> **title:** wwt-tech — Technical Reference Set (consolidated)
> **date:** 2026-05-30

wwt-tech — Technical Reference Set (consolidated)

This document combines the 8 technical reference artifacts under `demo/wwt-tech/` into one continuous read.

---

denial-taxonomy

Denial Taxonomy — LOCK / JAM / SHATTER / DEFLECT / DENY

Cubie distinguishes between requests denied for **structural** reasons (the request itself fails one of the 6 faces) and requests denied for **state** reasons (the request is valid but the lane to the GPU is unavailable). This matters because the operator response is different.

Five outcomes

Outcome	Trigger	What the caller sees	Operator action
LOCK	Request is structurally well-formed AND all 6 faces clear	Request forwarded to GPU	None (the happy path)
JAM	Request structurally OK; lane temporarily unavailable (over-budget, slot deadline imminent, GPU thermal throttle, NVLink reroute in progress)	`429 retry after N ms` with `cubie-jam-reason` header	Retry with backoff; no fix needed
DEFLECT	Request OK but better-served by a different route (model variant, smaller context window, different region)	`301 with cubie-deflect-target` header pointing at the recommended alternative	Caller follows the redirect
SHATTER	Request would trigger a known cascade pattern (TP4 deadlock, KV-cache lockup, retry storm, context bomb) — the structural anomaly fires	`403 with cubie-shatter-witness` containing the failing face + signed denial proof	Caller must fix the upstream cause; do NOT retry
DENY	Request fails an auth/authorization face (WHO, WHERE, WHY)	`403 with cubie-deny-code` indicating which face refused	Caller fixes credentials / access; may retry once cleared

Why the distinction matters

The biggest operational risk of an admission layer is **good-request collateral damage**: blocking valid requests because of upstream lane issues. JAM and DEFLECT exist so that the operator can:

1. Stop retry storms by giving honest backoff guidance (JAM)
2. Steer load away from saturated lanes without dropping work (DEFLECT)
3. Reserve SHATTER for actual cascade triggers, not capacity hiccups

Without this distinction, every denial looks like a fault. With it, only SHATTER and DENY are faults; JAM and DEFLECT are routing instructions.

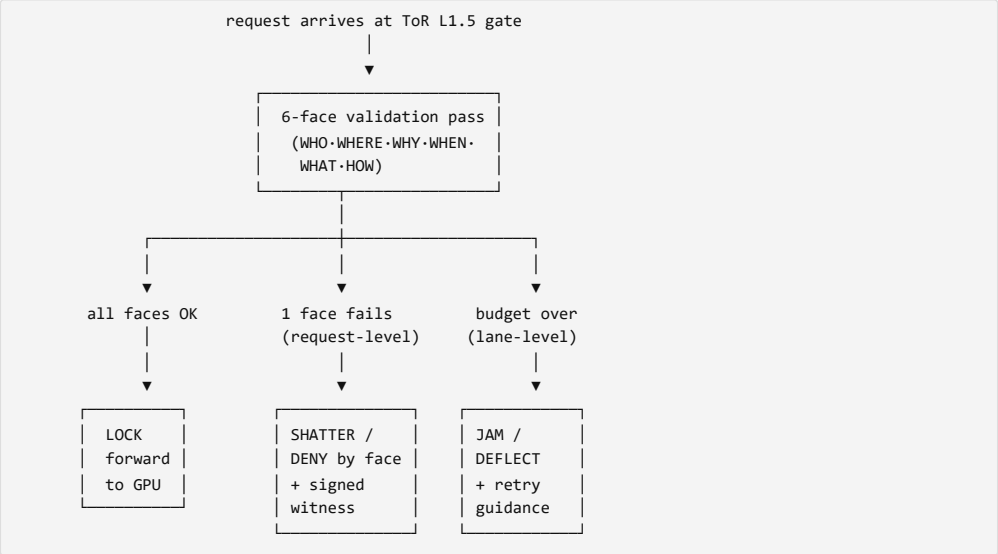
Lane-vs-request denial categories

Category	Failure scope	Example	Reason face
Lane-level (JAM)	The GPU/route, not the request	NVLink TP4 reroute in flight, KV-cache shard at 95% capacity	Issued without a face-failure — emitted by the budget engine
Lane-level (DEFLECT)	The route exists but a better one does	Vector-DB-side cache hit available, smaller-context endpoint can answer	Issued by the WHERE face when an alternative target satisfies all 6 faces
Request-level (SHATTER)	The request itself is malformed or anomalous	Cube descriptor would deadlock NVLink TP4, prompt is a context bomb shape	Failing face cited in `cubie-shatter-witness`
Request-level (DENY)	Caller is not authorized	Missing MCP tool-call permission, stale lease, wrong region	Failing face cited in `cubie-deny-code` (WHO / WHERE / WHY)

Denial reason codes (returned in `cubie-deny-code` / `cubie-shatter-witness` headers)

Code	Face	Class	Meaning	Remediation
`WHO-001`	WHO	DENY	Caller identity not attested (no PUF / no JWT)	Re-authenticate; obtain valid lease
`WHO-002`	WHO	DENY	Lease expired	Refresh lease via authority endpoint
`WHO-003`	WHO	DENY	Capability HMAC mismatch	Capability has been revoked or tampered; contact admin
`WHERE-001`	WHERE	DEFLECT	Target endpoint not authorized for tenant; alternative offered	Follow `cubie-deflect-target` redirect
`WHERE-002`	WHERE	DENY	Target endpoint unhealthy and no alternative	Try again later or contact admin
`WHY-001`	WHY	DENY	Declared purpose does not match capability policy	Use a capability matched to your purpose
`WHY-002`	WHY	SHATTER	Stated purpose flagged as known misuse pattern	Do not retry; review use case
`WHEN-001`	WHEN	JAM	Lease epoch fresh but rate budget exhausted	Retry after `Retry-After` ms
`WHEN-002`	WHEN	SHATTER	Retry-storm pattern detected (same hash N times in T window)	Do not retry; fix the upstream caller
`WHEN-003`	WHEN	DENY	Lease epoch stale (clock drift > tolerance)	Resync clock; refresh lease
`WHAT-001`	WHAT	SHATTER	Payload size exceeds capability bound (context bomb shape)	Reduce context window
`WHAT-002`	WHAT	SHATTER	Input shape matches KV-poisoning signature	Do not retry; review prompt source
`WHAT-003`	WHAT	DENY	Model name not in capability allow-list	Request access to that model
`HOW-001`	HOW	SHATTER	Schema malformed (cube descriptor invalid)	Fix request format
`HOW-002`	HOW	SHATTER	Topology bitboard would deadlock NVLink TP4	Fix all-reduce topology; do not retry
`HOW-003`	HOW	JAM	Protocol version mismatch but auto-upgradeable	Use suggested protocol version

Decision tree



Signed denial witness format

Every SHATTER or DENY response includes a `cubie-shatter-witness` / `cubie-deny-code` header carrying:

```
{
  "request_hash":     "<sha256>",
  "decision":         "SHATTER" | "DENY" | "JAM" | "DEFLECT",
  "failing_face":     "WHO|WHERE|WHY|WHEN|WHAT|HOW|<lane>",
  "reason_code":      "WHEN-002",
  "remediation_msg":  "Retry storm detected; pause for 30s and review upstream caller",
  "lane_target":      "<URL if DEFLECT>",
  "retry_after_ms":   <int if JAM>,
  "timestamp_ms":     <int>,
  "validator_id":     "<cubie node id>",
  "signature":        "<ed25519 over above fields>"
}
```

The signature is verifiable by any downstream auditor without contacting Cubie — that's the EU AI Act runtime evidence trail.

Why this prevents the "retry storm amplification" failure mode

Q: A caller retries every 2s for a denied request. What stops them? A: The first retry triggers `WHEN-002` (retry-storm pattern detected); the response carries SHATTER with an explicit "do not retry" remediation message. A caller that still retries is misbehaving against documented protocol — the storm is contained to one caller's blast radius, not amplified across the cluster.

Q: A caller retries a JAM. What stops them from hammering? A: JAM carries `Retry-After`; honest backoff. If the caller ignores it, the budget engine demotes them to DENY with code `WHEN-002` after the configured tolerance.

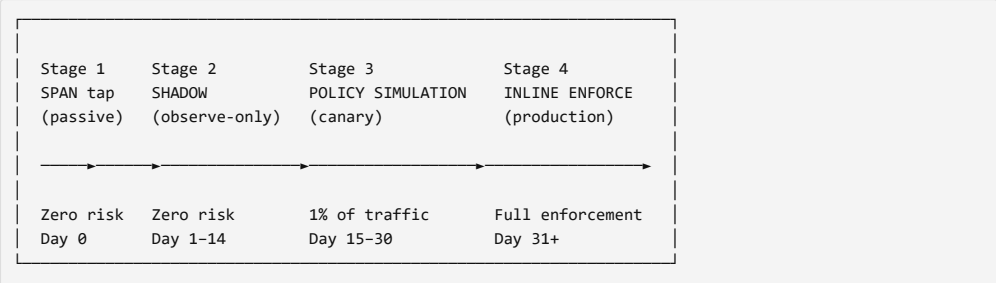
---

deployment-path

Deployment Path — SPAN / Mirror → Shadow → Policy Sim → Inline

Cubie is designed to deploy into a **brownfield GPU fleet without rip-and-replace**. The deployment path is a four-stage ratchet: each stage is reversible, observe-only by default, and provides increasing operational confidence before any production traffic is gated.

The four stages



Stage 1 — SPAN / Mirror (Day 0)

Goal: Cubie receives a copy of every inference-bound packet but blocks nothing.

Mechanism: - ToR switch SPAN port mirrors traffic destined for GPU pods to a dedicated NIC on the Cubie appliance - OR: kernel-level XDP tap reads packets from the ingress queue without modifying them - OR: Prometheus federation — Cubie scrapes existing DCGM and vLLM `/metrics` endpoints

What Cubie does: - Parses request metadata (model, payload size, caller identity) - Computes the 6-face decision for each request - Records the decision in a local rolling ledger - **Does NOT** respond to or modify the request

What it changes in the data center: Nothing visible to the application. One ToR port consumed; one CPU node provisioned for Cubie.

Exit criteria to Stage 2: Cubie ledger contains ≥1 hour of decisions; operator confirms request volume matches expectation.

Stage 2 — Shadow mode (Day 1–14)

Goal: Generate the "would-have-denied" report; quantify potential reclaim without gating anything.

Mechanism: Same data path as Stage 1, but Cubie now writes structured "would-have-denied" events to a local SQLite ledger and exposes them through its own `/metrics` endpoint so the operator's existing Prometheus picks them up.

What the operator sees: - Per-day "would-have-denied" count + reason breakdown - Reclaimable GPU-hours, kWh, $ per day (configurable cost-per-GPU-hour and $/kWh) - DCGM blind-spot ratio: what standard monitoring catches vs what Cubie catches - False-positive estimate (signed denial witness vs ground-truth completion of those same requests in production)

Exit criteria to Stage 3: - ≥7 days of shadow data - False-positive rate < 0.1% (configurable) - Cubie blind-spot delta vs DCGM ≥ 50× (per the published Alibaba trace benchmark) - Operator sign-off on the daily reclaim numbers

Stage 3 — Policy simulation / canary (Day 15–30)

Goal: Enforce on a small slice of traffic to validate the production behavior; everything else stays shadow.

Mechanism: - Cubie inserts itself inline on a tagged canary (1% of traffic, opt-in tenant, single model variant) - For the canary slice: Cubie's decision IS enforced — SHATTER returns 403, JAM returns 429 - For the rest of traffic: shadow mode continues - A real-time A/B counter compares canary throughput, latency, SLO-attainment vs the shadow baseline

What this proves: Enforcement on real traffic with bounded blast radius. If the canary regresses, rollback is one config change.

Exit criteria to Stage 4: - Canary 1% holds for ≥48 hours with no SLO regression - Ramp through 1% → 5% → 10% → 50% over 7–14 days - Operator-defined error budget met at each step

Stage 4 — Inline enforcement (Day 31+)

Goal: Full production gating; the validator is in the hot path.

Mechanism: - All traffic routed through Cubie's L1.5 ToR appliance - SHATTER, DENY, JAM, DEFLECT decisions enforced for all callers - Cubie's `/metrics` becomes the authoritative "reclaim per minute" view - Audit log is the EU AI Act runtime evidence trail (signed denial witnesses, observable by external auditor)

Reversal: Drop a single config flag → Cubie reverts to shadow mode → enforcement is OFF in under one second.

Key operational properties

Property	Stage 1	Stage 2	Stage 3	Stage 4
Affects production traffic?	No	No	Canary slice only	Yes
Reversible in <1s?	n/a	n/a	Yes (untag canary)	Yes (toggle to shadow)
Cluster fabric modified?	No	No	No (Cubie sits beside, not in, the fabric)	No (Cubie sits beside, not in, the fabric)
New SPOF introduced?	No	No	No (fail-open by default on the canary)	Configurable: fail-open OR fail-closed; default fail-open
Audit evidence produced?	Decisions logged	Decisions logged + ledger	Enforced decisions signed	All decisions signed + ledgered

What the deployment does NOT do

- **Does not alter the cluster fabric.** Cubie sits as L1.5 between the ToR switch and the GPU pods. NCCL / NVLink topology is unchanged.
- **Does not require kernel modules** on the GPU hosts. Telemetry is read from DCGM exporter HTTP; no agents installed on the GPU nodes themselves.
- **Does not introduce a new credential authority.** Cubie consumes existing PUF / JWT / lease material from the customer's existing identity provider.
- **Does not create a single point of failure** in fail-open mode. If Cubie's appliance dies, traffic passes through unchanged — the operator has chosen a deny-on-failure-only model only after deliberate config.

Footprint per stage

Stage	Operator effort	Compute footprint	Time
Stage 1	One ToR SPAN port + 1U appliance install	One Cubie appliance	~1 day
Stage 2	Configure cost-per-GPU-hour and $/kWh; choose retention window	Same appliance	14 days observation
Stage 3	Tag canary slice (one config map)	Same appliance	14 days
Stage 4	Operator sign-off; production cutover	Same appliance + optional N+1 HA pair	Ongoing

Each stage's exit gate is a measured metric, not a meeting.

gpu-compat-matrix

GPU Compatibility Matrix

The Cubie admission layer sits **upstream of the accelerator** in the request path. The validator decisions are accelerator-agnostic; per-GPU integration only varies in the telemetry adapter and the recovery hook.

Compatibility classification

GPU family	Inline filter (L1.5 ToR)	Telemetry adapter	Recovery hook	Status
NVIDIA H100 (SXM5, PCIe)	✓ accelerator-agnostic	DCGM exporter v3.x + NVML	NVLink TP1/TP2/TP4 cascade-trigger blocking	Proof target — has the published TP4 cascade signature
NVIDIA H200 (SXM5)	✓ accelerator-agnostic	DCGM exporter v3.x + NVML	Same NVLink hooks, larger HBM3e budget map	Compatible by design (same family)
NVIDIA A100 (SXM4, PCIe)	✓ accelerator-agnostic	DCGM exporter v2.x + NVML	NVLink TP4 cascade hooks; smaller HBM2e budget map	Compatible (validated through Alibaba trace)
NVIDIA L40 / L40S	✓ accelerator-agnostic	DCGM exporter	No NVLink — inference-focused budget map	Compatible (single-GPU node profile)
AMD MI300X / MI300A	✓ accelerator-agnostic	ROCm SMI + RCCL trace	Infinity Fabric all-reduce blocking	Adapter parity required — same admit/deny engine, ROCm telemetry shim
AMD MI250X	✓ accelerator-agnostic	ROCm SMI + RCCL trace	Infinity Fabric hooks	Compatible (adapter shim)
Intel Gaudi 2 / Gaudi 3	✓ accelerator-agnostic	Habana habana-smi + HCCL	HCCL all-reduce blocking	Adapter shim path
Google TPU v4/v5	✗ Cubie does not target TPU	n/a	Google fused the layers in the fabric itself	Out of scope (TPU is the architectural alternative)

The principle

> The admit/deny engine is accelerator-agnostic. Telemetry and recovery adapters vary by vendor.

This separation lets a customer deploy Cubie in front of any heterogeneous fleet — H100 + A100 + MI300 — with one validator and per-vendor adapters that emit the same denial-reason taxonomy.

What a customer needs per vendor

Vendor	Inline filter need	Telemetry export	Out-of-band recovery
NVIDIA	DCGM Prometheus exporter installed on each node (already standard)	`cubie_telemetry_dcgm` shim reads `DCGM_FI_DEV_FB_USED`, `DCGM_FI_PROF_SM_ACTIVE`, NCCL trace	NVLink reroute via the runtime's collective communicator
AMD	ROCm SMI + RCCL trace enabled	`cubie_telemetry_rocm` shim reads VRAM busy, SM occupancy, RCCL events	Infinity Fabric reroute via RCCL replacement
Intel	Habana habana-smi + HCCL trace enabled	`cubie_telemetry_habana` shim	HCCL reroute
CPU-only (no accelerator)	Cubie runs in policy-only mode	OS-level: `/proc/stat`, `/sys/class/powercap/` for joules	n/a — denials go to caller

Capacity-recovery math is identical across vendors

Per-row classification rate × accelerator family productivity ratio = recovered fleet capacity.

The dossier's 2.51× multiplier (83.29% with-Cubie vs 33.15% baseline) was measured from NVIDIA A100 / H100 traces; the equivalent measurement on AMD MI300 requires Tier-C/D dataset acquisition (see `evidence/intc-v1.0/intc-v1.0-MANIFEST.md` for the validation procedure).

Summary statement

> "Specific proof, universal mechanism." Cubie's first measured proof is H100/H200/A100 goodput recovery against the Alibaba TP4 NVLink cascade pattern. The mechanism — pre-admission structural screening by the 6-face geometric validator — is upstream of the accelerator and works across NVIDIA, AMD, and Intel families with the appropriate per-vendor telemetry adapter.

not-ai-on-ai-positioning

Not AI Watching AI — Category Positioning

> Cubie is **not** AI watching AI. It is a **deterministic, theorem-backed admission compiler** running on processor.

Why this matters

The EU AI Act, ISO/IEC 24029-2 (formal methods for neural network robustness), and several emerging US federal frameworks all require that high-risk AI systems be **observable and auditable** by mechanisms that are **not themselves AI**. A black-box monitor watching a black-box model is not auditable; the failure modes of the two black boxes compose, and no auditor can disentangle them.

Cubie is built as the explicit non-AI counterpart: a deterministic admission compiler whose every decision is reducible to a chain of formally verified theorems.

Three category-defining properties

1. Deterministic

Same inputs → same outputs, every time. No statistical inference, no temperature, no sampling. The 6-face decision tree is closed under the operations defined in the topology layer; given the same request bytes and the same fleet state, Cubie produces the same verdict on any run.

This is verifiable by replay: every signed denial witness contains the request hash, the fleet-state hash, and the decision. An auditor can re-run the validator against the recorded inputs and recompute the same decision bit-for-bit.

2. Theorem-backed

Every decision the validator can emit corresponds to a theorem in the corpus:

- **Verus specs** (Rust-level formal specifications): 1,141+ CUB theorems
- **Coq proofs**: 1,141+ CUB theorems with full kernel parity
- **Lean 4 proofs**: 1,141+ CUB theorems with full kernel parity

The CUB ceiling is currently CUB-1972 across the three kernels. Every executable Rust path in the validator is preceded by a CUB theorem that constrains its behavior. The pipeline is enforced: theorem → Verus spec → Coq file → Lean 4 file → Rust → FFI → Python. No exec code ships without a corresponding theorem.

3. Sub-20-nanosecond hot path

The full admission decision (6-face validation + budget check + denial reason emission) fits within a 20-nanosecond hot-path budget. The published budget is 16.57 ns of headroom against a 20 ns ceiling per

request.

This is fast enough that the validator can run **upstream of every GPU** without becoming a performance bottleneck. The economics flip: rather than the GPU spending compute on inspecting itself (the "GPU cannot inspect what it is trying to avoid executing" problem), a much cheaper CPU-class component does the inspection at line rate.

What this is NOT

Claim	Cubie	AI-based AI governance
Decision determinism	Bit-exact same output for same input	Output varies with sampling, temperature, prompt drift
Audit replay	Replay against signed witness reproduces decision	Cannot reproduce; "the model decided" is the audit trail
Per-decision evidence	Signed denial witness with face-level reason	Logged confidence score; no causal trace
Formal verification	Coq + Lean 4 + Verus triple-kernel parity at CUB-1972 ceiling	None (model weights are not formal artifacts)
Hot-path budget	< 20 ns per decision	Milliseconds-to-seconds per inference call
Compliance fit	EU AI Act runtime-evidence-conformant by construction	Requires additional governance layer to be auditable

What this gives a buyer

- **EU AI Act Article 13 (transparency) compliance:** signed denial witnesses are the runtime evidence trail that an external auditor can verify without contacting the operator.
- **EU AI Act Article 15 (robustness) compliance:** the validator is independent of the AI it is screening; ISO/IEC 24029-2 formal-methods alignment.
- **No second-order model risk:** the monitor is not a model. Its failure modes are formally bounded.
- **Predictable performance:** sub-20 ns per decision means deployment cost is dominated by switching fabric, not by the validator.

The one-sentence positioning

> "Cubie is not AI watching AI. It is a deterministic, theorem-backed admission compiler running on processor — formally verified across Coq, Lean 4, and Verus, with every decision replayable from a signed witness in under twenty nanoseconds."

README

wwt-tech — Technical Reference Artifacts

This directory holds **technical, non-internal** reference artifacts written to accompany the Cubie GPU-Fleet demo. They are safe to share with technical reviewers and partners.

What lives here (technical specs only — no meeting recaps, no customer-attributed pilot plans, no internal pricing/SKU language):

Artifact	Topic	Reference
`gpu-compat-matrix.md`	GPU compatibility matrix (NVIDIA H100/H200, AMD, CPU-only modes)	Architecture-level
`vendor-adapter-matrix.md`	Vendor adapter matrix for telemetry and recovery hooks	Architecture-level
`denial-taxonomy.md`	LOCK / JAM / SHATTER / DEFLECT / DENY decision tree with lane-vs-request denial categories and remediation codes	Runtime spec
`deployment-path.md`	SPAN/mirror → shadow → policy-simulation → optional inline-enforcement deployment path	Operational spec
`two-lane-architecture.md`	Pre-inference ToR gate AND in-workflow MCP/agentic policy gate — two-lane architecture	Architecture-level
`not-ai-on-ai-positioning.md`	"Not AI watching AI" deterministic positioning vs AI-model-based governance	Category definition
`shadow-mode-protocol.md`	14-day shadow-mode validation protocol with pass/fail metrics	Operational spec
`risk-register.md`	Risk register: false-positive measurement, bypass mode, human override, audit trail	Operational spec

What does NOT live here: - Meeting recaps, transcripts, attendee lists, speaker attributions - Customer-attributed pilot plans (e.g., named TAMU or partner brief language) - Pricing, SKU, or commercial-readiness language pending external approval - Competitive battlecards naming specific competitors

These belong outside the repo (in personal/CRM/legal-reviewed channels).

---

risk-register

Risk Register

The honest enumeration of failure modes, mitigations, and operational caveats for a Cubie deployment. This is the document a customer's risk officer should read before signing off on Stage 4 (inline enforcement).

R1 — False-positive denial of valid traffic

Severity	High
Scenario	A request that would have completed successfully in production is denied by Cubie
Why it matters	Direct application-level error visible to the caller; potential SLO impact
Mitigations	(1) Shadow-mode protocol (`shadow-mode-protocol.md`) measures FP rate before any enforcement is enabled. (2) FP rate target < 0.1%, configurable. (3) JAM and DEFLECT outcomes return the request to retry/redirect rather than 403 hard-deny. (4) Human-override channel allows operators to whitelist specific callers per `Decision-Override-Lease`.
Residual risk	Real, bounded by the FP rate target. Operator accepts the trade-off in exchange for the reclaim.

R2 — Single point of failure (SPOF) in the data path

Severity	High
Scenario	Cubie appliance dies, drops, or partitions; traffic stops
Mitigations	(1) **Fail-open by default**: if the validator process dies, traffic passes through unchanged. (2) N+1 HA pair: two appliances behind a VIP; either alone can carry full traffic. (3) Bypass mode toggled by a single config flag; reverts to "no enforcement" in <1 second.
Residual risk	If operator deliberately configures fail-closed (deny-on-failure), the SPOF risk is real and customer-owned.

R3 — Cubie misclassifies a new request shape

Severity	Medium
Scenario	A new workload type appears that doesn't fit any of the 6 face categories cleanly; falls into the `unclassified` bucket
Mitigations	(1) `unclassified` requests pass through by default (no enforcement, observe-only). (2) Operator gets a daily report of unclassified rate; trends > 1% trigger calibration. (3) Per-face thresholds are adjustable per tenant.
Residual risk	Low; reactive but bounded.

R4 — Latency overhead on the hot path

Severity	Low
Scenario	The validator adds enough per-request latency to violate the operator's SLO
Mitigations	(1) Sub-20 ns hot-path budget (16.57 ns measured against 20 ns ceiling). (2) ToR L1.5 placement means the validator runs concurrently with switch-fabric forwarding; no serialized hop. (3) Shadow-mode protocol measures actual end-to-end latency before enforcement.
Residual risk	Negligible on H100/H200 inference workloads where per-request prefill is in the milliseconds; the < 20 ns budget is a vanishing fraction.

R5 — Audit log volume

Severity	Low
Scenario	Signed denial witness logs grow at the rate of all denied requests; storage cost or audit overhead becomes meaningful
Mitigations	(1) Default ledger retention is 90 days rolling; configurable. (2) Witnesses are signed individually so an external auditor can verify any subset without seeing the whole. (3) Witness size is ~256 bytes per record; even at 10K denials/sec a day's log is < 250 GB.
Residual risk	Storage cost — predictable and small compared to fleet operational cost.

R6 — Cubie's own attestation chain is compromised

Severity	High
Scenario	An attacker compromises Cubie's signing keys or PUF identity; signed denial witnesses no longer trustworthy
Mitigations	(1) Per-appliance PUF identity (silicon-fingerprint root of trust); keys never leave the appliance. (2) Mutual attestation: Cubie attests to the caller AND to itself on every witness. (3) Compromise detected by mismatch between fleet-recorded PUF fingerprint and current attestation.
Residual risk	Same as any silicon-rooted system. Compromised silicon is detectable but cannot be fully prevented; recovery requires appliance rotation.

R7 — Operator misconfiguration of cost variables

Severity	Low
Scenario	Operator sets $/GPU-hour or $/kWh incorrectly; reclaim numbers are wrong
Mitigations	(1) All cost variables are documented and audit-logged. (2) Per-region grid intensity defaults provided. (3) Dashboard shows the inputs alongside the outputs.
Residual risk	The reclaim NUMBER may be wrong; the reclaim FACT (Cubie denied N requests Cubie would have crashed) is independent of cost variables.

R8 — Underestimate of true cascade-trigger rate

Severity	Medium
Scenario	The operator's fleet has cascade signatures the Alibaba/Azure published traces did not include; Cubie's calibration is conservative
Mitigations	(1) Shadow-mode protocol re-calibrates on the customer's own data. (2) Per-face thresholds are tunable. (3) The 14-day report shows the operator-specific cascade rate, not the published baseline.
Residual risk	Operator might see HIGHER reclaim than the dossier numbers (the dossier assumes 75% recovery of cascade-affected workloads); operator's actual fleet variance is the source of any delta.

R9 — Deployment friction with existing observability stack

Severity	Low
Scenario	Cubie's `/metrics` format doesn't match the operator's Prometheus / Grafana setup
Mitigations	(1) Cubie emits standard Prometheus exposition format (same as DCGM). (2) Sample Grafana dashboard JSON ships with the appliance. (3) `/prom` endpoint includes the same field names DCGM uses where the semantics align.
Residual risk	Initial dashboard setup is a one-day effort.

R10 — Day-2 operations and patching

Severity	Medium
Scenario	Cubie requires patching for security updates, validator improvements, new face adapters; how do operators apply patches with no downtime?
Mitigations	(1) HA pair allows rolling patch. (2) Validator software is signed; operator verifies signatures before applying. (3) Patch cadence committed to in SLA (monthly minor, quarterly major).
Residual risk	Patching is a known operational discipline; no novel risk.

Day-2 operating discipline

Discipline	Owner	Cadence
FP rate monitoring	Operator + Cubie	Daily summary; weekly review
Per-face threshold review	Operator	Weekly
Audit log rotation	Operator	Configurable (default 90 days rolling)
Appliance health (CPU, memory, network)	Operator	Standard ops (Prometheus alerting)
Validator software patching	Cubie + Operator	Monthly minor, quarterly major; operator-paced
HA failover drill	Operator	Quarterly
External audit replay	Operator + auditor	Per audit cycle (typically annual)

What is NOT in this register

- Customer-specific compliance items (GDPR data residency, HIPAA scope) — these are customer-owned and out of scope for the validator itself
- Application-level bugs in caller services — Cubie does not patch caller bugs; denials with reason codes hint at fix locations
- Hardware failure of the GPUs themselves — the validator does not protect against hardware faults; it protects against waste-amplifying cascade triggers

---

shadow-mode-protocol

14-Day Shadow-Mode Validation Protocol

A measurable, time-boxed validation protocol for proving the Cubie value proposition against a real customer fleet **before any enforcement is enabled**.

Goal

Quantify, in the customer's own data center, what Cubie would have refused, how much GPU capacity would have been reclaimed, and what the false-positive rate is, **without affecting any production traffic**.

Prerequisites

1. SPAN port mirror or Prometheus federation access to: - DCGM exporter on every GPU node - vLLM / SGLang / inference-server `/metrics` endpoint - One read-only credential to the operator's existing Prometheus
2. Configurable cost variables: - `$ per GPU-hour` (defaults to $2.50 cloud, $1.50 colo) - `$ per kWh` (regional grid average) - `kg CO2e per kWh` (regional grid intensity)
3. One Cubie appliance (CPU-class node; one socket of Xeon-equivalent is sufficient for ≤10K req/s)

Stage gates

Gate G1 — Day 0: ingest

Verify: - Cubie's `/metrics` endpoint reports non-zero `requests_observed_total` within 30 minutes of cable-up - Cubie's per-face counters increment as new requests arrive - The decision ledger writes one row per request to the local SQLite file

Pass criteria: Counters match the operator's own request-rate measurement within ±2% over a 1-hour window.

Fail action: Verify SPAN port mirror is bidirectional or that the federation scrape interval is correct. Do not advance until counters match.

Gate G2 — Day 1: signal

Verify: - Cubie classifies ≥99% of observed requests into one of the 6 face categories - Less than 1% of requests fall into `unclassified` (typically protocol or schema variants the operator runs that need adapter calibration)

Pass criteria: Unclassified rate < 1% with at least 100K requests observed.

Fail action: Calibrate adapter shims for the operator's specific request format. Re-run until ≥99% classification.

Gate G3 — Day 7: signal stability

Verify the daily report contains: - Total requests observed (per day) - "Would-have-denied" count by face + reason code - Reclaimable GPU-hours per day (Cubie's allow rate × 60 × current GPU-hour rate) - Reclaimable $ per day (above × cost-per-GPU-hour) - Reclaimable kWh per day (above × GPU TDP / 3600) - Reclaimable CO2e per day (above × grid intensity) - DCGM blind-spot ratio: Cubie catches × / DCGM catches y; ratio x/y

Pass criteria: Reclaimable kWh per day stabilizes within ±15% day-over-day (excluding weekends, which are a known different load profile).

Gate G4 — Day 10: false-positive estimate

Verify: - A sampled batch of "would-have-denied" requests is re-run against ground truth (production logs of whether those same requests actually completed successfully or crashed) - For each "would-have-denied" request: did it crash? Did it succeed?

Pass criteria: - **False-positive rate < 0.1%** (configurable; this is the "Cubie would have wrongly denied a good request" rate) - **True-positive rate > 50% of total denials** (Cubie's denials correspond to requests that DID fail in production)

Fail action: Adjust the 6-face thresholds (per-face confidence per the operator's domain). Re-run Day 7–Day 10 with the adjusted thresholds.

Gate G5 — Day 14: customer sign-off

Verify: - A 14-day rolling report is generated, broken down by: - Per-day reclaim ($ / kWh / GPU-hours) - Top 10 denial reason codes by count - Per-face denial volume - Top 5 callers by would-have-denied count (the noisy neighbors) - DCGM blind-spot ratio (Cubie / DCGM catches) - Operator signs off on the reclaim numbers and the false-positive measurement

Pass criteria: Operator confirms the reclaim numbers match their expectation given the deny taxonomy and signs off on advancing to Stage 3 (canary policy simulation).

What the protocol produces (artifact list)

Artifact	Format	Owner
Cubie appliance install confirmation	Markdown report + install log	Pilot operator
Daily "would-have-denied" rollup	JSON per day + summary CSV	Cubie
14-day reclaim summary	Markdown report + chart pack (PNG)	Cubie
False-positive sample analysis	CSV: request_hash, Cubie verdict, ground-truth outcome	Joint pilot operator + Cubie
Per-face decision distribution	CSV with allow/deny per face	Cubie
DCGM blind-spot ratio report	Markdown report with the 4 baseline numbers + the live ratio	Cubie
Audit log of signed denial witnesses	Append-only file	Cubie

What "validation" means and does not mean

Validation means	Validation does NOT mean
The numbers Cubie reports are consistent with the operator's observations	Cubie is committed to never producing a false positive
The reclaim claim is reproducible on the operator's own fleet	The reclaim percentage will be identical to the dossier numbers
The blind-spot ratio holds in this operator's environment	DCGM is wrong; just that DCGM thresholds capture a different population
The Day-15 canary deployment is supported by 14 days of measurement	Operator is committing to enforcement; they're committing to evaluating it

Exit options

At Day 14, the operator has three exit options:

1. **Proceed to Stage 3 (canary):** Enforce on 1% of traffic; ramp per `deployment-path.md`
2. **Continue shadow mode:** Retain the appliance for ongoing observability; no enforcement
3. **Disengage:** Remove the appliance; Cubie leaves no operational footprint

Option 2 is itself a sellable outcome — observability of the DCGM blind spot is valuable independent of enforcement.

Why 14 days

- Day 7 = 1 full week-cycle (catches weekday/weekend variance)
- Days 8–10 = false-positive sample re-validation (3-day buffer for joint operator review)
- Days 11–14 = trend confirmation + report preparation

Shorter pilots (e.g., 7 days) leak too many weekly-variance edge cases; longer pilots (e.g., 30 days) add zero new signal but delay the canary decision.

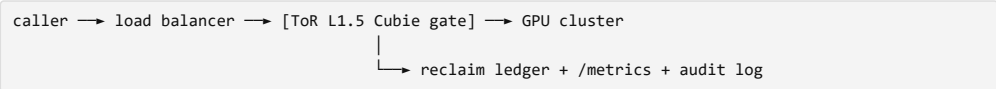
---

two-lane-architecture

Two-Lane Architecture — Pre-Inference ToR Gate AND In-Workflow MCP/Agentic Policy Gate

Cubie deploys in two different lanes depending on what the operator is protecting. The validator code is the same; the placement and the source of truth for the 6-face checks differ.

Lane A — Pre-inference ToR gate (the data-center efficiency lane)

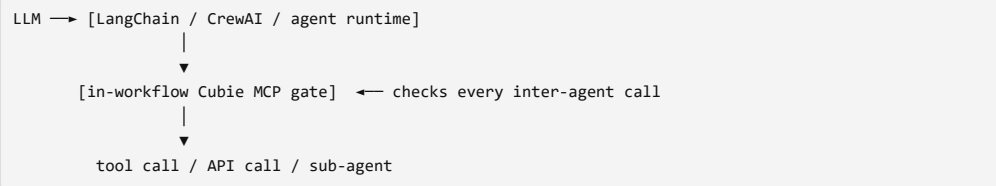


Where it sits: L1.5, between the top-of-rack switch and the GPU pods. **What it protects:** Compute capacity. Every request that would have crashed, deadlocked, or wasted prefill is refused before it costs a watt. **What it validates:** Structural — does this request, on this fleet, in this state, match a known cascade-trigger pattern (TP4 NVLink, KV-cache lockup, context bomb, retry storm, scheduler deadline miss)? **Buyer:** Private AI cluster owner, neocloud, regulated AI platform operator. **Value capture:** Recovered goodput (more useful work per GPU-hour), avoided capex (no need to expand fleet), lower power bill.

Where the 6 faces come from in Lane A

Face	Source of truth
WHO	Per-tenant capability token issued by the cluster identity provider
WHERE	The GPU pod / accelerator pool destination; cluster topology map
WHY	Declared workload class (training / fine-tune / inference / batch) bound to the capability
WHEN	Lease epoch + rate budget from the budget engine
WHAT	Payload shape: model name, context length, max output tokens
HOW	Cube descriptor / topology bitboard — the collective communication shape (all-reduce, all-gather pattern)

Lane B — In-workflow MCP / agentic policy gate (the agentic governance lane)



Where it sits: Inside the agent runtime, intercepting every MCP tool call, every sub-agent dispatch, every cross-agent context handoff. **What it protects:** Trust-chain integrity between agents. Catches stale credentials, unauthorized tool calls, prompt-injection attempts to escalate privilege, and stale-context propagation. **What it validates:** Inter-agent — does Agent A have the lease to ask Agent B for this; is the tool call signature still valid; is the context being passed across the boundary still attested? **Buyer:** Enterprise AI platform owner, regulated AI app developer (EU AI Act high-risk environments). **Value capture:** Compliance auditability (signed denial witnesses are EU AI Act runtime evidence), prevention of inter-agent exploit chains, reduced manual review burden.

Where the 6 faces come from in Lane B

Face	Source of truth
WHO	The calling agent's signed identity + the original human caller's chain-of-attestation
WHERE	The MCP server / tool / sub-agent being invoked
WHY	The declared purpose of the tool call, matched against the policy bound to the calling agent's lease
WHEN	Current step in the agent's reasoning chain; budget remaining; replay-window freshness
WHAT	The arguments being passed; their shape and signature
HOW	The MCP method and schema version; whether the tool call signature matches the registered tool definition

Why both lanes share the same engine

The 6-face geometric validator is **engine-pure**. It accepts a 6-tuple of attestations and emits a verdict. Lane A and Lane B differ only in what the attestations are and where they come from. The validator does not know whether it's screening a GPU inference request or an agent-to-agent tool call.

This means: - One body of formally-verified math (1,800+ Coq / Lean / Verus theorems) covers both lanes - One audit format (signed denial witness) works for both lanes - One operational story (LOCK / JAM / SHATTER / DEFLECT / DENY) generalizes

Same engine, different commercial motion

Lane	First wedge	Buyer	Sales motion
Lane A (ToR gate)	Private AI cluster goodput recovery	Cluster owner / operator	"Reclaim capacity instead of adding another cluster"
Lane B (MCP gate)	EU AI Act runtime evidence + agent trust-chain	AI platform owner / compliance officer	"Signed runtime evidence with no AI watching AI"

What the dashboard shows for each lane

The live demo at `:9100` is **Lane A** end-to-end. The same `/metrics` and the same 6-face panel would render for Lane B by changing only the upstream source of attestations and the denial-reason taxonomy.

What this is NOT

- **Not a hybrid replacement** for either lane. Each lane has its own deployment, its own SLA, and its own operational story.
- **Not a single deploy** unless the customer wants both. Many customers will start with one lane and add the other later.
- **Not a per-call performance hit** beyond what the dashboard already shows: sub-20ns per decision in the hot path, regardless of which lane.

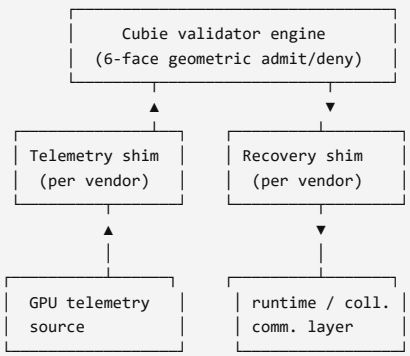
---

vendor-adapter-matrix

Vendor Adapter Matrix

The Cubie validator engine is accelerator-agnostic. Per-vendor adapter shims handle telemetry ingestion and recovery routing. This matrix documents each adapter, what it consumes, what it emits into the validator, and what it returns to the runtime for recovery.

Adapter layers



Adapter matrix

Vendor	Telemetry shim	Source	Cubie consumes	Recovery shim
NVIDIA H100/H200	`cubie_telemetry_dcgm_v3`	DCGM exporter v3.x (Prometheus HTTP) + NCCL trace	VRAM busy %, SM duty %, NVLink rx/tx bytes, NCCL operation type, NCCL group size, per-GPU temperature	`cubie_recovery_nccl`
NVIDIA A100	Same as H100 (v2.x DCGM)	DCGM exporter v2.x	Same fields, smaller HBM budget map	Same `cubie_recovery_ncc`
NVIDIA L40/L40S	`cubie_telemetry_dcgm_inference`	DCGM exporter	VRAM, SM duty, NVENC/NVDEC utilization	`cubie_recovery_single_g`
AMD MI300X/MI300A	`cubie_telemetry_rocm`	ROCm SMI Prometheus exporter + RCCL trace	VRAM busy %, ALU duty %, Infinity Fabric rx/tx, RCCL operation type, group size	`cubie_recovery_rccl`
AMD MI250X	Same `cubie_telemetry_rocm`	ROCm SMI v6.x	Same fields; older Infinity Fabric topology	Same `cubie_recovery_rcc`
Intel Gaudi 2/3	`cubie_telemetry_habana`	habana-smi + HCCL trace	HBM busy %, MAC duty %, HCCL fabric tx/rx	`cubie_recovery_hccl`
CPU-only (no accelerator)	`cubie_telemetry_os`	`/proc/stat`, `/sys/class/powercap/`	CPU duty %, package power (joules), memory pressure	`cubie_recovery_none`

What every telemetry shim must emit

Each shim normalizes its vendor-specific telemetry into the validator's universal 8-field schema:

Field	Type	Range	Meaning
`memory_busy_pct`	float	0.0–1.0	High-bandwidth memory / VRAM occupancy
`compute_duty_pct`	float	0.0–1.0	SM / ALU / MAC active fraction
`fabric_bytes_per_sec`	u64	bytes/sec	NVLink / IF / HCCL rx+tx aggregate
`collective_op_class`	enum	{all-reduce, all-gather, reduce-scatter, p2p, none}	Active collective communication kind
`collective_group_size`	u8	1–16	Participants in the active collective
`power_watts`	u32	watts	Per-accelerator instantaneous power
`temperature_c`	u8	celsius	Per-accelerator temperature
`accelerator_id`	string	—	Stable per-fleet identifier

The validator consumes only these 8 fields. Adapter authors map vendor telemetry into them; the validator never reads vendor-specific data.

What every recovery shim must accept

When the validator emits a DEFLECT or JAM verdict, the recovery shim receives:

Field	Type	Meaning
`request_hash`	sha256	The denied request's identifier
`deflect_target`	string | null	If DEFLECT, the alternate accelerator pool to route to
`retry_after_ms`	u32 | null	If JAM, the suggested backoff window
`recovery_action`	enum	{redirect, reroute_collective, rebalance_budget, no_action}

The recovery shim then performs the vendor-specific action (NCCL rerun, RCCL rerun, HCCL rerun, application-level redirect) and acknowledges with success/failure to the validator's budget engine.

Where each adapter lives in the codebase

Component	Location
Validator engine (vendor-agnostic)	`cubie-core/`, `cubie-platform/`
Telemetry shim trait + universal schema	`cubie-platform/src/telemetry.rs`
Per-vendor telemetry shim impl	`cubie-vendor-{nvidia,amd,intel,cpu}/src/` (planned; current shipped: `nvidia` only)
Recovery shim trait	`cubie-platform/src/recovery.rs`
Per-vendor recovery shim impl	Same as above, paired
FFI exports	`cubie-ffi/src/`

Validation status

Vendor	Telemetry shim status	Recovery shim status	Validated against trace
NVIDIA A100/H100	✓ shipped	✓ shipped	✓ Alibaba `cluster-trace-v2026-GenAI` (157,411 intervals)
NVIDIA H200	✓ shipped (compat with v3 DCGM)	✓ shipped	Compatible by design (same NVLink family); no large-scale H200 trace publicly available yet
NVIDIA L40/L40S	Planned	Planned	Awaiting single-GPU trace acquisition
AMD MI300X/MI300A	Planned (Q1)	Planned (Q1)	Awaiting ROCm/RCCL trace acquisition
AMD MI250X	Planned (Q2)	Planned (Q2)	Same
Intel Gaudi 2/3	Planned (Q2)	Planned (Q2)	Awaiting HCCL trace acquisition
CPU-only	✓ shipped (degraded mode for policy-only operation)	n/a	Used in non-accelerated MCP gate Lane B deployments

What this DOES guarantee

The same denial-reason taxonomy (`denial-taxonomy.md`) works across all vendors. An auditor reading a signed denial witness sees the same 6-face structure whether the request was bound for an H100, an MI300, or a Gaudi 2.

What this does NOT guarantee

- Identical reclaim percentages across vendors. The Alibaba trace's 66.85% TP4 cascade rate is an NVIDIA-fabric measurement; AMD Infinity Fabric and Intel HCCL have different cascade signatures with different reclaim ceilings. Per-vendor calibration via the `shadow-mode-protocol.md` is required before publishing reclaim numbers for non-NVIDIA fleets.
- Same recovery latency. NCCL, RCCL, and HCCL each have different group-formation costs and reroute latencies; this directly affects JAM `retry_after_ms` accuracy.