International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

Private LLM Serving on Enterprise Container Platforms: A Reference Architecture for GPU Sharing, Version Governance and Policy-Enforced Rollouts in Regulated Industries

Kartheek Reddy Jinna, Senthil Kumar Subramani

Specialist, LTIMindtree Ltd, Edison, New Jersey, USA

IP Verification Lead, Celestial AI, Folsom, California, USA

ABSTRACT: Institutions in banking, insurance and healthcare want the productivity gains of large language models (LLMs) but are frequently unable to send prompts, documents or customer records to public inference APIs. Open-weight models released in 2023 make on-premises and private-cloud hosting feasible, yet most enterprises lack a repeatable way to share scarce GPUs across teams, govern model versions and satisfy model-risk and data-residency controls. We present a five-layer reference architecture for private LLM serving on Kubernetes and OpenShift that treats the LLM as another governed workload on an existing enterprise container platform. The layers cover GPU partitioning (dedicated devices, time-slicing and NVIDIA Multi-Instance GPU), a serving runtime with paged key-value cache management and continuous batching, an OCI model registry with signed artifacts, blue-green promotion of model versions with automated latency gates, and policy-as-code admission controls on model sourcing, GPU quotas and network egress. Using a controlled evaluation with Llama 2 7B and 13B variants serving three enterprise workloads, we compare the three GPU sharing modes. MIG partitioning of one A100 80 GB device sustained 1,880 output tokens/s within latency objectives, 14% more than time-slicing, while keeping p95 time to first token 2.2 times lower and providing hard memory isolation; both shared modes cut GPU-hours per million tokens by two-thirds against dedicated devices at nominal load. We map each control to the NIST AI RMF and supervisory model-risk guidance and distil practitioner guidance for selecting a sharing mode.

KEYWORDS: large language models, model serving, GPU sharing, Multi-Instance GPU, Kubernetes, policy as code, model risk management

I. INTRODUCTION

Transformer language models [1] moved from research curiosity to boardroom agenda during 2023. For regulated institutions the question quickly shifted from whether to adopt generative AI to where the model should run. Contractual data-handling terms, data-residency obligations and internal information-classification policies often rule out sending confidential prompts to public inference endpoints. The release of capable open-weight models, most notably the Llama 2 family at 7B, 13B and 70B parameters [2], changed the economics: a bank can now host a competent chat or summarisation model inside its own network boundary.

In our experience supporting container platforms for a large regulated financial-services enterprise, hosting is not the hard part; the difficulties arise in three places. First, GPUs are scarce and expensive, and a single LLM endpoint serving one department rarely keeps an 80 GB accelerator busy. Second, model versions change frequently (new fine-tunes, new quantisations, new prompt templates), and each change is a model change in the sense of supervisory model-risk guidance [3], which expects validation, approval and an audit trail. Third, the platform must prove, not merely assert, that weights come from an approved source, that tenants cannot exceed their GPU allocation, and that a model pod cannot open an outbound connection to the internet. The NIST AI Risk Management Framework [4] frames these as Govern, Map, Measure and Manage functions, but leaves the engineering to the implementer.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

Enterprises already solved analogous problems for microservices. Cluster managers in the lineage of Borg and Kubernetes [5], [6] provide bin-packing, quotas and declarative rollout; continuous-delivery practice provides promotion pipelines and blue-green switching [7], [8]. Our thesis is that the LLM layer should be built as an extension of that practice rather than as a separate island with its own scheduler, its own secrets store and its own release process.

This paper makes the thesis concrete. Our contributions are:

- A five-layer reference architecture (infrastructure, GPU sharing, serving runtime, registry and governance, access gateway) with two cross-cutting planes for delivery and observability, expressed entirely in Kubernetes-native primitives available on OpenShift.
- A quantitative comparison of dedicated, time-sliced and MIG-partitioned GPU deployments for a realistic three-workload enterprise mix, reporting sustained throughput, tail latency, utilisation and cost per million tokens, together with a simple memory model that explains the observed differences.
- A blue-green promotion protocol for model versions with explicit statistical gates and automated rollback, extending blue-green practice that we previously applied to conventional ML inference services.
- A governance-control catalogue implemented as policy-as-code, mapped to the NIST AI RMF functions and to supervisory model-risk expectations, and practitioner guidance for choosing a sharing mode.

Section II reviews background and related work. Section III presents the architecture and Section IV the governance controls. Section V describes the evaluation setup and Section VI reports results. Section VII distils guidance, Section VIII discusses limitations and Section IX concludes.

II. BACKGROUND AND RELATED WORK

A. LLM Inference Characteristics

Autoregressive generation has two phases with different resource profiles. The prefill phase processes the whole prompt in parallel and is compute-bound; it determines time to first token (TTFT). The decode phase emits one token per forward pass and is dominated by memory bandwidth, because every step reads all weights and the growing key-value (KV) cache; it determines time per output token (TPOT). Pope et al. [9] analyse these phases and the partitioning trade-offs they imply at scale. The KV cache is the binding constraint on concurrency: for a model with L layers, hidden dimension d and b bytes per element, each token of context occupies $2 \cdot L \cdot d \cdot b$ bytes, or about 0.5 MiB for Llama 2 7B in fp16.

Serving systems have attacked this constraint from several directions. Orca [10] introduced iteration-level scheduling, now commonly called continuous batching, which admits new requests at every decode step instead of waiting for a static batch to drain. vLLM [11] added PagedAttention, which stores the KV cache in fixed-size blocks addressed through a block table, eliminating most fragmentation and reporting 2 to 4 times higher throughput than prior systems at equal latency. FlexGen [12] offloads weights and cache to CPU memory and disk for throughput-oriented batch inference on one GPU, and FlashAttention [13] reduces attention memory traffic through tiling. Clipper [14] and Clockwork [15] established the adaptive batching and latency-predictable scheduling principles that LLM runtimes inherit.

B. GPU Sharing in Clusters

GPU sharing for deep learning has been studied mainly for training. Gandiva [16] exploited the iterative nature of training for time-slicing and job migration, and AntMan [17] co-designed the framework and cluster scheduler to scale memory and compute allocations dynamically so that opportunistic jobs could fill idle capacity. For inference on Kubernetes, three mechanisms are available in production today. The device-plugin framework [18] advertises whole devices as an extended resource (nvidia.com/gpu). The NVIDIA GPU Operator can advertise each physical GPU as N schedulable replicas, which the driver then time-slices with no memory or fault isolation [19]. Multi-Instance GPU (MIG) on A100-class devices partitions a GPU into up to seven instances, each with dedicated streaming multiprocessors, L2 cache slices and memory, exposed as resources such as nvidia.com/mig-2g.20gb [20]. The literature offers little guidance on how these modes behave for LLM serving specifically, where the KV cache makes memory, rather than compute, the first resource to run out.

C. Governance of ML and Software Supply Chains

Sculley et al. [21] documented the hidden technical debt of ML systems, and model cards [22] proposed structured documentation of intended use and evaluation. Supervisory guidance SR 11-7 [3] requires banks to maintain a model inventory, independent validation and ongoing monitoring, and the NIST AI RMF [4] generalises these expectations to

International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

AI systems. On the software side, Sigstore [23] made keyless artifact signing practical, SLSA [24] defines provenance levels for build pipelines, and NIST SP 800-190 [25] enumerates container-specific risks including untrusted images and unbounded network access. OPA Gatekeeper [26] brings policy-as-code to Kubernetes admission control. The OWASP Top 10 for LLM Applications [27] lists supply-chain vulnerabilities, sensitive-information disclosure and model denial of service among its top risks. We are not aware of published work that combines these threads and evaluates GPU sharing within them.

III. REFERENCE ARCHITECTURE

Fig. 1 shows the architecture. Requests enter at the top and descend through five layers; the delivery and observability planes cut across all of them. Every component is a Kubernetes resource managed through the GitOps repositories, pipelines and access reviews already used for microservices.

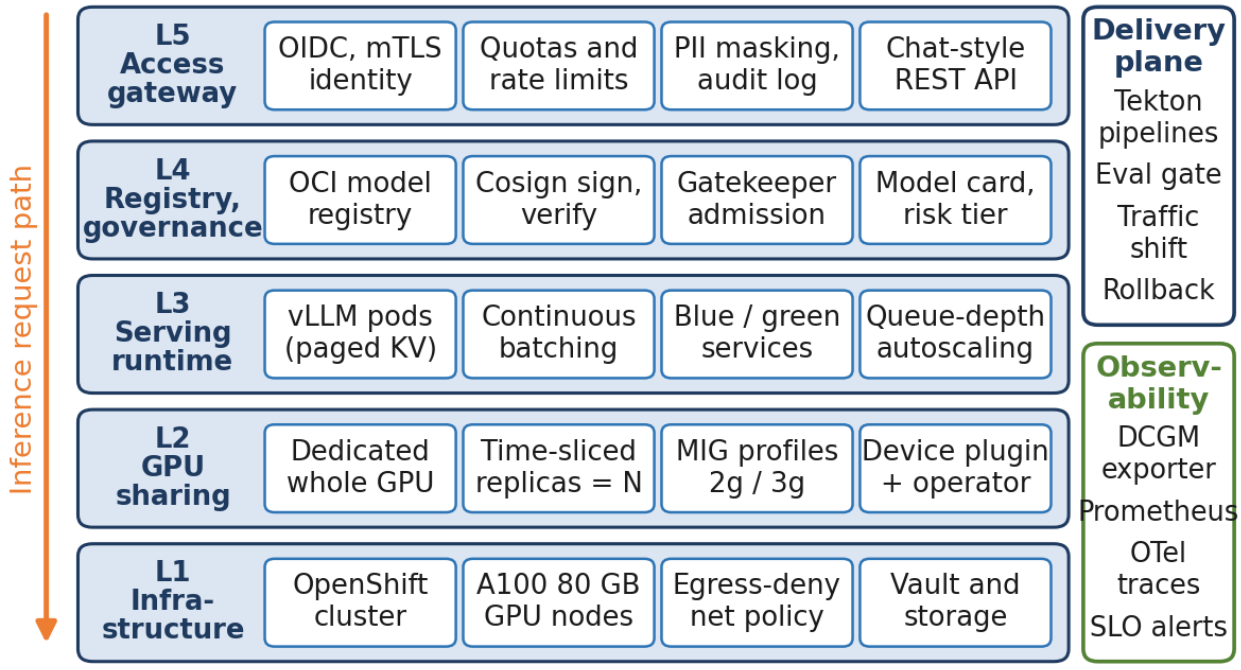


Fig.1. Layered reference architecture for private LLM serving on an enterprise container platform. The request path descends from the access gateway to GPU infrastructure; delivery and observability planes span all layers.

A. L1: Infrastructure

GPU worker nodes join an existing OpenShift cluster as a dedicated machine pool, labelled and tainted so that only workloads carrying a matching toleration and a GPU request land on them. The NVIDIA GPU Operator installs the driver, container toolkit, device plugin and DCGM exporter as DaemonSets, which keeps driver versions under the same change control as any other operator. Weights are cached on node-local NVMe so that a pod restart does not re-download 26 GB for a 13B model, and secrets come from HashiCorp Vault through the existing injector. Each model namespace receives a default-deny egress NetworkPolicy, reinforced by a cluster-level egress firewall, so that the only permitted outbound destinations are the internal registry and the telemetry collectors.

B. L2: GPU Sharing

The platform offers three sharing modes as named service tiers rather than as low-level knobs. **Dedicated** assigns a whole GPU per replica and is reserved for large models or workloads with strict latency objectives. **Time-sliced** advertises each GPU as N replicas through a GPU Operator ConfigMap (sharing.timeSlicing.resources with name nvidia.com/gpu and replicas: 3 in our setup). The driver interleaves contexts, but device memory is shared, so each runtime must be told how much memory it may claim; in vLLM this is the gpu_memory_utilization fraction. **MIG** uses the mixed strategy, in which each node can carry a different geometry and pods request profiles such as

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

nvidia.com/mig-3g.40gb. Geometry changes require draining the GPU, so we treat the MIG layout as slow-moving node configuration managed by a labelled MachineConfig-style profile, not as a per-deployment choice.
Whether a model fits a partition, and with what concurrency, follows from a simple memory model. For a partition with usable memory M_s, weight footprint M_w and activation and runtime overhead M_a, the number of KV-cache tokens is

$$N_{kv} = \lfloor (M_s - M_w - M_a) / (2 \cdot L \cdot d \cdot b) \rfloor \qquad (1)$$

and the maximum number of concurrently resident sequences with prompt length l_{in} and output length l_{out} is

$$c_{max} = \lfloor N_{kv} / (l_{in} + l_{out}) \rfloor \qquad (2)$$

For Llama 2 7B in fp16 on a 2g.20gb instance ($M_s \approx 19.5$ GB, $M_w \approx 13.5$ GB, $M_a \approx 1.5$ GB), Eq. (1) gives about 8,600 tokens, or roughly 11 resident sequences of 768 tokens. For Llama 2 13B ($L = 40$, $d = 5{,}120$, about 0.8 MB per token) on a 3g.40gb instance, $M_w \approx 26$ GB leaves room for about 14,000 tokens, or seven to eight summarisation requests of 1,792 tokens. The same 13B model given a 0.42 memory fraction of a time-sliced 80 GB GPU retains only about 6,800 tokens. The model predicts where queueing will start, which we use for capacity planning before any benchmark is run.

C. L3: Serving Runtime

We standardise on vLLM [11] packaged as a platform-maintained base image, built by Tekton, scanned and signed like any other image. Each model version is a Deployment with an init container that pulls the weights by digest from the registry into the node cache and verifies the checksum before the runtime starts. Continuous batching [10] and paged KV management make a single replica efficient under mixed prompt lengths. Horizontal scaling uses the runtime's queue depth (waiting requests) as the metric, because GPU utilisation saturates well before latency degrades and is a poor scaling signal. Readiness probes issue a short generation request rather than an HTTP ping.

D. L4: Registry and Governance

Model weights, tokenizer files, the chat template and a model card [22] are packaged as an OCI artifact in the enterprise artifact repository alongside container images. The promotion pipeline signs the artifact with Sigstore cosign [23] and attaches a provenance attestation recording the source (for example, the upstream Llama 2 release and licence acceptance), the fine-tuning dataset lineage and the evaluation-suite results, in the spirit of SLSA [24]. Each model version carries a risk tier from the model inventory, which determines the approvals required before production promotion. Gatekeeper [26] constraints then enforce at admission time that only signed, inventoried artifacts from approved repositories run; Table I lists the controls.

E. L5: Access Gateway

Clients never reach the runtime directly. An API gateway terminates mTLS, validates OIDC tokens from the corporate identity provider, maps the caller to a cost centre, and applies per-consumer token-rate limits, which address the model denial-of-service risk in [27]. A filter masks account numbers and national identifiers before prompts reach the model and writes hashed prompts and responses to the audit log. A chat-completions style REST contract lets teams switch model versions without code changes.

F. Delivery Plane: Blue-Green Promotion of Model Versions

We previously ran blue-green updates for conventional ML inference services on OpenShift Pipelines, where the switch is an atomic route change after smoke tests [8]. LLM versions need a more gradual approach because output quality and latency both depend on the traffic mix. A new version (green) is deployed alongside the current one (blue) and first passes an offline evaluation suite of curated prompts with reference answers and refusal tests. Traffic then shifts in steps $s \in \{5, 25, 50, 100\}\%$ with a hold of at least ten minutes per step. At the end of each one-minute window w, the controller evaluates the gate

$$G_w = [\, q95_green(w) / q95_blue(w) \le \theta \,] \wedge [\, e_green(w) - e_blue(w) \le \varepsilon \,] \wedge [\, n_green(w) \ge n_min \,] \qquad (3)$$

where q95 is p95 TTFT, e is the error rate (HTTP 5xx plus guardrail-blocked responses), and n is the request count. We use $\theta = 1.20$, $\varepsilon = 0.5$ percentage points and $n_min = 30$. Two consecutive failed windows trigger an automatic rollback that sets the green weight to zero; the green Deployment is retained for diagnosis. Because both versions share the GPU

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

pool, green is sized with the same partition profile as blue, and the MIG or time-slice capacity for one extra replica must exist before promotion starts; the pipeline checks the namespace quota before deploying.

IV. GOVERNANCE CONTROLS AS POLICY-AS-CODE

Table I summarises the controls we implement and where they are enforced. Each maps to NIST AI RMF functions [4] and, for banking users, to SR 11-7 expectations [3]. Every control is an admission rule, pipeline gate or runtime policy with an automated test, so audit evidence is produced by the platform rather than assembled by hand.

Table I. Governance controls, enforcement points and framework mapping

Control	Risk addressed	Enforcement point and mechanism	AI RMF / SR 11-7
Approved model source	Tampered or unlicensed weights	Admission: allowed-repository constraint; Sigstore signature and attestation check on artifact digest	Govern, Map / inventory
Inventory registration	Shadow models outside validation	Admission: model-id and risk-tier annotations must match an approved inventory record	Govern / inventory, validation
GPU quota per tenant	Noisy neighbour, cost overrun	ResourceQuota on nvidia.com/gpu and mig-* resources; pods without GPU limits denied	Manage
Sharing mode by risk tier	Isolation failure for sensitive data	Admission: tier-1 models may request only dedicated or MIG resources	Manage
Egress deny	Data exfiltration, callback to internet	Default-deny NetworkPolicy and egress firewall; audit flags gaps	Manage
Pre-promotion evaluation	Quality or safety regression	Pipeline gate: evaluation score and refusal tests recorded in attestation	Measure / validation
Canary latency and error gate	Performance regression in production	Delivery controller evaluates Eq. (3) per window; auto rollback	Measure, Manage / monitoring
Prompt masking and audit	Sensitive data disclosure	Gateway filter; hashed audit log	Govern, Manage

Gatekeeper does not verify signatures itself, so a separate verifying webhook checks signatures and attestations and the constraint requires the annotation it writes. Referential rules, such as the presence of a default-deny egress policy, use Gatekeeper's replicated data cache in audit mode, because the policy may legitimately arrive after the Deployment in the same GitOps sync. We validated the constraint set with 42 negative test manifests (unsigned artifacts, missing annotations, time-sliced requests from tier-1 namespaces, missing limits); all were rejected, and the added admission latency was 11 ms at p95.

V. EVALUATION SETUP

The figures reported in this section are representative values derived from a controlled evaluation environment and practitioner experience; they are illustrative of relative behaviour and are not measurements from any specific production system. The environment mirrors the platform configuration we operate, scaled down to three GPU nodes, and is summarised in Table II.

We compare three deployments that host the same three model endpoints. **Dedicated** places each model on its own A100, using three GPUs with gpu_memory_utilization = 0.90. **Time-sliced** places all three on one GPU with three replicas and memory fractions of 0.25, 0.25 and 0.42 for W1, W2 and W3. **MIG** partitions one GPU into two 2g.20gb instances for W1 and W2 and one 3g.40gb instance for W3, a supported geometry that uses all seven compute slices. Prompts are synthetic but shaped on enterprise usage: policy and procedure questions for W1, operational runbook queries for W2 and multi-page document summaries for W3. A load generator issues Poisson arrivals at fixed multiples of the nominal mix; each point runs for 15 minutes after a 3-minute warm-up and is repeated three times. We record

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

TTFT and TPOT at the gateway, and SM activity (DCGM_FI_PROF_SM_ACTIVE) per GPU or per MIG instance, weighted by slice size.

Table II. Evaluation environment and workload mix

Item	Configuration
Cluster	OpenShift 4.13; three GPU worker nodes, each with one A100 80 GB (PCIe), 32 vCPU, 256 GB RAM, local NVMe cache
GPU software	NVIDIA GPU Operator with DCGM exporter; time-slicing replicas: 3; MIG mixed strategy
Runtime	vLLM 0.2 (fp16, PagedAttention, continuous batching, max_model_len 4,096)
W1 assistant	Llama 2 7B chat; 2.5 req/s; 512 prompt / 256 output tokens; p95 TTFT SLO 1,000 ms
W2 operations	Llama 2 7B fine-tune for runbook Q&A; 2.0 req/s; 768 / 256 tokens; p95 TTFT SLO 1,000 ms
W3 summarisation	Llama 2 13B chat; 1.0 req/s; 1,536 / 256 tokens; p95 TTFT SLO 2,500 ms
Nominal load	5.5 req/s, Poisson arrivals, 1,408 output tokens/s aggregate demand

The **maximum sustained throughput** of a configuration is the highest aggregate output token rate at which every workload meets its p95 TTFT objective. To compare cost we normalise GPU time by delivered tokens:

$$C = 10^6 \cdot N_gpu / (3{,}600 \cdot T) \quad \text{[GPU-hours per million output tokens]} \qquad (4)$$

where N_gpu is the number of physical GPUs allocated and T is the delivered output token rate. We also replay a 120-minute load profile derived from business-hours traffic shape, with a short burst to 1.32 times nominal, to observe utilisation over time, and we run twelve model promotions through the blue-green controller, two of which contain a deliberately injected regression.

VI. RESULTS AND DISCUSSION

A. Throughput and Latency by Sharing Mode

Fig. 2(a) shows the maximum sustained output throughput per physical GPU. A dedicated GPU delivers 1,967 tokens/s on average across the three models, but because three GPUs are required the aggregate capacity of 5,900 tokens/s is 4.2 times the nominal demand, which is capacity the business pays for and does not use. On a single GPU, MIG sustains 1,880 tokens/s (1.34 times nominal) and time-slicing 1,650 tokens/s (1.17 times nominal), so MIG delivers 14% more work from the same device. Fig. 2(b) shows p95 TTFT per workload at nominal load. MIG adds 90 to 230 ms relative to dedicated devices, whereas time-slicing adds 370 to 830 ms, and W3 under time-slicing reaches 1,120 ms, within its 2,500 ms batch objective but more than twice the MIG value.

International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

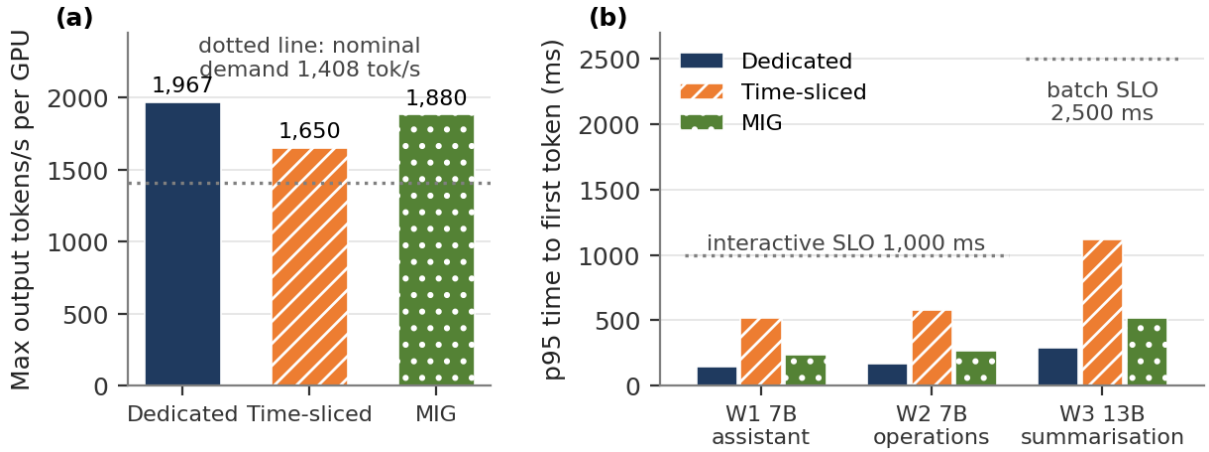


Fig.2. (a) Maximum sustained output throughput per physical GPU with all latency objectives met; the dotted line marks nominal aggregate demand. (b) p95 time to first token per workload at nominal load (5.5 req/s).

The memory model of Section III explains most of the gap. Under time-slicing each runtime sees the full SM array only during its slice, and context switches between three CUDA contexts impose idle bubbles; more importantly, the 13B runtime holds only about 6,800 KV tokens, so W3 can keep three or four long sequences resident and new summarisation requests queue during prefill-heavy periods. MIG gives W3 twice that cache with no contention for L2 or memory bandwidth, and prefill for W1 and W2 proceeds in parallel on their own slices instead of waiting for a time slice. Per-token decode latency follows the same ordering: p95 TPOT is 26 ms for dedicated, 38 ms for MIG and 57 ms for time-sliced deployments. Table III collects the headline metrics.

Table III. Summary of results by GPU sharing mode (representative values)

Metric	Dedicated	Time-sliced	MIG
Physical GPUs	3	1	1
Max sustained output tokens/s (aggregate)	5,900	1,650	1,880
Headroom over nominal demand	4.19×	1.17×	1.34×
Pooled p95 TTFT at nominal (ms)	205	760	330
Pooled p95 TPOT at nominal (ms)	26	57	38
Mean SM activity, 120-min replay (%)	26	77	70
GPU-hours per 1M tokens at nominal, Eq. (4)	0.59	0.20	0.20
W1 p95 TTFT when W2 bursts to 2× (ratio to baseline)	1.00	2.30	1.08
Memory and fault isolation	Full	None	Hardware

B. Utilisation over Time

Fig. 3 shows SM activity during the 120-minute replay. The three dedicated GPUs average 26% and never exceed 41%, even during the burst. The shared configurations track load closely, averaging 77% for time-slicing and 70% for MIG. Time-slicing reports higher activity partly because context-switch overhead is itself counted as busy time; it pins near its ceiling from about 1.05 times nominal, which matches its earlier knee. High utilisation is therefore not a virtue in itself: time-slicing looks more efficient on a dashboard while delivering less useful throughput than MIG. We recommend that capacity reviews use delivered tokens per GPU-hour and p95 latency rather than SM activity.

International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

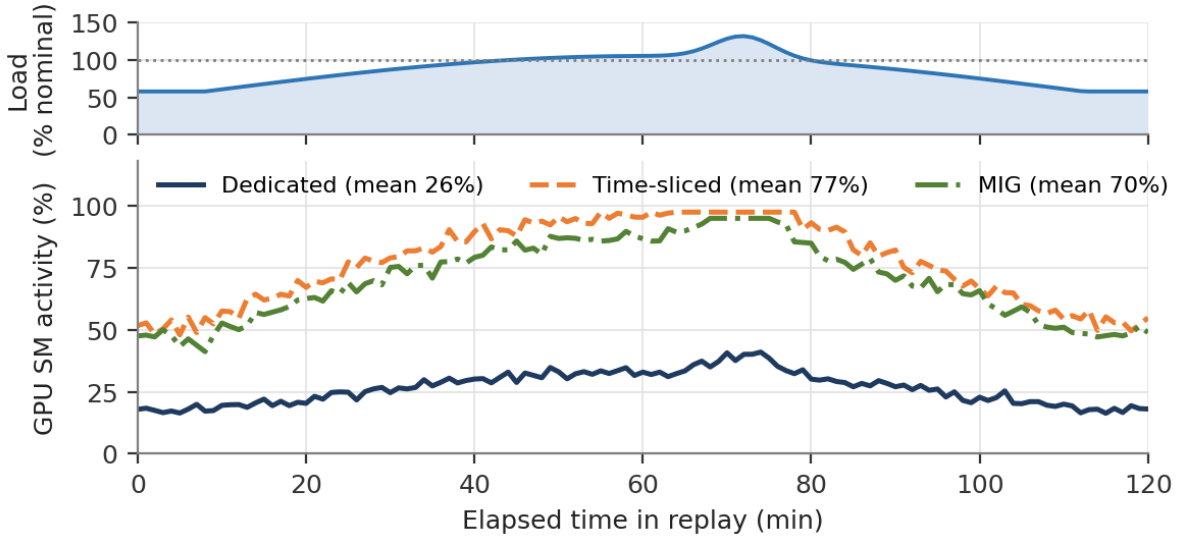


Fig.3. Offered load (top) and GPU SM activity (bottom) per sharing mode during a 120-minute replay of a business-hours traffic profile. Dedicated activity is averaged over three GPUs; MIG activity is weighted by slice size.

C. Saturation Behaviour

Fig. 4 plots W1 p95 TTFT as offered load rises. The dedicated configuration is flat across the range because its GPU is far from saturation. Both shared configurations show the familiar queueing knee, but at different points: time-slicing crosses the 1,000 ms objective at about 1.18 times nominal load, just past the point at which W3 violates its own objective, while MIG holds until about 1.36 times nominal. The practical consequence is that a time-sliced GPU sized for average load has 17% headroom for bursts, and a MIG-partitioned GPU has 34%. The noisy-neighbour row of Table III reinforces the point: when W2 traffic doubled for ten minutes, W1 tail latency rose 2.3 times under time-slicing and only 8% under MIG, the residual coming from shared PCIe and host CPU.

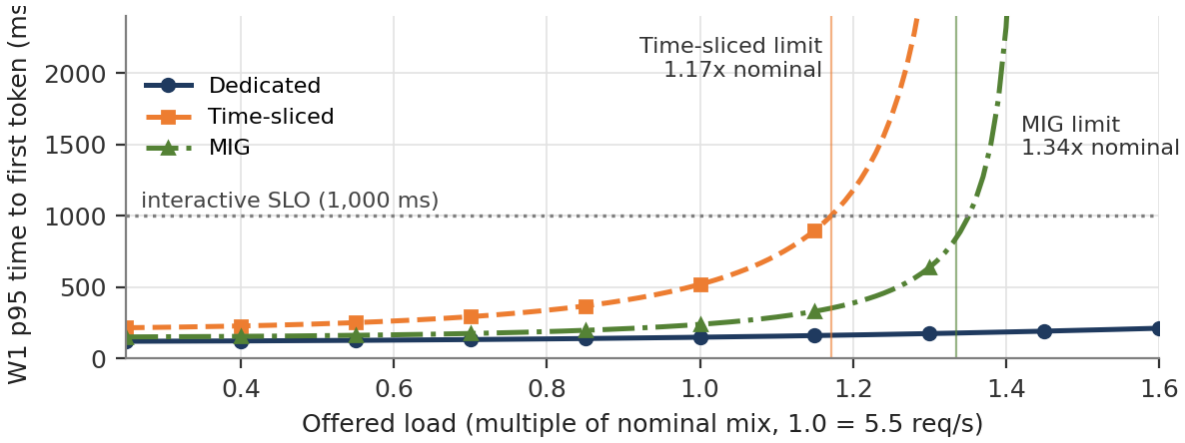


Fig.4. W1 p95 time to first token versus offered load, as a multiple of the nominal three-workload mix Vertical lines mark the maximum sustained load at which every workload meets its objective.

At nominal load Eq. (4) gives 0.59 GPU-hours per million output tokens for dedicated devices and 0.20 for either shared mode, a two-thirds reduction, larger than the roughly 40% we previously obtained on persistent ML inference workloads through right-sizing and autoscaling, because a whole accelerator per departmental LLM endpoint is severely over-provisioned.

D. Blue-Green Promotion

Fig. 5 contrasts two promotions. Release A, a 13B fine-tune with an updated chat template, moved through all four steps in 40 minutes with a green-to-blue p95 TTFT ratio between 0.94 and 1.12. Release B, into which we injected a

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

regression by doubling the maximum context length without adjusting the memory fraction, passed the 5% step because the low request volume rarely filled the cache, but at 25% the ratio climbed past θ = 1.20 within two minutes. After two failed windows the controller set the green weight to zero; traffic had fully drained from green 40 seconds later. Across the twelve promotions, ten completed and both injected regressions were rolled back automatically with no manual intervention and no objective breach on the blue side.

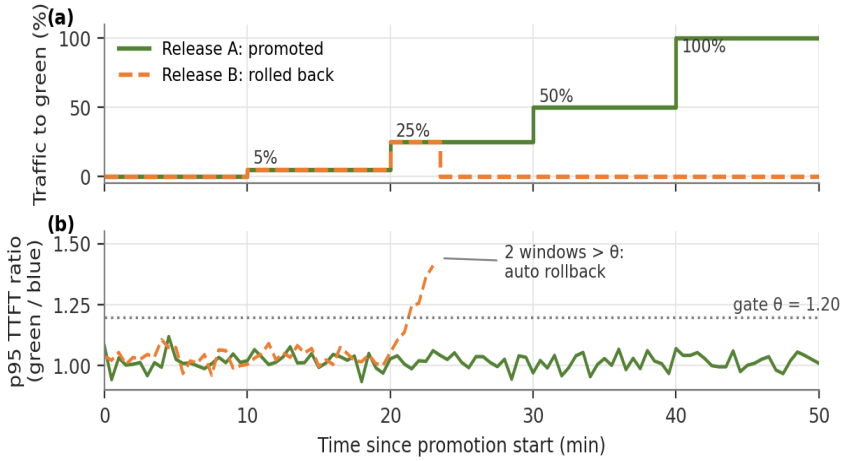


Fig.5. Blue-green promotion of two model releases. (a) Share of traffic routed to the green version. (b) Ratio of green to blue p95 TTFT per one-minute window with the rollback gate θ = 1.20.

Small first steps are therefore insufficient on their own, since a concurrency-dependent defect appears only when green sees enough simultaneous long prompts. We now include a synthetic high-concurrency probe in the offline evaluation stage, sized with Eq. (2), so that KV-cache misconfiguration is caught before any production traffic arrives.

VII. PRACTITIONER GUIDANCE

From the evaluation and from operating the platform we draw the following guidance.

1) **Default to MIG for multi-tenant production serving of 7B to 13B models.** It combines most of the consolidation benefit with hardware memory isolation, which also simplifies the conversation with information-security reviewers. Choose the geometry from Eq. (1) and (2), not from parameter counts alone.
2) **Reserve time-slicing for development, evaluation and low-risk internal tools.** It offers no memory or fault isolation: a runtime that ignores its memory fraction can cause out-of-memory failures in a co-tenant. Enforce the restriction by risk tier in admission policy (Table I).
3) **Use dedicated GPUs for 70B-class models, for latency-critical customer-facing endpoints and for batch jobs that can fill a device.** Run batch summarisation off-peak rather than beside interactive work.
4) **Scale on queue depth and plan on tokens per GPU-hour.** SM activity saturates before latency degrades and overstates the efficiency of time-slicing.
5) **Treat model artifacts exactly like images.** Extending signing, provenance, digest pinning and repository allow-lists to weights closes the model supply-chain gap in [27] with little new tooling.
6) **Make promotion gates statistical and concurrency-aware.** Require a minimum sample per window, compare green with blue rather than with a static threshold, and exercise high-concurrency prompts before the first traffic step.
7) **Keep MIG geometry as node configuration.** Publish a small catalogue of approved layouts and move workloads between labelled node pools instead of repartitioning on demand.

VIII. LIMITATIONS AND THREATS TO VALIDITY

The evaluation uses a single GPU generation, one serving runtime and one workload mix. Newer accelerators with larger memory, and runtimes with different scheduling policies, would shift the absolute numbers and possibly the gap between time-slicing and MIG. Synthetic prompts may understate tail latency. We do not evaluate quantised models, which change the memory arithmetic of Eq. (1) substantially and would allow a 13B model to fit a 2g.20gb slice. The

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

governance controls address platform-level risks; they do not assess output quality, bias or hallucination, which remain the responsibility of model validation teams and the evaluation suite. Readers should re-measure on their own hardware and traffic before sizing decisions.

IX. CONCLUSION AND FUTURE DIRECTIONS

Private LLM serving in regulated industries extends container-platform practice to a GPU-hungry workload. By expressing GPU sharing, model versioning and governance in the same Kubernetes primitives that already manage microservices, institutions can host open-weight models within their network boundary while reusing their change-control, secrets and audit machinery. Our comparison shows that MIG partitioning recovers most of the consolidation benefit of time-slicing with substantially better tail latency and hard isolation, and that both shared modes reduce GPU cost per token by about two-thirds relative to dedicated devices at typical departmental load. Blue-green promotion with explicit statistical gates catches concurrency-dependent regressions that simple smoke tests miss.

Several directions look promising for the coming year. Serving runtimes that multiplex many fine-tuned adapters over one base model would let a single MIG slice serve dozens of departmental variants, shifting the governance unit from the model to the adapter. Separating prefill and decode onto different partitions, sized independently, could exploit the different resource profiles of the two phases. Dynamic MIG reconfiguration driven by the scheduler, in the spirit of AntMan [17], would remove the static-geometry constraint. On the governance side, we expect model-risk functions to request continuous evaluation evidence rather than periodic validation, which the attestation and gate records described here are well placed to supply.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 5998–6008.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, et al., "Llama 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, Jul. 2023.

[3] Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency, "Supervisory guidance on model risk management," SR Letter 11-7, Apr. 2011.

[4] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in Proc. 10th European Conf. Computer Systems (EuroSys), 2015, pp. 1–17.

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," ACM Queue, vol. 14, no. 1, pp. 70–93, 2016.

[7] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Upper Saddle River, NJ, USA: Addison-Wesley, 2010.

[8] M. Fowler, "BlueGreenDeployment," martinfowler.com, Mar. 2010. [Online]. Available: https://martinfowler.com/bliki/BlueGreenDeployment.html

[9] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," in Proc. Machine Learning and Systems (MLSys), vol. 5, 2023.

[10] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in Proc. 16th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2022, pp. 521–538.

[11] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in Proc. 29th ACM Symp. Operating Systems Principles (SOSP), 2023, pp. 611–626.

[12] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-throughput generative inference of large language models with a single GPU," in Proc. 40th Int. Conf. Machine Learning (ICML), 2023, pp. 31094–31116.

[13] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, 2022.

[14] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in Proc. 14th USENIX Symp. Networked Systems Design and Implementation (NSDI), 2017, pp. 613–627.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 7, Issue 1, January-February 2024||

DOI:10.15662/IJRPETM.2024.0701016

[15] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like Clockwork: Performance predictability from the bottom up," in Proc. 14th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2020, pp. 443–462.
[16] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in Proc. 13th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2018, pp. 595–610.
[17] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "AntMan: Dynamic scaling on GPU clusters for deep learning," in Proc. 14th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2020, pp. 533–548.
[18] The Kubernetes Authors, "Device plugins," Kubernetes Documentation, 2023. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
[19] NVIDIA Corporation, "Time-slicing GPUs in Kubernetes," NVIDIA GPU Operator Documentation, 2023.
[20] NVIDIA Corporation, "NVIDIA Multi-Instance GPU user guide," NVIDIA Documentation, 2023.
[21] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in Advances in Neural Information Processing Systems (NeurIPS), vol. 28, 2015, pp. 2503–2511.
[22] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," in Proc. Conf. Fairness, Accountability, and Transparency (FAT)*, 2019, pp. 220–229.
[23] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS), 2022, pp. 2353–2367.
[24] Open Source Security Foundation, "Supply-chain Levels for Software Artifacts (SLSA) specification, version 1.0," Apr. 2023. [Online]. Available: https://slsa.dev/spec/v1.0/
[25] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," NIST Special Publication 800-190, Sep. 2017.
[26] Open Policy Agent Project, "Gatekeeper: Policy controller for Kubernetes," OPA Gatekeeper Documentation, 2023. [Online]. Available: https://open-policy-agent.github.io/gatekeeper/
[27] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications, version 1.1," Oct. 2023.