SEKHMET: Hash-Chained Perception Contracts for Heterogeneous Real-Time Edge Clusters

Mohamed El-Hadedy
Department of Electrical and Computer Engineering, California State Polytechnic University, Pomona, CA, USA
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, IL, USA

Abstract

Real-time perception pipelines on edge clusters are often scheduled as ordinary latency-sensitive pods, even when safety depends on sustained throughput and stable model outputs. This paper presents SEKHMET (Scheduling Edge Kubernetes with Hash-chained Monitoring of End-to-end Telemetry), a perception-aware orchestration layer for lightweight Kubernetes (K3s) clusters that exports window-level perception status as a control-plane signal. SEKHMET evaluates a perception–integrity contract (PIC) once per fixed-duration window and commits each window outcome into a hash-chained perception root that is published to an otherwise unmodified K3s control plane. The prototype uses a Raspberry Pi 5 perception-root node with a Hailo-8L accelerator, USB camera, and GPS receiver running a YOLOv8s detector, while up to five additional nodes generate elastic interference via `swarm-stress`. Under contract-unaware scheduling with multi-node interference, the end-to-end perception loop delivers ∼0.8–2.2 FPS and violates the PIC timing requirement in most of 214 windows, despite apparently healthy CPU and memory metrics. Under the same and heavier interference, SEKHMET sustains 27–30 FPS with `contract_ok = True` across 400 protected windows while publishing one 96-byte record per T=5 s window (19.2 B/s control-plane payload). These results show that making perception requirements control-plane-visible can turn fragile best-effort perception into a protected cluster-level resource on commodity edge hardware.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Cloud computing

Keywords and phrases edge clusters, K3s, Kubernetes, real-time perception, scheduling, integrity contracts, hash chaining, Hailo-8L

Digital Object Identifier 10.4230/OASIcs.NG-RES.2026.5

Acknowledgements This research was funded by the U.S. Navy NEEC (Grant N001742310002) and the Office of Naval Research (ONR) Summer Faculty Research Program (SFRP). It was also supported in part by the U.S. Department of Defense under award W911NF-24-1-0265 and by the Air Force Research Laboratory (AFRL) under agreement FA8650-24-2-2403. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

1 Introduction

Edge-AI systems increasingly run on small heterogeneous clusters that keep perception, localization, and control close to the physical process. Camera inference is often pushed onto low-power neural processing units (NPUs) such as Hailo-8L, GPS fixes provide context, and lightweight Kubernetes distributions such as K3s manage containers across Raspberry-Pi-class nodes and small servers. In this setting, the perception loop is mission-critical and easy to destabilize: it must sustain stable throughput and consistent outputs while sharing CPU time, memory bandwidth, I/O, and thermal headroom with elastic background work.

The limiting factor is not Kubernetes scheduling itself, but the signals available to the control plane. Kubernetes natively reasons about resource and health indicators (CPU, memory, network, and pod `Ready` status), yet it cannot directly express application obligations such as (i) minimum effective frame rate over a time window, (ii) bounded variation in task-level outputs (e.g., people count), or (iii) freshness of context aligned to inference. Under interference, perception can degrade while pods remain `Ready` and coarse utilization looks acceptable, leaving the control plane without a correctness-relevant signal to gate interference-increasing actions.

Perception contracts provide a principled language for stating such obligations. Prior work connects bounded perception error to downstream safety properties in ML-enabled control systems [2], refines contract reasoning for vision-based auto-landing [6], and characterizes operating design domains using Lyapunov-style conditions [5]. These mechanisms are typically enforced within a single device or vehicle, while cluster scheduling decisions about placement, co-location, and scale-out remain driven by generic resource metrics that are weak proxies for perception correctness.

Complementary systems and security work strengthens trust in execution and sensing. Execution-integrity and sensor-attestation mechanisms such as EILID [3] and IoT Notary [8] help establish what executed and what was captured. They do not, however, represent whether perception stayed within application-defined bounds under resource contention, nor are their artifacts intended as continuously refreshed inputs to orchestration decisions.

SEKHMET bridges this gap by making window-level perception compliance actionable at the control plane. A designated perception-root node evaluates a perception–integrity contract (PIC) once per fixed-duration window and commits contract-relevant summaries and the outcome into a compact, hash-chained perception root. The latest root is exported via standard Kubernetes interfaces so that a lightweight controller can conservatively gate elasticity: interference-increasing actions such as co-location, scale-out, or opportunistic placement are permitted only while recent windows remain compliant and fresh.

We evaluate SEKHMET on a heterogeneous K3s testbed with a Raspberry Pi 5 perception-root node (USB camera, Hailo-8L NPU, GPS receiver) and up to five worker nodes running elastic `swarm-stress` workloads. The results show that exporting contract status to the control plane preserves real-time perception behavior under sustained multi-node interference without modifying K3s, the container runtime, or accelerator firmware. Our focus is resource-driven interference and mis-scheduling; full physical compromise of the perception-root node is outside the threat model. SEKHMET contributes (i) a cluster-scoped interface that exports window-level perception compliance as a control-plane signal, (ii) compact hash-chained perception roots (96 bytes per window in the prototype) that serve as tamper-evident compliance tokens, and (iii) an end-to-end implementation and evaluation on a Raspberry Pi 5 + Hailo-8L K3s cluster demonstrating that contract status can constrain interference and protect perception under contention.

2 Background and Motivation

Figure 1 highlights a recurring failure mode in edge clusters: a perception pipeline can drift out of its required operating regime without triggering the signals the control plane normally monitors. In a conventional deployment (left), the perception workload runs as a standard pod on a designated node while other nodes host elastic workloads such as batch analytics, telemetry relays, and software updates. K3s responds to demand using generic health and resource indicators (CPU, memory, and `Ready` status). As load shifts, the cluster

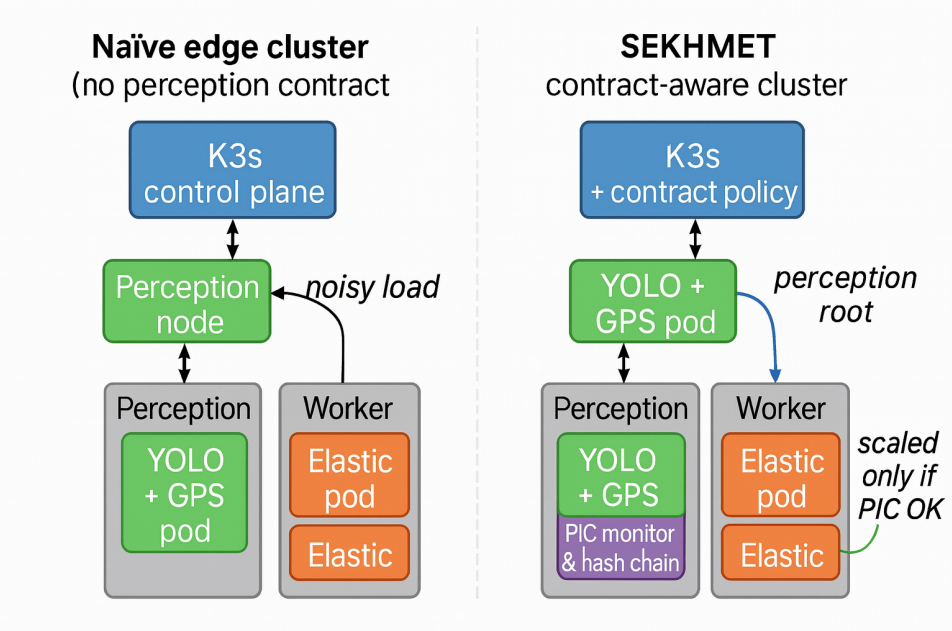

Figure 1 Naïve edge cluster (left) vs. SEKHMET (right). In a conventional K3s deployment, elastic workloads can contend with the perception pipeline and degrade effective frame rate, output stability, and context freshness. SEKHMET adds a lightweight monitor that emits a hash-chained root; the control plane uses this signal to constrain admission, scaling, and migration of elastic pods while the contract holds.

can introduce interference through co-location, shared-resource contention, and short bursts of activity. From the application's perspective, the perception loop degrades – throughput drops, context becomes stale, and outputs become less stable – even though pods remain healthy and no obvious resource limits appear violated. Section 6 quantifies this gap on our testbed.

SEKHMET (right) makes this degradation visible and actionable. A lightweight monitor runs alongside the perception pipeline on the perception-root node and emits a compact, per-window, hash-chained root that records whether the window satisfied the perception–integrity contract. The control plane consumes the latest root via standard Kubernetes interfaces and uses it to gate interference: placement and elasticity actions that would increase contention are permitted only while recent windows remain compliant. This reframes perception from a best-effort workload into a protected cluster resource with an explicit, auditable status signal.

This design addresses a blind spot across several research threads. Perception-contract work [2, 5] formalizes correctness obligations, but typically enforces them within a single device and does not influence cluster orchestration. Execution-integrity and sensor-attestation mechanisms [3, 8] establish what ran and what was captured, but do not report whether perception behavior stayed within acceptable bounds under contention. Edge-orchestration research [1, 9, 11, 12] improves coordination and placement in constrained clusters, yet still treats perception as an ordinary workload without a semantics-aware signal tied to model behavior.

The next sections describe SEKHMET's architecture and control-plane integration (Section 3), define the perception–integrity contract checked per window (Section 4), present the implementation on heterogeneous edge hardware (Section 5), and evaluate behavior under multi-node interference (Section 6).

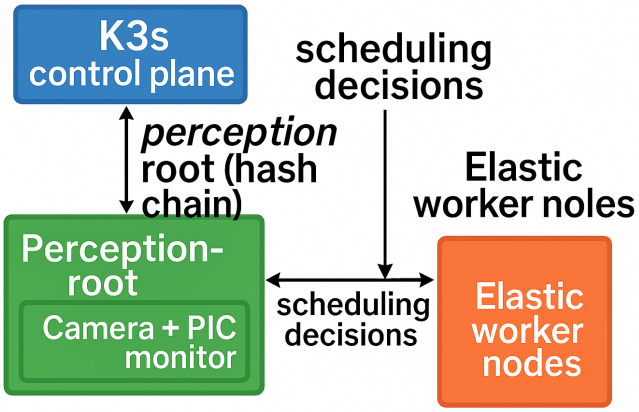

Figure 2 Cluster-level architecture of SEKHMET. A perception-root node runs the protected perception pipeline and a monitor that emits a hash-chained perception root. A lightweight controller publishes the latest root to the (unmodified) K3s control plane and applies policy that gates elasticity when the root indicates non-compliance or becomes stale.

3 System Architecture of SEKHMET

SEKHMET adds a contract-aware control loop to a standard K3s edge cluster (Figure 2). One node is designated as the *perception-root node* and reserved for the safety-critical perception pipeline; the remaining nodes act as elastic workers for best-effort workloads. K3s, the container runtime, and accelerator firmware remain unchanged. The objective is not to replace Kubernetes scheduling, but to provide a scheduler-actionable signal that reflects perception health rather than inferring safety from CPU and memory utilization alone.

At a fixed cadence, the in-pod monitor on the perception-root node emits a compact *perception root*: a fixed-format record that summarizes measurements for the most recent window and links to the prior window by including the previous hash. Each root commits to three classes of evidence: timing summaries (effective throughput and latency statistics), a task-level semantic summary derived from the model outputs (e.g., people-count variation), and context freshness summaries (e.g., GPS age). A lightweight controller exposes the latest root to the control plane using standard Kubernetes objects (e.g., a dedicated custom resource or ConfigMap updated once per window) and translates root status into ordinary scheduling constraints using existing primitives such as node labels, taints/tolerations, and workload placement rules. Table 1 lists the per-window fields committed by the monitor and the subset consumed online by the controller.

Policy and conservative gating

Elastic actions are permitted only while the recent root stream remains both compliant and fresh. We treat the root as *fresh* if the control plane observes an update within one window duration ($\leq T$) of its expected arrival; otherwise elasticity is blocked until updates resume. When a window fails, the controller blocks interference-increasing actions and can reduce elastic pressure using standard Kubernetes mechanisms (e.g., scaling down selected deployments, evicting lower-priority pods where configured, or applying throttling when available). The policy is fail-closed for elasticity: if roots stop arriving within the expected cadence (monitor failure, sensor loss, or timeout), scale-out and other interference-increasing actions remain blocked until valid updates resume.

Table 1 Per-window perception-root fields. The monitor commits a fixed-format record per window and hashes it into a chained root. The controller consumes freshness and compliance fields online; detailed summaries can be retained locally for offline analysis.

Field	Meaning (committed per window)	Controller uses?
`ver`	Record format/version identifier for forward compatibility.	No
`k`	Window index (monotone counter).	Yes (freshness/order)
`t_start`, `T`	Window start time and duration (e.g., T=5 s).	Yes (freshness)
`node_id`	Identifier of the perception-root node (stable name/UID).	Optional (sanity)
`timing_sum`	Timing summary or digest (e.g., effective FPS; latency stats such as P50/P95, or a hash of these values).	No (optional audit)
`semantic_sum`	Semantic-stability summary or digest (e.g., people-count min/max or Δ_{people}, or a hash).	No (optional audit)
`context_sum`	Context-freshness summary or digest (e.g., GPS-age statistics, or a hash).	No (optional audit)
`contract_ok`	Boolean window outcome (all enforced obligations satisfied).	Yes
`prev_hash`	Hash pointer to the prior window record (chain linkage).	Optional (audit)
`root_hash`	Hash of the current record (the published root identifier).	Yes (integrity token)

Figure 3 shows the per-window workflow. For each window, the monitor computes a bounded set of aggregates, evaluates the window outcome, and encodes the result into the next record in the chain. Hash chaining makes post-hoc modification of retained history evident. If deletion-detection is required, periodic off-node checkpointing can make missing history externally observable.

This division keeps measurement and root generation local to the perception-root node, while orchestration remains in the control plane – now guided by a semantics-aware signal rather than resource accounting alone. The next section formalizes the contract checked by the monitor (Section 4).

4 Perception–Integrity Contracts

SEKHMET evaluates perception over fixed-duration time windows. Let $T > 0$ denote the window length and S the step; window k is $W_k = [t_k, t_k + T)$. The prototype uses tumbling windows ($S = T$) with $T = 5$ s; the same definitions apply to sliding windows ($S < T$).

Within W_k, the perception-root node observes time-stamped camera frames $F_k = \{(t_i, x_i)\}_{i \in I_k}$ and corresponding model outputs $Y_k = \{(t_i, y_i)\}_{i \in I_k}$. It also observes a GPS-fix stream $G = \{(t_j, g_j)\}_{j \in J}$ that is not window-restricted so that each frame can be paired with the most recent fix even if it occurred before t_k. The output y_i is treated abstractly: it may be scalar (e.g., people count) or vector-valued (e.g., a class histogram). The prototype uses a people-count signal as a concrete instantiation.

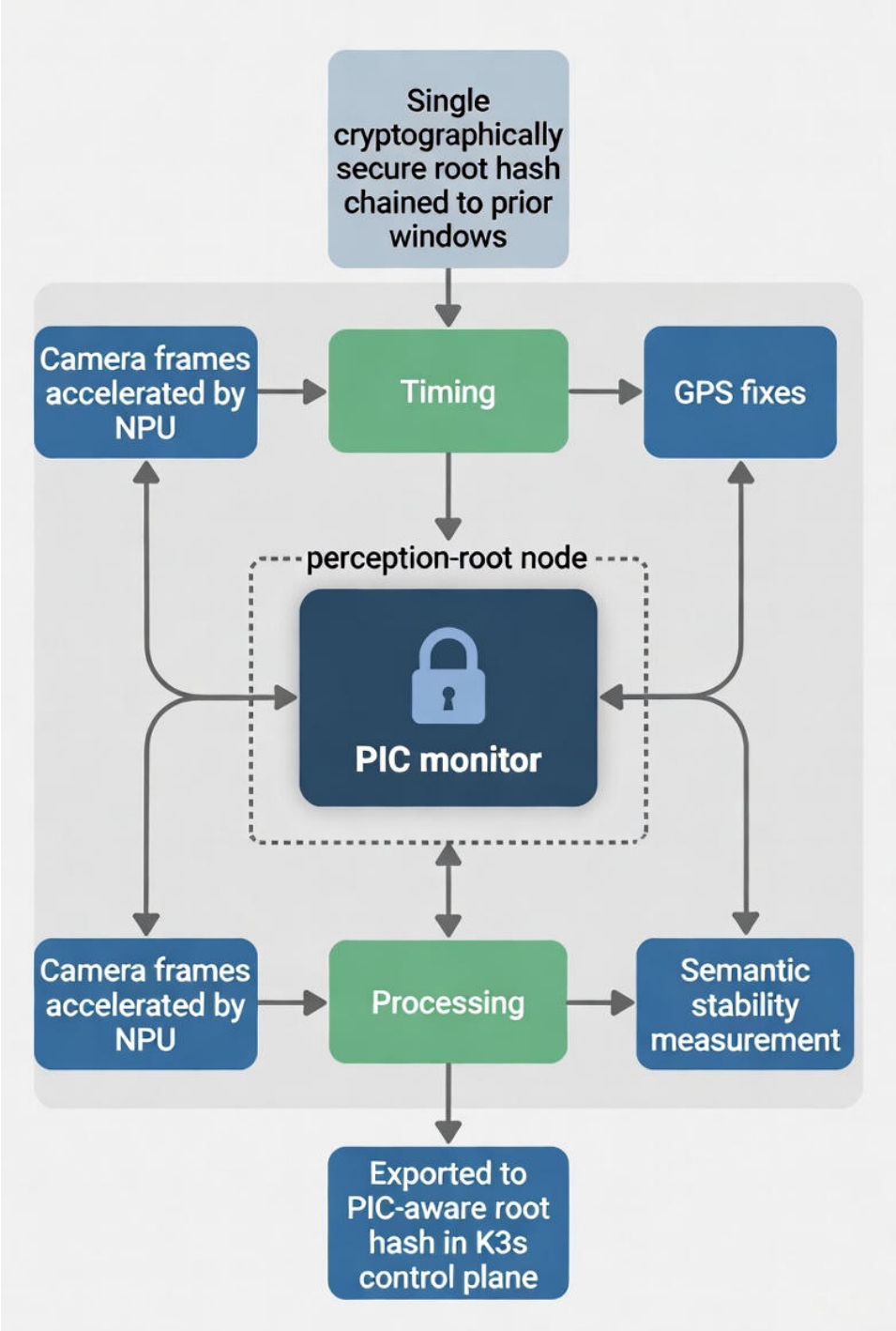

Figure 3 Per-window operation of the monitor. Camera frames and GPS fixes are summarized into timing, semantic-stability, and context-freshness statistics. The monitor evaluates the window outcome and hashes a fixed-format record with the previous hash to produce the next perception root.

A PIC holds on window W_k when three obligation classes are satisfied: timing, semantic stability, and context completeness. The timing obligation requires sustained throughput and bounded latency. The achieved frame rate is $f_k \triangleq |I_k|/T$; if $I_k = \varnothing$ then $f_k = 0$ and timing fails. Let ℓ_i denote end-to-end latency for frame i (capture to inference completion, measured on the perception-root node). A typical instantiation requires $f_k \geq f_{\min}$ together with either a per-frame bound $\ell_i \leq L_{\max}$ for all $i \in I_k$ or a tail constraint such as $\mathrm{P}_p(\{\ell_i\}_{i \in I_k}) \leq L_{\max}$ for a chosen percentile p.

The semantic-stability obligation bounds variation in outputs within the window. In a generic output space, one form is $\max_{i,j \in I_k} \|y_i - y_j\| \leq \Delta_{\mathrm{sem}}$, where $\|\cdot\|$ is an application-chosen norm consistent with the output representation. For a scalar signal such as people count, this reduces to a bounded within-window swing $(\max_{i \in I_k} y_i - \min_{i \in I_k} y_i) \leq \Delta_{\mathrm{people}}$.

The context-completeness obligation ensures that each inference is paired with sufficiently fresh metadata. For each frame time t_i, define the most recent GPS-fix time at or before t_i as $\hat{t}_i \triangleq \max\{\, t_j \;:\; (t_j, g_j) \in G,\ t_j \leq t_i \,\}$. The GPS age is $a_i \triangleq t_i - \hat{t}_i$, and if no such fix exists then $a_i \triangleq +\infty$. Context completeness requires $a_i \leq A_{\max}$ for all $i \in I_k$ (or under an agreed tail rule).

A window is contract-compliant when all three obligations hold: $C(W_k) \triangleq O_{\mathrm{time}}(W_k) \wedge O_{\mathrm{sem}}(W_k) \wedge O_{\mathrm{ctx}}(W_k)$. SEKHMET exports this compliance signal to the control plane and treats it as a hard constraint for elasticity: interference-increasing actions are permitted only while the most recent m windows satisfy $C(W_k)$ (configurable m). When a window fails, the policy loop blocks further interference and may shed elastic load to protect the perception pipeline.

At the end of each window, the monitor binds contract-relevant summaries into a compact, hash-chained perception root that commits to (k, t_k, T), digests of the timing/semantic/context summaries, a boolean `contract_ok` indicating $C(W_k)$, and a pointer to the previous-window hash. The resulting chain yields tamper-evident, per-window compliance tokens that an otherwise unmodified K3s control plane can consume as an admission and scheduling signal.

5 SEKHMET on a Heterogeneous Edge Cluster

SEKHMET is implemented on a lightweight K3s cluster with one designated *perception-root node* and up to five worker nodes that host elastic interference workloads. K3s remains unmodified: pods, services, and metrics collection operate normally. SEKHMET adds two components: (i) a perception pod that bundles the perception pipeline with an in-pod monitor, and (ii) a lightweight controller that maps per-window contract outcomes to standard Kubernetes placement and scaling constraints. To support reproducibility, each experiment run records a manifest containing the OS release, `k3s -version`, container runtime, Hailo runtime and software development kit (SDK) version, the YOLOv8s build and weights identifier, and the instantiated contract parameters (e.g., T, $f_{\min}$, and Δ_{people}). The manifest is stored alongside exported perception roots and per-window logs.

The perception-root node is a Raspberry Pi 5 connected to a Hailo–8L accelerator, a USB camera, and a serial GPS receiver. The protected perception pipeline runs YOLOv8s object detection accelerated on the Hailo–8L. Each frame is time-stamped at capture and again when inference completes, yielding per-frame end-to-end latency samples. Per-frame detections are reduced to a compact semantic signal (people count in this prototype). Camera frames are captured at a fixed resolution and resized to the detector input size prior to

inference; both the capture resolution and inference input dimensions are recorded in the manifest. GPS fixes are read from the serial device, parsed into time-stamped records, and exposed to the pod so that per-frame freshness can be checked at the window boundary.

The monitor runs in the same pod as the perception pipeline so that contract checks are derived from exactly the timestamps and outputs visible to the application. It evaluates the window-level obligations from Section 4 using tumbling windows with $T = 5\,\mathrm{s}$ and maintains a bounded set of summaries per window: effective frame rate and latency statistics, variation of the semantic signal, and GPS age relative to frame times. These summaries determine the window outcome. For each window, the monitor encodes the window identifier and summary digests into a fixed-format record and computes the next perception root by hashing the record with the previous-window hash, forming a per-window hash chain. Full window records are appended to a local log for offline analysis; online enforcement requires only the most recent root and its `contract_ok` bit.

Scheduler visibility is achieved through standard Kubernetes APIs rather than changes to K3s. The monitor publishes the latest root once per window by updating a namespaced ConfigMap reserved for SEKHMET, and the controller watches this object for new roots and freshness violations (a custom resource can replace the ConfigMap without changing the monitor logic). Freshness is defined as in Section 3; on stale or missing updates the controller fail-closes by blocking interference-increasing actions until updates resume. To enforce *no co-location* on the protected node, the perception-root node is tainted (e.g., `NoSchedule`) and only the perception pod carries the corresponding toleration; elastic workloads are deployed without that toleration and are therefore ineligible for placement on the perception-root node. Elastic workloads are admitted and allowed to scale only while recent windows remain compliant and the root remains fresh; on non-compliance or missing updates, the controller blocks interference-increasing actions, including additional elastic placement and further elastic scale-out.

For experiments, the monitor logs per-window summaries and outcomes, and the controller logs the actions it applies. These traces are paired with standard K3s resource metrics (via `metrics-server`) to produce the results reported in Section 6.

6 Evaluation

SEKHMET is evaluated on a heterogeneous K3s edge cluster with one dedicated perception-root node and up to five worker nodes. The perception-root node is a Raspberry Pi 5 equipped with a Hailo-8L accelerator, a USB camera, and a serial GPS receiver. The perception workload runs YOLOv8s object detection accelerated on the Hailo-8L. Cluster interference is generated with `swarm-stress`, a mix of CPU-, memory-, I/O-, and network-intensive workloads deployed as ordinary Kubernetes pods.

The perception–integrity contract (Section 4) is checked on non-overlapping windows of length $T = 5\,\mathrm{s}$. For the experiments in this section, the contract is instantiated with two enforced obligations: (i) minimum effective frame rate $f_{\min} = 10\,\mathrm{FPS}$ and (ii) bounded within-window variation in detected person count $\Delta_{\mathrm{people}} \leq 3$. A window is marked `contract_ok = True` only when both conditions hold. GPS fixes are collected and time-stamped throughout all runs; GPS is logged for traceability but not enforced in this evaluation, so reported violations reflect only the FPS and people-count checks.

Table 2 summarizes the five experiments. E1 and E2 capture contract-unaware behavior (93+121=214 windows total). In E1, the perception pipeline runs as a standard K3s pod on the Hailo-8L node for 93 windows ($93 \times 5\,\mathrm{s} = 465\,\mathrm{s} \approx 7.8\,\mathrm{min}$) without deploying `swarm-stress`.

Table 2 Evaluation scenarios (E1–E5). "Protected" indicates that SEKHMET policy prevents elastic co-location on the perception-root node and blocks interference-increasing actions when the contract is violated or the perception root becomes stale.

Exp.	Mode	Stress placement	Windows	Outcome (perception node)
E1	Unaware	None (no `swarm-stress`)	93	0.8–2.2 FPS; timing fails often
E2	Unaware	Co-located on perception-root node	121	0.8–2.2 FPS; persistent failures
E3	Protected	Workers only (moderate)	81	27–30 FPS; all windows OK
E4	Protected	Workers only (higher intensity)	118	27–30 FPS; all windows OK
E5	Protected	Workers only (heaviest/longest)	201	27–30 FPS; all windows OK

In E1 the perception-root node is not reserved, so routine system/cluster services and best-effort pods can still co-locate and intermittently contend for CPU, memory bandwidth, and I/O, which is sufficient to collapse end-to-end FPS even without explicit stress deployments. Even in this setting, end-to-end throughput at the application boundary is typically 0.8–2.2 FPS, causing the timing obligation to fail in most windows. This baseline reflects the full in-cluster loop (capture → inference completion → reporting) under normal K3s services and background activity, rather than an accelerator-only microbenchmark. In E2, `swarm-stress` pods are co-located on the perception-root node (121 windows, $121 \times 5\,\mathrm{s} = 605\,\mathrm{s} \approx 10.1\,\mathrm{min}$), producing similarly low effective FPS and persistent contract failures. Across E1 and E2, standard Kubernetes signals are largely non-diagnostic: pods can remain `Ready` and within nominal limits while contract-relevant timing and output stability degrade.

E3–E5 enable SEKHMET and use contract status to prevent elastic co-location and restrict elasticity actions that would increase interference on the perception-root node (81+118+201=400 protected windows total). In E3 (81 windows, $81 \times 5\,\mathrm{s} = 405\,\mathrm{s} \approx 6.8\,\mathrm{min}$), `swarm-stress` runs on worker nodes at moderate intensity; the perception workload operates at 27–30 FPS, person-count variation stays within the bound, and all collected windows satisfy the enforced obligations. E4 increases stress intensity and pod count across the worker pool (118 windows, $118 \times 5\,\mathrm{s} = 590\,\mathrm{s} \approx 9.8\,\mathrm{min}$) and preserves the same contract-compliant regime on the protected perception-root node. E5 is the longest and heaviest setting (201 windows, $201 \times 5\,\mathrm{s} = 1005\,\mathrm{s} \approx 16.8\,\mathrm{min}$) with sustained multi-node stress; the perception workload remains at 27–30 FPS with low people-count jitter, and no contract violations are observed in the collected windows.

At the end of each window, the monitor exports a 96-byte, hash-chained perception root summarizing the window measurements and `contract_ok`. With $T = 5\,\mathrm{s}$, this corresponds to a steady control-plane payload of $96/5 = 19.2\,\mathrm{B/s}$ (one update per window), which is small relative to typical cluster control traffic. The control plane uses root freshness and contract status as an execution-time signal: elastic workloads are permitted to scale or migrate only while recent windows remain compliant, and placement that would increase interference with the perception pipeline can be blocked.

Overall, these scenarios show that conventional edge-cluster scheduling can violate modest perception obligations while appearing healthy under standard resource and pod-status metrics. In contrast, exporting per-window contract status as a control-plane signal keeps the perception workload in its expected operating regime (27–30 FPS on this testbed) under sustained multi-node interference. No changes are required to K3s, the container runtime, or the accelerator firmware, and the user-space monitor does not measurably alter steady-state FPS in the protected configuration while emitting one fixed-size record per 5 s window.

7 Related Work

Perception contracts formalize obligations that connect perception quality to downstream safety properties. Astorga et al. introduced perception contracts to bound perception error and relate it to safety in ML-enabled control systems [2]. Li et al. refined these ideas for vision-based auto-landing [6] and later developed Lyapunov-based formulations that characterize operating design domains [5]. This line of work primarily supports per-platform verification and control synthesis. SEKHMET operates at a different layer: it treats contract outcomes as an execution-time signal that constrains cluster orchestration under contention, rather than as an analysis artifact confined to a single embedded platform.

Execution-integrity and sensor-attestation mechanisms establish trust in what executed and what was captured. EILID provides lightweight control-flow attestation for constrained IoT devices [3], and IoT Notary captures tamper-evident sensor readings before they reach applications [8]. These mechanisms do not encode perception-specific obligations such as sustained window-level throughput or bounded variation in model outputs under interference, and their artifacts are typically consumed for audit or verification rather than as a continuously refreshed control-plane input. SEKHMET follows the same integrity mindset, but commits contract-relevant summaries into a compact, per-window root intended for online consumption by the orchestrator.

Lightweight Kubernetes distributions such as K3s, MicroK8s, and k0s are widely used for edge orchestration, motivating substantial work on improving placement and coordination. Recent studies explore QoS-aware coordination across distributed clusters [1], network-aware placement policies [9], Kubernetes networking behavior in constrained settings [4], and comparative analyses of lightweight distributions [11, 12]. Across this literature, schedulers primarily act on system-level signals (CPU, memory, and network), while perception pipelines are treated as ordinary workloads. SEKHMET addresses this gap by exporting a perception-specific compliance signal to the control plane, enabling admission and scaling decisions to reflect application-level correctness constraints rather than resource utilization alone.

Tamper-evident logging using hash chains and related immutable-log designs is well studied for auditability [10]. SEKHMET reuses hash chaining with an operational goal: producing a small per-window root that is checked for freshness and consumed during execution as a scheduling token, not only after the fact as a forensic artifact.

Finally, 6G visions argue for integrated sensing and perception as first-class infrastructure services [7]. SEKHMET instantiates a concrete version of that direction on current edge clusters by combining contract-style obligations, tamper-evident roots, and orchestration hooks that act on those roots without modifying the underlying K3s control plane.

Overall, SEKHMET sits at the intersection of perception contracts, integrity/attestation, and edge orchestration by turning window-level contract outcomes into a compact, tamper-evident control-plane signal that constrains interference from elastic workloads in a heterogeneous K3s cluster.

8 Conclusion

SEKHMET shows that perception health can be elevated from an internal application concern to a control-plane signal in lightweight edge clusters. A designated perception-root node evaluates window-level timing and output-stability obligations and commits each window into a compact, hash-chained perception root. By publishing this root through standard Kubernetes interfaces, SEKHMET exposes a semantics-aware status signal to an otherwise unmodified K3s control plane. The resulting policy loop gates elasticity conservatively:

interference-increasing actions (e.g., co-location, scale-out, and opportunistic placement) are conditioned on fresh, contract-compliant windows rather than on CPU and memory utilization alone.

Experiments on a Raspberry Pi 5 + Hailo-8L cluster demonstrate why this signal matters. Under contract-unaware scheduling, the end-to-end perception loop frequently collapses to 0.8–2.2 FPS and violates the timing obligation in many windows, even while pods remain `Ready` and coarse resource metrics appear nominal. Under the same and heavier multi-node interference, SEKHMET maintains 27–30 FPS with stable detections across protected runs while emitting one fixed-size record per 5 s window. Achieving this behavior requires no changes to NPU firmware, the container runtime, or K3s; it relies on lightweight user-space monitoring and standard control-plane mechanisms.

Several extensions follow directly from this design. Contracts can incorporate additional context streams (e.g., LiDAR, radar, IMU) and richer semantics beyond count stability, including task-specific invariants and cross-sensor consistency checks. Multi-pipeline deployments can generalize the single-root setting by publishing multiple roots and composing their compliance signals into cluster policy for competing protected workloads. Stronger persistence and deletion detection can be obtained by periodically checkpointing roots off-node or anchoring them to external storage when audit requirements demand it. Finally, coupling online contract signals with formal analysis and controller synthesis is a promising direction for linking window-level perception obligations to end-to-end safety guarantees.

Overall, SEKHMET indicates that making application-level perception obligations explicit, attestable, and scheduler-visible can convert fragile best-effort perception into a protected cluster resource on commodity edge hardware.

References

1 Haci Ismail Aslan, Syed Muhammad Mahmudul Haque, Joel Witzke, and Odej Kao. QONNECT: A QoS-aware orchestration system for distributed Kubernetes clusters, 2025. doi:10.48550/arXiv.2510.09851.

2 Angello Astorga, Chiao Hsieh, P. Madhusudan, and Sayan Mitra. Perception contracts for safety of ML-enabled systems. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2):299:1–299:28, 2023. doi:10.1145/3622875.

3 Sashidhar Jakkamsetti, Youngil Kim, Andrew Searles, and Gene Tsudik. EILID: Execution integrity for low-end IoT devices, 2025. doi:10.48550/arXiv.2501.09216.

4 Georgios Koukis, Sotiris Skaperas, Ioanna Angeliki Kapetanidou, Lefteris Mamatas, and Vassilis Tsaoussidis. Performance evaluation of Kubernetes networking approaches across constraint edge environments, 2024. doi:10.48550/arXiv.2401.07674.

5 Yangge Li, Chenxi Ji, Jai Anchalia, Yixuan Jia, Benjamin C. Yang, Daniel Zhuang, and Sayan Mitra. Lyapunov perception contracts for operating design domains. In *Proceedings of the 7th Annual Learning for Dynamics and Control Conference (L4DC)*, volume 283 of *Proceedings of Machine Learning Research*, pages 1053–1065. PMLR, 2025. URL: https://proceedings.mlr.press/v283/li25c.html.

6 Yangge Li, Benjamin C. Yang, Yixuan Jia, Daniel Zhuang, and Sayan Mitra. Refining perception contracts: Case studies in vision-based safe auto-landing. arXiv preprint arXiv:2311.08652, 2023. doi:10.48550/arXiv.2311.08652.

7 Zhiyan Liu, Xu Chen, Hai Wu, Zhanwei Wang, Xianhao Chen, Dusit Niyato, and Kaibin Huang. Integrated sensing and edge AI: Realizing intelligent perception in 6G, 2025. doi:10.48550/arXiv.2501.06726.

8 Nisha Panwar, Shantanu Sharma, Guoxi Wang, Sharad Mehrotra, Nalini Venkatasubramanian, Mamadou H. Diallo, and Ardalan Amiri Sani. IoT notary: Attestable sensor data capture in IoT environments. *ACM Transactions on Internet of Things*, 3(1):3:1–3:30, 2022. doi:10.1145/3478290.

9 Ying Qiao, Junhan Xiong, and Yiguo Zhao. Network-aware container scheduling in edge computing. *Cluster Computing*, 28, 2025. doi:10.1007/s10586-024-04733-8.

10 J. D. Morillo Reina and T. J. Mateo Sanguino. Decentralized and secure blockchain solution for tamper-proof logging events. *Future Internet*, 17(3):108, 2025. doi:10.3390/fi17030108.

11 Diyaz Yakubov and David Hästbacka. Comparative analysis of lightweight Kubernetes distributions for edge computing: Performance and resource efficiency, 2025. doi:10.48550/arXiv.2504.03656.

12 Diyaz Yakubov and David Hästbacka. Comparative analysis of lightweight Kubernetes distributions for edge computing: Security, resilience and maintainability. In *Service-Oriented and Cloud Computing (ESOCC 2025)*, volume 15547 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2025. doi:10.1007/978-3-031-84617-5_8.