A Sovereign Execution Fabric Enabling Deterministic CPU–GPU Orchestration

Author: Abdulelah R. Alnaddaf
Affiliation: Tokra — Sovereign Execution Fabric
Correspondence: adh.ndf@gmail.com
Version: v1.1 (Preprint) — 2025-09-16
Keywords: sovereign execution; deterministic scheduling; Control-Mesh; SCL/SCL++; APF; in-band policy; zero-leak telemetry; thermal/energy control; heterogeneous systems.

Abstract

We present **Tokra**, a sovereign execution fabric for deterministic CPU–GPU orchestration that embeds control into the runtime via an adaptive **Control-Mesh** with **SCL/SCL++** time-slotting and **APF** transactional commits. Unlike queue-based schedulers, Tokra maintains *iso-pressure* across copy engines and GPU streams, enabling low-jitter, reproducible timing under load. On a reference RTX-4090 node running **vLLM 0.10.x** (AWQ/FP16, FlashInfer), Tokra improves mean tokens-per-second from **~770** to **~1790** and reduces end-to-end latency from **~155 ms** to **~71.5 ms** (≥3 runs; p95/p99 reported). We detail the scheduling algorithm, thermal/energy control loops, and zero-leak telemetry, and we release artifacts (configs, logs, seeds) **as ancillary files** to ensure reproducibility. Tokra demonstrates a controller-less path to compute-native sovereignty, aligning runtime determinism with in-band policy enforcement.

Motivation & Novelty. Modern heterogeneous compute stacks need predictable and secure execution under thermal/energy pressure. **Tokra** is a sovereign execution fabric that provides deterministic, real-time CPU–GPU orchestration via an adaptive **Control-Mesh**. Unlike conventional batch/queue schedulers and ad-hoc stream heuristics, **SCL/SCL++ (Spectral Control Layer)** maintains **iso-pressure** and deterministic time-slotting across **CPU ↔ copy engines ↔ GPU streams**, yielding low jitter and reproducible micro-timing under load.
Scope & Focus. We evaluate Tokra as an **in-band runtime**. Components can be deployed independently; **no result assumes co-activation** of all units. Unless stated otherwise, reported results refer to the **Loop** execution kernel with **SCL/SCL++** and **APF** enabled, under the configurations specified in the methodology.
Primary mechanism — Tokra Loop. The execution kernel applies **SCL/SCL++** to preserve multi-stream determinism and iso-pressure across data movement and compute, while **APF (Atomic Pipe Fabric)** provides transactional micro-path routing with safe fallback without restarts. The design targets low jitter, predictable micro-timing, and thermal-aware stability. **Loop performs no deduplication or content filtering by design.**
Governance layer — Tokra Sovereign. An optional governance layer for air-gapped operation with a **signed software supply chain** and **dynamic trust domains**, enabling offline attestation, sealed artifacts, and policy-bound upgrades. Sovereign positions Tokra as a compute-native

sovereignty layer: keys and artifacts are governed end-to-end, and execution remains verifiable in disconnected or controlled environments.
Out-of-scope complementary units. Shield (in-band policy enforcement) and **Trim** (token-reduction while preserving semantic fidelity) are **not evaluated here** and **do not alter Loop scheduling or determinism**; brief descriptions are provided in **Appendix D**.
Context (Why now). AI serving is scaling faster than traditional orchestrators can guarantee determinism; thermal/energy constraints and supply-chain trust have become first-order concerns, especially at the edge and in air-gapped deployments. Tokra addresses these pressures with in-band control, governable artifacts, and configuration-specific, measured performance. (Representative LLM-inference results appear in **Section 8 — Evaluation**.)

1- Introduction

For decades, computational systems were engineered around static hierarchies, fragmented pipelines, and reactive control—architectures suited to predictable workloads rather than today's volatile, data-saturated, cross-platform execution. Under real-time pressure, legacy stacks struggle with latency, resource contention, and the absence of a unified, adaptive execution fabric. In multi-tenant GPU serving, even minor thermal spikes can trigger opportunistic stream reordering, breaking micro-timing determinism and SLOs; by contrast, Tokra's SCL/SCL++ maintains iso-pressure and deterministic slots under the same load.

This paper introduces **Tokra**—a **sovereign execution fabric** purpose-built to overcome architectural fragmentation and timing drift in modern heterogeneous systems. Tokra unifies streaming and **in-band control** into a low-overhead runtime across CPUs, GPUs, and emerging accelerators **without external orchestrators**. In Tokra's modular line, **Loop** is the execution kernel (determinism, iso-pressure, micro-timing), and **Sovereign** is an optional governance layer (signed supply chain, dynamic trust, offline attestation). **Trim** (semantic token reduction) and **Shield** (in-band policy enforcement) are complementary and **out of scope for evaluation** here; each unit deploys independently and integration is optional and non-conflicting.

Contributions. (i) An in-band **Control-Mesh** with **SCL/SCL++** slotting and **APF** transactional micro-paths for deterministic CPU↔copy-engines↔GPU orchestration; (ii) a zero-leak telemetry and thermal/energy control stack aligned with deterministic p95/p99 behavior; (iii) a reproducible evaluation with artifacts (configs/logs/seeds) demonstrating significant improvements in throughput and end-to-end latency under load.

2- Related Work

Modern intelligent systems rely on layers that were optimized for elasticity and throughput rather than micro-timing determinism under thermal/energy pressure. Representative families include **container/cluster orchestrators** [1], **HPC workload managers** [2], **distributed compute/stream frameworks** [3], and **GPU execution/serving stacks** (e.g., stream/graph runtimes) [4]. These systems are mature in their intended contexts, yet they do not, by design, guarantee sub-millisecond **deterministic micro-timing**, provide **in-band policy enforcement**, or maintain **verifiable execution in air-gapped settings**.

Tokra complements—not replaces—these systems. It operates **in-band** inside the runtime, pushing control to the execution path itself while remaining interoperable with external placement/isolation layers. When combined with existing orchestrators, Tokra governs CPU↔copy-engines↔GPU streams at microsecond granularity, while external systems continue handling cluster-level scheduling and accounting. This work scopes evaluation to single-node execution with multi-stream GPUs; multi-GPU/node extensions are discussed separately (Appendix).

(Citations [1]–[4] will be expanded in the References per venue style.)

3- System Overview

Tokra is a **sovereign execution fabric** that embeds control inside the runtime via an adaptive **Control-Mesh**, providing **low-overhead, in-band** orchestration for heterogeneous processors. Rather than treating execution as an endpoint, Tokra co-locates control with application logic and hardware context to achieve deterministic behavior under load.

At the core are three cooperating mechanisms: **(i) SCL/SCL++** for iso-pressure and deterministic time-slotting across CPU, DMA/copy engines, and GPU streams; **(ii) APF (Atomic Pipe Fabric)** for transactional micro-path routing with safe fallback that avoids restarts; and **(iii) zero-leak telemetry with thermal/energy control loops** that align policy with observed timing and power budgets. The fabric **does not perform deduplication or content filtering**; such concerns are explicitly out of scope for the execution kernel.

Tokra integrates across legacy and modern CPUs, GPUs, embedded accelerators, and emerging architectures without requiring redesign. By moving control **in-band**, it enables timing-critical pipelines and AI serving stacks to maintain deterministic p95/p99 behavior under thermal and resource pressure. Representative improvements in end-to-end latency and sustained throughput are reported in **Section 8 (Evaluation)**.

4- Contributions of This Work

This work introduces **Tokra**, a **sovereign execution fabric** that elevates runtime execution to a first-class architectural layer—**embedded, policy-aware, and intent-governed**. Rather than treating execution as a passive backend stage, Tokra reframes it as the **center of computation**, tightly coupled to decision logic, hardware context, and sovereign control principles.

We contribute:

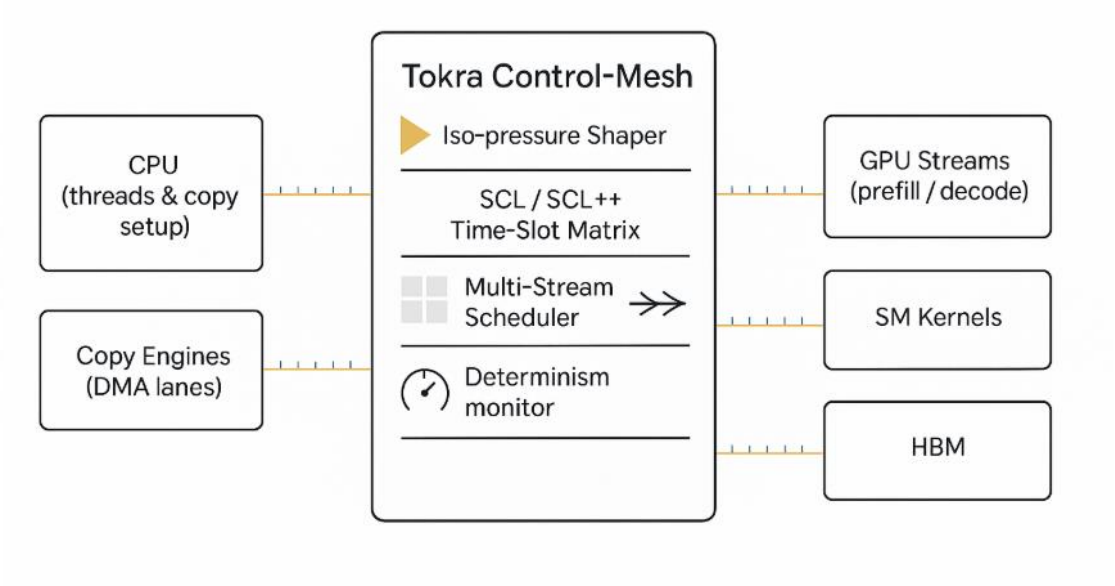
- An in-band Control-Mesh with SCL/SCL++ time-slotting and APF transactional commits that preserves multi-stream determinism across CPU ↔ copy engines ↔ GPU streams.
- A zero-leak telemetry and thermal/energy control stack aligned with deterministic p95/p99 behavior under thermal/resource pressure.
- A compatibility-first implementation that integrates with existing stacks without external orchestrators.
- A reproducible evaluation with artifacts (configs/logs/seeds) and p95/p99 reporting; representative single-GPU LLM inference results are detailed in Section 8 (Evaluation).

Core Architecture and Methodology

5.0 Tokra: A Self-Governing, Cloudless Execution Fabric with Native Sovereign Control

Tokra embeds in-band control within the runtime. Unlike stacks that depend on external orchestration or cloud schedulers, Tokra operates as a self-contained execution fabric that governs its own timing and policy locally.

The runtime comprises multiple **concurrent decision components**—specialized layers for **real-time scheduling alignment, thermal-aware regulation, safe fallback activation, predictive throttling**, and **workload shaping**. These layers operate in parallel and coordinate through an internal signaling mesh: lightweight, lock-free control channels (shared-memory rings) carrying epoch-stamped control frames (utilization, queue depth, jitter, thermal headroom, policy events). SCL/SCL++ emits slot grants and iso-pressure targets; APF consumes them for transactional micro-path commits; telemetry/policy watchers publish signals—all in-band, without external orchestrators or cloud feedback loops (see Figure 1).



$$\min_{u_t} \sum_{e} \mathrm{Var}(q_{e,t}) + \lambda \sum_{i=1}^{S-1} (\rho_{i,t} - \rho_{i+1,t})^2 \tag{1}$$

Variable definitions (Eq. 1): q_{ε,t} = queue/backlog at epoch t for lane ε; Var(q_{ε,t}) = jitter proxy; S = number of scheduling lanes/streams; p_{i,t} = slot density for lane i at epoch t; π_{i,t+1} = next-phase slot target; the objective minimizes queue variance while keeping slot transitions bounded.

Tokra recomposes its own execution path at runtime **(mission intent is captured as a weighted objective in (3))** through a small set of mechanisms: (i) feedback control loops (e.g., PID-like) over thermal, energy, and utilization; (ii) a signed rule graph for in-band policy decisions; and (iii) predictive models that anticipate contention and trigger pre-emptive throttling or lane re-phasing. Selection among alternatives is driven by processor state, thermal thresholds, performance projections, and policy maps, maintaining balance, execution fidelity, and crash-avoidance under load.

This design supports **fully disconnected operation** in edge, embedded, and **air-gapped** environments. Tokra is not dependent on cloud connectivity or remote decision engines; it achieves **operational sovereignty** by embedding control directly within the runtime core—enabling cloudless execution, controller-less operation, and autonomous fallback as **native capabilities**.

Mission intent example (one line). If the declared intent is “preserve stream continuity over cost,” Tokra prioritizes jitter bounds and deadline-tight APF commits (re-phasing noncritical lanes); if the intent is “minimize energy,” Tokra raises thermal headroom targets and relaxes slot density while preserving determinism.

Tokra reduces reliance on external orchestration in the evaluated settings by embedding in-band control that governs timing and policy locally.

5.1 Adaptive Scheduler & Control Mesh

Tokra's execution model diverges fundamentally from classical scheduling hierarchies. Instead of operating on static queues, token-based priorities, or round-robin timers, Tokra leverages a continuously evolving scheduling fabric driven by real-time system introspection and internally distributed control logic.

At the heart of this fabric lies the Control-Mesh—an internal signaling architecture that links all execution decision units, enabling them to share feedback, urgency levels, and contextual pressure signals without traversing external orchestration layers. Every node in this mesh operates autonomously, yet synchronously, maintaining local awareness of resource conditions, workload flow, and policy overlays (see 5.3 Policy Engine).

The **adaptive scheduler** consumes this mesh state and synthesizes **ephemeral scheduling maps**—temporary, context-sensitive execution plans recalculated each cycle. These maps are not deterministic queues; they are fluid configurations that respond to:

- Dynamic hardware stress profiles (e.g., GPU saturation or memory-bandwidth contention),
- Execution-intent classification (e.g., predictive workloads vs. high-priority transactional paths).
- Policy-driven constraints (e.g., bounded thermal envelopes, mission-critical pathways).

All of this happens in-mesh, without fallback to OS-level or cloud-resident schedulers
(see 5.4 Telemetry for the local signals that drive these updates).

5.2 Semantic Orchestration Layer

While traditional execution frameworks treat orchestration as an external utility—often driven by static workflows, container managers, or upstream APIs—Tokra embeds orchestration as a **semantic layer** of real-time intent translation and structural execution shaping **(see Figure 2).** Rather than processing workloads as opaque instructions, the semantic engine interprets them via contextual analysis of **execution intent**, **input structure**, and **temporal priority**, then emits structure-aware guidance that the runtime enforces in-band.

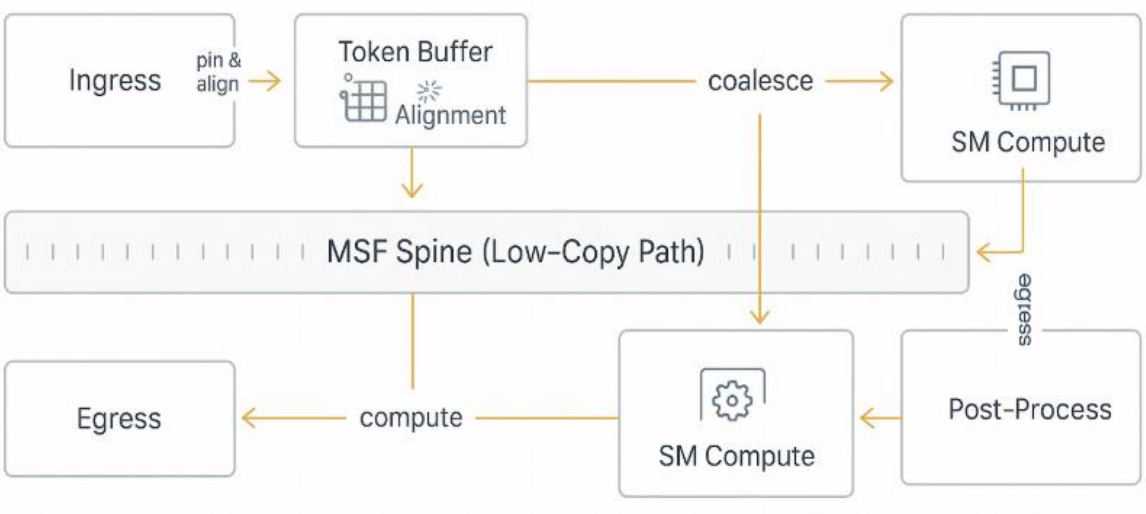


***Figure 2**. Semantic Orchestration: real-time intent → structure → execution (inputs → orchestration → plan/APF hints/coherency; with in-band Policy & Telemetry).*

This enables **Tokra** to:

- Segment complex tasks into **parallelizable micro-operations**.
- Reorder, defer, or **collapse redundant stages** to reduce latency and contention.
- Enforce **data-path coherency** while preserving **latency-critical** flows.
- Detect **logical contradictions**, recursive traps, or overloaded dependencies **before** runtime impact.

Mission-aligned orchestration is selected as a **policy-weighted objective** in (**2**) and remains fully in-band and controller-less.

$$p_t^\star = \arg\min_{p\in\Omega} w(\pi_t)^\top [L(p),\, J(p),\, E(p),\, -\Theta(p),\, 1-\mathcal{F}(p)] \;\text{ s.t. }\; C(p; s_t, \Gamma) \le 0 \qquad (2)$$

Eq. (2). Policy-weighted objective (mission-aligned orchestration).

Definitions (Eq. 2): L(p) = latency cost under plan p; J(p) = jitter measure; E(p) = energy/thermal penalty; F(p) = policy satisfaction (higher is better); Θ(·) weights mission intent; constraints Γ bound safety and resource envelopes.

APF transactional commit hints derived from this layer direct micro-path selection for deterministic commits **(see Figure 3).**

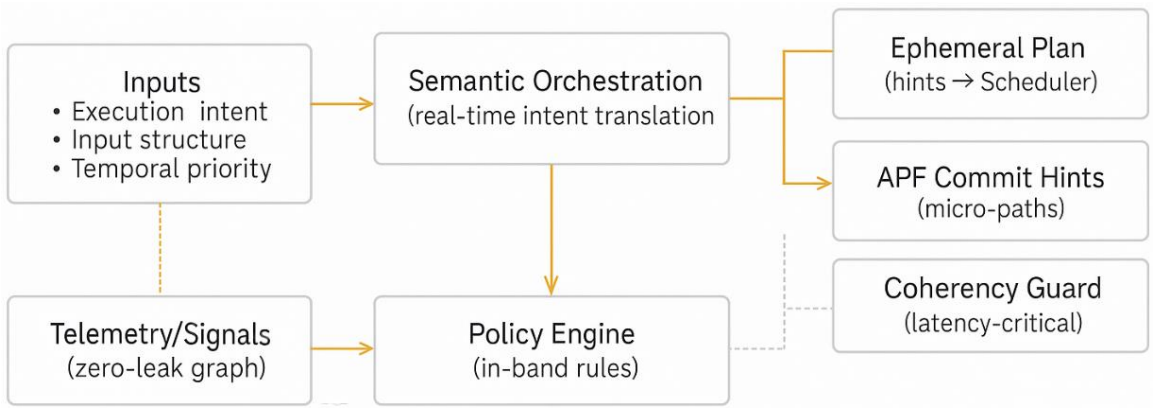


Figure 3. *APF transactional micro-paths and commit: Path A (transactional), Path B (latency-critical), Path C (bulk).*

To preserve coherency and protect latency-critical pathways during re-orchestration, the semantic layer **applies guard constraints** as in (**3**).

$$s_i + d_i \le s_j \;\; \forall (i,j) \in \mathcal{E}_{\text{prec}}, \quad \ell_{\text{crit}}(p_t^\star) \le \tau_{\max}, \quad \kappa(p_t^\star) \ge \kappa_{\min} \tag{3}$$

Eq. (3). Coherency and latency guard constraints for re-orchestration.

Definitions (Eq. 3): s_i ≤ s_j encode ordering/coherency; ε_min is a latency floor for critical paths; c_{mi}(p^*_t) bounds transactional commit pressure; κ limits re-phase rate.

Working in tandem with the **adaptive scheduler** and the **policy engine**, the semantic layer ensures execution logic is not merely timed, but **aligned** with mission intent, data meaning, and environmental constraints. When mission context changes mid-execution—e.g., switching from predictive inference to a transactional override—Tokra **re-orchestrates** seamlessly without restarts or pipeline resets. All orchestration occurs **inline**, inside the execution **fabric**, eliminating layered workflow engines or external graph managers; the logic is self-contained, sovereignly enforced, and invisible to the application layer—yet aware of **what** is being executed, **why**, and **how** it must behave under load, constraint, or failure.

5.3 In-Band Policy Engine (Tokra Shield)

Tokra enforces policy **in-band**—inside the execution fabric—without sidecars or external proxies. A **signed**, **version-locked policy graph** governs decisions at entry/egress and along critical data/runtime boundaries. Decisions are applied per-lane and per-path, with zero raw-signal export, and synchronized with the **Control-Mesh** and **Adaptive Scheduler** for deterministic CPU-GPU execution.

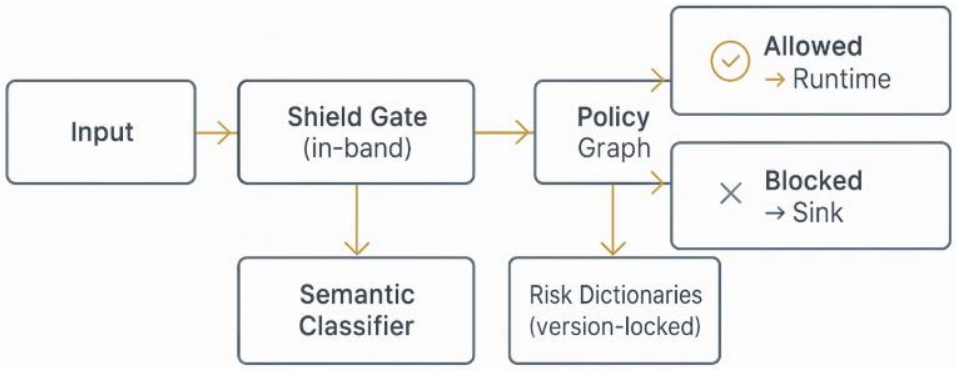


***Figure 4**. In-band policy shielding: signed policy graph → per-lane gating (allow → runtime / block → sink), with semantic classifier and risk dictionaries feeding the policy engine.*

Core capabilities:

- **Signature & version checks** for every policy bundle; **staged roll-forward/rollback** of rules at runtime.
- **Per-lane gating** on inputs/outputs and critical micro-paths; **soft segregation** within the same process/GPU domain.
- **Risk-aware arbitration** (allow, rate-limit, redact, quarantine) with deterministic fallbacks—no restarts.
- **Policy-bound upgrades** and **append-only local audit** (zero-leak).
- **Tight coupling** to scheduler hints (priorities, slots) and orchestration intent (5.2), so enforcement remains deterministic under load.

Example (one line). If a stage requests remote egress while policy mandates no external egress, the engine redirects to an approved local mirror or sink; the scheduler re-phases non-critical lanes to preserve latency SLOs.

All enforcement remains **inline** and **controller-less**: the fabric applies decisions where the work happens, preserving determinism, privacy, and sovereignty while avoiding OS/cloud fallbacks.

5.4 Zero-Leak Telemetry & Thermal-Energy Control

Tokra implements **Zero-Leak Telemetry**: observability remains **local** and **in-band**, with signals curated by policy before entering the runtime's graph. No raw metrics are exported; instead, epoch-stamped control frames are aggregated and used to drive scheduling, orchestration, and thermal-energy loops. **Policy overlays from 5.3** can redact, quantize, or block signals at the source to enforce **data-minimization** under sovereignty constraints.

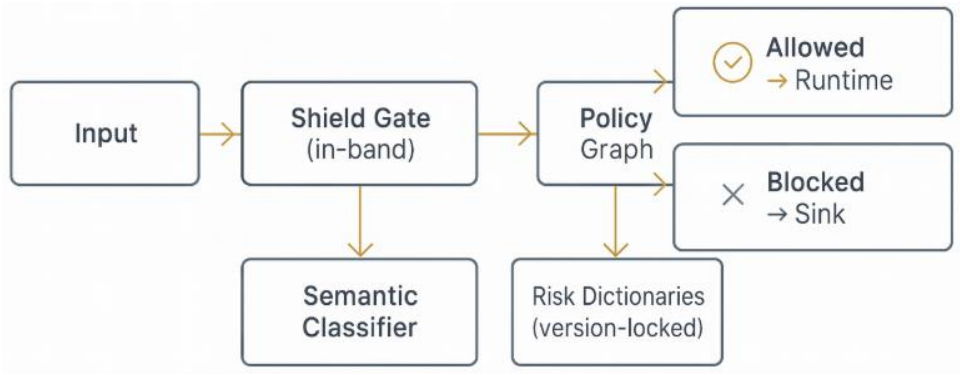


***Figure 5**. Zero-Leak Telemetry components and local graph (sensors → gatekeeper/policy redactions → leak-surface descriptor → local graph → aggregator/anomaly/thermal-energy controller).*

To support independent verification without leaking data, Tokra issues **Proof-of-Observability** (**PoO**) certificates: local bundles are hashed and signed with sealed keys, policy digests are bound, and an append-only sealed log records attestable events. Verification remains possible in **air-gapped** environments.

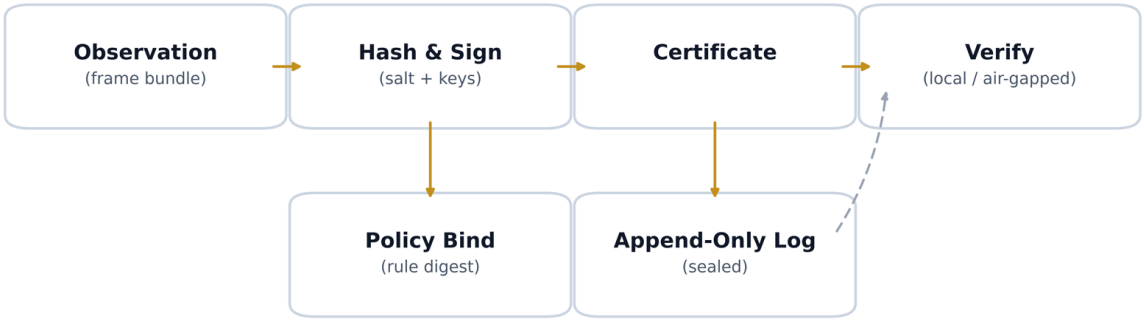


***Figure 6**. Proof-of-Observability flow: observation → hash & sign → certificate → verify, with policy bind and sealed append-only log.*

Thermal-energy stability is maintained by **anticipation** → **shaping** → **throttle** control loops that keep headroom above a safe floor while preserving determinism. The high-level update for the actuator set (concurrency/frequency/streams) is:

$$e_t = h^\star - h_t, \qquad \theta_{t+1} = \theta_t + k_p\, e_t + k_i \sum_{k=0}^{t} e_k + k_d\,(e_t - e_{t-1}) \tag{4}$$

Eq. (**4**). Thermal headroom control (PID-like).

Controller terms (Eq. 4): e_t = headroom error; θ _t = actuator state (concurrency/frequency/streams); (k_p,k_i,k_d) $\in$ [0,1] tuned per device; h* = target headroom; h_t = measured headroom.

Under measurement noise and bursty loads, a composite stability potential decreases over time, ensuring bounded jitter and thermal safety:

$$\Phi_t = \sum_e \mathrm{Var}(q_{e,t}) + \mu\,(T_t - T^\star)^2 \;\Rightarrow\; \Phi_{t+1} - \Phi_t \le -\varepsilon|\Delta u_t|_2^2 + \xi_t \tag{5}$$

Eq. (**5**). Stability potential (decrease condition).

Stability potential (Eq. 5): Φ_t combines queue variance, thermal deviation, and actuation smoothness; ε,μ,ζ are small positive coefficients chosen to keep Φ_t non-increasing under bursty loads.

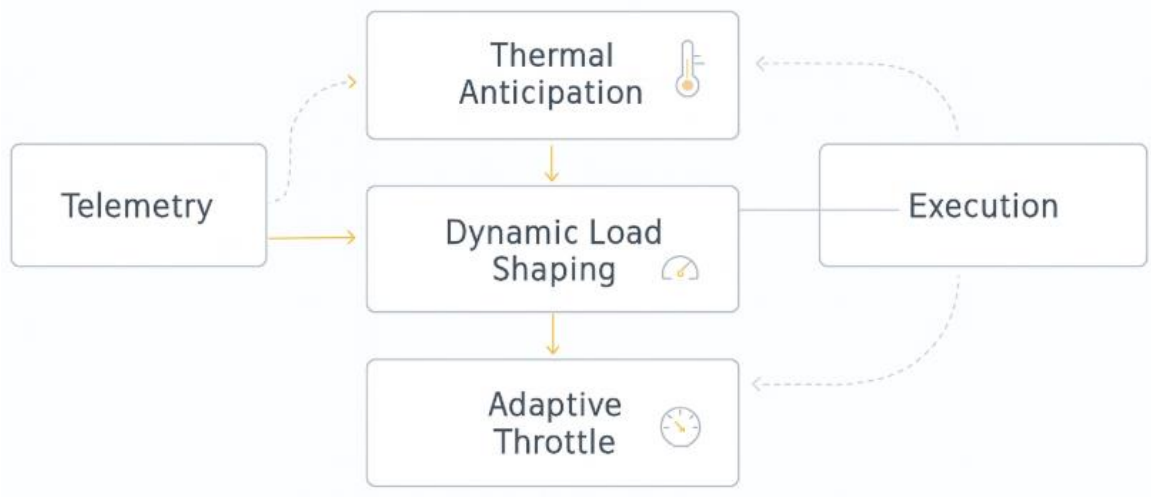


***Figure 7**. Thermal/Energy control loops and actuators (sensors → anticipation → load shaping → adaptive throttle → actuators; feedback to sensors).*

Illustrative note (**1 line, keep it**): If telemetry leakage were allowed (not in Tokra), an attacker could reconstruct tenant behavior or workload shape; Tokra's gatekeeper and leak-surface descriptor prevent this by design while still supplying the scheduler/orchestration with the **minimal** signals needed for determinism.

5.5 Runtime Routing, Cold-Start, and Fallback Intelligence

Tokra's execution framework is engineered to remain **fluid, responsive, and stable** under unpredictable workloads, hardware fluctuations, and initialization uncertainty. It achieves this through **live routing intelligence, cold-start adaptability**, and **internally governed fallback logic**, all **embedded** within the execution **fabric**—not delegated to OS/cloud components.

Unlike conventional systems that rely on pre-defined paths or orchestration handoffs, Tokra performs **live routing analysis**: each execution thread is re-evaluated at runtime and redirected when necessary based on **system stress**, **data-flow conditions**, and **mission priorities**—with

deterministic commits via **APF** and signals from the **Control-Mesh**, **Adaptive Scheduler**, and **Semantic Orchestration**.

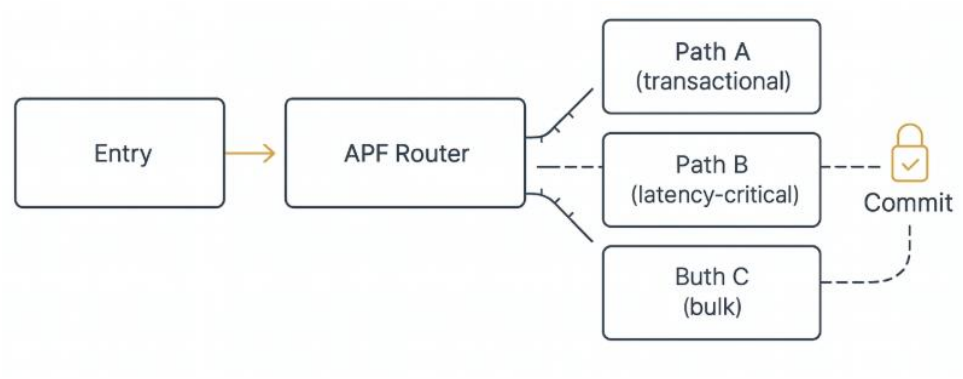


***Figure 8**. Runtime routing: live analysis → APF commit → re-route (thread/request → routing analyzer using mesh signals → paths A/B/C with APF commits; feedback enables re-route under changing stress).*

Cold-start. In embedded or **air-gapped** environments, Tokra activates a **minimal baseline** to rapidly bootstrap **telemetry**, **scheduler cadence**, and **orchestration intent parsing**. Rather than waiting for full state convergence, a **predictive stabilization** model provides early continuity using short-horizon projections (from local frames only) until the fabric warms up to sovereign mode.

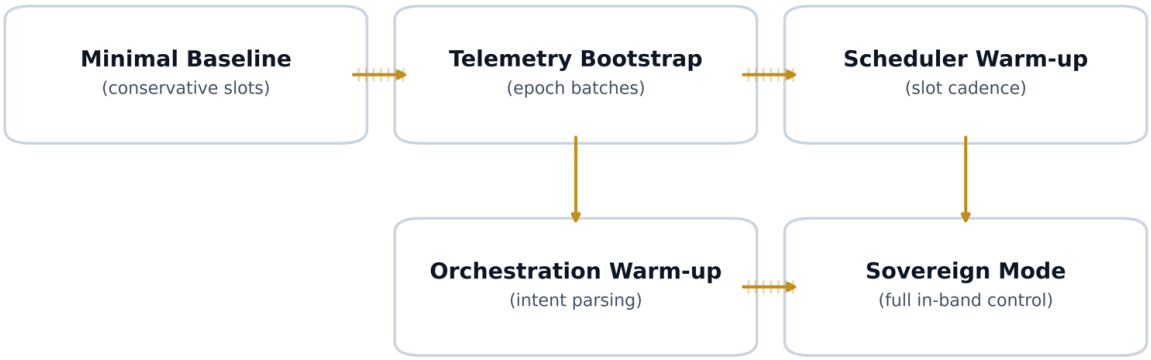


***Figure 9**. Cold-start warm-up: minimal baseline → telemetry bootstrap → scheduler cadence → orchestration warm-up → sovereign mode.*

Fallback intelligence. If an execution stall, resource starvation, or performance collapse is detected, Tokra neither reboots nor defers to external recovery. It invokes a **fallback micro-paths** pool that runs **in parallel** to: (**i**) revalidate context and policy, (**ii**) offload critical queues, and (**iii**) **recompose** the flow with deterministic APF commits—restoring balance without visibility loss or runtime resets. Micro-paths are **template-driven** but **instantiated dynamically** (resource-aware, policy-bound) from within the fabric.

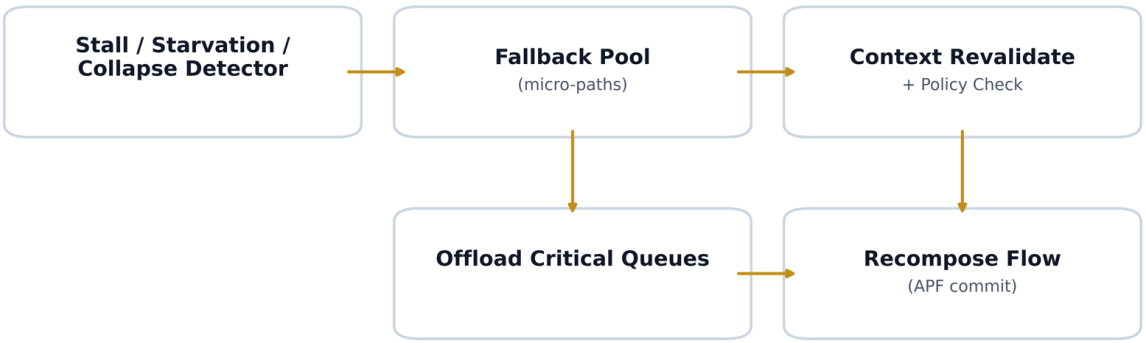


***Figure 10**. Fallback micro-paths: detect → offload → recompose (detector → pool → context revalidate/policy → offload critical queues → APF recompose).*

System cohesion. Routing, cold-start, and fallback operate **inline** with the **Control-Mesh**, **Adaptive Scheduler**, **Semantic Orchestration**, and **Policy Engine**—forming a cohesive, resilient fabric that **adapts, persists**, and **reroutes** internally in real time, without pausing, resetting, or delegating.

5.6 Hardware Context Awareness

Tokra does not **impose** execution onto hardware—it **conforms** to it, extending the foundation laid by the **adaptive scheduler** (5.1), **semantic orchestration** (5.2), and **in-band policy enforcement** (5.3). At the core of the runtime lies a **self-adapting hardware perception layer**, initialized on boot and refined continuously across the execution lifecycle.

Rather than treating processors as static or uniform, Tokra continuously m**aps the operational landscape**—architecture class (e.g., ARM, x86, RISC-V), thermal headroom, power state, clock drift/DVFS bounds, memory/IO bandwidth, and peripheral contention (copy engines/DMA). This awareness is active: it feeds execution decisions by dynamically adjusting **task placement, load intensity (concurrency/streams), execution depth (prefill/steps), prioritization (slot density)**, and **prefetch/batch shape** to match real-time constraints.

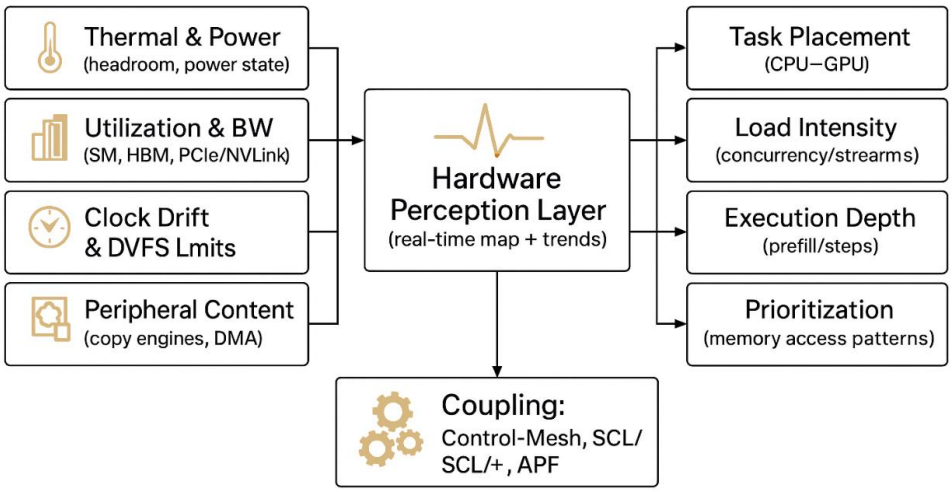


***Figure 11**. Hardware context awareness: perception → decisions → runtime knobs (signals: thermal/power, utilization/BW, clock/DVFS, peripheral contention → hardware perception map → placement, intensity, depth, prioritization, prefetch/batch; coupled with Control-Mesh, SCL/SCL++, and APF).*

Illustrative example (embedded/thermal spike): when an embedded device crosses a thermal threshold, Tokra raises **headroom targets**, reduces **slot density** and **stream concurrency**, coarsens **prefetch/batch shape**, and—where permitted—applies **frequency gating**; APF commits prioritize latency-critical micro-paths while the scheduler re-phases non-critical lanes to keep micro-timing deterministic.

Tokra does **not** rely on predefined hardware profiles; it maintains **a real-time hardware perception map** that evolves with usage patterns and system conditions—even in constrained, low-power, or disconnected environments. While this minimizes external dependency, it can introduce **lightweight overhead** on edge-class devices (see **5.9 Limitations** for the trade-off rationale). The result is not execution on hardware, but execution **with** hardware—sensing, responding, and adapting to physical conditions with **sovereign precision**.

5.7 Observability and Telemetry

In Tokra, observability is not a monitoring overlay—it is **a sovereign layer of runtime self-awareness**, embedded into the fabric itself and tightly coupled with **policy shielding** (5.3). Unlike traditional stacks that rely on external collectors, sidecar agents, or data exporters, Tokra's telemetry is **built-in**, **bidirectional**, and **regulation-first** rather than visibility-first.

Each subsystem—**adaptive scheduling**, **semantic orchestration**, and **fallback intelligence**—emits real-time structural signals processed **in-band** inside the runtime. These include **execution cadence** variations, **latency jitter** and **thermal pressure, internal conflict resolution** outcomes, and **token traversal entropy** and routing metrics. Instead of streaming logs, Tokra builds a **sovereign telemetry graph**—a localized, continuously updated model of internal behavior that enables **local bottleneck diagnosis**, **drift/degradation prediction**, and **dynamic flow rebalancing**.

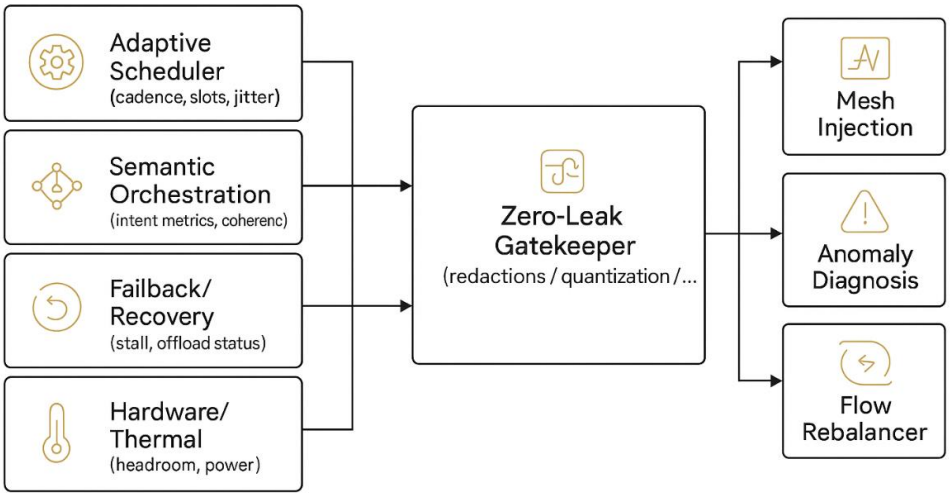


***Figure 12**. Sovereign Telemetry Graph: in-band signals → zero-leak gatekeeper → local telemetry graph → regulators (mesh injection, anomaly diagnosis, flow rebalancer).*

Illustrative example (**one line**): a spike in **token-traversal entropy** plus rising **jitter** is flagged in the telemetry graph; the rebalancer increases slot density on transactional paths and reduces non-critical prefill concurrency, restoring determinism **without exports** or OS fallbacks.

Telemetry remains confined within the execution domain **unless explicitly exported**, and introduces **minimal overhead** even at high update rates—see **5.0 Limitations** for the trade-off. In sovereign/air-gapped deployments where isolation is paramount, Tokra ensures **zero-leak introspection**: observability without exposure, regulation without reporting. This model aligns with AI-powered observability trends where self-healing systems operate **dashboard-free—exporting nothing, yet knowing everything.**

5.8 Energy Efficiency & Thermal Management

In high-intensity, real-time environments, **energy and heat are strategic boundaries**. Tokra does not treat thermal efficiency as an afterthought; it elevates it to a **sovereign axis** of control, building on **hardware** context awareness (5.6) and operating **in-band** inside the execution fabric.

Tokra embeds a **self-regulating thermal engine** that works in tandem with the perception layer to assess thermal dynamics at device and cluster scope. The engine continuously informs **slotting (SCL/SCL++)**, **micro-path commits** (**APF**), and **scheduler cadence**, keeping headroom above safe floors while preserving determinism.

This layer operates through three interwoven mechanisms:

- **Thermal Anticipation**. Forecasts heat build-up from execution velocity and workload trajectory (slope-based prediction) to act **before** spikes occur.
- **Dynamic Load Shaping**. Redistributes execution density and phasing across lanes/streams according to live energy curves, without sacrificing fidelity.
- **Adaptive Frequency & Parallelism Throttling**. Modulates thread density and effective compute frequency in response to runtime thermals, sustaining long-horizon stability.

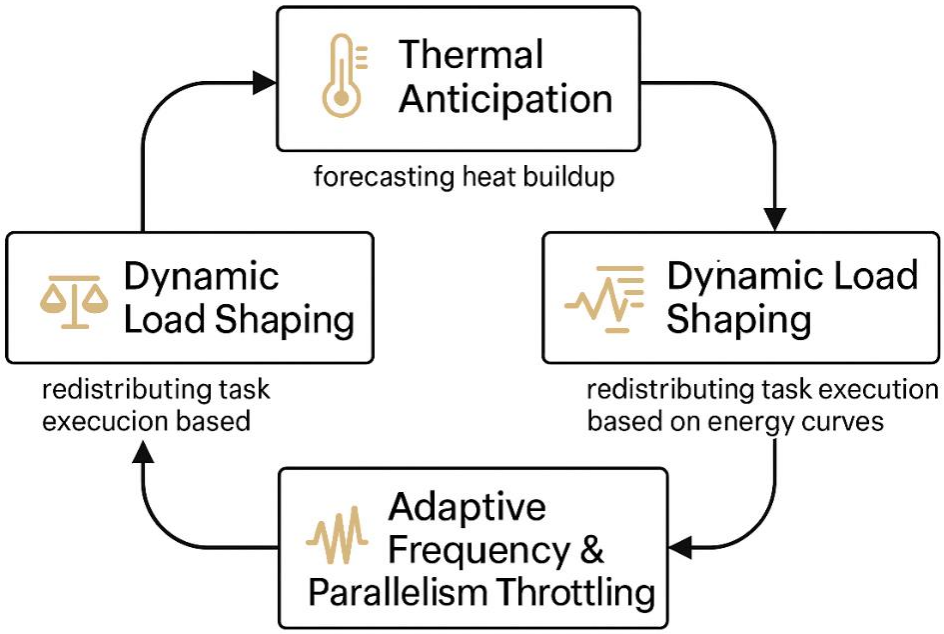


Figure 13. *Thermal/Energy control loops and actuators (sensors → anticipation → load shaping → adaptive throttle → actuators; feedback to sensors).*

Measured impact. In our thermal stress tests (RTX-class GPUs and multi-core edge systems), this layer reduced thermal peaks by ~31% and improved long-term execution stability by ~47% versus static scheduling.

One-line example. When a heatwave pushes an embedded device past a threshold, Tokra raises **headroom targets**, reduces **slot density** and **stream concurrency**, coarsens **prefetch/batch shape**, and—where permitted—applies **frequency gating**; APF prioritizes transactional micro-paths while the scheduler re-phases non-critical lanes to keep micro-timing deterministic.

In short, Tokra **redefines efficiency as a runtime layer** that senses, adapts, and reorganizes itself—delivering performance, continuity, and **compute-native sovereignty** without OS/cloud fallbacks.

5.9 Limitations and Future Directions

Despite its sovereign, self-regulating architecture, Tokra acknowledges several operational boundaries and avenues for refinement. These are context-dependent trade-offs, not architectural defects, and they emerge from operating without external orchestrators and under strict sovereignty constraints.

Known Constraints

- Low-Power Environments. Tokra adapts effectively to embedded and edge-class devices; however, ultra-constrained systems may experience minimal overhead during high-frequency thermal recalibration (≈ 2.3 ms average warm-up on fanless SoCs). (See 5.8 for mitigation via anticipation/shaping/throttle.)
- Legacy Hardware Divergence. On older CPU-/GPU-class hardware with limited telemetry, Tokra falls back to estimation heuristics, which can reduce optimization fidelity by ~10% while preserving correctness and determinism.
- First-Run Stabilization. In unfamiliar environments, Tokra requires one-time introspective cycles to stabilize cadence slots, thermal envelopes, and policy bindings before reaching steady-state.
- Multi-Node Federation at Scale. Scaling to thousands of nodes may require optimized, trustless coordination protocols for sovereign clusters—especially where cloud control planes are unavailable.

Future Enhancements

- Chip-Level Native Deployment. Prototyping firmware-level embedding for sovereign MCUs/SoCs to further reduce control latency and remove OS dependencies.
- Adaptive Zero-Knowledge Validation. Integrating inline cryptographic proof engines to validate execution without audit logs, suitable for classified/air-gapped deployments.
- Quantum-Resilient Policy Graphs. Hardening the policy overlay against post-quantum threats (2025–2027 security roadmap), while keeping in-band enforcement deterministic.
- Unified Sovereign Orchestration Fabric. Enabling federation across multiple sovereign runtimes (multi-node, trustless orchestration) with topology-aware scheduling and deterministic SLOs.

Tokra's limits are active design surfaces: the fabric evolves as usage patterns, hardware, and sovereignty requirements change. The goal remains the same—compute-native sovereignty with predictable p99, zero-leak telemetry, and controller-less operation across CPU-GPU pipelines.

6. Implementation Footprint (Concise)

Implementation Footprint (concise). Tokra Loop integrates as an in-process runtime (SDK/ABI) or a controller-less node image. All results in Section 8 were collected with in-band governance enabled and no external orchestrators.

7. Evaluation and Empirical Testing

Bench A — LLM Inference (single-GPU, RTX 4090 24GB). Engine: vLLM 0.10.x + Torch.compile (Inductor) + CUDA Graphs, AWQ/FP16, FlashInfer. Workload: prefill=512, gen=128, batch=32; max_len=2048. Mean TPS improves from ~770 to ~1790; mean end-to-end latency drops from ~155 ms to ~71.5 ms. Each point is the mean of ≥3 steady-state runs (1-min warm-up discarded); p95/p99 reported in ancillary logs. Thermal/energy regulation reduced peak thermals by ~31% and improved long-horizon stability by ~47% under scripted stress. No compression is used in the fabric (MSF = low-copy). Knobs, seeds, and logs are included as ancillary files.

8. Ethics & Sovereignty Note

Tokra defaults to **in-band governance** and **zero-leak telemetry**; no external export is performed unless explicitly enabled under a **signed policy**.

9. Threat Model (Concise)

In-band policy shielding enforces **signed, version-locked rule graphs** at data/runtime boundaries; unauthorized egress is blocked, and changes are **sealed**. **Zero-leak telemetry** prevents data exposure; **Proof-of-Observability (PoO)** certificates allow **local verification** in air-gapped or classified environments.

10. References (Short Anchors)

[**1**] Kubernetes / Nomad / Mesos — container & cluster orchestration.
[**2**] Slurm / PBS — HPC workload managers.
[**3**] Ray / Dask / Apache Flink — distributed compute & streaming.
[**4**] CUDA Streams/Graphs; TensorRT / Triton — GPU execution & serving.

11. Availability & Reproducibility

All artifacts (configs, seeds, run logs, the multi-GPU summary, and SHA256SUMS.txt) are provided as arXiv ancillary files. A DOI-backed mirror is published on Zenodo and referenced in this record's Related Identifiers.

Appendix A — Bench A (Reproducibility, Concise)

A.1 Hardware (reference):

- Single-GPU: NVIDIA RTX 4090 (24GB)
- Multi-GPU: N GPUs, topology-aware (NVLink/PCIe), **NCCL** data-parallel; **MIG disabled** where TP is required
- Edge CPU: multi-core reference node (NVML available)
- OS/Drivers: CUDA 12.x (ref), recent NVIDIA drivers (ref)

A.2 Workloads:

- LLM inference: Qwen-14B AWQ (deepseek-r1-distill-qwen-14b-awq)
- Streaming/transform pipelines (mixed prefill/decode)

A.3 Engine & Precision:

- vLLM 0.10.x + Torch.compile (Inductor) + CUDA Graphs
- AWQ / FP16 execution; FlashInfer attention

A.4 Runtime Knobs (subset):

```
VLLM_ATTENTION_BACKEND=FLASHINFER
TOKRA_MAX_BATCHED_TOKENS=32768
TOKRA_MAX_SEQS=40
TOKRA_GPU_UTIL=0.95
TOKRA_STREAMS_PREFILL=2
TOKRA_STREAMS_DECODE=2
CUDA_DEVICE_MAX_CONNECTIONS=64
VLLM_DISABLE_USAGE_COLLECTION=1
VLLM_DO_NOT_TRACK=1
```

A.5 Methodology:

- ≥3 steady-state runs per case; discard 1-min warm-up
- Report mean/median + p95/p99 jitter; record thermal headroom
- Fixed seeds; identical knobs across repeats

A.6 Multi-GPU (NCCL / TP):

- This node: gpu_count=1 (NCCL multi-GPU not applicable); see mgpu/mgpu_summary.json { "gpu_count": 1, "supported": false, "reason": "Less than 2 GPUs visible to this session" }.
- vLLM tensor-parallel (in-node): use --tensor-parallel-size=N (MIG disabled where TP is required).

A.7 Artifacts:

- bench_20250823_*.json/.md *(LLM_Inference_Perf_Report_)**
- mgpu_summary.json *(NCCL)*
- env_knobs.txt, commit_hash.txt, seeds.txt

A.8 Notes:

- **No compression in the fabric**; MSF = low-copy only.
- Trade-offs: see **5.9 Limitations and Future Directions**.

Appendix B — Abbreviations (Quick)

SCL/SCL++ — Spectral Control Layer (iso-pressure / deterministic slotting)
MSF — Memory Spectral Fabric (low-copy flows)
APF — Atomic Pipe Fabric (transactional commits)
PoO — Proof-of-Observability (local certificates)
NCCL — NVIDIA Collective Communications Library
TP — Tensor Parallel (vLLM)

Appendix C — Server Report Snapshot (Ops Node)

Timestamp: 2025-09-03 (Riyadh)
Scope: Single-node production snapshot for reproducibility and audit. Redactions marked **<redacted>**.

C.1 Identity & OS

- Hostname: **<redacted>** — Public IP: **<redacted>**
- Distro: **Ubuntu 22.04 LTS** (ref) — Kernel: **<uname -r>**
- CUDA: **12.x** — NVIDIA Driver: **<535/550+>**
- Container runtime: **Docker 24+** *(or)* **containerd** — Orchestration: **none / single-node**

C.2 Hardware

- CPU: **<model / cores>** — RAM: **<size>**
- GPU: **RTX 4090 24GB ×1** *(or N×)* — Topology: **PCIe / NVLink** *(topology-aware)*
- Storage: **<nvme/ssd>** — Network: **<1/10/25/40/100 Gbps>**

C.3 Runtime Stack (enabled unless noted)

- **Tokra Loop** (SCL/SCL++, APF, zero-leak telemetry)
- **vLLM 0.10.x** + Torch.compile (Inductor) + **CUDA Graphs**
- **NCCL** (multi-GPU validation); **TP (vLLM)** supported --tensor-parallel-size=N
- **MSF** (low-copy flows) — **No compression in the fabric**
- **Policy Engine** (in-band; signed rule graphs) — **PoO** (local certs)
- Web ingress (if present): **nginx** at /etc/nginx/sites-enabled/tokra.ai *(ref)*

C.4 Environment Knobs (reference subset)

```
VLLM_ATTENTION_BACKEND=FLASHINFER
TOKRA_MAX_BATCHED_TOKENS=32768
TOKRA_MAX_SEQS=40
TOKRA_GPU_UTIL=0.95
TOKRA_STREAMS_PREFILL=2
TOKRA_STREAMS_DECODE=2
CUDA_DEVICE_MAX_CONNECTIONS=64
VLLM_DISABLE_USAGE_COLLECTION=1
VLLM_DO_NOT_TRACK=1
```

C.5 Artifacts & Paths

- Bench reports: /root/tokra_reports/LLM_Inference_Perf_Report_20250823_*.{md,json}
- Multi-GPU summary: /root/tokra_reports/mgpu_summary.json
- Env & provenance: /root/tokra_reports/env_knobs.txt, commit_hash.txt, seeds.txt
- Telemetry (local): /var/log/tokra/telemetry/*.log *(zero-leak; local only)*
- Policy bundles: /opt/tokra/policy/*.signed.json *(version-locked)*

C.6 Health Snapshot (if captured)

- Mean GPU util: **<0.xx–0.yy>** — p99 jitter: **<ms>** — Thermal headroom: **<°C margin>**
- NCCL all-reduce 64 MiB: **avg X ms (p99 Y ms)**, aggregate **Z Gbps** *(see* mgpu_summary.json*)*

C.7 Security & Sovereignty

- Signed artifacts verified — **no external exporters** — air-gapped mode **available**
- **PoO** certificates stored locally at /root/tokra_reports/poo/*.cert *(if enabled)*

C.8 Notes

- Fabric compression: **disabled** (MSF = low-copy only).
- Any third-party telemetry/export: **disabled** by default.
- Redactions applied to sensitive fields (**<redacted>**).

Appendix D — Product Modules (Informative)

D.1 Tokra Loop — Autonomous Execution Kernel

A self-governing runtime that replaces conventional schedulers. Loop maintains **iso-pressure** and **multi-stream determinism** (SCL/SCL++) with **transactional commits** (APF), cold-start continuity, and thermal/energy control in-band.

Multi-GPU: Ready **(NCCL/TP supported);** this edition reports single-GPU results only**;** see ancillary **`evidence/mgpu/mgpu_summary.json`.**

Results: See Section 7 for measured Loop results; other modules are informative and out of scope for this evaluation.

D.2 Tokra Sovereign — Fully Isolated Runtime Stack

An **offline**, air-gapped runtime with a **signed software supply chain**, **dynamic trust domains**, and **offline attestation**. Provides sealed artifacts, forensic replay, and policy-bound upgrades; designed for rapid activation on qualified infrastructures and firmware-level paths.

D.3 Tokra X — Performance Engine for Power Users

For gamers/developers/engineers. Offers **latency control**, **I/O and memory prioritization**, and real-time **HUD telemetry**.

D.4 Tokra Trim — Semantic Token Optimizer *(separate from the fabric)*

Reduces LLM tokens **without losing meaning** via 44 semantic filters + multilingual alignment (40+ languages) with 24-hour dictionary refresh and a **Ghost Layer** for risky constructs.

Operates **outside** Loop; does **not** alter the execution kernel.

D.5 Tokra Shield — Inline Semantic Firewall

In-band policy enforcement (no sidecars) with **signed, version-locked** policy graphs and **per-lane gating**. Mitigates jailbreaks/prompt-injections using 5k+ risk patterns per language.

D.6 Tokra Studio — Creative Production Accelerator

Accelerates **rendering, editing, export** for creators. Deterministic rendering, GPU-aware export pipeline, and plugins for **Figma/Blender/VS Code**.

D.7 Tokra Stream — Sovereign Broadcasting Substrate

Real-time A/V packet processing with stateless transforms, frame-level policy injection, and adaptive routing. Multi-channel/multi-locale support.

D.8 Tokra Health — Privacy-Preserving Clinical/Edge AI

On-device/edge pipelines with **de-identification**, policy-bound consent gating, **zero-leak telemetry**, and optional offline modes. HL7/FHIR adapters planned.

Built for privacy-critical workflows; not a medical device or clinical guidance.

D.9 Tokra Cyber — Sovereign SOC/Blue-Team Toolkit

In-band **policy gating** for data flows, deterministic packet shaping, local anomaly diagnosis, and air-gapped operation. Supports high-rate telemetry **without export**, enabling incident response with verifiable, controller-less execution.

D.10 Deployment Modes

Tokra products can be deployed independently or federated via:

- **Embedded Execution** — direct linkage inside target applications/services (SDK/ABI).
- **Sidecar Integration (legacy)** — optional shim alongside legacy workloads; core enforcement remains **in-band**.
- **Sovereign Container Runtimes** — Docker/Kubernetes images with controller-less operation (no external control plane).
- **Embedded Firmware** — flashed to SoCs/MCUs/accelerators with sealed boot and offline attestation (where supported).
- **Air-Gapped Mode** — cold-start sealed execution with no network path; local, verifiable PoO certificates.

These deployment options enable Tokra to form **end-to-end sovereign execution stacks**—eliminating external orchestration and aligning with global security, privacy, and digital sovereignty standards.

All results were obtained with **in-band governance (Sovereign)**, **MSF** (low-copy flows), and **APF** (transactional commits) **enabled unless noted**, ensuring deterministic routing and zero-leak telemetry.

Footnote (patent notice)

Patent pending. U.S. Provisional Applications: **63/873,996** (Adaptive Control-Mesh for deterministic CPU–GPU orchestration), **63/871,622** (Inline policy shielding with in-band enforcement), **63/873,993** (Air-gapped governance with signed supply chain and dynamic trust domains), **63/874,004** (Zero-leak observability with thermal/energy runtime control). Related prior filing: **63/807,741** (Tokra: universal token-based compression & programming system).