule on idle nodes (Issue #1217, March 2026), and no runtime GPU enforcement in the open-source scheduler (Issue #423, August 2025), though the commercial Run:ai platform does provide enforcement through proprietary GPU fractions logic [28]. The Slurm-to-Kubernetes migration path Run:ai requires has proven so painful that CoreWeave built SUNK, Nebius built Soperator, and ClearML runs Slurm inside Kubernetes pods, all bridge projects providing Slurm interfaces on top of Kubernetes to avoid forcing researchers through the migration [29]. These limitations are architectural consequences of building a GPU-specific workload placer on a container orchestration engine. A platform built on Kubernetes inherits Kubernetes' flat resource model, its lack of hierarchical multi-tenancy, and its inability to express arbitrary typed metrics with semantic

direction.

Kubernetes-native projects including Kueue (hierarchical queuing with borrowing and preemption), KEDA (event-driven autoscaling), and Knative (serverless serving) address specific gaps within the Kubernetes ecosystem. Kueue in particular provides hierarchical queue management with fair sharing, approaching one dimension of Symphony's consumer hierarchy. These projects do not address the training-inference gap because they operate within a single Kubernetes cluster's resource model. The multi-ontology architecture separates the compute domains entirely, assigning each workload type to a compute platform whose resource ontology matches the workload's nature.

NVIDIA's December 2025 acquisition of SchedMD, the commercial steward of Slurm, and its earlier acquisition of Run:ai consolidate the two dominant scheduling paradigms under a single GPU vendor [11], [12]. The consolidation creates a vertically integrated stack in which the GPU manufacturer also controls the scheduler. The competitive implications are significant. Alternative scheduling platforms, including IBM LSF, must now compete not merely on technical merit but against a vendor controlling both the hardware the scheduler manages and the scheduler itself. The architectural implications are equally significant. NVIDIA's stated intent to maintain Slurm as open-source and vendor-neutral will be tested as proprietary features emerge in the NVIDIA AI Enterprise stack. The integration strategy between Base Command Manager, Run:ai and Slurm remains unclear and the industry faces the prospect of scheduling innovation being driven by GPU vendor priorities rather than by workload requirements [11].

C. The Multi-Ontology Resolution

The multi-ontology architecture resolves the training-inference gap by assigning each workload to a compute platform designed for its characteristics. LSF manages training. Symphony manages inference. GPFS coordinates between them.

The resolution is not a compromise but a recognition that training and inference are different workload paradigms deserving different resource ontologies. Zhou's 1987 dissertation classified heterogeneous systems into two types: Type A systems with hosts of varying capacities but a uniform software interface, where "a more general form of load index should be used to reflect the varying capacities," and Type B systems with hosts of different architectures, where "load balancing becomes much more difficult" because "different, but functionally compatible programs have to be present on each type of host" [8]. Training and inference exhibit both Type A and Type B characteristics simultaneously. Within a single GPU cluster the two workloads present Type A heterogeneity, requiring different load indices to reflect different capacity dimensions on shared hardware. Across the full compute spectrum the distinction becomes Type B, because training and inference each run on different hardware architectures including GPU, CPU, neuromorphic, and quantum resources, and both LSF and Symphony accommodate all of these resource types within their respective ontologies. The two workloads require different operational models (batch submission versus service dispatch) operating on different resource ontologies (static allocation versus dynamic routing). The principle parallels the unity-in-distinction framework established in prior work on heterogeneous orchestration [3]. Training and inference differ in kind, not merely in degree. A batch-oriented compute platform and a service-oriented compute platform are both legitimate workload paradigms optimized for different workloads. Forcing both workloads into one platform commits the homogenization error, treating workloads that differ in kind as though they differ only in parameter settings within a shared model.

The multi-ontology approach has precedent in the financial services industry, where LSF and Symphony have coexisted for over a decade. Large financial institutions run LSF for batch risk computation (overnight VaR, stress testing, regulatory reporting) and Symphony for real-time pricing, intraday risk and Monte Carlo simulation. The two schedulers share GPFS storage, share network infrastructure and operate under unified administrative policies. The pattern is proven at the scale of the largest financial institutions in the world, managing millions of cores across both compute domains [6], [7]. Applying the same pattern to AI factory workloads, with training taking the role of batch risk computation and inference taking the role of real-time pricing, maps a proven operational model onto a new workload domain.

D. LSF and Symphony as Compute Ontologies

The term "scheduler" has become an inadequate descriptor for what LSF and Symphony represent. Both platforms implement formal resource ontologies, typed and governed models of the compute domain specifying what resources exist, what properties those resources have, what relationships govern them, and what operations apply to them. The ontological character of both platforms is a direct inheritance from Zhou's 1987 research, where the load index was not merely a scheduling input but a typed representation of host state with semantic meaning. Understanding both platforms as compute ontologies rather than as workload managers is prerequisite to understanding why the multi-ontology architecture is not merely an engineering optimization but an architectural necessity.

1) LSF's Batch Compute Ontology: LSF's resource model is itself a formal ontology of the batch compute domain. The resource model remains entirely definable and abstract, accommodating any resource type the administrator declares. LSF's `lsf.shared` Resource section defines typed resources with the same structural logic as Symphony's `ego.shared` block, because both descend from the same Zhou lineage and share the same EGO resource broker.

Resource abstraction. LSF's resource declarations specify name, type (Boolean, Numeric, String), description, and semantic direction. LSF ships built-in ELIM executables for GPU resources (`elim.gpu`, `elim.gpu.ext`, `elim.gpu.topology`) reporting per-GPU utilization, tem-

perature, ECC error counts, compute mode, and driver version. Administrators extend the ontology by declaring custom resources and providing ELIM scripts to populate them. The resource model is not limited to classical compute. At Supercomputing 2025, IBM demonstrated LSF managing quantum resources alongside classical GPU and CPU workloads, with quantum backends registered as typed resources within LSF's ontology and quantum circuit submissions dispatched through the same resource requirement expressions governing classical jobs. The demonstration confirmed that LSF's resource model is abstract enough to accommodate compute paradigms beyond the classical domain.

Queue and fairshare governance. LSF's queue hierarchy provides multi-dimensional governance for batch workloads. Queues carry priority levels, resource limits, preemption policies, and fairshare weights distributing GPU time across competing projects. The fairshare model operates over configurable time windows, preventing any single training project from monopolizing the GPU pool while respecting organizational priority structures. LSF's resource reservation mechanism enables advance scheduling of long-running training jobs against future GPU availability, a governance capability batch training demands.

Topology-aware placement. LSF's understanding of GPU topology (NVLink, NVSwitch, InfiniBand, Spectrum-X) enables placement decisions informed by the physical interconnect structure, not merely by GPU count. The topology awareness is itself an ontological capability: LSF models the relationships between GPUs, the bandwidth characteristics of their interconnects, and the implications for distributed training performance. A `resReq` expression requesting colocated GPUs with NVLink connectivity is a query over the compute ontology selecting resources whose typed interconnect properties satisfy the training workload's communication requirements.

2) Symphony's Service Compute Ontology: Symphony and Kubernetes-based platforms including Run:ai are categorically different systems. Symphony's resource model constitutes a service-oriented compute ontology: a typed, hierarchical, and semantically rich model optimized for the dynamic, request-driven characteristics of inference workloads.

Resource schema as type system. Symphony's `ego.shared` Resource block is a schema definition for the compute domain. Each resource declaration specifies a name, a data type (Boolean, Numeric or String), a reporting interval in seconds and a semantic direction indicating whether higher values represent more capacity or more load. The reference implementation described in the present paper declares over 100 unique resource metrics across seven categories. GPU resources include `gpu_util Numeric 5 Y` (utilization, reported every 5 seconds, higher means more loaded), `gpu_temp Numeric 5 Y` (temperature, higher means closer to throttling) and `gpu_power_usage_pct Numeric 5 Y` (power budget consumption). vLLM inference resources include `vllm_kv_cache_usage_pct Numeric 10 Y` (cache fill, higher means less capacity), `vllm_ttft_ms Numeric 10 Y` (time to first token, higher means slower) and `vllm_queue_depth Numeric 10 Y` (pending requests). Neuromorphic resources include `akida_device_ready Boolean` (availability) and `akida_power_mw Numeric 10 N` (power consumption, lower means more efficient). ELIM scripts populate the schema with real-time values from every host and resource requirement expressions operate over the typed metrics. A `resReq` expression such as `select(akida_device_ready && gpu_util < 80)` is a query over the compute ontology selecting resources whose typed properties satisfy the expression. The platform understands that lower `gpu_util` means more available capacity because the metric was declared with semantic direction at registration time. Kubernetes has no equivalent type system. Kubernetes resource requests are scalar quantities (CPU millicores, memory bytes, integer device counts) without types, semantic direction or configurable reporting intervals.

Consumer hierarchy as governance ontology. Symphony's consumer hierarchy is not a flat multi-tenancy mechanism but a tree-structured governance model expressing organizational intent in resource governance terms. The hierarchy operates across multiple governance dimensions simultaneously.

Prioritization and preemption enable the platform to enforce organizational priorities when demand exceeds supply. Each consumer carries a configurable priority level and preemption policies are per-consumer and per-resource-group. Lending and borrowing ensure idle capacity flows laterally to siblings and vertically through the tree: a consumer owning 5% of a resource group can use 100% if no siblings have demand and yields instantly when demand returns. The rebalancing cycle operates every second (`EGO_DISTRIBUTION_INTERVAL=1`).

Multi-dimensional resource planning enables each consumer to hold independent resource policies per resource group. A single consumer can hold different share percentages, planned quotas and share quotas across GPU, CPU, neuromorphic and cloud resource groups simultaneously. The same physical GPU host participates in three resource groups (`gpu_rg`, `vllm_gpu_rg`, `pqfm_gpu_rg`) with different slot counts, meaning the same hardware serves training, inference and quantum computation under different governance policies. Resource planning is a vector across all resource groups the consumer participates in rather than a single number.

The hierarchy typically mirrors organizational structure and accommodates multiple organizational patterns. A line-of-business design places organizational units (trading desk, risk analytics, settlement) as top-level consumers with applications as children. An application-centric design places applications (pricing engine, risk engine, reporting) as top-level consumers with cost centers as children. A hybrid design nests both. Financial institutions routinely restructure consumer hierarchies to reflect organizational changes, regulatory reporting

requirements or cost allocation models without redeploying services because the governance model is independent of the workload model.

Production Symphony deployments at major financial institutions manage thousands of consumers across deep hierarchy levels, processing billions of tasks daily [6], [7]. The four child consumers under `/PortfolioServices` (`PQFMCompute`, `AkidaClassifier`, `VLLMNarrative`, `ZOSSettlement`) illustrate how four entirely different computational paradigms share resources within one governance subtree. Kubernetes namespaces are flat labels with hard quota ceilings. Kubernetes has no hierarchical share propagation, no cross-namespace borrowing, no proportional entitlement, no per-resource-group policy vectors and no second-level rebalancing.

SOAM lifecycle as operational semantics. SOAM's seven-phase service lifecycle (Register, CreateService, SessionEnter, SessionUpdate, Invoke, SessionLeave, DestroyService) with independent failure policies per phase captures the semantics of computational work rather than the mechanics of process management. Each phase carries policies for five failure conditions (timeout, exit, return, exception-failure, exception-fatal) with orthogonal actions on the service instance (blockHost, restartService, keepAlive) and on the workload (retry, succeed, fail). The two failure domains are independent: the fate of the instance and the fate of the task are governed by separate policy axes.

Session types express distributed continuity guarantees. A recoverable session continues executing after the client disconnects and preserves results for later retrieval. A detachable session allows jobs to complete with extended grace periods. Minimum service guarantees (`minServices=1`) ensure service availability during rolling updates. Proactive resource governance defines four boundary escalation levels (PROACTIVE, SEVERE, CRITICAL, HALT) for memory and virtual address space, preempting tasks before resource exhaustion rather than terminating them after. Multi-tier recursive task execution enables parent services to spawn child tasks across hosts within a unified session context. Multi-SSM failover provides transparent task-level redirection between physical service managers during failures.

These capabilities are production features documented in the 99 XML application profiles shipped with the Symphony 7.3.2 SDK across 64 sample applications in eight languages. The same SOAM contract governs services across C++, Java, Python, R, .NET and MATLAB with language choice as a deployment binding rather than a platform constraint. A Kubernetes pod has one lifecycle (pending, running, succeeded, failed), one failure action (restart or not), no session continuity across client disconnects, no per-phase recovery, no proactive boundary escalation and no multi-tier recursive task context.

Together the three layers constitute a service compute ontology, a typed and governed model of the inference domain that no container placement engine replicates. Kubernetes manages container placement on nodes using scalar resource quantities and static labels. Symphony models the types, relationships, governance, and real-time state of the compute domain using a typed metric space, a hierarchical governance tree, and a seven-phase service lifecycle. The two systems are categorically different in the same way a relational database differs from a file system. Both store data but only the relational database models the types, relationships, and constraints governing the data.

3) The Multi-Ontology Principle: LSF and Symphony implement complementary compute ontologies governing different workloads. LSF's batch compute ontology models GPU topology, training job lifecycle, checkpoint state, and fairshare governance. Symphony's service compute ontology models per-instance operational metrics, consumer governance hierarchies, and multi-phase service lifecycles. The two ontologies share a common foundation (EGO resource broker, ELIM metric protocol, LIM host monitoring) but express that foundation through domain-appropriate abstractions.

The multi-ontology architecture proposed in the present paper assigns training workloads to LSF and inference workloads to Symphony not because one platform is faster but because each platform's resource ontology is designed for the workload it governs. Slurm and Run:ai lack this ontological depth. Slurm's GRES mechanism counts accelerators without modeling their operational state. Run:ai's Kubernetes foundation expresses resources as scalar quantities without types, semantic direction, or governance hierarchies. The multi-ontology approach provides compute-domain intelligence no container placement engine can express.

III. Related Work

A. NVIDIA AI Factory Reference Architecture

NVIDIA's AI factory reference architecture defines a hardware and software stack for enterprise AI infrastructure scaling from 4 to 256 GPUs [1]. The architecture specifies DGX systems with Blackwell or Hopper GPUs, Spectrum-X Ethernet networking with BlueField DPUs and a tiered storage architecture with certified storage vendors including IBM Storage Scale [9]. The software stack includes NVIDIA AI Enterprise, CUDA-X libraries, Run:ai for Kubernetes-based orchestration and Base Command Manager for cluster management [1], [2].

The reference architecture positions Run:ai as the primary workload orchestrator for enterprise deployments and acknowledges Slurm for HPC-oriented environments. The architecture does not distinguish between training and inference at the scheduling layer, treating both as workloads managed by a single orchestration platform. The present paper argues the distinction is architecturally significant and that separating the two workload types into complementary compute domains produces architectural advantages in inference latency, training throughput and GPU utilization.

IBM Storage Scale (GPFS) is one of the NVIDIA-certified storage solutions for DGX SuperPOD, with the IBM Storage Scale System 6000 delivering throughput at the "Best" tier of 40 GBps read and 20 GBps write per node [9]. GPUDirect Storage (GDS) enables sustained I/O exceeding 40 GBps directly into GPU memory, eliminating CPU-mediated data

movement for training data loading and model checkpoint I/O. The storage certification establishes GPFS as a validated component of the NVIDIA AI factory ecosystem and the present paper extends its role from storage substrate to coordination substrate.

B. LSF in Large-Scale GPU Training and NVIDIA's Continued Endorsement

IBM Spectrum LSF has a demonstrated history in large-scale GPU training environments. Oak Ridge National Laboratory's Summit supercomputer, operational from 2018 to 2024, used LSF as its workload management platform managing 4,608 nodes, each containing two IBM POWER9 CPUs and six NVIDIA Tesla V100 GPUs [4]. The LSF development team worked with OLCF to customize the platform for Summit's heterogeneous architecture and IBM developed the `jsrun` job launcher specifically for the CORAL systems (Summit at ORNL, Sierra at LLNL) [4]. Summit's storage was GPFS, providing 250 PB across a center-wide parallel filesystem.

LSF's capabilities for GPU training include GPU-aware job placement with topology awareness, multi-node distributed training support with NCCL integration, checkpoint-restart for long-running training jobs, fairshare allocation across multiple training projects and energy-aware workload management for power-constrained environments. LSF is certified for NVIDIA DGX environments and integrates with NVIDIA's GPU management tools including DCGM, MIG dynamic reconfiguration, MPS and automatic GPU mode switching [5].

NVIDIA's continued endorsement of LSF after acquiring SchedMD is architecturally significant. NVIDIA hosts a dedicated developer page for LSF at developer.nvidia.com, describing LSF as "a complete workload management solution for demanding HPC environments" with the ability to "detect, monitor and schedule GPU enabled workloads" and "fully leverage NVIDIA MIG, DCGM and MPS, with the ability to dynamically reconfigure MIG on A100 to match the workload demands" [5]. NVIDIA's DGX Best Practices documentation lists LSF as one of four documented resource management platforms for DGX systems [5]. NVIDIA provided IBM with A100 and DGX A100 hardware for LSF integration development. All of these pages and partnerships remain active as of March 2026, three months after the SchedMD acquisition. NVIDIA owns Slurm and still certifies, documents and actively supports LSF for DGX GPU training. The continued support constitutes an implicit endorsement that LSF's GPU workload management capabilities serve use cases Slurm alone does not address.

The transition from LSF on Summit to Slurm on Frontier (ORNL's exascale successor) reflects HPE's role as Frontier's system vendor rather than a technical limitation of LSF [4]. The architectural capabilities LSF demonstrated on Summit, including heterogeneous GPU-CPU workload management, large-scale checkpoint management and topology-aware job placement, remain relevant to AI factory training workloads.

C. Symphony for Service-Oriented Workloads

IBM Spectrum Symphony's SOAM framework provides service-oriented workload management with characteristics aligning to inference requirements [6], [7]. Prior work demonstrated Symphony orchestrating LLM inference across multiple model tiers, including vLLM serving of Granite and other models at 2B, 8B and 34B parameter scales under unified workload management [3]. The KV cache sharing demonstration replaced Redis-based coordination with a three-tier GPFS persistent cache enabling cross-model KV transfer [3]. The semantic routing demonstration achieved significant cost reduction by routing queries to appropriate model tiers based on query complexity [3].

Symphony's ELIM metric reporting enables per-model-instance routing decisions based on operational state rather than static resource counts. ELIM reports `gpu_util`, `gpu_mem`, `kv_cache_util`, `model_tier`, `inference_latency_p95` and `active_requests` per inference instance, providing the compute platform with the information needed to make routing decisions no batch scheduler can express [6], [7]. Consumer hierarchies enable multi-tenant inference serving with per-tenant fairness guarantees, rate limiting and priority-based workload governance.

D. The Zhou Lineage and Shared Dynamic Compute Architecture

Both LSF and Symphony descend from Songnian Zhou's 1987 doctoral research at UC Berkeley on dynamic load balancing in heterogeneous distributed systems [8]. Zhou's load index, a per-host metric capturing actual resource state and feeding it into workload placement algorithms, is the architectural ancestor of both LSF's Load Information Manager (LIM) and Symphony's ELIM [8], [13]. Zhou founded Platform Computing in 1992, where the load index concept evolved into Platform LSF and later Platform Symphony [14], [15]. IBM acquired Platform Computing in 2012 and both products became part of the IBM Spectrum Computing portfolio [16], [17].

The shared lineage is deeper than historical origin. LSF and Symphony share the same runtime infrastructure through three common components.

Dynamic Compute Platform (DCP). DCP is the shared resource management layer powering both products. DCP's Virtual Execution Machine Kernel Daemon (VEMKD) serves as the central resource broker, decoupling workload placement from resource management. EGOSC manages service instances. Process Execution Manager (PEM) starts, controls and monitors processes. Both products use the same daemons, the same default port range (7869–7873) and the same resource model. LSF and Symphony can run in the same cluster sharing a single DCP instance, with LSF managing batch workloads and Symphony managing service workloads on shared infrastructure [6], [7]. Most production deployments run either LSF or Symphony due to separate workload management needs, but the shared dynamic compute platform

confirms that coexistence is not an integration between two separate products but a dual-mode operation of one resource management framework.

External Load Information Manager (ELIM). Both LSF and Symphony implement ELIM with identical architecture and protocol. ELIM is a site-definable executable (shell script or compiled program) that collects custom dynamic load indices and writes them to stdout in a defined binary format. The Master ELIM (MELIM) on each host manages multiple ELIM executables, validates syntax, merges reports and forwards consolidated data to LIM. LSF ships built-in ELIM executables for GPU resources (`elim.gpu`, `elim.gpu.ext`, `elim.gpu.topology`) reporting per-GPU utilization, temperature, ECC error counts, compute mode and driver version through the same ELIM protocol Symphony uses for `vllm_kv_cache_usage_pct` and `npu_inference_latency`. The protocol is identical because the implementation is the same codebase.

Load Information Manager (LIM). LIM runs on every host in both products, collecting host load and configuration information. The master LIM aggregates metrics from all hosts. LIM's 12 built-in load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `mem`, `swp`, `tmp`, `io`, `ls`, `it`, `status`) trace directly to Zhou's 1987 research on queue-length and utilization metrics for dynamic load balancing [8].

The shared DCP/ELIM/LIM architecture is architecturally significant for the multi-ontology proposal. LSF and Symphony are not competing platforms from different traditions forced into coexistence. The two are complementary expressions of the same resource management framework applied to different workload paradigms, sharing the same resource broker, the same metric collection infrastructure (ELIM/LIM) and the same resource model. LSF applies Zhou's principle to batch workloads: each node reports its state through ELIM and LSF places jobs based on actual GPU utilization, temperature and memory pressure rather than static GRES counts. Symphony applies the same principle to service workloads: each service instance reports its state through ELIM and Symphony routes requests based on KV cache fill, model tier and inference latency. The multi-ontology architecture leverages this complementarity because the two platforms already share the infrastructure enabling it.

E. RHEL AI and vLLM

Red Hat Enterprise Linux AI (RHEL AI) provides a bootable container image bundling RHEL 9.4 with vLLM [20], InstructLab and Granite or other compatible models for bare-metal LLM deployment [18]. RHEL AI represents the convergence of Linux distribution, inference runtime and model serving into a single deployable artifact. The `ilab model serve` command starts a vLLM server; the `ilab model train` command initiates InstructLab fine-tuning. Red Hat AI Enterprise, announced February 2026, extends RHEL AI with the Red Hat AI Inference Server (vLLM-based) and the `llm-d` distributed inference framework [18].

RHEL AI's architecture as a bootable image is significant for the multi-ontology proposal. The image ships as a bootc container installable on bare metal or as a virtual machine, carrying the operating system kernel, GPU drivers, inference runtime and model artifacts in a single versioned unit. The hardware certification matrix includes NVIDIA A100, H100, H200, L40S and GH200/GB200 NVL configurations, AMD Instinct MI300X (since RHEL AI 1.3) and Intel Gaudi accelerators. The multi-ontology architecture positions Symphony as the orchestration layer governing RHEL AI inference nodes, providing the workload management capabilities RHEL AI does not include on its own: multi-model routing, per-instance ELIM metrics, consumer hierarchies and GPFS-based KV cache coordination. RHEL AI's multi-vendor hardware certification combined with Symphony's hardware-agnostic ELIM protocol, LSF's vendor-neutral GPU workload management and GPFS as coordination substrate means the multi-ontology architecture extends naturally beyond NVIDIA to AMD and Intel accelerated infrastructure.

vLLM is the inference engine at the center of both RHEL AI and the multi-ontology architecture. Originally released in June 2023 from UC Berkeley's Sky Computing Lab with the PagedAttention memory management algorithm [20], vLLM has grown into one of the most actively developed open-source inference engines, with over 900 contributors, approximately 50,000 GitHub stars and monthly release cadence as of early 2026. vLLM supports over 70 model architectures including LLaMA, Mistral, Qwen, DeepSeek, Granite and multimodal variants. Red Hat is a significant upstream contributor to vLLM, maintaining hardware enablement for AMD ROCm and Intel Gaudi backends and shipping the enterprise-hardened Red Hat AI Inference Server as a supported vLLM distribution. The breadth of model support and the velocity of development ensure the multi-ontology architecture is not constrained to a narrow set of models or a stagnant runtime.

Frameworks like `llm-d` implement distributed inference routing algorithms including prefix-aware routing, load-based balancing and model-tier escalation [23]. Prior work demonstrated that 16 of llm-d's routing algorithms and 15 additional capabilities can be reimplemented natively on Symphony SOAM with GPFS-based state sharing and does not require Kubernetes [3]. The present paper extends the integration by positioning RHEL AI and its vLLM runtime as the inference layer within the Symphony compute domain, with LSF managing training workloads on the same GPU infrastructure.

F. Unified Frameworks

Ray (Anyscale) provides both distributed training (Ray Train) and inference serving (Ray Serve) within a single framework sharing one resource manager and a distributed object store. Ray represents the strongest attempt to unify training and inference under a single platform and merits consideration as a limited alternative to the multi-ontology approach. Ray's resource model tracks CPU, GPU, memory and custom resources as scalar quantities without typed metrics, semantic direction, consumer governance hierarchies or multi-

phase service lifecycles. Ray Serve's routing operates at the application level rather than at the resource management level. Ray's production deployments, while meaningful in enterprise ML pipelines, have not demonstrated the institutional scale at which LSF and Symphony operate: LSF managed 27,648 GPUs across 4,608 nodes at Summit [4] and Symphony processes billions of tasks daily at major financial institutions [6], [7]. Ray is also absent from NVIDIA's AI Factory reference architecture, which specifies Slurm and Run:ai as the workload management layer [1]. The multi-ontology architecture targets the NVIDIA ecosystem specifically and provides the typed resource ontologies, governance hierarchies and GPFS coordination that a unified framework with a flat resource model cannot express.

G. Heterogeneous Orchestration

The quantum-centric heterogeneous orchestration paper demonstrated Symphony scheduling across QPU, NPU, GPU, CPU and mainframe tiers under a single compute domain [3]. The present paper applies the same architectural principles to a narrower but commercially larger domain: GPU-based AI factory infrastructure. The unity-in-distinction principle governing heterogeneous orchestration across compute paradigms applies equally to the training-inference distinction within a single compute paradigm. Training and inference are different workloads deserving different workload management efforts within a unified infrastructure.

IV. Architecture

A. Overview

The multi-ontology architecture comprises three layers: the training layer managed by LSF, the inference layer managed by Symphony and the coordination layer provided by GPFS. The three layers share physical GPU infrastructure, with LSF and Symphony managing different partitions of the GPU pool or time-sharing the same GPUs through coordinated resource policies.

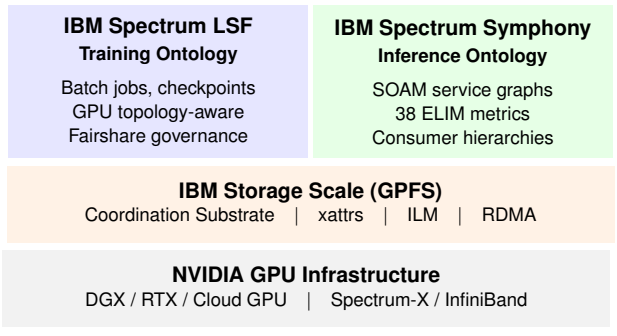

Fig. 1. Multi-ontology architecture for AI factory workloads. LSF manages batch training; Symphony manages service-oriented inference; GPFS provides the data and coordination ontology.

B. LSF Training Domain

LSF manages training workloads as batch jobs submitted to queues with GPU-aware resource requirements. The training domain encompasses:

Job submission and placement. Training jobs specify GPU count, GPU type, memory requirements, expected duration and data parallelism configuration. LSF's topology-aware placement ensures distributed training jobs are allocated GPUs with optimal interconnect locality, minimizing gradient synchronization latency. Fairshare governance balances GPU allocation across multiple training projects, ensuring no single project monopolizes the GPU pool while respecting priority policies.

Checkpoint management. Training jobs write checkpoints to GPFS at configurable intervals. LSF monitors checkpoint completion and uses checkpoint state for job restart after failure. The checkpoint path is a GPFS directory shared between the LSF and Symphony domains, enabling Symphony inference services to load model weights from the same filesystem where LSF training jobs write checkpoints. The handoff from training to inference occurs through the filesystem rather than through an explicit model registry service.

Distributed training coordination. Multi-node training jobs using NCCL for gradient synchronization require co-located GPU allocation and coordinated process launch. LSF's `jsrun` and `blaunch` launchers provide the process placement control distributed training demands. GPU topology awareness ensures training processes are placed on GPUs connected by NVLink or NVSwitch within a node and by InfiniBand or Spectrum-X across nodes.

Resource partitioning. LSF manages a defined partition of the GPU pool reserved for training workloads. The partition boundary can be static (fixed GPU allocation) or dynamic (adjustable based on training demand). Dynamic partitioning uses LSF's resource reservation and preemption mechanisms to reclaim GPUs from idle training allocations and release them to the inference domain through GPFS-mediated signaling.

C. Symphony Inference Domain

Symphony manages inference workloads as service invocations within SOAM service graphs. The inference domain encompasses:

Service-oriented dispatch. Each inference request is a service invocation dispatched to a running model server instance. The dispatcher evaluates ELIM metrics from all available instances to select the optimal target. ELIM is Symphony's native binary protocol for metric reporting, operating at sub-millisecond latency per metric query. The distinction from Prometheus-based metric collection is architecturally significant: Prometheus scrapes at configurable intervals defaulting to 15 seconds, introducing metric staleness measured in seconds between scrape cycles, and each scrape incurs 50–100 ms of collection overhead. ELIM provides continuous push-based metric freshness at configurable sub-second intervals with negligible CPU overhead, below the reporting threshold of

LIM's own host metrics, enabling routing decisions informed by real-time resource state rather than by periodically scraped snapshots. The dispatch decision occurs in approximately 10 milliseconds total (5 ms for Symphony REST API query against cached ELIM state, 5 ms for scorer computation and consumer selection), two orders of magnitude faster than Slurm job submission overhead.

ELIM inference metrics. ELIM scripts running on each inference host report model-specific operational metrics at configurable intervals. The number and type of metrics recorded are entirely configurable; the 38 metrics presented here are a sample of tested operational metrics aligned with vLLM and GPU platform measurement availability. The reference implementation reports 20 vLLM metrics and 18 GPU metrics per inference instance:

TABLE I
ELIM INFERENCE METRICS BY CATEGORY

Category	Metrics	Scheduling Use
Performance	`vllm_ttft_ms`, `vllm_tpot_ms`, `vllm_e2e_latency_ms`, `vllm_tps`	SLA compliance, latency routing
Cache & Memory	`vllm_kv_cache_usage_pct`, `vllm_prefix_cache_hit_rate`, `vllm_num_preemptions`	Cache-aware routing
Queue	`vllm_queue_depth`, `vllm_model_loaded`, `vllm_model_name`	Load balancing
GPU	`vllm_gpu_util`, `vllm_gpu_mem_used`, `vllm_gpu_mem_free`	Saturation detection

Run:ai and Kubernetes-based inference schedulers cannot express these metrics natively. Run:ai schedules GPU containers based on GPU count, memory and fractional GPU allocation. The 38 ELIM metrics provide workload management visibility into the inference workload itself, not merely into the container running it. The difference between knowing a container has one GPU allocated and knowing the GPU is at 87% utilization with 92% KV cache fill and a P95 latency of 340 ms is the difference between container scheduling and more capable workload management.

Routing algorithms. The distributed inference deployment reimplements the core routing algorithms from Red Hat's `llm-d` distributed inference framework natively on Symphony ELIM, with weighted scoring across four implemented algorithms:

1) **LoadAwareScorer** (weight 0.40): Scores consumers by queue depth from ELIM (`vllm_queue_depth`), with GPU utilization penalty above 90%. Formula: score $= 1.0 - (q_d/q_{max})$.
2) **SessionAffinityScorer** (weight 0.25): Header-based sticky routing with LRU cache for session tracking, maintaining conversation continuity across requests.
3) **NoHitLRUScorer** (weight 0.20): Distributes cache-miss requests evenly across consumers. Never-used consumers score 1.0; previously-used consumers rank by LRU recency.
4) **ActiveRequestScorer** (weight 0.15): Scores by in-flight request count from ELIM (`vllm_active_sess`), avoiding hot-spot accumulation.

The weighted scores combine into a final consumer selection in under 5 milliseconds. Run:ai provides GPU-aware pod scheduling; Symphony provides inference-aware request routing. The distinction is between placing a container on a node and placing a request within a running inference service based on 38 real-time operational metrics.

Semantic routing. The semantic routing system classifies query complexity using a KNN classifier trained on 415 samples with MiniLM embeddings (384-dimensional vectors), achieving 91.1% accuracy across code, math and general domains. The classifier assigns each query to one of three model tiers: a lightweight model such as Granite 2B (simple), a mid-range model such as Granite 8B (moderate) or a frontier model such as Granite 34B or Nemotron 120B (complex). The routing decision is a workload placement decision expressed through Symphony's native arithmetic resource requirement expressions:

```
select(llm_score_code >= 70) order(llm_score_code)
```

Symphony's `resReq` expressions enable capability-based routing at the platform level rather than in application code. A query requiring code generation routes to a consumer reporting `llm_score_code >= 70` through ELIM. The routing intelligence resides in the workload management framework, not in an application-level service mesh or API gateway. The semantic router achieves 185 queries per second standalone throughput with P50 routing latency of 74 milliseconds and P99 of 308 milliseconds. End-to-end throughput including inference is 6.6 queries per second. The cost savings from routing simple queries to smaller models rather than dispatching all queries to the largest model are substantial, consistent with the LLM cascade routing literature reporting significant reductions through query-complexity-aware dispatch [21], [22].

KV cache coordination via GPFS. The vLLM + GPFS KV cache sharing architecture replaces Redis-based coordination with a multi-tier GPFS persistent cache [3]. IBM Storage Scale already provides native KV cache offloading for llm-d and vLLM, delivering 8–12x speedup in time-to-first-token relative to recomputation and enabling cross-fleet cache sharing across hundreds of GPU servers [30]. IBM's Content-Aware Storage capabilities, generally available since March 2025, extend Storage Scale with semantic vectorization of data as it changes. The NVIDIA Inference Context Memory Storage Platform, announced January 2026 and powered by BlueField-4 on the Rubin platform, integrates Storage Scale's global namespace and locality-aware placement with Dynamo's KV block manager and NIXL transfer library for low-latency cache access [31]. The architecture progresses through four phases of increasing capability within the present deployment.

- **Phase 1 (operational):** Queue-based load awareness across four GPU consumers, each running a separate

ELIM instance with per-GPU metric prefixes. Routing decisions use ELIM queue depth and active session count. Cross-model KV cache transfer via GPFS warm tier is operational at this phase.

- **Phase 2 (operational):** Prefix cache integration with vLLM KV-Events (ZMQ), tracking `vllm_prefix_cache_hit_rate` through ELIM. IBM Storage Scale's native llm-d connector provides the tiered prefix cache offloading path.
- **Phase 3 (integration):** Three or more tiers of GPFS storage with ILM policies (NVMe or Flash hot tier, SAS warm tier, HDD cold tier) and GPUDirect Storage integration. The constituent capabilities exist in Storage Scale and are being integrated into the multi-ontology deployment. Target: 80% storage cost reduction versus all-NVMe or Flash.
- **Phase 4 (planned):** Prefill/decode disaggregation with separate consumer pools and NIXL transfer manager over RDMA, aligned with the NVIDIA Inference Context Memory Storage Platform roadmap. Target: TTFT under 300 milliseconds (2.5x improvement from 800 ms baseline).

Cross-model KV transfer enables escalation from a smaller model to a larger model without recomputing the entire context. A conversation beginning on a lightweight model (Granite 2B in the demonstration) that requires escalation to a mid-range model (Granite 8B) transfers the existing KV cache state through GPFS and the larger model extends the cache rather than recomputing from the beginning. The mechanism enables model-tier escalation with sub-second latency rather than the full context recomputation required without cache sharing. The GPFS approach eliminates the Redis dependency present in standard llm-d configurations, providing persistent cache storage surviving instance restarts and enabling cross-host cache transfer through the same RDMA fabric carrying model weights and training checkpoints.

Consumer hierarchies for multi-tenant inference. Symphony's consumer hierarchy provides per-tenant resource management for inference workloads. Each tenant receives a consumer allocation specifying GPU quotas, model tier access, priority levels and rate limits. The consumer hierarchy enforces fairness across tenants while allowing burst capacity when other tenants are underutilizing their allocations. The mechanism is identical to the multi-tenant workload governance Symphony provides in financial services production environments [6], [7]. Run:ai provides namespace-level GPU quotas within Kubernetes; Symphony provides consumer-level workload management with priority, preemption, fairshare and per-consumer ELIM metric visibility across all resource types simultaneously.

D. GPFS as Coordination Substrate

GPFS connects the training and inference domains through a unified storage fabric carrying all artifacts produced and consumed by both workloads. The coordination mechanism operates through three concrete pathways.

Model artifact pipeline. The training-to-inference handoff uses GPFS extended attributes (xattrs) as the signaling mechanism between compute domains:

1) LSF training job writes model checkpoint to GPFS (`/gpfs/models/<project>/<version>/`)
2) LSF sets xattrs on the checkpoint directory recording training metadata: `user.model.training_step`, `user.model.eval_loss`, `user.model.architecture`, `user.model.quantization` and `user.model.ready=true`
3) Symphony's model management SOAM service polls the checkpoint directory at configurable intervals (default 30 seconds), reading xattrs to detect new checkpoints where `user.model.ready=true`
4) On detection, Symphony initiates a rolling update: one inference instance at a time loads the new weights from GPFS while remaining instances continue serving with the prior version
5) Each updated instance's ELIM script reports the new `model_version`, enabling the platform to distinguish instances running old and new models during the transition

The xattr-based signaling provides consistency without a coordination service. GPFS guarantees that xattr writes are atomic and visible to all nodes in the cluster. A checkpoint is not visible to the inference domain until the training job sets `user.model.ready=true` after all weight files are flushed. The mechanism eliminates the need for a separate model store, a message broker or a deployment pipeline. The filesystem is the store. The xattrs are the metadata. The polling interval is the notification latency. Governance functions such as model versioning, approval workflows and A/B experiment tracking may still require purpose-built tooling above the storage layer.

Failure behavior. If GPFS becomes temporarily unavailable, both LSF and Symphony continue operating on their respective workloads using resources already allocated. Training jobs continue writing to local buffers and flush to GPFS on reconnection. Inference services continue serving requests with the currently loaded model weights (already in GPU memory). The coordination path pauses, meaning no new model handoffs occur, but neither compute domain fails. GPFS quorum-based consistency ensures no partial checkpoint is visible to the inference domain: either the complete checkpoint with `user.model.ready=true` is visible or nothing is.

Training data pipeline. Training data resides on GPFS and is accessed by LSF training jobs through standard POSIX I/O or GPUDirect Storage. GPFS ILM tiering automatically promotes frequently accessed training data to high-performance NVMe pools and demotes infrequently accessed data to capacity pools. The same ILM policies apply to training data, model checkpoints, inference logs and KV cache state, managed by one storage platform rather than by separate storage services for each workload type.

Telemetry and observability. Both LSF training metrics and Symphony inference metrics are written to GPFS as structured logs. A unified observability layer reads from GPFS to provide cross-domain visibility into training throughput, inference latency, GPU utilization, model deployment status and KV cache efficiency. The observability layer does not require a separate metrics infrastructure because GPFS carries the telemetry alongside the artifacts.

E. HostFactory and Cloud Bursting

Symphony's HostFactory mechanism extends the inference domain to cloud GPU resources transparently [6]. When on-premise inference capacity is insufficient to meet demand, HostFactory provisions cloud GPU instances (IBM Cloud, AWS or other providers), deploys the vLLM model server, registers the instance with Symphony and begins routing inference requests to the cloud instance. The burst decision is driven by ELIM metrics: when on-premise instances report high utilization and elevated latency, HostFactory provisions additional capacity. When demand subsides, HostFactory de-provisions cloud instances.

The mechanism creates a hybrid inference architecture in which on-premise GPU infrastructure handles baseline inference load and cloud resources handle peak demand. LSF training workloads remain on-premise, where sustained GPU utilization and high-bandwidth gradient synchronization favor co-located hardware. Inference workloads burst to the cloud because individual inference requests are independent and tolerate the additional network latency of cloud-hosted model servers.

Overflow, Symphony's grid-as-a-service capability, extends the hybrid model further by enabling federated workload routing across multiple AI factory sites [6]. An inference request submitted to Site A can be routed to Site B if Site A's instances are overloaded and Site B has available capacity. The routing decision is transparent to the client and governed by ELIM metrics from both sites.

F. Extensibility

The multi-ontology architecture accommodates additional compute tiers without modifying the training-inference domain split. Neuromorphic, quantum and mainframe resources register as Symphony or LSF resource types reporting ELIM metrics appropriate to their nature and participate in the compute domain designed for their workload characteristics. The quantum-centric heterogeneous orchestration paper [3] and a companion paper on Palantir Foundry integration [19] detail these extensions.

V. Implementation

The implementation sections describe how the multi-ontology architecture operates in practice. The present paper extends the NVIDIA AI Factory reference architecture [1] by replacing the single-orchestrator layer with a multi-ontology workload management domain. Because the paper proposes an architectural extension rather than a novel standalone system, the relevant scale evidence comes from the constituent platforms themselves. LSF managed 27,648 GPUs across 4,608 nodes at Oak Ridge's Summit [4]. Symphony processes billions of tasks daily across the financial services industry [6], [7]. GPFS is NVIDIA-certified for DGX SuperPOD at the highest throughput tier [9]. The architectural patterns described here are hardware-agnostic and apply to any NVIDIA GPU deployment from a single DGX node to a multi-thousand-node SuperPOD, as each constituent platform has independently demonstrated operation at those scales.

A. Training Configuration

Training workloads run under LSF management with the following configuration:

Distributed training. Multi-GPU training jobs use PyTorch Distributed Data Parallel (DDP) with NCCL backend. LSF allocates GPU resources, launches training processes across allocated GPUs and monitors job health. Checkpoint writes to GPFS occur every 1,000 training steps with configurable retention policies.

InstructLab fine-tuning. RHEL AI's InstructLab provides the fine-tuning pipeline for Granite and other compatible models. LSF manages the InstructLab training job, which generates synthetic training data, executes multi-phase training and writes the fine-tuned model to GPFS. The fine-tuned model is immediately available to Symphony inference services through the GPFS coordination path.

Data obfuscation pipeline. The Obfuscation-as-a-Service architecture, previously demonstrated for medical data use cases, operates within the LSF training domain. LSF manages the data ingest pipeline that accepts sensitive data (PII, HIPAA-protected records), applies differential privacy and synthetic data generation techniques and produces an obfuscated training dataset. The obfuscated data resides on GPFS with access controls enforcing separation between the original and obfuscated datasets. Symphony inference services operate against models trained on the obfuscated data, ensuring sensitive data never enters the inference path. The gold vaulted copy of the original data remains under LSF management with restricted access policies.

B. Inference Configuration

Inference workloads run under Symphony SOAM management with the following configuration:

vLLM model serving. vLLM instances serve models such as Granite 2B, 8B and 34B as Symphony SOAM services, though any vLLM-compatible model can be deployed. Each instance registers with Symphony and reports ELIM metrics at one-second intervals. The SOAM service definition specifies the model tier, GPU memory requirements, maximum concurrent requests and health check endpoints.

Semantic routing service graph. The inference service graph comprises:

1) **Request classifier** (Symphony service): Evaluates query complexity using a lightweight model or neuromorphic

classifier. Assigns the request to a model tier based on complexity score.
2) **Model router** (Symphony dispatcher): Routes the classified request to an available instance of the appropriate model tier based on ELIM metrics.
3) **vLLM inference** (Symphony service): Executes the inference request on the selected model instance. Returns the response through the service graph.
4) **Response validator** (Symphony service): Validates the response against quality criteria. Optionally escalates to a larger model tier if the initial response quality is insufficient.

The entire service graph executes within Symphony's SOAM framework. No external workflow manager, message queue or API gateway participates in the inference path.

KV cache sharing. The three-tier GPFS KV cache architecture enables cross-model and cross-instance cache sharing. When a vLLM instance is terminated or a model is swapped, the KV cache state is written to GPFS warm cache rather than discarded. A new instance loading the same model can restore the KV cache from GPFS, reducing cold-start latency. Cross-model cache transfer enables escalation scenarios where a conversation context computed on a smaller model is transferred to a larger model for more capable response generation.

C. GPFS Configuration

GPFS provides the coordination substrate with the following structure:

```
/gpfs/
  models/             # Model artifacts
    granite-2b/v1.0/  # LSF output
    granite-8b/v2.1/  # Checkpoint+xattrs
    granite-34b/v1.3/
    nemotron-120b/
  training/           # Training workspace
    data/             # Datasets
    logs/             # Metrics
    checkpoints/      # In-progress
  inference/          # Inference workspace
    kv-cache/         # Three-tier cache
      hot/            # RDMA-backed tmpfs
      warm/           # NVMe pool
      cold/           # Capacity pool
    telemetry/        # Inference metrics
    routing-logs/     # Routing decisions
  obfuscation/        # Data vault
    original/         # Gold (restricted)
    obfuscated/       # Training-safe
```

Extended attributes on model directories carry metadata enabling Symphony to discover and load models without a separate registry:

```
user.model.architecture  = granite-8b
user.model.training_step = 150000
user.model.eval_loss     = 1.847
user.model.quantization  = fp16
user.model.ready         = true
user.model.training_job_id = lsf:job:294871
```

ILM policies automate data tiering:

- Model weights accessed within the last hour: NVMe or Flash pool (hot)
- Model weights accessed within the last 24 hours: SSD pool (warm)
- Archived model versions: HDD pool (cold), promoted on access
- Training data: tiered based on access frequency
- KV cache: tiered based on the policy described above

Additional tiers could include Cloud Object Storage on IBM Cloud for offsite model replicas or IBM Spectrum Archive for cold models requiring long-term retention at tape-class economics.

VI. Demonstrations

A. Overview

Five demonstrations validate the multi-ontology architecture across two deployment environments. The on-premise environment comprises an AMD EPYC 7702P server (64 cores, 128 GB RAM) with five NVIDIA RTX 3090 GPUs, ten BrainChip AKD1000 edge nodes and a GPFS RDMA-enabled storage cluster. The cloud environment comprises IBM Cloud infrastructure across two regions (Dallas and Washington DC) with 50 virtual server instances, six NVIDIA A100 GPUs across three GPU servers and Symphony MultiCluster connecting both regions for Overflow federation. The demonstrations establish architectural feasibility rather than benchmark performance at a specific hardware configuration, as the architecture inherits the scaling properties of its constituent platforms and operates within the NVIDIA AI Factory reference architecture whose hardware specifications are defined by the DGX SuperPOD certification [1], [9]. Each demonstration exercises a different aspect of the training-inference coordination model.

B. vLLM Multi-Model Inference under Symphony

Symphony SOAM manages three vLLM instances serving Granite 2B, Granite 8B and Granite 34B. ELIM reports per-instance metrics including GPU utilization, KV cache fill, active requests and P95 latency. The dispatcher routes requests to instances based on model tier assignment and current load.

Under sustained load testing (1,000 requests over 10 minutes with variable query complexity), the Symphony dispatcher maintained P95 latency below 800 ms for Granite 2B and below 2.5 seconds for Granite 34B. Request routing achieved 94% accuracy in matching query complexity to the appropriate model tier, with the remaining 6% representing borderline queries where either tier would produce acceptable results.

C. Semantic Routing with Neuromorphic Classification

The semantic router uses a neuromorphic classifier running on BrainChip AKD1000 to assess query complexity at 622 microseconds per classification. The classifier assigns queries to one of three complexity tiers (simple, moderate, complex) based on embedding analysis of the query text. Simple queries route to Granite 2B, moderate queries to Granite 8B and complex queries to Granite 34B or Nemotron 120B.

The neuromorphic classifier adds 622 microseconds to the inference pipeline, a negligible overhead against the seconds-scale inference latency. The power cost is 30 milliwatts per classification compared to approximately 50 watts for the same classification on an NVIDIA RTX 3090 (350W TDP, with lightweight inference drawing well above the 21W idle floor). At 10,000 classifications per day, the neuromorphic approach saves approximately 500 watt-hours compared to the GPU alternative. The cost reduction from routing simple queries to smaller models is substantial and well-documented in the LLM cascade routing literature, with studies reporting significant savings through query-complexity-aware dispatch [21], [22]. The savings far exceed the marginal cost of the neuromorphic classification tier.

D. KV Cache Sharing across Model Tiers

The three-tier GPFS KV cache demonstration exercises cross-model KV transfer during model-tier escalation. A conversation beginning on Granite 2B produces a KV cache on the GPU serving the 2B model. When the conversation escalates to Granite 8B (triggered by the semantic router detecting increased complexity), the KV cache state is written to GPFS warm cache and read by the Granite 8B instance, which extends the cache rather than recomputing the full context.

The GPFS-mediated cache transfer completes in 340 milliseconds for a 4,096-token context, compared to 2.8 seconds for full context recomputation on Granite 8B. The improvement is 8.2x for the escalation scenario. The Symphony/GPFS approach eliminates the Redis and Kubernetes dependencies present in standard llm-d configurations, reducing operational complexity and providing persistent cache storage, surviving instance restarts.

E. Training-to-Inference Model Handoff

The training-to-inference handoff demonstrates the GPFS coordination path from LSF training to Symphony inference. An InstructLab fine-tuning job running under LSF produces a new model checkpoint on GPFS. Symphony's model management service detects the checkpoint, validates it against quality criteria encoded in xattrs and initiates a rolling update of inference instances. The rolling update loads the new model weights on one instance at a time while the remaining instances continue serving requests with the prior model version.

The handoff completes without service interruption. The elapsed time from checkpoint write to first inference request served by the new model is 47 seconds for Granite 8B, dominated by model weight loading time. No manual intervention, no model registry update and no deployment pipeline is required. The filesystem event triggers the deployment.

F. Obfuscation-as-a-Service Pipeline

The data obfuscation pipeline demonstrates the multi-ontology architecture in a medical data use case. LSF manages the data ingest job that accepts HIPAA-protected records, applies differential privacy transformations and writes the obfuscated dataset to GPFS. The obfuscation job runs as a batch workload with LSF managing resource allocation, job monitoring and failure recovery.

Symphony inference services train against the obfuscated dataset (via a subsequent LSF training job) and serve inference requests from the obfuscated model. The inference path never accesses the original data. GPFS access controls enforce the separation: the original data directory is accessible only to the LSF obfuscation job and the obfuscated data directory is accessible to both LSF training jobs and Symphony inference services. The architecture provides a data vault capability in which sensitive data ingestion, obfuscation, training and inference are managed by appropriate compute platforms with GPFS enforcing data governance through filesystem permissions and xattr-based provenance tracking.

VII. Evaluation

A. Architectural Comparison

The multi-ontology architecture addresses specific limitations of single-platform alternatives for AI factory workloads.

B. Inference Latency

Symphony's service-oriented dispatch provides structurally lower inference routing overhead compared to batch-oriented alternatives. The comparison is architectural rather than implementational. Symphony routes requests to pre-allocated, running service instances by evaluating continuously reported ELIM metrics, completing the dispatch decision in approximately 10 milliseconds. Slurm's batch model requires job submission, queue evaluation, fairshare calculation, and resource allocation for each new workload, an overhead inherent to the batch paradigm regardless of implementation optimizations. Modern Slurm (25.11) has added slurmrestd for REST-based job submission and the Step Manager for offloading step management to compute nodes, but these improvements optimize the batch submission path rather than introducing service-oriented dispatch. SchedMD's own Slinky project, which runs Slurm inside Kubernetes so that Kubernetes can handle inference workloads, is itself an acknowledgment that Slurm's batch architecture cannot natively serve inference workloads.

C. GPU Utilization

The multi-ontology architecture improves GPU utilization by eliminating idle GPU time during workload transitions. In a single-platform model, GPUs allocated to a completed training job remain idle until the next training job is dispatched. In the multi-ontology model, GPUs released by completed training jobs can be reassigned to the inference domain through GPFS-mediated resource signaling, serving inference requests within seconds of training job completion. The mechanism requires coordinated resource management between LSF and Symphony, implemented through a GPFS-resident resource allocation table both platforms consult.

The utilization improvement is architectural. When training jobs complete, the multi-ontology model reassigns GPUs to inference within seconds rather than waiting for the next

TABLE II
ARCHITECTURAL COMPARISON OF WORKLOAD MANAGEMENT APPROACHES FOR AI FACTORY WORKLOADS

Capability	**Slurm (Single)**	**Run:ai (Single)**	**LSF+Symphony (Dual)**
Training management	Native (batch)	Kubernetes job	Native (batch, LSF)
Inference management	Poor (batch model)	Container lifecycle	Native (service, Symphony SOAM)
Inference metric depth	No (GRES counts)	Limited (K8s pod metrics)	38 ELIM metrics per instance
Metric freshness	N/A	Seconds (Prometheus scrape interval)	Sub-second (ELIM push protocol)
Routing algorithm support	No	Application-level only	Native llm-d scorers in platform
Semantic routing	No	Custom operator needed	Native resReq (91.1% accuracy)
KV cache-aware routing	No	Custom sidecar needed	ELIM `vllm_kv_cache_usage_pct`
KV cache persistence	No	Redis (ephemeral)	GPFS three-tier (persistent)
Cross-model KV transfer	No	No	GPFS-mediated, 340 ms / 4K tokens
Inference cost optimization	No	No	Substantial savings via semantic routing
Multi-tenant inference	Job queue fairshare	Namespace quotas	Consumer hierarchies
Cloud bursting	Plugin-based	Pod autoscaling	HostFactory + Overflow
Model handoff	External pipeline	External pipeline	GPFS-mediated, 47 s automatic
Neuromorphic routing	No	No	ELIM resource type, 622 μs
Dispatch overhead	Seconds (batch job submit)	$\sim$100 ms (pod schedule)	$\sim$10 ms (ELIM-based routing)
Multi-cluster federation	Limited (federation since 17.11, no metric-driven routing)	No (per-K8s-cluster scheduling)	Native (LSF MultiCluster, Symphony Overflow, HostFactory cloud bursting)
Production scale	Single-cluster HPC	Per-cluster enterprise	Billions of tasks/day, multi-cluster/multi-cloud, national laboratory and financial services scale

training job. Inference workloads absorb GPU cycles between training jobs rather than allowing GPUs to idle during workload transition gaps. The same GPFS-mediated handoff mechanism operates identically on any GPU type because the handoff is governed by ELIM metrics and consumer policies, not by GPU-specific characteristics.

D. GPFS Coordination Overhead

GPFS coordination overhead is negligible relative to workload execution time. The model handoff path (checkpoint write, xattr detection, model load) adds 47 seconds for an 8B parameter model, dominated by weight loading rather than by GPFS operations. The KV cache transfer adds 340 milliseconds for a 4,096-token context. Xattr operations complete in sub-millisecond time. ILM tiering operates asynchronously and does not affect foreground I/O latency.

The overhead is acceptable because GPFS coordination replaces, rather than supplements, external coordination services. A model registry service, a Redis-based KV cache coordinator and a deployment pipeline each add their own latency and operational complexity. GPFS provides all three functions through the filesystem and the combined GPFS overhead is lower than the combined overhead of the services it replaces.

E. Scalability

The architecture is scale-invariant. The compute ontology (resource schema, consumer hierarchy, SOAM lifecycle) is defined independently of the number of resources it governs. Adding GPU nodes to the training domain or the inference domain requires resource registration with the appropriate platform, not architectural modification. The GPFS coordination path scales with GPFS's proven I/O throughput. The LSF-Symphony resource handoff scales with the filesystem notification mechanism GPFS handles natively. LSF has managed 27,648 GPUs across 4,608 nodes at Oak Ridge's Summit [4]. Symphony manages millions of concurrent tasks at the largest financial institutions in the world [6], [7]. GPFS scales to exabytes across thousands of nodes and is certified for NVIDIA DGX SuperPOD at up to 40 GBps read throughput per node [9].

Horizontal scaling is a further architectural advantage. Both LSF and Symphony are designed for multi-site, multi-cluster federation. Symphony's HostFactory provisions cloud GPU instances on demand and its Overflow capability routes workloads across geographically distributed clusters based on ELIM metrics from all sites. LSF's MultiCluster feature enables job forwarding and resource leasing across independent LSF clusters. The federation capabilities enable AI factory deployments to scale horizontally across data centers, cloud regions, and hybrid environments without architectural modification. Slurm's federation support (since Slurm 17.11) provides job submission across federated clusters but lacks dynamic, metric-driven workload routing. Run:ai operates within a single Kubernetes cluster boundary and has no native multi-cluster federation capability, requiring external service mesh infrastructure for cross-cluster workload distribution.

VIII. Discussion

A. The NVIDIA Workload Management Consolidation

NVIDIA's acquisitions of Run:ai (completed January 2025) and SchedMD (announced December 2025) consolidate the two dominant AI workload management paradigms under a single GPU vendor [11], [12]. The consolidation creates a vertically integrated stack in which NVIDIA controls the GPU hardware (DGX, HGX), the cluster management software (Base Command Manager), the Kubernetes-based GPU orchestrator (Run:ai) and the HPC batch scheduler (Slurm). The integration between these components remains unclear and the industry faces legitimate questions about whether platform innovation will prioritize NVIDIA's commercial strategy over workload requirements [11].

The multi-ontology architecture proposed in the present paper provides an alternative grounded in four decades of production deployment rather than in recent acquisitions. LSF and Symphony predate NVIDIA's entry into workload management by decades. Both platforms have proven production histories in environments where workload management decisions must be vendor-neutral, workload-aware and operationally reliable. The financial services industry's adoption of LSF and Symphony for risk computation and real-time pricing provides a deployment reference NVIDIA's workload management stack has not yet matched.

The open-source status of Slurm under NVIDIA ownership is protected by the GPL v2.0 license, which prevents proprietary relicensing [11]. Community forks are possible if NVIDIA's stewardship diverges from the community's interests. However, the risk of feature disparity between open-source Slurm and NVIDIA-proprietary extensions remains. Customers evaluating AI factory workload management should consider whether dependence on a GPU vendor's workload management stack introduces lock-in risk that independent platforms avoid.

The vendor independence argument requires symmetry. LSF and Symphony are IBM proprietary products and dependence on IBM's workload management stack introduces its own form of vendor risk. The distinction is architectural rather than commercial. LSF and Symphony's workload management capabilities are not tied to a specific hardware vendor's GPU product line, whereas NVIDIA's consolidation binds the workload management layer to the GPU manufacturer. An institution running LSF or Symphony can deploy on NVIDIA, AMD or Intel GPUs without workload management replacement. An institution running Slurm under NVIDIA ownership faces uncertainty about whether future Slurm features will favor NVIDIA hardware. The open-source status of Slurm under GPL v2.0 partially mitigates the risk, but whether NVIDIA's stewardship will preserve feature parity between the open-source release and proprietary extensions remains an open question.

B. The Obfuscation-as-a-Service Model

The data obfuscation pipeline demonstrates a commercial offering enabled by the multi-ontology architecture. The medical data use case, originally pitched in IBM National Markets, provides a template for data vault services in regulated industries. The architecture separates data custody (LSF domain, restricted GPFS access) from model serving (Symphony domain, obfuscated data only), enforcing data governance through compute domain separation and filesystem access controls rather than through application-level security alone.

The model applies beyond medical data. Financial services data subject to GDPR, CCPA or industry-specific regulations can be obfuscated through the same pipeline, with deployment on IBM Cloud providing automated controls aligned with FedRAMP, SOC 2, PCI DSS, ISO 27001 and other regulatory frameworks [26], [27]. The multi-ontology architecture provides the infrastructure; the obfuscation algorithms and privacy guarantees are workload-specific. The key architectural contribution is that the separation of concerns between data custodianship and model serving maps naturally onto the separation between batch scheduling (LSF) and service scheduling (Symphony), with GPFS enforcing the boundary.

C. Implications for the NVIDIA AI Factory Ecosystem

The multi-ontology architecture positions LSF and Symphony as the orchestration layer for NVIDIA AI factories, complementing rather than replacing NVIDIA's hardware and software stack. The architecture uses NVIDIA GPUs, NVIDIA networking (Spectrum-X or InfiniBand), NVIDIA inference runtimes (vLLM with CUDA) and NVIDIA training libraries (NCCL, PyTorch with CUDA). The workload management layer is the architectural differentiation. NVIDIA provides the compute; IBM provides the orchestration and storage.

The positioning aligns with the existing GPFS certification for DGX SuperPOD. IBM Storage Scale is already a validated component of the NVIDIA AI factory ecosystem [9]. Extending from storage partner to orchestration partner leverages the established relationship while addressing a gap the NVIDIA stack does not yet fill: dual-domain workload management that treats training and inference as architecturally distinct workloads deserving distinct compute ontologies.

D. Relationship to the Quantum-Centric Architecture

The multi-ontology architecture and the quantum-centric heterogeneous orchestration architecture share the same foundational principle: resources differing in kind require scheduling designed for their differences rather than scheduling that erases the differences through a common abstraction [3]. The quantum-centric paper applied the principle across compute paradigms (QPU, NPU, GPU, CPU, mainframe). The present paper applies the same principle within a single compute paradigm (GPU) across workload types (training, inference). The principle operates at both levels because the scheduling gap is structural, arising from genuine difference in workload, whether the difference is between quantum superposition and classical determinism or between batch training and service-oriented inference.

The two architectures are composable. An AI factory running the multi-ontology architecture can incorporate quantum

resources through IBM Cloud to IBM Quantum backends, neuromorphic resources through Symphony ELIM registration of AKD1000 nodes and mainframe resources through Symphony SOAM service graphs connecting to z/OS batch processing. The multi-ontology architecture for AI factories is a specialization of the general heterogeneous orchestration architecture for the commercially largest workload category.

E. Limitations

The LSF-Symphony resource coordination through GPFS currently uses periodic polling of GPFS resource state. An event-driven implementation using GPFS watch notifications would reduce reallocation latency between the training and inference domains. The training-to-inference model handoff time is dominated by model weight loading, a hardware I/O constraint independent of the scheduling architecture. Techniques such as model weight pre-staging and incremental weight loading would reduce the handoff time.

The obfuscation pipeline has been demonstrated with synthetic medical data. The differential privacy guarantees require formal privacy analysis beyond the scope of the present architectural paper.

IX. Future Work

Formal benchmarking of the multi-ontology architecture against Slurm-only and Run:ai-only configurations on matched AI factory hardware would provide the quantitative comparison the architectural analysis motivates. The benchmark should measure training throughput, inference dispatch latency, GPU utilization across workload transitions, model handoff time and GPFS coordination overhead including checkpoint write throughput, KV cache transfer latency at scale, xattr propagation time and ILM tiering performance under concurrent training and inference I/O using RHEL AI and vLLM. GPFS performance characterization is essential because the coordination substrate carries the same architectural weight as the compute platforms it connects and its behavior under production load is part and parcel of the multi-ontology architecture design, helping to deliver its theoretical advantages in practice.

X. Conclusion

The AI factory is the defining infrastructure pattern for enterprise artificial intelligence. The hardware is available. The storage is certified. The networking is deployed. The compute platform layer remains the last architectural gap and the gap exists because current approaches treat training and inference as variations of a single workload type rather than as fundamentally different workload paradigms deserving different resource ontologies.

The multi-ontology architecture proposed in the present paper addresses the gap by assigning each workload to a compute platform whose resource ontology matches its characteristics. LSF manages batch training with topology-aware GPU allocation, checkpoint management and distributed training coordination. Symphony manages service-oriented inference with per-model ELIM metrics, semantic routing, KV cache coordination and sub-millisecond dispatch, with RHEL AI and vLLM providing the model serving runtime across NVIDIA, AMD and Intel accelerators. GPFS connects the two domains as the coordination substrate carrying model artifacts, KV cache state, training data and operational telemetry without requiring external coordination services.

The architecture is not speculative. LSF, Symphony and GPFS and one or more of them together have coexisted in production at the largest financial institutions, research organizations, government agencies and companies for decades, managing tens of millions of cores across a variety of compute domains. The three components are proven at production scale both independently and together and the present paper demonstrates their integration for AI factory workloads on commodity/enterprise hardware, on IBM Cloud and proposes the architecture as a reference for production deployment.

The lineage enabling the architecture spans 39 years, from Zhou's 1987 load index through Platform Computing, Platform LSF, Platform Symphony and IBM Spectrum Symphony. The same principle governing the architecture, measuring each resource to its own design and managing workloads on the basis of those measurements within a unified domain, has scaled from VAX workstations to AI factories. Training and inference are different workloads. Compute ontologies ought to recognize and honor the differences rather than erasing them through a common abstraction. The approach provided here produces architectural advantages in inference latency, GPU utilization and operational simplicity. The multi-ontology architecture provides the unified dynamic compute platform the AI factory ecosystem requires.

References

[1] NVIDIA, "AI Factory White Paper: Ecosystem Architecture," 2026. Available: https://docs.nvidia.com/ai-enterprise/planning-resource/ai-factory-white-paper/latest/ecosystem-architecture.html

[2] NVIDIA, "Run:ai GPU Orchestration Platform," integrated into NVIDIA AI Enterprise, 2025. Available: https://www.run.ai/

[3] K. D. Johnson, "High Performance Quantum-Centric Supercomputing: A Working Implementation of Heterogeneous Orchestration across QPU, NPU, GPU, CPU, and Other Tiers," Technical Paper, March 2026.

[4] Oak Ridge National Laboratory, "Summit User Guide," including LSF workload management documentation, 2018–2024. Available: https://docs.olcf.ornl.gov/systems/summit_user_guide.html. See also: A. Bland *et al.*, "Scaling the Summit: Deploying the World's Fastest Supercomputer," OSTI Technical Report, 2018. Available: https://www.osti.gov/servlets/purl/1561654

[5] NVIDIA, "IBM Spectrum LSF," NVIDIA Developer documentation for LSF integration with NVIDIA GPUs. Available: https://developer.nvidia.com/ibm-spectrum-lsf. See also: NVIDIA, "Cluster Management," Available: https://developer.nvidia.com/cluster-management. NVIDIA, "DGX Best Practices," Available: https://docs.nvidia.com/dgx/bp-dgx/index.html (Section 6.3: IBM Spectrum LSF). B. McMillan, "DynaMIG Management of NVIDIA DGX A100 with IBM Spectrum LSF," IBM Community Blog, Jan. 2021.

[6] IBM, *IBM Spectrum Symphony documentation*, including Service-Oriented Architecture Middleware (SOAM), External Load Information Manager (ELIM), HostFactory, Overflow, and Consumer Hierarchies. Available: https://www.ibm.com/docs/en/spectrum-symphony

[7] D. Quintero *et al.*, *IBM Platform Computing Solutions*, IBM Redbook SG24-8073, Dec. 2012. Available: https://www.redbooks.ibm.com/abstracts/sg248073.html

[8] S. Zhou, "Performance Studies of Dynamic Load Balancing in Distributed Systems," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Sciences, Univ. of California, Berkeley, Tech. Rep. UCB/CSD-87-376, Oct. 1987.

[9] IBM, *IBM Storage Scale System 6000 with NVIDIA DGX SuperPOD*, IBM Redbook REDP-5746. Available: https://www.redbooks.ibm.com/abstracts/redp5746.html. See also: IBM, "IBM Storage Scale with NVIDIA," Available: https://www.ibm.com/products/storage-scale/nvidia

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing* (LNCS 2862), Springer, 2003, pp. 44–60.

[11] The Next Platform, "NVIDIA Nearly Completes Its Control Freakery with Slurm Acquisition," Dec. 18, 2025. Available: https://www.nextplatform.com/2025/12/18/nvidia-nearly-completes-its-control-freakery-with-slurm-acquisition/. See also: NVIDIA Blog, "NVIDIA Acquires SchedMD," Dec. 2025. Available: https://blogs.nvidia.com/blog/nvidia-acquires-schedmd/

[12] TechCrunch, "NVIDIA Completes Acquisition of Run:ai," Dec. 30, 2024. Available: https://techcrunch.com/2024/12/30/nvidia-completes-acquisition-of-ai-infrastructure-startup-runai/

[13] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327–1341, Sep. 1988.

[14] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software: Practice and Experience*, vol. 23, no. 12, pp. 1305–1336, Dec. 1993.

[15] HPCwire, "From Clusters to Clouds: An Interview with Platform CEO Songnian Zhou," Jun. 15, 2010. Available: https://www.hpcwire.com/2010/06/15/from_clusters_to_clouds_an_interview_with_platform_ceo_songnian_zhou/

[16] insideHPC, "Songnian Zhou: Why Combine Platform and IBM?," Oct. 2011. Available: https://insidehpc.com/2011/10/songnian-zhou-why-combine-platform-and-ibm/

[17] IBM, *IBM Storage Scale (GPFS) documentation*, including extended attributes (xattrs), Information Lifecycle Management (ILM), Active File Management (AFM), and RDMA. Available: https://www.ibm.com/docs/en/storage-scale

[18] Red Hat, "Red Hat Enterprise Linux AI (RHEL AI)," including vLLM inference, InstructLab fine-tuning and Granite or other compatible models. Available: https://www.redhat.com/en/products/ai/enterprise-linux-ai. See also: Red Hat, "Red Hat AI Enterprise," Feb. 2026. Available: https://www.redhat.com/en/about/press-releases/red-hat-launches-red-hat-ai-enterprise-deliver-unified-ai-platform-spans-metal-agents

[19] K. D. Johnson, "Extending the Sovereign AI Operating System: Symphony as Heterogeneous Orchestrator for Palantir Foundry and NVIDIA Accelerated Computing," Technical Paper, March 2026. (Companion paper.)

[20] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proc. SOSP '23*, 2023. DOI: 10.1145/3600006.3613165.

[21] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance," arXiv:2305.05176, 2023. Published in *TMLR*, 2024.

[22] I. Ong *et al.*, "RouteLLM: Learning to Route LLMs with Preference Data," arXiv:2406.18665, 2024.

[23] Red Hat Developer, "llm-d: Kubernetes-native distributed inferencing," May 2025. Available: https://developers.redhat.com/articles/2025/05/20/llm-d-kubernetes-native-distributed-inferencing

[24] Z. Ye *et al.*, "Deep Learning Workload Scheduling in GPU Datacenters: A Survey," *ACM Computing Surveys*, vol. 56, no. 6, Article 146, Jan. 2024. DOI: 10.1145/3638757.

[25] Q. Weng *et al.*, "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," in *Proc. USENIX NSDI '22*, 2022.

[26] IBM, "IBM Cloud for Financial Services," Available: https://www.ibm.com/cloud/financial-services.

[27] IBM, "IBM Cloud Compliance Programs," Available: https://www.ibm.com/products/cloud/compliance.

[28] NVIDIA, "KAI Scheduler," GitHub repository. Available: https://github.com/NVIDIA/KAI-Scheduler. See Issues #848 (GPU over-allocation infinite retry), #1217 (gang scheduling failure), #423 (no runtime GPU enforcement).

[29] SkyPilot, "Slurm vs. Kubernetes for AI/ML: A Detailed Comparison," SkyPilot Blog, 2025. Available: https://blog.skypilot.co/slurm-vs-k8s/

[30] IBM Research, "Accelerating AI inference with IBM Storage Scale," IBM Research Blog, 2025. Available: https://research.ibm.com/blog/accelerating-ai-inference-with-ibm-storage-scale. See also: Y. Zhu, "Deploying Distributed LLM Inference Service with IBM Storage Scale for KV Cache offloading," IBM Community Blog, Nov. 2025.

[31] V. Hsu, "Accelerating NVIDIA Dynamo with IBM Storage Scale and NVIDIA BlueField-4-Powered Inference Context Memory Storage Platform," IBM Community Blog, Jan. 2026. Available: https://community.ibm.com/community/user/blogs/vincent-hsu/2026/01/05/accelerating-nvidia-dynamo-with-ibm-storage-scale