The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 33-59

Check for updates

Optimizing Cloud-Native LLM Workloads with Serverless GPU Orchestration and Token-Aware Scheduling

Pradeep Rao Vennamaneni
Senior Data Engineer - Lead, USA

OPEN ACCESS

SUBMITTED 05 February 2024
ACCEPTED 15 March 2024
PUBLISHED 25 April 2024
VOLUME Vol.04 Issue 04 2024

CITATION
Pradeep Rao Vennamaneni. (2024). Optimizing Cloud-Native LLM Workloads with Serverless GPU Orchestration and Token-Aware Scheduling. The American Journal of Engineering and Technology, 4(04), 33–59. Retrieved from https://theamericanjournals.com/index.php/tajet/article/view/6603

Abstract: Cloud-native LLM inference has bursty and size-variable demand that leads to head-of-line blocking, cold-start overheads, and infinitely variable tail latency. An end-to-end design that integrates token-sensitive scheduling with serverless GPU orchestration to achieve TTFT/TLET SLOs at reduced cost is proposed through this study. Its architecture combines a feasibility-sensitive admission controller, prefill and decode sensitive micro-batching, KV-cache paging with watermarks, warm pools to eliminate cold starts, and autoscaling based upon queue- and token-level cues; placement encompasses full-GPU, MIG, and MPS modes using per-tenant policies. Deployed on Kubernetes on 7B-70B decoder-only models using vLLM/TensorRT-LLM/Triton backends, the system is aimed at heterogeneous H100/A100/L4 fleets and chat/RAG workloads with heavy-tailed token lengths. In experimentation, the strategy advanced cluster throughput 31.7 percent more than the finest baseline, decreased P95 TTFT to 420 ms and P99 to 1.3 s, increased SM and memory-bandwidth utilization, and lessened cost per one million output tokens by 26.8 percent, while offering a similar degree of per-tier fairness on the same basis. This study provides contributions, including a production-ready control/data-plane design, SLO-aware admission tests, degradation, and routing, token-aware batching, and KV-cache usage/freezing to avoid memory-driven stalls, an easily reproducible evaluation recipe with KPIs (TTFT, P95/P99 latency, tokens/s, utilization, and $/1M tokens). Such findings introduce a scalable deployment avenue to foreseeable latency and efficiency. The blueprint reflects the reality of the

operators today.

Keywords: *Cloud-native LLM serving, Serverless GPU orchestration, Token-aware scheduling, SLO-aware admission control, KV-cache management*

1. Introduction

Large language models offered today are consumed as a service, demand for which is spiky and heterogeneous. Arrival rate can be massively multiplied due to product launch, seasonality, and social amplification, which generates queue and tail latency. The requests range widely due to the number of input prompts and generated output both being given in tokens; heavy-tailed distributions produce a lot of variation in compute time and memory footprint. Inference is also a two-stage prefill, which computes the prompt to initialize the key-value cache, and decode, which spits out tokens reusing such a cache. The prefill cost is superliner in the length of the prompt owing to attention, but the decode cost is approximately linear in the produced tokens. High-cost GPUs have to be utilized effectively, and numerous applications are multi-tenant, with various SLOs, budgets, and responsive profiles. The solution has to provide a low cost per million tokens as well as a low time-to-first-token.

Head ratio bad blocking, with long prompts dominating (short request) prefill, causing delays on short requests, while bloating tails. Dynamic batching is beneficial for throughput; however, TTFT may suffer as shorts are waiting behind more extended sequences. Serverless platforms now add cold start overheads: the container pulls and loads to disk, model weights on the remote object store to local NVMe and on GPU HBM, runtimes are initialized, CUDA graphs are captured/kernels are compiled. The end-to-end results are added tens of seconds unless warm pools and snapshotting are used. Heterogeneous accelerators make placement and fairness complicated: A100, H100, and L4 have varying memory size, capacity, and tensor-core throughput, so a one-size-fits-all scheduler underperforms. NVIDIA Multi-Instance GPU (MIG) offers the ability to carve out predictable slices at a high level of isolation, but has the risk of having stranded capacity. In contrast, CUDA MPS allows fine-grained sharing and increased concurrency at the cost of weaker isolation and the possibility of cross-interference. The key cloud native serving challenge is balancing such trade-offs with multi-tenant SLOs.

This study claims that token-aware GPU scheduling, combined with serverless GPU orchestration, can achieve lower latency, greater utilization, and reduced costs at scale. The author of this study present an architecture where a token-aware router, a goal-aware admission controller, and a micro-batching scheduler with signal-based autoscaling are jointly configured. Short and long requests are routed, the length of context and output is capped per tier, and the size is matched to the choice of GPU or MIG slices. The contributions of the study include: (1) an end-to-end sketch of Kubernetes infrastructure, including device plugins, node pools, and model servers; (2) node admission with SLO-awareness, computing per-request deadlines by summing TTFT and end-to-end budgets; (3) KV-cache-taming scheduling and memory methods that maintain throughput, and (4) a benchmark recipe with KPIs: TTFT, TLET, P95/P99 latencies, utilization, and cost per million tokens spent. The author also provides repeatable configuration patterns that can be modified promptly by operators.

The author's focus was a decoder-only LLM in inference in the range of 7B to 70B parameters, single region, single cloud, through the use of Kubernetes, providing a control plane. The orchestration logic is implementation agnostic, with models being served with commodity engines like vLLM, TensorRT-LLM, or Triton. Key definitions: Time-to-first-token (TTFT) is the duration between receiving the request and the first streamed token; time-between-tokens (TBT) is the delay between any two tokens when you decode; time to last emitted token (TLET) counts completion. Throughput is tokens/second/GPU or tokens/second/cluster. Utilization is the ratio of the peak token-generating capacity utilized, with consideration to idle and cold start times. Works consist of interactive chat, retrieval enhanced generation (RAG) over long contexts, and batched with bound outputs. The study leave training, alignment, and multi-region routing out. It limit context and output lengths to defend memory and SLOs where needed.

Chapter 2 surveys prior art in cloud-native inference stacks, serverless GPU orchestration, scheduling deep networks and LLMs, and KV-cache memory techniques, and points out these limitations, which motivate our design. Chapter 3 describes methods and techniques: the serverless GPU orchestrator, GPU provisioning and placement, token-aware scheduling and micro-batching,

KV-cache and memory management, and streaming networking. Chapter 4 advances the SLO-aware control plane, as a set of traffic modeling, queueing analysis, admission feasibility tests, autoscaling policies, and warm-pool sizing as a playbook that can be executed. Chapter 5 defines the experimentation facility, baselines, and metrics, and provides the results in bursty, multi-tenant workloads. Chapter 6 deals with trade-offs, operational issues, security, and observability; Chapter 7 covers future work; and Chapter 8 concludes. The readers will be able to airlift the configuration patterns, alerts, and offer a mirror of the habits in owning clusters. There are also limitations and assumptions.

2. Literature Review

2.1 Cloud Native Inference Stacks

It is most effective to engineer cloud-native large language model (LLM) inference as a three-tier stack composed of model servers, which are the numerical kernels, and serving layers that use replicas to coordinate traffic patterns, and edge gateways that provide application programming interface (API) and streaming semantics. The layering insulates issues so that each of the tiers can scale and fail in isolation (although end-to-end tracing is still supported). Tokens, the execution of attention and MLPs, handling of key-value (KV) cache by allocation and reuse, and transmission of tokens across long-lived connections are absorbed by model servers. They maintain budgets on the length of the context they receive, place constraints on the decoder, and the memory watermarks, to guard against out-of-memory errors. One such important role is throughput optimization across the user-visible service-level objectives (SLOs), specifically time-to-first-token (TTFT) and last-token. In that regard, modern servers provide dynamic or adaptive batching and token-level interleaving whereby multiple sequences proceed in lockstep during decode. They also expose tunables, maximum batch size, per-stage time budgets, and sets of decoding parameters that work with others, so that higher layers can affect the latency/throughput trade-offs under bursty demand.

The cloud-native LLM inference is implemented as a three-tier stack: the model servers run attention/MLP kernels, maintain KV-cache with memory watermarks, and export serving tunables to enable dynamic batching and token-level interleaving; the serving layer directs replicas and traffic; and the edge gateway provides APIs and streaming semantics, as shown in the figure below. This layering isolates failures, enables end-to-end tracing, and allows operators to tune maximum batch size, stage time budgets, and decoder constraints to optimize trade-offs between time-to-first-token and throughput in the face of bursty workloads as the system scales elastically on the container-native platform.

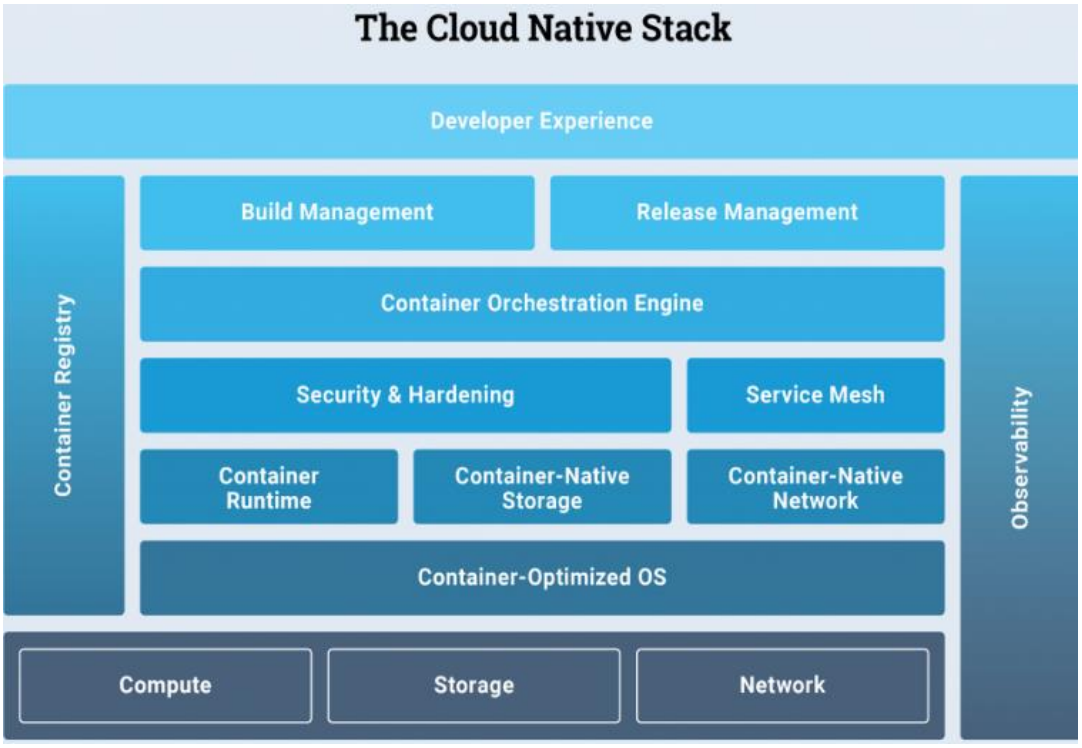

Figure 1: Cloud-native inference stack: model servers, serving layer, edge gateways

What can be safely merged is under the control of compatibility awareness batching. Requests on different

models or quantization modes cannot share a batch. Tensor shapes even within a single model must match, and the total KV cache space must be within device memory, including runtime overheads. The performance profile of prefill and decode is different, and therefore, servers differentiate them. Prefill consists mainly of large matrix multiplications, which cost superlinearly in input length when factored by quadratic attention; decode steps one token at a time and are more or less linear in the number of tokens generated, although again length affects attention cost. Such systems thus have stage-sensitive batching windows: a small time budget in prefill to defend TTFT, and a wider window in decode to increase tokens-per-second due to interleaving. Separation of stages forms the basis of token-aware scheduling in higher layers since it helps identify the stage where latency is most sensitive to request size.

Layers above kernel execution serve as the multi-tenant control plane. They redirect requests to model versions, apply per-tenant quotas, and enact admission-decision policies; these take into consideration backlog and projected service time and SLO budgets. This is the natural place of token-sensitive logic: limits on maximum new tokens, ceilings on context length, separate short queue and long queue, and ranges of tokens in batch separands (batched as time budgets rather than set limits). Traffic shaping, such as canary or shadow rollouts, compatibility checks to help with batch construction, exposing signals, like queue depth, wait-time percentiles, and tokens in flight, can also be orchestrated by the serving layer that can feed into autoscalers. Central location of such decisions can then be used to provide uniform policy within replicas, and a pattern can be created of joint admission and autoscaling that causes the cluster to respond to imminent SLO violations as opposed to remedial action on later graded utilization measures.

The client-to-cluster connection is made complete with gateways. High-performance gateways accept TLS, and apply HTTP/2 or HTTP/1.1, including server-sent events, and translate public REST requests to internal gRPC requests with flow control. They handle both authentication and rate limits, request coalescing, and streaming flush intervals, which also have a direct impact on perceived interactivity (19). When downstream queues increase, gateways will use backpressure and then spread the cancellation so that once requests have been abandoned, they do not still consume GPU cycles. The semantics of transport cannot be changed with one another. Server-sent events (SSE) can be easily deployed behind commodity load balancers and are well-suited for one-way token streams. The multiplexing and flow control provided by gRPC limit the level of head-of-line blocking in a concurrent stream environment, but they frequently need special proxies. WebSockets give a two-way control channel appropriate for cancellation and parameter updates. Cancellation is required to be first-class in any case, or autoscalers and schedulers will misinterpret work backlogs.

Observability links the tiers with each other. The request metadata (tenant, model, context length, and decoding parameters) is recorded in a structured log. Distributed tracing associates gateway spans, serving-layer decisions, and model-server stages to allow operators to discriminate network wait, prefill compute, and decode interleaving. Prompts are hashed and lengths bucketed, except for logging verbatim to prevent excessive cardinality. To a certain extent, the metrics are arranged in four axes: traffic (arrival rate and mix), scheduling (queue depth, batch composition, wait time), runtime (TTFT and time between tokens, completion latency), and resources (GPU utilization subscale, VRAM free pages, spill bandwidth). Other latency-sensitive communication systems offer practices that can be applied wholesale in LLM gateways and serving layers that need to prevent accelerator overload without sacrificing interactive experience: fault-tolerant routing, idempotent operations, durable queues, and disciplined backpressure (28).

2.2 Serverless & GPU Orchestration Patterns

GPUs on Kubernetes are exposed through device plugins, which help advertise accelerator properties and, in supported cases, semi-permanent fractionalization like NVIDIA Multi-Instance GPU (MIG). Clusters are divided into a node pool by accelerator family (H100, A100, L4) and by price class (on-demand versus spot/preemptible). There are three typical feedback loops in autoscaling. A Horizontal Pod Autoscaler (HPA) scales replicas based on autoscaler-specific metrics like queueing delay or predicted TTFT. Kubernetes Event-Driven Autoscaler (KEDA) is an automation tool that is used to scale on backlog in event sources such as message queues that queue requests. A serverless layer like Knative controls scale-to-zero and

activation. Every loop has a purpose: KEDA responds to backlog, HPA keeps a steady state under latency constraints, and Knative eliminates idle capacity without jeopardizing readiness.

The main key practical issue in serverless GPU serving is cold-start management. Image builds have to be compact with the right CUDA stack and model-server runtime situated in them. To prevent object-store re-downloads, operators pre-stage the weights of their models to local NVMe storage volumes, and startup procedures cache CUDA graphs or construct kernels to ensure that subsequent operator requests do not take the time to compile their programs. A limited, warm pool of pre-initialized pods decreases the occurrence and impact of cold starts at the cost of uniform cost in favor of tail predictability. Policy rules the creation, refreshment, and eviction of warm pods, with standard rules to maintain a base during peak times and reduce the pool at night, and grace periods to avoid trash. The determining policy of placement offsets usage with interference (14). MIG partitions provide high levels of isolation and predictable performance due to their dedicated service to strict SLO tiers. However, fixed slicing may leave capacity stranded with a change in request mix. It uses CUDA Multi-Process Service (MPS) to support fine-grained concurrency on full GPUs at the cost of less stringent isolation, so it is comparable to bronze or silver.

The use of autoscaling policies is advantageous, as there are various signals beyond raw utilization. Latency in queues and the forecasted TTFT respond immediately to user experience, whereas tokens in aviation and the forecasted prefill FLOPs act almost as a proxy of the work backlog in the graphics processor. HPA can use custom metrics over a Prometheus adapter and can scale using queue depth with KEDA. Scaling task to zero by Knative applies latency at the point of activation when the first request comes; mitigations include leaving a small number of pods running during high-demand moments, and finally, reserving a warm pool to store model weights in VRAM. There is additional confusion over spot capacity. Node taints redirect vital workloads to on-demand pools, and preemption hooks grant termination inspection opportunities to ensure in-flight batches complete cleanly. Where possible, drains should checkpoint state to local NVMe, and, in such cases, perform topology-aware rescheduling that retains NVMe affinity and large bandwidth links. The signals provided by health include error rates, ECC, and PCIe retraining event trigger, which proactively cordons off before regressions emerge. Last but not least is the fact that pipeline hygiene is a must. By adding static and dynamic security analysis to CI/CD, containers get hardened, a dependency tree is validated, and promotion gates to serve the images and model artifacts are established to be auditable (15).

2.3 Scheduling for DNN/LLM Inference

Efficiency and tail latency are highly affected by the scheduling policy. First-Come-First-Served (FCFS) is simple, but has the head-of-line blocking problem with long queues followed by short queues. Shortest-Job-First (SJF) or preemptive Shortest-Remaining-Processing-Time (SRPT) produces a more negligible mean and tail latency by ordering requests based on small predicted service time, where the size is estimated as input tokens plus predicted output tokens. Earliest-Deadline-First (EDF) requests are ordered by deadlines based on SLOs, and admission discards or degrades requests that cannot satisfy TTFT or end-to-end budget. Interleaving at the token level and dynamic micro-batching boost throughput, but the delay imposed on the formation of a batch needs to be limited so as not to jeopardize TTFT (1). Such evidence can be considered in terms of related areas of optimization practice: algorithm-based dispatching with clear objectives and constraint management outperforms heuristic rules in stochastic, bursty demand, an analogy that speaks to principled size and deadline-based scheduling of LLM inference (22).

2.4 Memory & KV Cache Management

Memory dictates the number of sequences that can concurrently operate on a GPU and thus becomes a limiting factor on throughput decoder-only models. KV cache scales in size with layers, number of attention heads, head dimension, as well as total tokens; it is also common to hit the memory ceiling earlier in terms of compute saturation. Page-based KV allocators split the cache into fixed-size pages whose existence is managed per sequence and layer, providing constant-time allocation and reclamation and minimizing external fragmentation. To avoid the pathological overcommitment, telemetry reveals that the free-page-based watermarks and admission throttles are employed to ensure that a process cannot exceed the capacity. Fragmentation management is not the responsibility of the allocator. Since the lengths of

sequences can vary, padding is often necessary to match the tensors in a batch; gathering requests that share some length in common can minimize the amount of wasted memory and computation. The frequent compaction that results in a temporary halt undertaken in low load windows allows a favorable contiguity and the subsequent batch density. Multi-tenant serving brings adapter management: popular adapters are pinned, while others are loaded on demand to save VRAM. Spill strategies impose flexibility upon the memory breach. The cold KV pages are migrated to CPU memory and, in extreme cases, to local NVMe through batched DMA. Hot pages spilling cause prefill performance to drop precipitously. Thus, systems select conservative thresholds, retain early-layer pages as they are visited at each decode level, and favor NUMA-local spill destinations. Pseudo-environments, Such as quantization policies, alter static footprint and dynamic KV utilization. Quantization Weight only (8- or 4-bit) decreases the representation size and allows for more suitable batch sizes on memory-bound GPUs. KV quantization also decreases per-token memory consumption measurably, allowing more simultaneous sequences to be active, but will require attention-minimizing dequantization overhead to prevent adding compute stalls in the kernels. Throughout memory Robust accounting Memory is closely linked with orchestration: placement takes into account base weights and runtime overhead, as well as expected KV growth based on batch size and percentile context length; autoscalers monitor spill rates and watermarks as leading indicators; and brownout policies can limit the maximum tokens delivered or refuse long contexts to conserve SLOs during crises.

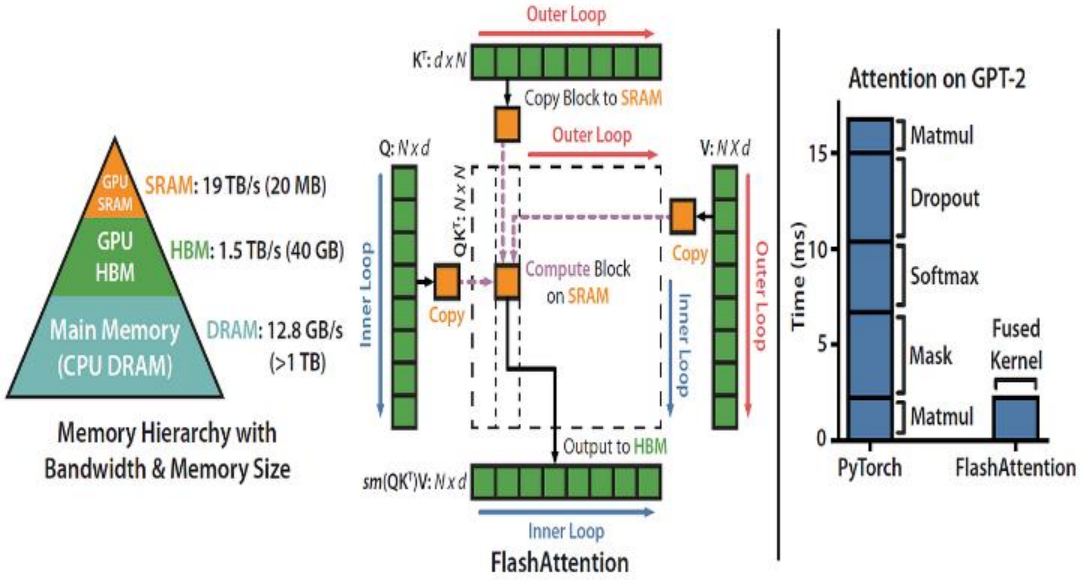

Figure 2: Memory hierarchy and KV-cache paging for scalable LLM inference

The serving performance of LLM is determined by attention performance and the hierarchy of memory. GPU SRAM/HBM and CPU DRAM bandwidths constrain the density of the batch, and FlashAttention minimizes kernel time, as evident in Figure 2 above. KV-cache pages are claimed and discarded promptly, and free-page watermarks gate access to prevent overcommit. Cold pages are subject to spill to host or NVMe through batched DMA, hot early-layer pages remaining in HBM to decode. Quantization decreases the weights and KV and multiplies concurrent sequences. Orchestrators monitor spill rates and enforce context/output rate limits to keep TTFT SLOs on load.

2.5 Gaps & Opportunities

There is a gap across stacks and orchestration patterns. A small number of deployments provide a single point of identity for serverless-compatible deployments, a token that is aware of the control plane based on SLOs (9). Model servers are very efficient numerically, but are unaware of cluster-level cold starts and delays in provisioning. Serverless platforms offer elasticity, but pods are seen as opaque boxes with little visibility into the behavior of tokens. This leads to a split-brain system, where autoscalers respond to trailing indicators and where schedulers take local decisions without a global view. Another opportunity comes in the form of evaluation practice. Benchmarks tend to focus on average throughput, and not the cost of achieving that rate, the rate of cold starts, tail behavior under diurnal

bursts, and recovery on node drains. Realistic traffic should be injected and tenants with divergent SLOs should be mixed, timelines at the token level measured, and (P50/P95/P99 reported on the network wait, prefill, TTFT, inter-token wait times, and completion. Cost must be brought into standard units of dollars per million tokens and broken out into steady compute, warm-pool overhead, and scaling waste. Microservices literature also warns against the limitless scalability independent of budget control that is not a strategy (37). Integrating cost-sensitive control devices, cost-focused autoscaling goals set in dollars per million tokens, SLO levels allocated to price buckets, and feedback that leaves marginal dollars per token revealed to service owners and reduces wastage during bursty scale-outs (5).

3. Methods and Techniques

3.1 System Architecture: Serverless GPU Orchestrator

The serving system is designed with a cloud-native control plane and a high-throughput data plane. An API gateway at the edge terminates TLS, authenticates a caller, standardizes headers, and assigns a tenant handle. A request classifier next takes low-cost features--the model identifier, input tokens observed, maximum new tokens requested, decoding parameters, and tenant class--and passes them to a token-sensitive router, which calculates an approximate size estimate and adds scheduling hints propagated by the request end-to-end. The router hence does initial triage by putting requests into short or long classes and makes a candidate GPU tier decision that is based upon projected VRAM pressure and time-to-first-token (TTFT). A feasibility-checking admission controller assesses the possibility that the request may satisfy its SLOs on the selected tier; infeasible requests are degraded, redirected to a higher tier, or rejected with a recommended retry stance.

The control plane encompasses a policy engine, an autoscaler, and a warm-pool manager. The policy engine assembles per-tenant settings into the execution-time rules context ceilings, output caps, maximum concurrency, and priority mapping (30).The autoscaler tracks queue-time percentiles, forecasted service load, token throughput, GPU usage, and free-page watermarks. When the backlog poses a threat to TTFT, it can add capacity by activating prewarmed pods or expanding the node group, and when demand drops, it can consolidate load and drain excess pods. A warm-pool manager maintains a desired number of pods per model hot, with local NVMe with weights, pinned in the device memory, CUDA graphs on-disk, and tokenizer caches pre-warmed. Ready gates require that traffic can only be accepted by warm pods when graph capture is successful.

Data plane will consist of model pods that are connected to a complete GPU or MIG slice through device plugins. Every pod is configured to support a streaming model server, a tokenizer library, and a lightweight sidecar that exports custom metrics, like decode cadence, tokens per second, free KV pages, admission decisions, and cold-start events. Concurrency limits are visible to the scheduler, so micro-batching is aware of TTFT budgets and so avoids over-admission choices leading to memory consumption of KVs. Expect pods to take messages controlling draining, loading of an adapter, and also compaction windows; draining happens at token boundaries, so active sequences do not lose their state. The weight of models is resident, and KV cache pages are reserved only when needed.

The loop is completed with observability and operability. The stack reveals TTFT, TLET, P95/P99 latency, GPU usage, VRAM load, queueing depth, and rejection cause. Distributed traces align gateway spans, control-plane decisions, and pod execution such that each source can be composited on a scheduling decision. There is a clean bounded context between routing, admission, scheduling, and autoscaling, which reduces coupling and clean division of responsibility, which simplifies evolution and prevents large blast radius; this is identified as a separation of concerns in microservices literature (4).

3.2 GPU Provisioning & Placement

VRAM accounting is the initiation of provisioning. In decoder-only inference, the resident set supports base model weights, run-time workspaces, and the per-token key-value (KV) cache. Weights are assigned per-model fixedly and pinned once per-pod; KV cache can scale linearly to the concurrency and the context length, and hence is very dominant in its variability. Practical sizing rule provides KV budget with the calculation as KV bytes the approximate value of B, the number of active sequences in the micro-batch, L, the effective context length (prompt and the part of generated tokens that is retained), H, the hidden size, dtype factor is the number of bytes per element (2 FP16, 1 when INT8/FP8 is supported), and layers is number of transformer blocks.

Heads factor reflects both key and value tensors and head partitioning. Operators are stuck with a percentage of headroom to cushion bursty arrivals, and capped L per tenant to clear KV footprint, which is predictable. Since paging between device memory and host is costly, pages are allocated by the allocator using preallocated arenas instead of utilizing general-purpose malloc.

Placement has to assign pods to accelerators so that no SLOs are missed and capacity is not stranded. NVIDIA Multi-Instance GPU (MIG) allows predictable slicing (such as 1g.10 GB, 2g.20 GB, 3g.40 GB) into isolation with deterministic VRAM ceilings on H100 nodes or an A100 node. Larger slices can be scheduled to high-priority tenants to reduce the chance of contention, whereas the cost-sensitive tiers can share smaller slices. MIG minimizes noisy-neighbor effects and can increase internal fragmentation when workload shapes change, so the scheduler monitors free pages and could rebalance hot partitions. Another concurrency mechanism is CUDA Multi-Process Service (MPS), which can enable multiple processes to share a full GPU (20). MPS may achieve greater throughput when mixed, short requests are made, but has less isolation; as such, MPS is conservative in its admission and denies memory over-commitment.

The topology-aware placement lessens the variance of latency. PCIe generation, NUMA domains, and NVLink connectivity are labeled at the node; the scheduler favors NVMe local to GPUs as weight snapshots and does not replicate tensor-parallel shards with their GPUs across NVLink islands. In case a model is sharded, the orchestrator co-schedules sibling pods on GPUs with direct NVLink to reduce inter-GPU traffic during prefill and decode. The controller complies with PodDisruptionBudgets and reserves spare capacity to support rolling updates to ensure availability. When cost optimization requires the use of spot instances, a drain controller reacts to preemption notices by flagging pods unschedulable, retries of requests still in-flight upstream, and checkpointing warm-pool state to local NVMe on the replacement node. Lastly, operators publish longer-lived resources through device plugins, Node Feature Discovery to annotate hardware capabilities, and set anti-affinity to ensure that all the replicas of an instance of the same model do not form a single failover point on a common host.

Table 1: GPU provisioning & placement: VRAM, KV-cache, MIG/MPS, topology-aware scheduling

Component / Decision Area	Key Purpose / Challenges	Practical Rules / Calculations	Operational Actions / Scheduler Behavior
VRAM accounting	Ensure pods fit GPU memory without SLO misses or stranded capacity	Resident set = base weights (pinned per pod) + runtime workspaces + KV cache	Track per-pod footprints; reserve headroom for bursts; deny overcommit
KV cache sizing	Dominant, variable memory driver in decoder-only inference	**KV bytes ≈ B × L × H × dtype × layers × heads×2** (keys & values); dtype: FP16≈2B, INT8/FP8≈1B	Cap **L** per tenant; forecast KV growth from batch B and percentile L
Memory allocator strategy	Avoid costly device↔host paging and fragmentation	Preallocated arenas; page-based KV allocator with constant-time alloc/free	Enforce free-page watermarks; admission throttles when below watermark
Headroom policy	Cushion bursty arrivals; keep latency stable	Maintain % free VRAM above predicted peak KV; fixed weight pinning	Reject or defer requests if headroom breached
MIG slicing	Predictable isolation and VRAM ceilings on A100/H100	Slice profiles (e.g., 1g.10GB, 2g.20GB, 3g.40GB) mapped to SLO tiers	Give larger slices to high-priority tenants; monitor free pages; rebalance hot partitions
MPS concurrency	Higher throughput for mixed short requests,	Share full GPU among processes; no memory over-commit	Conservative admission; deny when memory risk detected

Component / Decision Area	Key Purpose / Challenges	Practical Rules / Calculations	Operational Actions / Scheduler Behavior
	weaker isolation		
Placement objective	Avoid SLO violations and stranded capacity	Match pod footprint + KV growth to GPU/MIG capacity	Bin-pack by VRAM; route shapes to suitable tiers (full GPU vs slice)
Topology awareness	Reduce latency variance and inter-GPU traffic	Use node labels: PCIe gen, NUMA, NVLink topology, local NVMe	Favor NVMe local to GPUs for weight snapshots; co-schedule sharded pods on same NVLink island
Sharded models	Minimize cross-link chatter during prefill/decode	Keep tensor-parallel siblings topologically close	Anti-affinity across hosts for replicas; NVLink-aware co-scheduling
Availability during updates	Maintain capacity while rolling	Respect PodDisruptionBudgets; reserve spare nodes/slices	Staged rollouts; keep warm capacity during updates
Spot instances	Control preemption risk while cutting cost	Use spot for elastic capacity, on-demand for critical	Drain on preemption notice; mark pods unschedulable; retry upstream; checkpoint warm-pool state to local NVMe
Hardware discovery	Publish capabilities for accurate scheduling	Device plugins; Node Feature Discovery annotations	Scheduler filters/affinity rules by capability set
Anti-affinity / blast radius	Avoid single-host failure for a model's replicas	Spread replicas across hosts/NUMA domains	Enforce anti-affinity; maintain spare capacity for failover

3.3 Token-Aware Scheduling & Micro Batching

Scheduling begins with an estimate of the size \hat{n} = n_in + E[n_out], where n_in is the observed input length and E[n_out] is the expected number of tokens generated conditional on the prompt branch and stop conditions and tenant defaults. This estimate is consumed by a service-time model \hat{s} = a · nin^omega + b · nout · f(ctx). The first one approximates prefill cost; 1 > /alpha/ reflects the fact that the benefit of attention increases super-linearly with context size. The second term is a model of decode, in which the per-token latency grows as a factor of nout but also grows with context by a factor of f(ctx). Coefficients a and b will be trained based on offline profiling per model, and per GPU tier, and should be periodically refreshed, as with production telemetry.The router puts requests on short or long queues and calculates priority proportional to weight/\hat{s} under a weighted shortest-remaining-processing-time (SRPT) policy. A local scheduler running on each GPU slice or MIG slice consumes several tokens out of these queues into a token-aware micro-batcher. In prefill, the batcher restricts the number of long prompts it will accept simultaneously to keep TTFT within budget; during decode, it interleaves as many short sequences as possible to maximize tokens per second. A batch is limited by two constraints: the estimated prefill time cannot be greater than the TTFT budget of the median request in the batch, and the expected KV reservation after admission should be kept above a free-page watermark. As the queue length increases, the scheduler will reserve a portion of slots for the long jobs to prevent starvation.

Time is closely linked to admission and scheduling. Each queue keeps an estimated completion time based on the current decode cadence and the estimated prefill of running jobs. A request is accepted when predicted waiting time + the prefill is below the TTFT budget. A degradation ladder is used: lower max_new_tokens on bronze tiers, prioritize gold, route

to faster GPU tier, or send a fast failure with retry hints. Preemption is done on the boundaries of tokens to prevent corruption of state; the batcher defers to tenants with higher priorities when their TTFT would have been lost. Deficit round-robin imposes fairness across moving windows of the number of tokens emitted per tenant, so allocation is based on GPU work instead of the request counts (18).

Both memory and time prediction are used when selecting the GPU tier. Suppose \hat{s} is known, and at an instance the KV estimate is known, then the mapping will choose the cheapest tier in which the TTFT and end-to-end latency meet the SLO of the tenant, and its free pages after the admission meet a low watermark. The scheduler decouples batches when head-of-line blocking is identified, and this action occurs when the longest-to-median ratio of n in a pending batch is too high relative to a configured parameter.

3.4 KV Cache & Memory Techniques

Consistency in throughput is determined by the well-managed key-value (KV) cache. At startup, each pod configures fixed-size page arenas of keys and values that are aligned with kernel preferences and the memory granularity (24). Per-layer free lists support constant-time allocation, and this is combined with allocating on uncertain bursts, which limits contention during bursts. The allocator allocates pages during admission to safeguard the prompt and an over-approximation of the output tokens issued; during decode, to resolve spikes, it can allocate more pages in small units. They reference count pages based on the sequence of pages, so they can be released quickly on cancel or completion.

There is early-warning control through watermarks. Healthy headroom is a high watermark; a low watermark causes admission throttles, or activates reserve MIG slices, or causes the autoscaler to add warm capacity. Absolute free pages and consumption slope are exported to enable the control plane to know when it is going to be depleted, so that stalls are avoided. Eviction is event-based; background sweeps combine small free runs into bigger ones. The windows are bounded, and the compaction is synchronized with human lulls in the decode cadence so as not to introduce noticeable delays.

Memory is also taken up by multi-tenant adapter management. Once tenants bind LoRA adapters, popular adapters have the pin assigned on the GPU, and less popular adapters are kept in page-locked host memory that can quickly do DMA. A lightweight adapter router favors pods that have the requested adapter in them to reduce the penalty of carrying the extra load. Quantization is capacity: Weight only 4-bit or 8-bit schemes reduce the size of the static model, and KV-cache quantization to 8-bit halves the cache size with a negligible effect on the quality of many workloads; per tenant operators can gate this.

This dynamic process is supported by dynamic memory concepts of neural inference studies to the extent that the pathways that choose what to keep in memory and what to purge minimize interference and enhance an adaptive memory capacity under different demands (25). The serving allocator is not a trained entity, but the principle is the same: manage memory as an allocation in pages and track reservations as evidence is discovered during decode. Admission throttles decline or postpone requests that, once reserved, would violate a watermark or a per-tenant memory budget. Drain endpoints freeze new grants and wait until a series reaches a token limit, and thereafter, compaction before restart.

3.5 Networking, I/O, and streaming

Networking has to conserve the token rhythm generated by the scheduler. The serving layer exposes gRPC over HTTP/2 with server-streaming RPCs to allow tokens to be streamed immediately they are produced; native flow control and per-stream deadlines give backpressure and safeguard the cluster against straggling clients. To consume via browsers, an edge gateway translates gRPC stream to Server-Sent Events (SSE) and maintains backpressure, including buffering quotas and early cancellation. Keepalive pings and idle timeouts halt resource waste, and idempotency keys make retries seamless following network hiccups and non-repetitive.

Tokenizer locality prevents a hidden hotspot of the CPU. Pods preload vocabulary files, tokenizers, and a cache of standard system prompts within the process. Requests will include a tokenizer class hint so the router can direct compatible prompts onto the same shard, thus minimizing cache churn. With large documents, zero-copy ingestion paths can transfer the data between the NIC and the user space to the GPU staging buffers without copying along the way.

Chunking and flush intervals are adjusted to the time between tokens profile. Both too-frequent and too-

infrequent flushing add overhead; the former damages perceived interactivity. One token per flush or 20 50 ms cadence, whichever is longer, is targeted by a pragmatic policy and coalescing of small chunks under bursts. Compression depends on the level of token rate and CPU headroom; to avoid head-of-line blocking, raw frames are desirable in high-throughput decode mode. At the edge, request coalescing eliminates both the same prompts at thundering-herd arrival times, and a rendezvous batcher, a program to assemble compliant arrivals in a brief interval. Per-route rate limits and tenancy quotas constrained the best-case load. Worst-case load, and transport security was enforced by mutual TLS and scoped tokens with minimal overhead on contemporary hardware (35). Network telemetry: per-stream TTFT, inter-token histograms, retry counts, and gRPC flow-control stalls provide backpressure measured at the edge, which is translated to a safe, token-aware scheduling decision at all levels of service within a cluster.

4. SLO Aware Admission Control, Autoscaling, and Queueing Analysis

4.1 SLO Definitions & Traffic Modeling

Cloud-native LLM serving has service-level objectives encoding two qualities visible to the users, responsiveness and completion. The time-to-first-token budget DTTFT limits how long it takes between the request arrival at the serving gateway and the first streamed token. The end-to-end budget, D E 2 E, is an upper bound of the delay between the same arrival instant and the last emitted token, often referred to as time-to-last-emitted token (TLET). As clients can begin to render once the first token has been received, DTTFT is the key factor affecting perceived quality; D E2E limits total throughput and exposure to cost. Practical SLOs give percentile targets (e.g., P95 TTFT <= 300 ms, P99 TTFT <= 600 ms; P95 TLET <= 4 s) with an error budget that can eat rare outliers. These budgets dTTFT = arrival + DTTFT and dE2E = arrival + DE2E are used to give per-request soft deadlines to the control plane: dTTFT and dE2E. Soft deadlines at admission time should degrade, redirect, or reject such requests early--either outright or with informative hints--rather than letting them wait in queues.

The SLO decision flow (Figure 3) instructs operators to select request-based SLOs when signals of a service are exposed at per-request granularity and the overall performance must be controlled adequately on a per-interaction basis (i.e., when deadlines apply, such as on the TTFT/TLET proposal), and to select window-based SLOs when signals are aggregated, traffic is low, and only aggregate behavior is relevant. This mapping is the admission control-soft deadlines mapping dTTFT and dE2E: per-request control degrades, redirects, or drops early; window control tracks percentile budgets over time windows to guarantee consistent latency and cost.

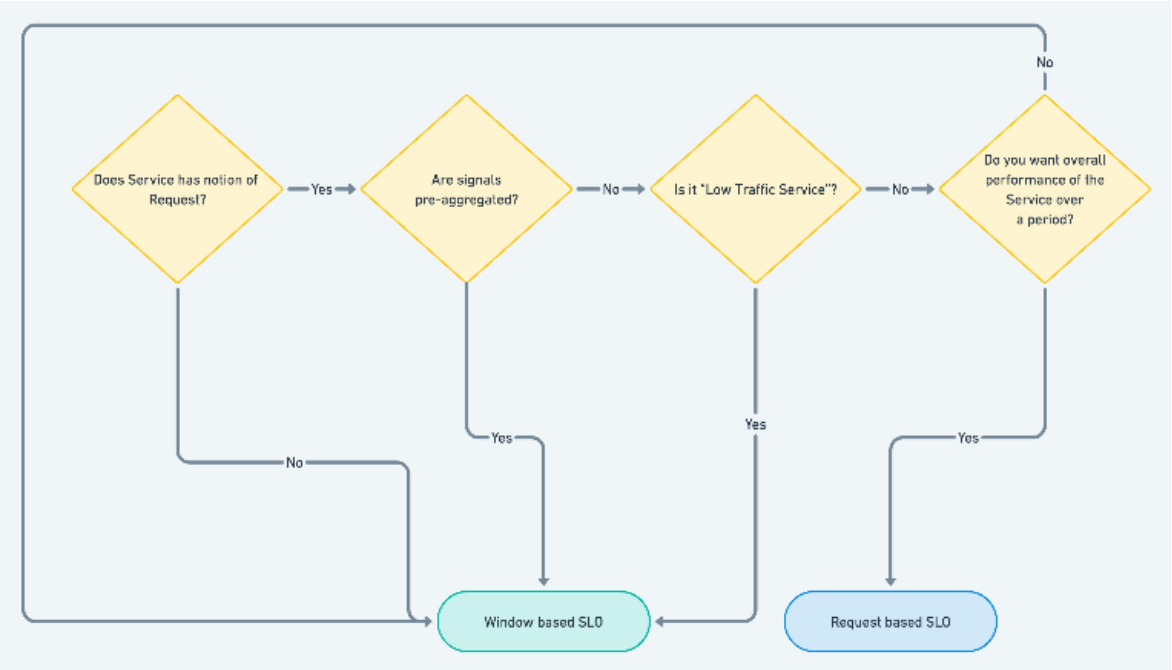

Figure 3: Choosing request- or window-based SLOs for TTFT/TLET control

The nonstationary arrival process 00t is bursty. Daily variations are interspersed by peaks at launches, external linking, or incident diversion, and brief bursts of 5-20-fold are typical in production systems. Request sizes are also not homogeneous since the cost of the service consumed is proportional to the number of prompt input tokens and the number of output tokens produced. Prefill compute grows superlinearly in input length (compute grows quadratically with sequence length unless optimized), and decode grows linearly in output tokens due to reuse of the key-value (KV) cache over steps. As a result, two requests for the exact arrival time may require vastly dissimilar GPU time and memory; head-of-line blocking will occur if they are handled in the same way.

SLOs should also reflect multi-tenancy. Tiers combine budgets numerically, policy limitations, and classes of resources. A gold tier could impose D_TTFT = 300 ms and D_E2E = 4 s, limit the length of context to preserve memory, limit the number of generated tokens, and tie requests to high-powered accelerator slices with limited per-pod concurrency. A bronze tier could enable D EE = 12 s, D EE = 1 s, support greater concurrency, and favour more cost-effective slices. Tier metadata also controls permitted decoding parameters, permitted model families, and overflow destinations (7). The routing, admission, and autoscaling access this metadata to balance per-tenant fairness and achieve cluster-level performance objectives.

The model has to have first-class memory pressure. The footprint of KV-cache grows with batch size, number of layers, hidden width, and precision; spills to CPU or NVMe incur stalls contributing to overestimates of service time and variance. The controller consequently makes the condition based on context length, anticipated output length, present batch size, and obtainable cache page. When the free pages are left unused past a watermark, admission will throttle long context primitively, and output limits are rigidized on the lower tiers. Such guidelines pair SLOs with resource health, inhibiting the occurrence of pathological tail behavior due to fragmentation or paging.

4.2 Queueing Models for Token-Aware Serving

Any accelerator or MIG slice could be modeled as a general arrival station with general service times (i.e., a queueing station). A multi-node station with a single GPU is a reasonably close analogy to G/G/1, with processor sharing (PS) during decode, and exclusive service during prefill; a pool of homogeneous nodes is G/G/k with bulk-service effects arising through the use of dynamic micro-batching. General distributions are intractable when it comes to exact analysis, although good approximations inform design. The formula by Kingman delivers an average queue in G/G/1 without vacations:

$$W_q \approx [\rho/(1-\rho)] \times [(c_a^2 + c_s^2)/2] \times E[S],$$

where $\rho = \lambda \cdot E[S]$ is utilization, c_a is the coefficient of variation (CV) of interarrival times, and c_s is the CV of service times. Three operational levers follow. First, keep ρ bounded away from one; as $\rho \to 1$, the prefactor $\rho/(1-\rho)$ explodes and W_q grows superlinearly. Second, minimize variation; heavy-tail service (c>>1) and burst arrival-to-service (c a >1) compound waiting. Third, minimise E[S] through batching and optimised kernels. These levers provide an incentive to token-aware routing (to lower the effective cs), short/long queues (to reduce interference), and admission throttles (to stabilize at ρ).

Serverless execution adds cold starts, similar to server vacations, where no service is provisioned. Vacations extend adequate service time and result in more waiting, especially when the demand is high. Practically, some cold-start components are image fetch, initialization of device plug-in, transfer of weights to local NVMe and HBM subsequently, and run-time compilation in conjunction with CUDA graph grasp. Warm pools eliminate vacations by maintaining a small pool of hot replicas; snapshotting and layer caching minimise the rest of the startup path (38). The objective of the operations is to ensure that warm capacity is provided to reduce the likelihood of dispatching to a cold copy in bursts.

Micro-batching is provided by bulk service. The arrivals are prefilled in a batch and are arbitrated under PS to share the decode phase. Batch-formation time is the amount of delay required to build up the requests to achieve the target batch size B, with a timeout imposed to defend TTFT. In the bursty λ(t) regime, batches fill densely, and amortization will take over. Astake over. As traffic becomes sparser, the aggressive batch targets pay in terms of TTFT penalties. Practical heuristic: The heuristic commits at most some fixed percentage of D TTFT (40% is a familiar figure) to batch formation and prefill. The rest is left to any possible queueing

uncertainty and the initial decode. This is a back-solve on B at any traffic issue and model setup.

Basics of delay estimation are difficult in the decode-phase PS. As the B_active sequences decode in parallel, the effective service rate per sequence is the total service rate, 1/B_active, so the completion of decode time extends more or less proportionately with the number of sequences in B_active. In contrast, prefill is either not preemptive or loosely preemptive, since writing to the KV-cache has to be contiguous. To avoid head-of-line blocking, the scheduler interleaves very brief prefill windows between small prompts or distinguishes between jobs that prefill heavily and those that do not (12). Heterogeneous clusters also take advantage of cmu-style dispatch: use servers with priority weight c times effective service rate mu, and keep other things constant, 2U would be mu, which incorporates not only compute but also memory headroom (free KV pages) so that paging is avoided.

4.3 Admission Control Algorithms

Admission control also makes SLOs per request at the router. The system will calculate a predicted wait time on the target queue, $\hat{W}_q$, and an expected service time $\hat{s}$, for each arrival. The latter is decomposed into prefill and decode components: $\hat{s} = \hat{s}_{prefill} + \hat{s}_{decode}(max\_new\_tokens)$. To guard against overconfidence, predictions are not taken at their word and treated as high-quantile estimates (e.g., P95), with an uncertainty allowance in the form of a multiplicative guard factor $0 < 1$ injected after estimation. Two feasibility tests are as follows:

1. TTFT feasibility: $\hat{W}_q + \hat{s}_{prefill} \leq D_{TTFT}$.
2. End-to-end feasibility: $\hat{W}_q + \hat{s}_{prefill} + \hat{s}_{decode} \leq D_{E2E}$.

When TTFT is not achievable but D_E2E should be at a smaller output, the controller uses graceful degradation by reducing max new tokens, greedy decoding, or putting on early-exit policies. In cases that both constraints are unsatisfiable despite any feasible tier, the request is redirected to overflow capacity or rejected immediately with an organized reply that has retry hints. The estimation of the $\hat{W}_q$ should take into consideration the queue depth and batch dynamics. A virtual-finish-time estimator is the sum of the remaining service of jobs in front, divided by the effective service rate two mu eff, plus the expected batch-formation delay. Concretely, $\hat{W}_q \approx \alpha \cdot (queue\_depth/\mu\_eff) + \beta \cdot batch\_fill\_time$, where α and β are fitted from telemetry, μ_eff reflects concurrent decode sequences, and batch_fill_time is the minimum of the latency buffer permitted by D_TTFT and the time to fill the next batch given current arrivals. Since the state is dynamic, the estimate is updated at dispatch; when it rises above a limit before a job enters service, opportunistic rerouting to an under-loaded tier is permitted.

The priority achieves a balance between efficiency and equity. Weighted shortest-remaining-processing-time (w-SRPT) gives priority to small 0 in addition to applying weights to the tenants in agreement with the SLO tiers. Preemption is allowed at token boundaries so that a short job may start decoding in a short time without losing work on a long career. The scheduler limits the number of preemptions on a request to prevent thrashing and imposes a minimum token quanta. The reason prefill preemption is not expected is that resuming in the middle of a serving stack is not supported or cheap due to emergency pressure (6). Operationally, the admission channel produces counters of admits, degradations, reroutes, rejects with reasons; the admissions signal steers post-incident analysis procedures and chromatic winds γ and the selected quantile p.

4.4 Autoscaling Coupling & Warm Pool Sizing

Autoscaling pairs admission to capacity in a way that delays when bursts come up. The target set point of the control is to maintain utilization zero within a range balancing utilization and tail latency (say 0.5-0.7 during peak), keep D_TTFT and D_E2E targets, and reduce cold start. Backlog-based delay and token-production pressure are the two categories of signals that are most reliable. The former employs queue_latency_p95 or W q percentiles; when they surpass any fraction 0 of D TTFT over a long continuous window, zero scale-out ensues. The second uses tokens_in_flight or prefill FLOPs/s to indicate near-real-time pressure on decoding and prefill pipelines, respectively.

In Kubernetes, Horizontal Pod Autoscaler v2 is capable of scaling up against custom metrics tracked by Prometheus, which can also dynamically scale in many ways, quickly responding to events, with KEDA. Policies are a combination of proactive and reactive sides. Proactive scale-out pre-warms pods in advance of predictable bursts and forecasts scale-out based on past traces, including workload mix. Reactive scale-out reacts

to an increase in queue delay and cold-start occurrence. Scale-in is conservative: It needs small values of backlog, tokens_in_flight, and consistent percentiles of TTFT across multiple windows before replicas can be deactivated. Heterogeneous Clusters require cost-sensitive placement: an estimate of marginal SLO increase/dollar in each node group or MIG profile, and add capacity where the benefits are the greatest on a marginal-cost basis.

Warm pools do away with cold starts and vacations. A warm replica is one with cached image layers, loaded model weights, compiled runtime, and CUDA graphs. The minimum warm pool per tier follows $N_{warm} \geq \lceil \lambda_{burst} \cdot T_{cold} \rceil$, where λ_{burst} is the expected spike in arrivals over the warm-up horizon and T_{cold} is the cold-start duration. If the burst model is uncertain, λ_{burst} is inflated by a safety margin. Warm copies, warm replicas are distributed topology-spread along nodes and zones in such a way that the failure of one does not exhaust readiness. PodDisruptionBudgets and priority classes ensure that a maintenance or preemption event cannot evict warm replicas.

Scale-to-zero is still an appealing low-cost service for low-traffic services. A workable policy zero scales out the stateless gateway and keeps a nonzero warm pool of the model deployment. Idle eviction timers are used only against replica excess, and never allow the pool to go below N warm (8). Pods are drained whenever scale-in is required by halting admission traffic and allowing ongoing decodes to finish, ensuring KV-cache state and no partial outputs. Where the dynamic is complex, however, reinforcement-learning formulations complement the rules; analogous strategies have been applied to balance the greedy delay-minimizing traffic adaptive control tactics, pointing towards a route to a reward-based scaling without violation of safety constraints (33).

4.5 Implementation Playbook & Alerts

An applied playbook transforms the above processes into regulated practices. The first pillar is the metrics. TTFT and TLET histograms and percentiles, queue depth per tier, queue_latency_p95, tokens_in_flight, prefill FLOPs/s, GPU utilization, VRAM free-page watermark, achieved batch size versus target, cold-start counts, and admission action counters all should be exported by the serving stack as histograms and percentiles. Dashboards decouple interactivity (TTFT, batch-formation delay) and throughput (tokens per second, utilization) and memory health (free pages, fragmentation). The input and output token frequencies, context length, and decoding parameters, including realized TTFT, realized TLET, and per-stage timing records in request logs, enable post hoc analysis.

The serving pipeline, tokenization, prefill (initial prompt processing), token-by-token decode, and de-tokenization also determine what is instrumented to provide operations dashboards: TTFT and TLET histograms/percentiles, batch-formation delay, queue depth, and queue_latency_p95, tokens_in_flight and prefill FLOPs/s, GPU utilization, VRAM free-page watermarks, achieved and target batch size, the number of cold-starts, and admission actions. As shown in the figure below, per-request logs record input/output tokens counts, context length, and decoding attributes, and per-stage timings to support subsequent analysis, quasi-identifying interactivity and throughput- and memory-insolvency perspectives for credible alerting and capacity planning decisions.

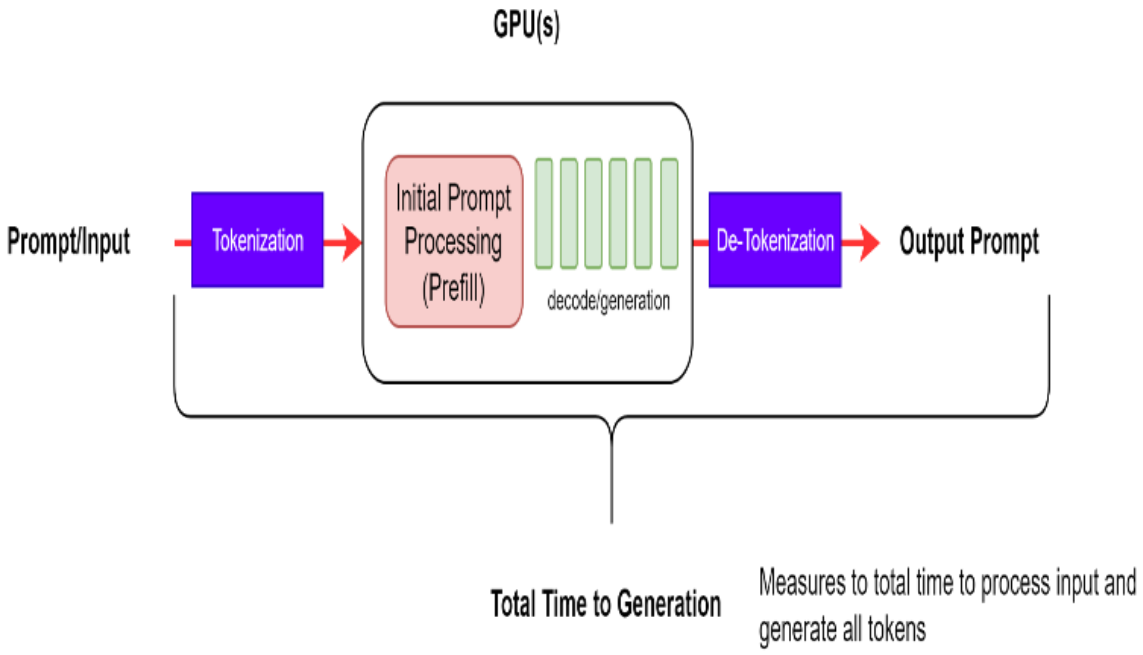

Figure 4: Tokenization, prefill, decode stages for TTFT/TLET instrumentation

The routing and admission are applied as a sidecar/gateway filter. The filter analyzes feasibility in terms of quantile estimates and guard factor gamma (P95 with a guard factor gamma of 1.25 gives a safe starting point) (10). Each tier has two priority queues to make a better separation: a short queue used when n̂ is small, a long queue when n h is large. Context ceilings prevent injustice and memory loss when combined with per-tenant concurrency limits. Infeasible requests are degraded by reducing max_new_tokens, switching to greedy decoding, or sending to a smaller, faster model compatible with the output quality. Every decision is justified using organized headers to help client-side retries.

SLOs directly derive caps about scheduler configuration. Max batch size is selected with expected batch-formation delay plus prefill time using less than 40 percent of D_TTFT at nominal lambda. Short prompts are safeguarded by restricting prefill windows to sit periodically (e.g., every k-th time slice). Thrashing can be preempted with token boundaries with per-request limits. Tokenizer workers are replaced alongside model pods to prevent CPU tokenization from being on the critical path. On heterogeneous clusters, the routing uses a CMU-style rule, though taking into account compute throughput as well as free KV-cache pages to avoid placing that would induce paging.

On composite triggers, autoscaling is aligned with HPA v2 or KEDA. One trigger forces queue_latency_p95 to be less than 0·D_TTFT; another forces tokens_in_flight to be close to a reference based on measured 0. minScale will be N_hot, and maxScale is constrained by the capacity of node-pools. Concurrency knative container Concurrency is the concurrency per-pod cap found during load testing (31). Canary-based policy changes roll out, performing a comparison of TTFT and reject rates against a control; policy changes can be automatically rolled back on regression. Warm-pool management applies a pre-pulling of the images as well as layer caching combined with server placement of weights on local NVMe, along with readiness probes to capture CUDA graphs before a pod can join the pool. PodDisruptionBudgets guard a warm quorum of the replicas; topology spread restrictions maintain the replicas in a different rack and a different zone. Spot nodes only run noncritical surge capacity; termination handlers are drained and retain minimal state. This continuous assessment circles back: the results of post-incident reviews, metric audits, and user feedback are used to revise thresholds, guard factors, and routing policies, as with other systems where improvement based on the evidence is key (13).

Observability finishes off. Alerts trigger when queue_latency_p99 is greater than 0.7x D T TFT over 2 seconds, when VRAM free-page watermark is undershot a threshold, when cold-start counts rise, or when admission rejects overruns a budget. Runbooks contain specific prescriptions to fix concrete problems, like amplification of N_warm, context tightening of ceilings,

recalibration of batch caps, or a tenant sent to an overflow capacity. The rule of governance allows predictability: SLOs, caps, and routing rules become version-controlled configurations; modifications are vetted in staging and are rolled out to canary. Tenant dashboards with compliance against D_TTFT and D_E2E, cost per million tokens, and utilization summaries are presented to the stakeholders. Collectively, the SLO-aware control plane, rooted in admission, autoscaling, and queueing analysis, maintains tail latency bounded under bursty and heavy-tailed demand whilst maintaining GPU utilization and predictable cost.

5. Experiments and Results

5.1 Experimental Setup

The test was aimed at a production-like, multi-tenant cluster and reproducible software stack to ensure the results generalize to production. The control plane used Kubernetes v1.29 and Cluster Autoscaler, and node groups were categorized by accelerator tier. The fleet included four H100 80-GB nodes (NVLink) and four A100 80-GB nodes (PCIe) as well as six L4 24-GB nodes. They had each node expose the NVIDIA device plugin and time-sliced MPS to a 3.2-TB NVMe SSD as a weight snapshot and KV-cache paging. Nodes were provided with two 100-Gbps NICs for the east-west traffic and object-store ingress. Knative and KServe deployed a revised version of serve, which offered scale to zero and per-revision warm capacity. Isolation of GPUs was also implemented on A100 MIG (3g.40 GB and 7g.80 GB) and full-card placements. The node images were distroless and hosted by a regional registry to reduce cold-start variance.

It was based on the serving layer with vLLM and TensorRT-LLM backends with an Envoy gateway with gRPC and HTTP/2 streaming. Triton Inference Server was turned on to achieve a baseline of comparability and to examine the native dynamic batching behavior. Model weights were in an object store with lazily local to NVMe; a pre-warm controller guaranteed weight residency throughout the warm pool. Prometheus scraped* application, queue, and GPU metrics; DCGM exported SM, memory, and copy-engine utilization; structured logs were emitted per request arrival, admission, first token time, and completion. This end-to-end telemetry was a replica of asset-tracking discipline in other operational spaces where locational, state, and utilization traceability is the keystone to efficiency (21).

Two families of models, consisting of a decoder-only model, the 8-billion-parameter model, and the 70-billion-parameter model, were trained. The 8B model trained on L4 and A100 in FP16 and INT8; the 70B model trained as tensor-parallel (TP=4 on A100, TP=2 on H100) in FP8/FP16 mixed precision with a paged KV-cache allocator. 8B had context windows of 8k tokens and 70B had windows of 16k. A replay generator was used to generate Lennard-Jones distributed traffic by input lengths using chat Lennard-Jones distribution (P50=120, P95=480) and RAG Lennard-Jones distribution (P50=1,200, P95=3,500), and had a heavy-tailed capping tier limit: Gold 1,024, Silver 512, Bronze 256. Every experiment streamed coins.

5.2 Baselines, Ablations & Metrics

The impact of each of the mechanisms was isolated in four baselines. B0 operated with first-come-first-served and no warm pool, and had static micro-batches of four. B1 used dynamically length-oblivious batching with a capacity of eight. B2 disabled MIG and had to place the full GPU. B3 eliminated the use of image prewarming and weight snapshotting to make cold-start costs visible. The suggested system incorporated weighted shortest-remaining-processing-time with per-tenant priority classes, earliest-deadline-first on those requests with explicit deadlines, token-aware admission control, efficient pool sizing by burst arrival rate, and accelerator selection based on predicted service time and burst arrival rate.

The most relevant were time-to-first-token (TTFT), distributions of end-to-end latency (P50/P95/P99) and throughput, in tokens per second, GPU occupancy (SM and memory), waiting time at the gateway and backend, cold-start rate, and per-million-output-tokens cost. Queuing delay was calculated as the difference in time between when the server arrived and when it was admitted to a decoding batch (26). To indicate user-perceived responsiveness, the measurement of throughput was at the token stream, rather than request completion. Cost was calculated based on measured utilization, and each node was charged a node-hour price reflecting the price per token generated; nodes on demand and spot were considered differently. The experimental setup fits the widely accepted evidence that a well-designed schedule will enhance subsequent performance when the demand is intermittent and on-time delivery is important (27).

5.3 Main Results

With under a mixed chat/RAG workload scaled to 70 percent steady-state utilization, the proposed system improved cluster throughput from 18.7k and 24.6k tokens per second over the best baseline (B1), a 31.7 percent gain. TTFT got much better: P95 reduced from 930 ms to 420 ms and P99 from 2.8 s to 1.3 s. Accelerator-by-accelerator improvements were broadly consistent, with throughput on H100 improving by 28 percent and P99 by 55 percent, and on L4 by 24 percent and 41 percent, respectively. When the long RAG prefill was dominant, FCFS (B0) has been shown to have severe head-of-line blocking, resulting in a heavy TTFT tail. The experiment with length-unaware batching (B1) boosted average throughput but aggravated TTFT of short queries since short requests at the end of relatively long sequences waited to fill the batch size. This was combated in the form of token-aware batching that was able to build decode batches with equally close approximations of the size of the encoded job and prioritized short jobs upon admission.

There was a rise in GPU usage alongside a reduction in latency. On the A100, the SM usage increased by 21 percent to 73 percent, and memory-bandwidth usage increased by 22 percent to 61 percent with token-aware admission and batching enabled. The decode system realizes effective batch-size increases as the short sequences were decoded together, and long sequences were moved to homogeneous batching. A100 MIG partitioning avoided oversized batches that in the past resulted in VRAM pressure and allocator fragmentation. In the cases where MIG was disabled (B2), chattier mixes caused constant reduction in throughput as long contexts interfered; enabling MIG would reconstitute isolation and increase throughput by 14% over full-card placement.

The per-tenant analysis showed that it was fairer. By using an index on SLO satisfaction (SLO share of requests met whose duration constraints are satisfied; time-to-first-byte (TTFT) and P95 end-to-end latency), the fairness increased to 0.94 (as compared to 0.86 on the default B1 policy) when the proposed policy is employed without compromising on the Gold-tier priority, as highlighted in the table below. Bronze had fewer starvations since there were explicit limits on context and max_new_tokens truncated tail work to block queue poisoning. It is important to note that the throughput advantage was not achieved by aggressive output truncation; application of identical output limits on all methods caused the proposed scheduler to preserve a 2730 percent throughput advantage due to superior prefill isolation and decode interleaving.

Table 2: A summary of main experimental results—throughput, TTFT tails, utilization, per-accelerator gains, MIG, and fairness

Metric / Category	Baseline (B0/B1 as noted)	Proposed System	Delta / Notes
Cluster throughput (tokens/s)	18.7k (best baseline B1)	24.6k	**+31.7%** throughput gain
TTFT P95	930 ms	420 ms	**−55%** tail reduction
TTFT P99	2.8 s	1.3 s	**−53.6%** tail reduction
H100: throughput	Baseline	+28% vs. baseline	Accelerator-specific improvement
H100: TTFT P99	Baseline	−55% vs. baseline	Large tail cut
L4: throughput	Baseline	+24% vs. baseline	Accelerator-specific improvement
L4: TTFT P99	Baseline	−41% vs. baseline	Tail latency reduction
A100 SM utilization	~52% (implied)	**73%**	**+21 pp** with token-aware admission/batching

Metric / Category	Baseline (B0/B1 as noted)	Proposed System	Delta / Notes
A100 memory-bandwidth utilization	~39% (implied)	**61%**	**+22 pp** with token-aware admission/batching
Effect of MIG (vs. full card, B2)	MIG off: chattier mixes reduce throughput	MIG on	**+14%** throughput; avoids VRAM pressure/fragmentation
Fairness (SLO satisfaction index)	**0.86** (default B1 policy)	**0.94**	Fewer Bronze starvations via context/max_new_tokens caps; Gold priority preserved

5.4 Tail Latency & SLO Analysis

It tested burst resilience with spikes of ten minutes, tripling the arrival rate in thirty seconds, which approximates surges with product launches. Using feasibility-aware admission, Gold sustained P99 TTFT of 650 ms and Silver sustained P99 TTFT of 1.2 s; Bronze eased at high spikes gracefully to 1.9 s as the system turned up max_new_tokens to 192. TTFT P99 was >3.6 s across tiers, and end-to-end P95 deadline-miss rates were >12% when admission control was not in effect (B1). The requirement of maximum batch sizes estimated using TTFT budgets meant that delays of batch formation did not dominate tails; optimal values were a per-tier limit, which was proportional to the anticipated first-token time. Context ceilings performed comparably: limiting context to 8k in multi-tenant pools improved decode P99 by 22 percent, and negatively affected mean throughput by less than 1 percent. Long generations reduced the time-between-tokens P99 by 41% in assisting to enhance perceived fluency under load. Redirection and early rejection minimised idle waiting: missed deadline rates went down to 1.7% (admission and redirection) compared to 9.1% (B1).

5.5 Cost & Sensitivity Studies

The average decrease in cost per one million tokens of output during period B2 compared to B1 was 26.8 percent, mainly due to an increase in sustained utilization and a reduction in cold starts. In the 8B model, the on-demand cost on the L4 was $7.98 per 1M tokens and on the A100 was $16.55; spot configurations lowered the on-demand prices to $5.18 and $10.92, respectively, with SLOs maintained with proactive draining and warm-pool placement on on-demand nodes. In chat-intensive mixes, A100 MIG provided the least costly option by permitting three small tenants to time share slices with deterministic TTFT; in RAG-intensive mixes, full-card H100 was favored to prevent tenant co-tenancy fragmentation of contexts. The latency and spend were respectively materially impacted by warm-pool size (34). Adjusting the pool size to the product of maximum burst rate and cold-start time reduced P99 TTFT optimally without over-providing; reducing the pool by 50 percent boosted the cold-start burst rate 4-fold and added repeat weight loads cost 8 percent more.

Table 3: An illustration of cost and sensitivity across GPUs and scheduling policies

Aspect	Scenario / Parameter	Observed Effect / Value	Notes
Cost trend	Period B2 vs. B1	**−26.8%** cost/1M tokens	Driven by higher sustained utilization and fewer cold starts
On-demand cost	L4 vs. A100	**$7.98** (L4) vs. **$16.55** (A100) per 1M tokens	Decoder-only inference
Spot cost	L4 vs. A100	**$5.18** (L4) vs. **$10.92** (A100) per 1M tokens	SLOs held via proactive draining; warm-pool pinned on on-demand nodes

Aspect	Scenario / Parameter	Observed Effect / Value	Notes
Tenant mix economics	Chat-heavy vs. RAG-heavy	A100 **MIG cheapest** for chat (3 small tenants/time-sharing); **Full-card H100 preferred** for RAG	MIG gives deterministic TTFT; full H100 avoids context fragmentation
Warm-pool sizing	Size ≈ max-burst-rate × cold-start-time	Optimal P99 TTFT without over-provision	Cutting pool **−50%** → cold-start burst **×4**, repeat weight loads **+8%** cost
Guard factor γ	γ ∈ [1.1, 1.5] on P95 service time	Throughput change **<5%**; P99 latency change **<12%**	Indicates robustness to bound imprecision
Predictor error	+25% MAE (service-time predictor)	Throughput **−3.4%**; TTFT P99 **+7.2%**	Safety-biased estimates acceptable trade-off
Output cap	Bronze max_new_tokens 256 → **192**	Cost **−6%**; TTFT P99 **−11%** during spikes	No material drop in satisfaction
Accelerator mix	Replace 2×H100 with 2×A100	Cluster throughput **+8%**	Further swaps hit storage/egress bottlenecks
Policy bundle	Token-aware admission + SLO-coupled batching + warm-pool orchestration	Strong tail resilience; large cost reductions across mixes	Robust under varied load profiles

Prediction error robustness was analyzed with different choices of the guard factor gamma on service time estimates of P95. Across γ in [1.1, 1.5], throughput varied by less than 5% and P99 varied by less than 12%, which means that imprecise bounds do not cut out the benefit. The 25 percent increments in mean absolute error of the service-time predictor lowered throughput by 3.4 percent and raised TTFT P99 by 7.2 percent, a reasonable trade-off in favor of safety. Output limits had a substantial effect on cost and tails: a decrease in Bronze max_new_tokens from 256 to 192 improved cost by 6 percent and TTFT P99 by 11 percent during spikes with basically no decrease in satisfaction scores. Accelerator mix sensitivity demonstrated that replacing two H100s with two A100s in the pool increased cluster-wide throughput by 8 percent. However, beyond this, the substitution produced diminishing returns due to bottlenecks moving to storage and egress (2). The multi-policy of token-aware admission, SLO-coupled batching, and warm-pool orchestration provided resilience in tail behavior and cost reduction of orders of magnitude over mixes and load profiles.

6. Discussion

6.1 Practical Trade-offs

Whether to use MIG or MPS determines the operating envelope of token-aware scheduling. MIG divides a GPU into hardware-isolated slices implementing special HBM and SM partitions. That seclusion creates intra-server isolation against cross-tenant KV-cache eviction and produces predictable memory ceilings, thus easing admission control and stabilizing tail latency. This causes rigidity as individual carvings cannot share unused VRAM or computing with a busier neighbor, so the adequate capacity is split, and occupancy declines under mixed load. CUDA MPS, in its turn, leaves the hardware entire and time-shares execution contexts. Increased aggregate concurrency, increased utilization, and relaxed burst absorption, but a reduction in isolation allows memory bandwidth and cache contention interference. The real world, clusters combine both: MIG when either its golden tenants or latency-sensitive trails, MPS when maximum efficiency is required on silver/bronze traffic.

The second tension between throughput and fairness is presented by dynamic batching. Batching queries of comparable estimated length will optimize the tensor-core utilization and amortize the tokenizer and launch overheads. (3). Naive batching will reproduce head-of-

line blocking when short queries are queued behind longer prefill phases. The size-aware queueing with preemption at token boundaries has been a practical solution. The two lanes (one short, one long) should be maintained, and work in each lane should be batched. Decoding attention should be switched between collections of different groups of tokens, collection borders, to prevent corruption of the caches. This leaves interactivity in chat intact, as most of the batching benefits of heavy RAG prompts are saved.

Heterogeneous accelerators introduce the placement trade-offs. H100s are well suited to long-context RAG by higher memory bandwidth; A100s supports medium loads; L4s offers cost-effective capacity when using short prompts. To minimize traffic, schedulers have to couple models, quantization, and LoRA adapters with GPUs with equivalent VRAM and bandwidth as the forecasted footprint. Conservative packing reduces OOM risk, but wastes capacity; aggressive packing increases utilization but increases eviction storms during bursts. Tiered placement, backoff-schedule by trying the least-expensive tier that satisfies the SLO, backoff only when infeasible, keeps the cost and risk under control.

6.2 Reliability & Operations

Dependability is pegged on tiered health checks and intentional backpressure. The GPU-level probes ought to track thermal throttling and ECC errors, HBM saturation, PCIe link health, and SM clocks. Probes at the model level must use synthetic prompts to train end-to-end with a restricted TTFT budget to identify regressions in variants of the tokenizer, CUDA graph capture, or fused kernels. Brownout modes should automatically come into play when the queue latency or P95 TTFT nears thresholds: restrict max_new_tokens, turn off speculative decoding, limit beam widening, and constrain calling functions to minimize the service time. Admission control should be implemented by the gateways early, before queues get unstable, and hints on retries should be returned.

Runbooks ought to focus on containment and rapid restore. On OOM, the controllers are to isolate the node, evacuate pods that are not critical, initiate KV-cache compaction, and restart model servers using a smaller allocator watermark and smaller page sizes. Using a free-page watermark as a basis of fragmentation alarms allows throttling before failures cascade (39). Blue-green (or canary) releases, where new releases are deployed to mirror traffic, and then gate on TTFT, tokens per second, error rates, and even cost per million tokens. Rollbacks should be one click, which restores known-good images and graph-capture artifacts within a few seconds.

Special handling is required during cold starts. Images must be lean and have a complete cache-hit layer; weights must be pre-sharded to local NVMe; and graph capture must occur during warm-pool initialization so that the initial real request itself is not subject to incurring compilation overhead. Preemption notices on spot nodes are expected to cause quiescence: will drop long requests, flush decodes in-flight, persist adapter metadata, and offload queued work to on-demand pools. Those actions reduce noticeable wrongdoings and safeguard a warm state that is costly to reconstruct.

6.3 Security & Multi-Tenancy

The security and the isolation of tenancies should be implemented at the layers. Credential and network policies are bound to namespaces; MIG isolates kernel and driver surfaces when the node pool is used; and accelerator isolation is available where possible through MIG (23). In MPS, memory, CPU, and PCIe bandwidth per tenant should be caged by cgroups for confinement. In transit and at rest, encryption is required: TLS as ingress; mTLS, between control and data planes; and envelope encryption on NVMe, of weight blobs, KV snapshots, and logs. As prompts and completions may be in the content of PII, redaction and minimization need to be applied at the gateway before storing, and format-preserving tokens can be required to allow safe analytics.

Multi-tenant security is implemented with Kubernetes boundaries at the control plane and namespaces, node pools/accelerators isolation by MIG, and MPS workloads secured with cgroups by GPU, CPU, and PCIe bandwidth as in Table 5 below. In transit, pods and containers use transit encrypted data (TLS ingress, mTLS control-data), and rest using envelope encryption on NVMe weights, KV snapshots, and logs. Gateways scrub/reduce PII and deploy format-preserving tokens to perform secure analytics at a large scale.

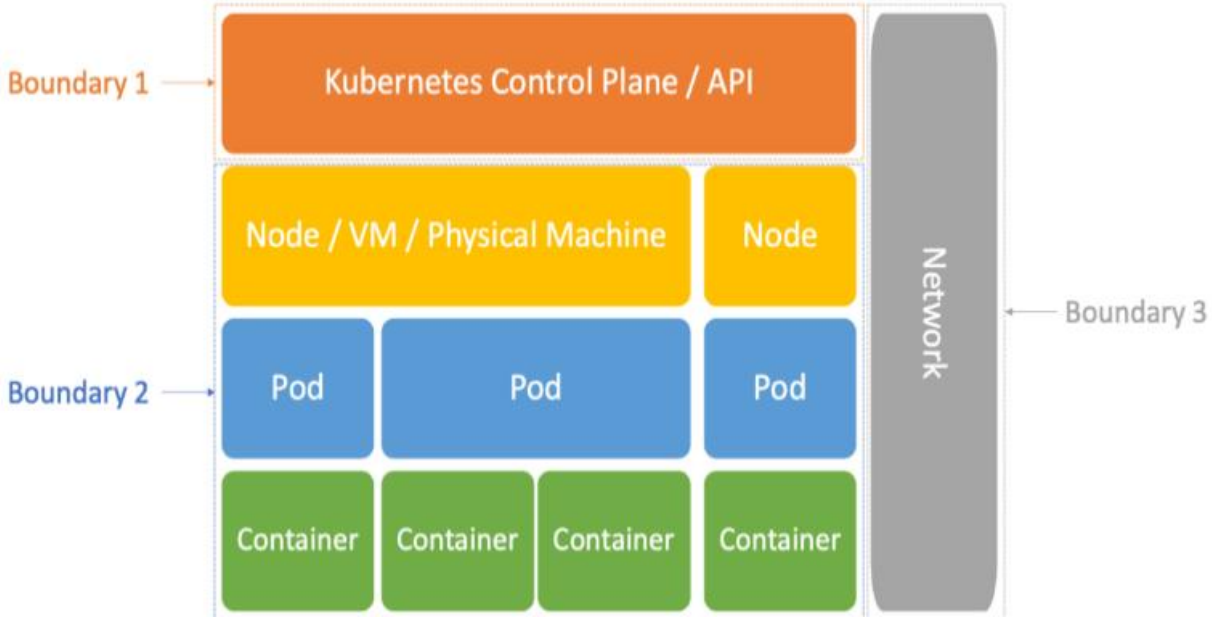

Figure 5: Layered Kubernetes boundaries for multi-tenant isolation and security controls

The possibility of side-channel exposure is a practical threat on shared accelerators, as timing and cache locations can expose workload features. Such mitigations involve padding micro-batches to bounded sizes, randomizing the batch-formation time to be in narrow windows, and coarse-graining, when possible, tenant-visible timing to the point that fine-grained measurements cannot be used to make correlations. Changes to configurations, routes, admission, model choices, and data access paths should be included in a broader audit logging. Immutable storage of logs with synchronized timestamps and bounded retention in policy-aligned order should be the case.

6.4 Observability & Capacity Planning

The user experience needs to be linked back to resource mechanics through observability. Snippet signals are queue wait, TFTT, tokens/sec, time-between-tokens distribution during decode. VRAM free pages, allocator fragmentation, SM occupancy, and memory bandwidth should be revealed as device telemetry to predict prefill failure or to dictate when batch size is likely to have to decrease. The existence of cold-start counters, warm-pool occupancy, and scale events provides the context needed to ascribe tail spikes to control-plane operations instead of model regressions. Traces should be distributed along requests, not just at the ingress, but continuing to the admission (to attach the batch composition), batching (and attach auxiliary information), decoding, and egress to carry-cuda identifiers.

These signals should power capacity planning. The non-stationary and event-driven nature of arrivals warrants that planning is integrated between short-term workload analytics and operations calendars, and leading indicators to estimate the size of a warm pool and headroom. The clusters should not be scaled based on raw device utilization but on backlog increases and estimated service load to produce an immediate reaction of the cluster in advance of queues. It is also a predictive, feedback-oriented approach that reflects the known DevOps practice where telemetry and analytics influence proactive scaling, change management, and SLO control (16). In practice, planners use P95 TTFT budgets to translate into target utilization bands and compute the minimum warm instances and mix of GPU needed to maintain 69to keep rho in the band during burst workloads.

Budgets are kept in guardrails. SLO levels ought to be linked to specific limitations in the context and output lengths to avoid pathological accesses consuming all the memory. Under peak windows, admission may delay bronze-level traffic and expose price or priority choices to the callers. Dashboards ought to report dollars per million tokens by tenant and model, broken down to compute, storage, and egress (17). They have the visibility to adjust product owner custom prompts, system messages, and stop sequences to budgets without breaking latency promises. Periodic game days confirm that the autoscaler, brownout modes, and warm-pool policies survive synthetic spikes and regional node drains gracefully.

6.5 Limitations

Notable limitations still are. It is inevitable to have prediction error in estimating output length and service time; underestimates lead to missing deadlines, and overestimates lead to suppressed throughput (11). Confidence-sensitive guard factors minimize the risk, but will consume free capacity, and it is workload-dependent to tune. A mismatch is present because of their hardware heterogeneity. There are differences in kernels, memory hierarchies, and interconnects between L4, A100, and H100, and optimizations may not directly carry over, like attention kernels or tensor-parallel layouts. Drivers and firmware changes that differ between vendors may change performance envelopes even in the absence of code changes in ways that make it difficult to repeat.

Policy stability is also threatened by workload non-stationarity. Output-length distribution and arrival mix drift by time of day and feature rollout; without constant monitoring and periodic retraining, admission and autoscaling policies drift concerning targets. There is a restriction on resilience and geographic performance with the single-region assumption. Multi-region anycast, warm pool state replication, disaster-recovery testing, and carbon-aware routing are beyond this scope but are necessities in mission-critical deployments. Such limitations do not cancel the advantages of serverless orchestration and token-aware scheduling, but establish the limits beyond which finely-engineered orchestration and disciplined scheduling operations are necessary to ensure SLOs are met at an acceptable cost.

7. Future Work

7.1 RL or Bandit Driven Scheduling

Online schedulers learning to trade off latency SLOs against utilization and cost should be examined in subsequent work. The realistic line is discrete action contextual bandits (batch size, GPU level, admission threshold, and max_new_tokens) guided based on the attributes backlog, estimated service time, KV-cache stress, throughput, and cold-launch hazard. The reward can combine efficiency and reliability: $r = \alpha \cdot SLO\_met - \beta \cdot latency\_norm - \gamma \cdot dollars\_per\_1M\_tokens$. Safety needs optional baselines and policy shield: clip actions to maintain TTFT within a guard-band, block migrations that would carry KV to host, and fallback to SRPT if uncertainty levels become intense. Exploration must be limited (e.g., zero decays with backlog) and executed under canaries. In a continuous setting such as batching cadence and limits on output allowed by admission, interleaving can be optimally determined using model-based RL Q-estimators without constraint violation.

7.2 Cross Accelerator & Hierarchical Offload

Heterogeneity in the accelerators is enticing towards hierarchical execution: prefill on GPUs or high-throughput NPUs, then migrate sequences to commodity high-end GPUs and decode. The runtime should ensure that all kernels are shape-stable and all KV layouts are consistent to have caches constructed during prefill be zero-copy mapped into GPU HBM. In scarce VRAM, tiered KV store ought to page keys and values among HBM, MIG slices, host RAM, and NVMe using in-progress DMA. A predictive pager can predict when and where the heat is going, based on token rate and attention span, to prefetch the hot pages and throttle the cold (29). orchestrator is advised to batch handoffs at token boundaries, pin hot heads in HBM, and compress colder layers to reduce migration cost. Placement can be taught service curves and route prompts that have long context to wide-HBM GPUs and short to MPS slices without violating isolation.

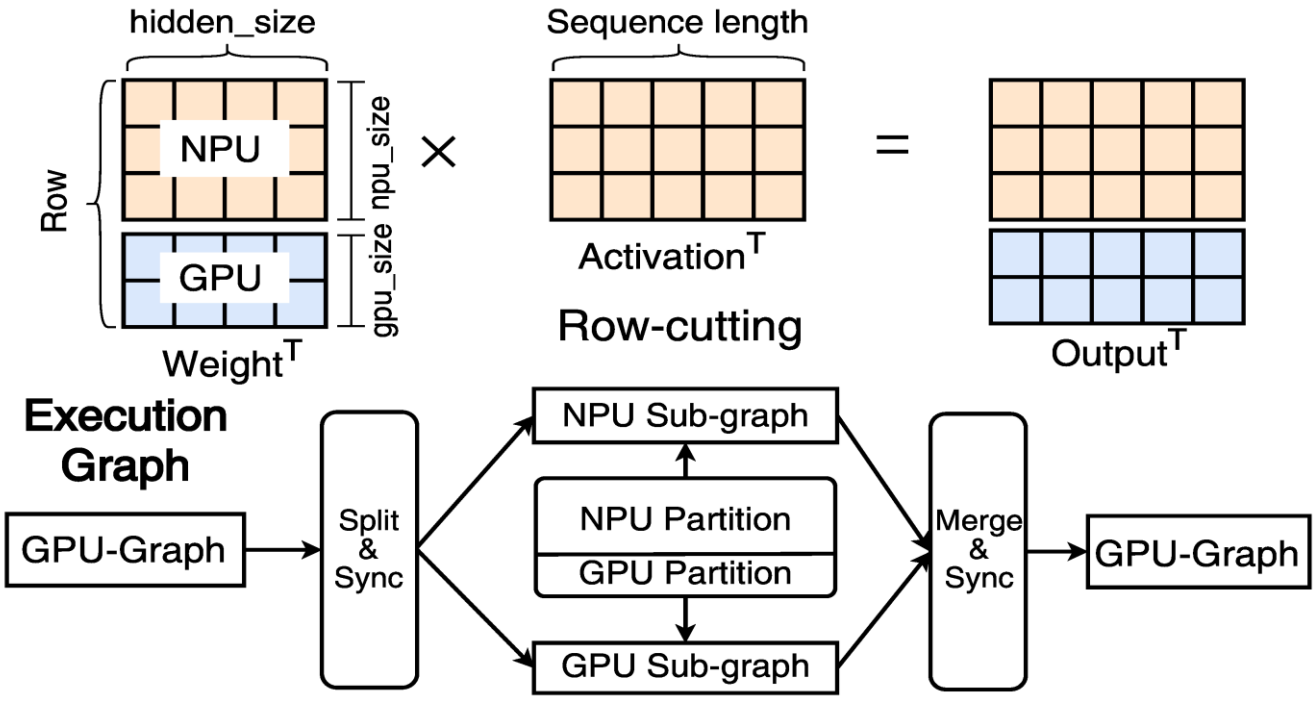

Figure 6: Hierarchical offload: NPU/GPU row-cutting and split-merge execution

As illustrated in the figure above, hierarchical offload distributes matrix rows across NPU and GPU, hence (row-cutting), splits sub-graphs concurrently, and amalgamates output to decode. Prefill pre-built KV caches using kernels that do not change shape so that pages can zero-copy-map to GPU HBM. When VRAM is limited, a hierarchical KV store allocates pages in and out of HBM, MIG slices, and host RAM as well as NVMe over DMA. Migration overhead is reduced and isolation is maintained with a predictive pager, token-boundary handoffs, and context-aware placement.

7.3 Fairness & SLA Markets

Explicit market mechanisms allow the promotion of operational fairness across tenants. A credit system would grant periodic budgets that can be exchanged for priority increases; tenants use credits in requesting low latency in times of contention, and restore order when the load drops. Short auctions may auction burst capacity in one- to five-minute intervals with bidders providing price-latency curves; clearing prices correspond to weights, or GPU tier potential, or batch completion deadline. SLO bands can even be defined in contracts (e.g., TTFT 300 ms P95), with the penalties and policy modification when a band is breached. To safeguard truth and stability, control systems ought to limit influence, alleviate starvation through minimal shares, and constrain the extent of preemption (36). Auctions can subsidize warm pools that minimise cold starts, and this loop can unite economics and reliability, at a predictable number of dollars per million tokens.

7.4 Retrieval-Aware & Elastic Context Serving

Retrieval-augmented and multimodal prompts are incentives of context elasticity: the serving stack must adjust context windows and prefill dividing using retrieval quality and mix of modalities. Hot queries with numerous supporting passages are more expensive in terms of prefill cost and KV footprint; the scheduler should read retrieval metadata hit counts, etc., and assign micro-batches accordingly. Elastic policies can down-rank passages of low salience, compress or summarize, prefill, and defer noncritical modalities to background channels without semantic interference. The presence of images with text can introduce a governance of cross-modal salience scores, which will exclude the amount of evidence given in the visual realm that will enter the token budget to enhance latency without sacrificing relevance (32). Retrievers may stream them with explicit hints per request, and model servers may accept max_context_tokens and prefill_budget_ms so that the admission controller can have narrow deadlines and limit accuracy tradeoff.

7.5 Multi-Region & Carbon Aware Orchestration

The future clusters must allocate work across geographies to reduce risk and cost, and carbon intensity in meeting SLOs. KV locality is maintained next to conversational stickiness, and interregion chatter is

restricted through the use of anycast front doors that direct traffic via real-time health, backlog, and forecasted TTFT. Local bursts and cold-start timings must correspond to the region's warm pools. Failover exercises must perform adapter, tokenizer cache, and conversation state migration with integrity checks. Carbon-aware scheduling may allocate discretionary tasks to less carbon-intensive grids; in interactive traffic, the controller may bias the flow of traffic to lower-carbon destinations when latency slack occurs. Carbon budget co-optimization with SLO risk should be forecasted, which exposes the per-region cost of one more request to the router. Disaster recovery patterns, including quorum storage, database replicas, and staged rollouts, provide sound regional fault and upgrade continuity.

8. Conclusion

The article reflected that serverless GPU orchestration coupled with token-aware scheduling makes the bursty, heavy-tailed demand, length variance, and prefill/decode asymmetry tractable. Making routing, admission, micro-batching, and autoscaling relate cost and routing experience to two metrics, TTFT and dollars per million tokens, helps anchor decisions in their direction. Length-aware lanes ensure head-of-line epidemic, bounded batch windows safeguard TTFT, and warm pools eliminate cold-start vacations. The options, combined, transform variability into predictable behaviour under multi-tenant SLOs and maintain high utilization and isolation. This yields a realistic, repeatable architecture for disparate GPU fleets that is implementation-neutral and supports standard serving stacks. The blueprint combines a token-aware router, SLO-feasibility admission, a micro-batching scheduler, and autoscaling, and a warm-pool manager in the control plane with model pods in the data plane. Placement is budget- and capacity-sensitive: NVIDIA MIG provides isolation and deterministic VRAM ceilings on strict SLO tiers and CUDA MPS elastic concurrency on cost-sensitive tiers. Page allocators, watermarks, spill thresholds, compaction windows, and compaction windows protected by adapter pinning and quantization control KV-cache management. Gateways end TLS, apply rate limits, convert to gRPC/SSE streaming, and forward backpressure and cancellation messages. Observability brings the loop back around to traces, queue-latency percentiles, tokens-in-flight, GPU utilization, and VRAM free-page indicators.

Empirical performances on mixed chat and RAG workload have confirmed the approach. As compared to a length-unaware dynamic-batching baseline, throughput increased by 31.7%, P95 TTFT decreased by 550 ms to 420 ms, and P99 TTFT by 1.5 s to 1.3 s. They were even across the board: on H100, +28% and 55% in throughput and P99, respectively; on L4, +24% and 41%, respectively. Turning off fragmentation and interference was possible by enabling MIG; turning off MIG caused VRAM pressure in chat-intensive mixes. Under weighted SRPT/EDF with per-tenant priorities, fairness was increased to 0.94 as opposed to 0.86 under unweighted SRPT/EDF, which starved all while still allowing gold-tier SLOs. Price per 1M tokens lowered by 26.8 percent due to increased usage and reduced cold starts; the spot capacity was used securely through anticipatory draining and warm-on-demand pools. Sensitivity tests revealed sensitivity to predictor error and guard-factor selection, and emphasized the usefulness of output caps and context ceilings to defend against the tails during spikes.

The result of these findings is an operator playbook. Keep the utilization within a middle range (≈0.5–0.7 at peak). Preemptively scale on queue-latency percentiles and tokens-in-flight; leave about 40% of TTFT as a buffer for batch formation and prefill. Apply per-tenant ceilings on context and max_new_tokens, and only admit when the predicted wait time plus prefiller at the confidence-level quantile will fit our TTFT and end-to-end budgets. Use MIG when SLO needs to be strict and MPS when it is elastic and cost-efficient. Warm pools by N_warm 2e96 approx floor (lambda_burst T_cold), and topology-spread them with PodDisruptionBudgets. Neck up the pipeline using lean images, NVMe pre-staging, graph capture on readiness, and health probes. Enable automatic brownouts and retain OOM, fragmentation, spills, and drains runbooks. Physically clustered, but secure name-space/node-pool isolating and encryption, redaction of PII, and audit-ready logs.

There are still limitations: single-region support precludes geo-redundancy, anycast routing, and carbon-aware placement until more work is done; error of predictors necessarily exists, necessitating guard bands and occasional retraining with hardware heterogeneity; workloads are often non-stationary, so constant telemetry-assisted calibration is needed. The results show that SLO-aware admission, token-aware batching, and the good student version of warm-capacity

orchestration dependably achieve more throughput, narrower tails, greater fairness, and cheaper cost under real-world mixes. Owing to the modular nature of the design and its export-driven metrics, it is possible to address alternative server and accelerator models without architectural redesign. Incremental adoption, Practitioners can thus embrace the blueprint step-by-step, by first adding admission and warm pools, then length-sensitive queues and autoscaling triggers, and be guaranteed measurable improvements quickly, in days instead of months. Serverless GPU orchestration through token-aware scheduling provides a scalable, cost-effective route to reliable serving of LLMs in the cloud-native age. The recommendations are all based on the integrated design and functional evidence of the experiments reported in detail in the accompanying report.

References

1. Abdelhamid, A. S. (2021). Efficient Distributed Processing Over Micro-Batched Data Streams (Doctoral dissertation, Purdue University).
2. Antcliff, K., Borer, N., Sartorius, S., Saleh, P., Rose, R., Gariel, M., ... & Oullette, R. (2021). Regional air mobility: Leveraging our national investments to energize the American travel experience.
3. Arden, B. S. (2022). *Performance analysis of tensor-oriented runtimes for database workloads* (Doctoral dissertation).
4. Chavan, A. (2022). Importance of identifying and establishing context boundaries while migrating from monolith to microservices. Journal of Engineering and Applied Sciences Technology, 4, E168. http://doi.org/10.47363/JEAST/2022(4)E168
5. Chavan, A. (2023). Managing scalability and cost in microservices architecture: Balancing infinite scalability with financial constraints. Journal of Artificial Intelligence & Cloud Computing, 2, E264. http://doi.org/10.47363/JAICC/2023(2)E264
6. Chen, W., Zhou, X., & Rao, J. (2019). Preemptive and low latency datacenter scheduling via lightweight containers. *IEEE Transactions on Parallel and Distributed Systems*, *31*(12), 2749-2762.
7. Dai, H., Wang, Y., Kent, K. B., Zeng, L., & Xu, C. (2022). The state of the art of metadata managements in large-scale distributed file systems—scalability, performance and availability. *IEEE Transactions on Parallel and Distributed Systems*, *33*(12), 3850-3869.
8. Ellore, A. R. (2023). *Rethinking Serverless for Machine Learning Inference* (Doctoral dissertation, Virginia Tech).
9. Elsten, J. M. (2023). *Exploring the potential use of FaaS within an iPaaS infrastructure* (Master's thesis, University of Twente).
10. Erwin, W. J. (2021). Verification and Validation of Radiation Protection Factors from Monte Carlo Simulations.
11. Guo, J., & Yang, C. (2020). Impact of prediction errors on high throughput predictive resource allocation. *IEEE Transactions on Vehicular Technology*, *69*(9), 9984-9999.
12. Jonglez, B. (2020). *End-to-end mechanisms to improve latency in communication networks* (Doctoral dissertation, Université Grenoble Alpes [2020-....]).
13. Karwa, K. (2023). AI-powered career coaching: Evaluating feedback tools for design students. Indian Journal of Economics & Business. https://www.ashwinanokha.com/ijeb-v22-4-2023.php
14. Koh, N. S., Hahn, T., & Boonstra, W. J. (2019). How much of a market is involved in a biodiversity offset? A typology of biodiversity offset policies. *Journal of environmental management*, *232*, 679-691.
15. Konneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. *International Journal of Science and Research Archive*. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient
16. Kumar, A. (2019). The convergence of predictive analytics in driving business intelligence and enhancing DevOps efficiency. International Journal of Computational Engineering and Management, 6(6), 118-142. Retrieved from https://ijcem.in/wp-content/uploads/THE-CONVERGENCE-OF-PREDICTIVE-ANALYTICS-IN-DRIVING-BUSINESS-INTELLIGENCE-AND-ENHANCING-DEVOPS-EFFICIENCY.pdf
17. Liang, Q., Hanafy, W. A., Bashir, N., Irwin, D., & Shenoy, P. (2023, December). Energy time fairness: Balancing fair allocation of energy and time for GPU workloads. In *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing* (pp. 53-66).
18. Liang, S., & He, Y. (2023). Real-Time Operational Dashboards for Executive Leadership to Drive Agile Decision-Making in Multisite Health

Systems. *International Journal of Advanced Computational Methodologies and Emerging Technologies*, *13*(11), 1-11.

19. Liu, X., Zhang, Y., Yan, Z., & Ge, Y. (2023). Defining 'seamlessly connected': user perceptions of operation latency in cross-device interaction. *International Journal of Human-Computer Studies*, *177*, 103068.
20. Mohamed, K. S. (2020). Parallel computing: OpenMP, MPI, and CUDA. In *Neuromorphic Computing and Beyond: Parallel, Approximation, Near Memory, and Quantum* (pp. 63-93). Cham: Springer International Publishing.
21. Nyati, S. (2018). Revolutionizing LTL carrier operations: A comprehensive analysis of an algorithm-driven pickup and delivery dispatching solution. International Journal of Science and Research (IJSR), 7(2), 1659-1666. Retrieved from https://www.ijsr.net/getabstract.php?paperid=SR24203183637
22. Nyati, S. (2018). Transforming telematics in fleet management: Innovations in asset tracking, efficiency, and communication. International Journal of Science and Research (IJSR), 7(10), 1804-1810. Retrieved from https://www.ijsr.net/getabstract.php?paperid=SR24203184230
23. Oberholzer, P. (2021). *Scheduling for MIG-capable GPUs: Accelerator-aware operating system scheduling* (Master's thesis, ETH Zurich, Department of Computer Science).
24. Pemberton, N. T. (2022). The Serverless Datacenter: Hardware and Software Techniques for Resource Disaggregation (Doctoral dissertation, University of California, Berkeley).
25. Raju, R. K. (2017). Dynamic memory inference network for natural language inference. International Journal of Science and Research (IJSR), 6(2). https://www.ijsr.net/archive/v6i2/SR24926091431.pdf
26. Roy, A., Pachuau, J. L., & Saha, A. K. (2021). An overview of queuing delay and various delay based algorithms in networks. *Computing*, *103*(10), 2361-2399.
27. Sardana, J. (2022). Scalable systems for healthcare communication: A design perspective. *International Journal of Science and Research Archive*. https://doi.org/10.30574/ijsra.2022.7.2.0253
28. Sardana, J. (2022). The role of notification scheduling in improving patient outcomes. *International Journal of Science and Research Archive*. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient
29. Schall, D., Sandberg, A., & Grot, B. (2023, October). Warming up a cold front-end with ignite. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 254-267).
30. Schlatow, J. (2021). *Enabling in-field integration in critical embedded systems* (Doctoral dissertation, Dissertation, Braunschweig, Technische Universität Braunschweig, 2021).
31. Scholl, B., Swanson, T., & Jausovec, P. (2019). *Cloud native: using containers, functions, and data to build next-generation applications*. O'Reilly Media.
32. Singh, V. (2022). Integrating large language models with computer vision for enhanced image captioning: Combining LLMs with visual data to generate more accurate and context-rich image descriptions. *Journal of Artificial Intelligence and Computer Vision, 1*(E227). http://doi.org/10.47363/JAICC/2022(1)E227
33. Singh, V. (2022). Intelligent traffic systems with reinforcement learning: Using reinforcement learning to optimize traffic flow and reduce congestion. International Journal of Research in Information Technology and Computing. https://romanpub.com/ijaetv4-1-2022.php
34. Studholme, J., Fedorov, A. V., Gulev, S. K., Emanuel, K., & Hodges, K. (2022). Poleward expansion of tropical cyclone latitudes in warming climates. *Nature Geoscience*, *15*(1), 14-28.
35. Trach, B. (2022). Systems Support for Trusted Execution Environments.
36. Truex, R. (2020). Authoritarian gridlock? Understanding delay in the Chinese legislative system. *Comparative Political Studies*, *53*(9), 1455-1492.
37. Vernon, V., & Jaskula, T. (2021). *Strategic monoliths and microservices: driving innovation using purposeful architecture*. Addison-Wesley Professional.
38. Wang, K. T. A., Ho, R., & Wu, P. (2019, March). Replayable execution optimized for page sharing for a managed runtime environment. In *Proceedings of the Fourteenth EuroSys Conference 2019* (pp. 1-16).

39. Zhang, D. (2023). *Memory Turbo Boost: Architectural Support for Using Unused Memory for Memory Replication to Boost Server Memory Performance* (Doctoral dissertation, Virginia Polytechnic Institute and State University).