Tensorial MLOps: Unified Algebraic Formalization and Algorithmic Realization with a Case Study in AGI via Reinforcement Learning

Jaime Andres Menéndez Silva
Director Science & Technology ©
of the Theoretical Department of Research, Development and Technological Innovation
Barion Technologies
August 9, 2025

***Abstract*—We present a tensor-based algebraic formalization to unify MLOps artifacts (datasets, pipelines, models, metrics, and constraints) under a tensor calculus that enables traceability, validation, and controlled deployment. We define spaces, indices, operators, and validation predicates, and materialize them in generic pseudocode. Finally, we include a case application to Artificial General Intelligence (AGI) with Reinforcement Learning (RL), showing how the global metrics tensor governs training, evaluation, and promotion to production.**

***Index Terms*—MLOps, Tensor Algebra, Model Governance, AGI, Reinforcement Learning, Algebraic Validation**

I. Introduction and Motivation

Machine learning science and engineering in production demands a framework that unifies representation, computation, and verification. We propose *Tensorial MLOps* that describes all artifacts as tensors, transformations as operators, and validation as algebraic predicates over a global metrics tensor.

A. Limitations we address

Semantic heterogeneity: each tool manages its own metadata; **absence of algebraic semantics:** there is no formal calculus to compose data, pipelines, and metrics; **scattered validation:** *gates* are coded ad hoc without algebraizable traceability; **multimodality and AGI:** a structure is lacking for decision-making and learning in mixed domains.

B. Principles of the tensorial approach

(P1) *Unified representation:* data, models, and processes $\to$ tensorial objects with indices; **(P2)** *Closed operations:* transformations as linear or nonlinear compositions/contractions; **(P3)** *Algebraic validation:* constraints as predicates over $\mathcal{R}$; **(P4)** *Index-based traceability:* every run/version is indexed; **(P5)** *Scalability:* reductions and contractions are parallelizable.

C. Scope

The framework covers: formal definitions; operations and properties; generic algorithms (pseudocode); and a case application to AGI+RL, without relying on figures or tables.

D. Contributions

(1) General algebraic model; (2) Global metrics tensor $\mathcal{R}$ and logical *gates* $\mathcal{V}$; (3) Canonical pseudocode for evaluation, validation, and promotion; (4) AGI+RL case with performance and safety metrics.

II. Formal Definitions and Tensor Notation

A. Indices and spaces

Let the indices be $i \in \{1, \ldots, I\}$ (datasets), $j \in \{1, \ldots, J\}$ (pipelines), $k \in \{1, \ldots, K\}$ (models), and $m \in \{1, \ldots, M\}$ (metrics). We denote relevant vector (or tensor) spaces:

$$\mathcal{X}_i \subseteq \mathbb{R}^{N_i \times d_i \times c_i}, \quad \mathcal{Y}_i \subseteq \mathbb{R}^{N_i \times o_i}, \quad \mathcal{W}_k = \prod_{r=1}^{R_k} \mathbb{R}^{\prod_t l_{k,r,t}}$$

where each weight block $W_{(r)}^{(k)} \in \mathbb{R}^{\prod_t l_{k,r,t}}$.

B. Artifacts as tensorial objects

a) Dataset: $D_i = (X^{(i)}, Y^{(i)})$ with $X^{(i)} \in \mathcal{X}_i,\ Y^{(i)} \in \mathcal{Y}_i$.

b) Pipeline: $P_j = T_{j,L_j} \circ \cdots \circ T_{j,1}$, where each $T_{j,\ell}$ is (i) linear with kernel $K_{j,\ell}$ and action $X \mapsto K_{j,\ell} \cdot X$ (contraction), or (ii) nonlinear σ.

c) Model: $M_k(x) = f(x; W_k)$ with $W_k \in \mathcal{W}_k$. The composed evaluation is

$$\hat{Y}^{(i,j,k)} = M_k\big(P_j(X^{(i)})\big).$$

C. Global metrics tensor

We define $\mathcal{R} \in \mathbb{R}^{I \times J \times K \times M}$ with entries

$$R_{i,j,k,m} = \frac{1}{N_i} \sum_{n=1}^{N_i} \ell_m\Big(y_n^{(i)},\ \hat{y}_n^{(i,j,k)}\Big),$$

for loss/score functions ℓ_m (e.g., acc, F1, MAE, latency, toxicity). $\mathcal{R}$ admits *slices* by dimension for comparisons and historical analysis.

D. Constraints and validation

For each pair (j,k) we define a predicate

$$C_{j,k} : \mathbb{R}^M \to \{\text{true}, \text{false}\}, \quad C_{j,k}(v) = \bigwedge_{m=1}^{M} \big[v_m \ \text{op}_m \ \tau_{j,k,m}\big].$$

Global algebraic validation requires

$$\mathcal{V}(j,k) = \bigwedge_{i=1}^{I} C_{j,k}\big(R_{i,j,k,:}\big).$$

If $\mathcal{V}(j,k) = \text{true}$, the pair (P_j, M_k) is *promotable*.

E. Index-based traceability

Each artifact maintains a version: $D_i^{(v)}, P_j^{(v')}, M_k^{(v'')}$. Every run records the quadruple

$$\big(D_i^{(v)},\ P_j^{(v')},\ M_k^{(v'')},\ R_{i,j,k,:}\big),$$

enabling reproducibility and algebraizable auditing.

F. Typing examples (multimodal)

For a multimodal problem (vision+text+sensors), it can be modeled as

$$X^{(i)} = \big(X^{\text{img}} \in \mathbb{R}^{N_i \times H \times W \times 3},\ X^{\text{text}} \in \mathbb{R}^{N_i \times L \times d_e},\ X^{\text{sens}} \in \mathbb{R}^{N_i \times s}\big),$$

and a fusion pipeline $P_j = O_{\text{img}} \oplus O_{\text{text}} \oplus O_{\text{sens}} \to O_{\text{fusion}}$, without needing figures to express its algebraic semantics.

III. Basic and Advanced Tensor Operations

A. Applying pipelines and models

Let $X^{(i)} \in \mathcal{X}_i$. The pipeline output is $X_j^{(i)} = P_j(X^{(i)})$. The model prediction is $\hat{Y}^{(i,j,k)} = M_k(X_j^{(i)})$. When $T_{j,\ell}$ is linear with kernel $K_{j,\ell}$, the action over a batch is a contraction:

$$(X_\ell^{(i)})_{n,\alpha'} = \sum_\alpha (K_{j,\ell})_{\alpha',\alpha}\,(X_{\ell-1}^{(i)})_{n,\alpha}, \quad X_0^{(i)} = X^{(i)}.$$

If it is a nonlinearity σ, then $X_\ell^{(i)} = \sigma\Big(X_{\ell-1}^{(i)}\Big)$. In deep networks, M_k is an analogous composition with parameters W_k.

B. Metric computation and aggregation

For each metric m,

$$R_{i,j,k,m} = \frac{1}{N_i}\sum_{n=1}^{N_i} \ell_m\Big(y_n^{(i)}, \hat{y}_n^{(i,j,k)}\Big).$$

Aggregations over subsets (e.g., demographic *slices* g):

$$R_{i,j,k,m}^{(g)} = \frac{1}{|S_g|}\sum_{n \in S_g} \ell_m\Big(y_n^{(i)}, \hat{y}_n^{(i,j,k)}\Big).$$

We define *reductions* along $\mathcal{R}$'s dimensions: temporal averages, worst case by dataset, percentiles, etc.

C. Composite constraints and gate logic

Let $v \in \mathbb{R}^M$ be the metrics vector for a pair (j,k) under a given dataset. Basic gates:

$$C_\wedge(v) = \bigwedge_m (v_m \ \text{op}_m \ \tau_m), \quad C_\vee(v) = \bigvee_m (v_m \ \text{op}_m \ \tau_m).$$

Hierarchical (multi-criteria) gates:

$$C_{\text{lex}}(v) = \big(C_1(v)\big) \ \text{lex} \ \big(C_2(v)\big) \ \text{lex} \ \cdots,$$

where C_1 is primary (safety), C_2 secondary (latency), etc. The global validation:

$$\mathcal{V}(j,k) = \bigwedge_i C_{j,k}(R_{i,j,k,:}).$$

D. Drift detection and system health

Let $F^{(a)}, F^{(b)} \in \mathbb{R}^{N \times d_f}$ be intermediate representations. Normalized Frobenius similarity:

$$\text{sim}(F^{(a)}, F^{(b)}) = \frac{\big\langle F^{(a)},\ F^{(b)} \big\rangle_F}{\big\|F^{(a)}\big\|_F \ \big\|F^{(b)}\big\|_F}.$$

If $\text{sim} < \tau_{\text{drift}} \Rightarrow$ *trigger* retraining. Collapse diagnosis: SVD over batch-feature F and ratio $\sigma_{\max} / \sum_r \sigma_r$.

E. Fairness and robustness as dimensions of $\mathcal{R}$

We include fairness/robustness metrics $m \in \mathcal{M}_{\text{fair}} \cup \mathcal{M}_{\text{rob}}$. Example: gap by subgroup g,

$$\Delta_{j,k}^{(g)} = \Big|R_{i,j,k,m}^{(g)} - R_{i,j,k,m}^{(\text{ref})}\Big|, \quad C^{(g)} : \Delta_{j,k}^{(g)} \le \epsilon.$$

The total gate requires $\bigwedge_g C^{(g)}$ in addition to classical metrics.

F. Complexity and parallelization

The dominant cost comes from (i) evaluating P_j and M_k, (ii) reductions for $\mathcal{R}$. For B batches, the approximate cost is:

$$\mathcal{O}\left(B \cdot (\text{cost}(P_j) + \text{cost}(M_k)) + IJKM\right).$$

Reductions of $\mathcal{R}$ are $\mathcal{O}(IJKM)$ and can be parallelized by shards of i or k.

IV. Base Algorithms for Tensorial MLOps

A. Evaluation and logging of $\mathcal{R}$

```
def evaluate_and_log_R(datasets, pipelines, models,
    metrics):
    I, J, K, M = len(datasets), len(pipelines), len(
    models), len(metrics)
    R = zeros((I, J, K, M))
    for i,(X_i,Y_i) in enumerate(datasets):
        for j,Pj in enumerate(pipelines):
            X_proc = Pj.transform(X_i)
            for k,model in enumerate(models):
                Y_pred = model.predict(X_proc)
                vals = compute_metrics(Y_pred, Y_i,
    metrics=metrics)
                for m_idx, name in enumerate(metrics
    ):
                    R[i, j, k, m_idx] = vals[name]
    return R
```

Listing 1. Full evaluation and construction of $\mathcal{R}$.

B. Algebraic validation and promotion

```
def validate_and_deploy(R, constraints, pipelines,
    models, deploy):
    I, J, K, M = R.shape
    for j,Pj in enumerate(pipelines):
        for k,model in enumerate(models):
            sl = R[:, j, k, :]            # slice
    over datasets
            ok = validate_tensor_slice(sl,
    constraints[(j, k)])
            if ok:
                deploy(model, pipeline=Pj)
            else:
                log_rejection(model, pipeline=Pj)
```

Listing 2. Validation of $\mathcal{R}$ slices and deployment.

C. Constraints and validators

```
def validate_tensor_slice(R_datasets, gate):
    # gate: dict {metric_name: (op, threshold)} plus
     modes 'all'/'any'/'lex'
    mode = gate.get('mode', 'all')  # 'all' = and; '
    any' = or; 'lex' = priority
    for i in range(R_datasets.shape[0]):
        v = R_datasets[i]  # vector of M metrics
        if not check_vector(v, gate, mode):
            return False
    return True

def check_vector(v, gate, mode):
    checks = []
    order = gate.get('order', range(len(v)))  # for
    'lex'
    for m_idx in order:
        op, thr = gate['rules'][m_idx]
        if op == '>=': checks.append(v[m_idx] >= thr
    )
        elif op == '<=': checks.append(v[m_idx] <=
    thr)
        # ... other operators ...
    if mode == 'all': return all(checks)
    if mode == 'any': return any(checks)
    if mode == 'lex':
        # lexicographic: first violation -> False
        for c in checks:
            if c is False: return False
        return True
```

Listing 3. Schema of algebraic constraints.

V. Extension to AGI: States, Policies, and Memory

A. Multimodal space and latent states

The state s_t is modeled as a direct sum of subspaces:

$$s_t = s_t^{\text{vis}} \oplus s_t^{\text{text}} \oplus s_t^{\text{aud}} \oplus s_t^{\text{sens}} \oplus s_t^{\text{sym}}.$$

Let P_j be a perception and fusion pipeline: $h_t = P_j(s_t) \in \mathbb{R}^{d_h}$.

B. Hierarchical policy and planner

We define a hierarchical policy $\pi_\theta(a_t|h_t, g_t)$ conditioned on a goal g_t, and a planner $G_\phi(h_t) \to g_t$. The pair (π_θ, G_ϕ) constitutes the *agent*.

C. Working memory and update

Let $M_t \in \mathbb{R}^{d_m}$ be a recurrent memory; the internal dynamics are:

$$M_{t+1} = U_\psi(M_t, h_t, a_t, r_t).$$

The *stack* (h_t, M_t, g_t) defines the decision context.

D. Metrics for AGI

We extend $\mathcal{R}$ with dimensions: $\{\text{average reward}, \text{failure rate}, \text{safety}, \text{latency}\}$. Safety constraints:

$$C_{\text{safe}}: \text{ violations} \le \tau_{\text{safe}}, \quad C_{\text{lat}}: \text{ latency} \le \tau_{\text{lat}}.$$

Promotion requires $C_{\text{safe}} \wedge C_{\text{lat}} \wedge C_{\text{perf}}$.

VI. Tensorial Reinforcement Learning

A. Formulation

Environment $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$. Episode data e: trajectory $\tau_e = (s_0, a_0, r_0, \ldots, s_T)$. We define the episodes tensor:

$$\mathcal{T} \in \mathbb{R}^{E \times T \times d_\tau}, \quad d_\tau = \dim(s) + \dim(a) + 1.$$

Policy π_θ, values V_ω, or Q-values Q_η.

B. Metrics and gates in RL

For each episode e, metrics:

$$R_{e,j,k,\text{rew}} = \frac{1}{T}\sum_{t=0}^{T-1} r_t, \quad R_{e,j,k,\text{viol}} = \sum_t \mathbf{1}\{\text{rule\_violation}(s_t, a_t)\}.$$

For promotion:

$$\overline{R}_{j,k,\text{rew}} = \frac{1}{E}\sum_e R_{e,j,k,\text{rew}}, \quad \overline{R}_{j,k,\text{viol}} = \frac{1}{E}\sum_e R_{e,j,k,\text{viol}}.$$

RL gate:

$$C_{\text{RL}}: \ \overline{R}_{j,k,\text{rew}} \ge \tau_{\text{rew}} \ \wedge \ \overline{R}_{j,k,\text{viol}} \le \tau_{\text{safe}}.$$

C. Optimization with constraints

Problem:

$$\max_\theta \ \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t\right] \quad \text{subject to} \quad \mathbb{E}_{\pi_\theta}[c_q(s_t, a_t)] \le \kappa_q, \ \forall q,$$

where c_q are costs (safety, energy, etc.). Lagrangian dual:

$$\mathcal{L}(\theta, \lambda) = J(\theta) - \sum_q \lambda_q\Big(\mathbb{E}[c_q] - \kappa_q\Big), \ \lambda_q \ge 0.$$

Alternating updates $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$, $\lambda \leftarrow [\lambda + \beta(\mathbb{E}[c_q] - \kappa_q)]_+$.

D. Convergence diagnostics

Over $\mathcal{R}$ we define

$$\Delta_{\text{rew}} = \left|\overline{R}^{(t)}_{\text{rew}} - \overline{R}^{(t-1)}_{\text{rew}}\right|, \quad \Delta_{\text{viol}} = \left|\overline{R}^{(t)}_{\text{viol}} - \overline{R}^{(t-1)}_{\text{viol}}\right|.$$

Training stop gate: $\Delta_{\text{rew}} \le \epsilon_{\text{rew}} \wedge \Delta_{\text{viol}} \le \epsilon_{\text{viol}}$ for H iterations.

VII. RL Algorithms under the Tensorial Structure

A. On-policy training (PPO-style scheme)

```
def train_on_policy(env, policy, value, pipelines, J
    , K, metrics, E, T):
    R = zeros((E, J, K, len(metrics)))  # episodes x
     pipelines x models x metrics
    for e in range(E):
        s = env.reset()
        Pj = pipelines[e % J]
        traj = []
        for t in range(T):
            h = Pj.transform_state(s)
            a, logp = policy.sample(h)
            s2, r, done, info = env.step(a)
            traj.append((s, a, r, logp, info))
            s = s2
            if done: break
        # update policy/value (GAE, clip, etc.) --
    omitted for brevity
        vals = compute_ep_metrics(traj, metrics=
    metrics)
        for m_idx, name in enumerate(metrics):
            R[e, e % J, 0, m_idx] = vals[name]   #
    model 0 as an example
    return R
```

Listing 4. On-policy loop with constraints and logging in $\mathcal{R}$.

B. Off-policy training (actor–critic scheme)

```
def train_off_policy(env, actor, critic, Pj, metrics
    , E, T, buffer, validate_every, gate):
    R_hist = []
    for e in range(E):
        s = env.reset(); ep = []
        for t in range(T):
            h = Pj.transform_state(s)
            a = actor.select(h)
            s2, r, done, info = env.step(a)
            buffer.add(s, a, r, s2, done)
            ep.append((s,a,r,info))
            s = s2
            if done: break
        # batch updates from the buffer
        for _ in range(updates_per_ep):
            batch = buffer.sample()
            critic.update(batch)
            actor.update(batch, critic)
        vals = compute_ep_metrics(ep, metrics=
    metrics)
        R_hist.append([vals[m] for m in metrics])
        if (e+1) % validate_every == 0:
            R_block = to_tensor(R_hist[-
    validate_every:])
            if validate_tensor_slice(R_block, gate):
                maybe_checkpoint(actor, critic)
    return array(R_hist)
```

Listing 5. Off-policy loop with buffer and periodic validation.

C. Promotion plan

```
def promotion_decision(R_hist, windows=50, rew_thr
    =..., safe_thr=...):
    if len(R_hist) < windows: return False
    blk = R_hist[-windows:]
    rew = mean([x[0] for x in blk])       # assumes
     index 0 = mean reward
    viol = mean([x[1] for x in blk])      # assumes
     index 1 = violations
    return (rew >= rew_thr) and (viol <= safe_thr)
```

Listing 6. Promotion based on moving average and safety limits.

VIII. Case Study: AGI with RL under Tensorial MLOps

A. Scenario

A generalist agent operates in a continuous environment with alternating goals $g \in \{g_1, \dots, g_G\}$. The state s_t is multimodal; P_j performs perception and fusion. The policy π_θ decides actions; the planner G_ϕ updates g_t.

B. Quantitative definitions

For each episode e: average reward R_{rew}, violations R_{viol}, average latency R_{lat}. The gate is:

$$C: \ \mathbb{E}[R_{\text{rew}}] \geq \tau_{\text{rew}} \ \wedge \ \mathbb{E}[R_{\text{viol}}] \leq \tau_{\text{safe}} \ \wedge \ \mathbb{E}[R_{\text{lat}}] \leq \tau_{\text{lat}}.$$

C. Computation of $\mathcal{R}$ and promotion

```
def end_to_end_AGI_RL(env, Pj, policy, planner,
    metrics, gate, E, T):
    R = zeros((E, 1, 1, len(metrics)))  # one
    pipeline, one model (for this case)
    for e in range(E):
        s = env.reset(); ep = []
        g = planner.init_goal()
        for t in range(T):
            h = Pj.transform_state_goal(s, g)
            a = policy.select(h)
            s2, r, done, info = env.step(a)
            planner.update(h, a, r, info)    #
    optional: meta-control
            ep.append((s, a, r, info))
            s = s2
            if done: break
        vals = compute_ep_metrics(ep, metrics)
        for m_idx, name in enumerate(metrics):
            R[e, 0, 0, m_idx] = vals[name]
        # rolling validation
        if e >= 32:
            sl = R[e-31:e+1, 0, 0, :]
            if validate_tensor_slice(sl, gate):
                save_candidate(policy, planner, tag=
   f"ep{e}")
    # final decision
    if validate_tensor_slice(R[:, 0, 0, :], gate):
        deploy_agent(policy, planner)
    else:
        log_rejection("AGI-RL agent", reason="
    gate_failed")
    return R
```

Listing 7. Complete integration in a single use case.

D. Convergence and health metrics

We define the rates:

$$\text{SPR} = \frac{1}{W}\sum_{t=1}^{W} \mathbf{1}\{\Delta_{\text{rew}}^{(t)} \leq \epsilon_{\text{rew}}\}, \quad \text{SVR} = \frac{1}{W}\sum_{t=1}^{W} \mathbf{1}\{\Delta_{\text{viol}}^{(t)} \leq \epsilon_{\text{viol}}\}.$$

The agent is stable if $\text{SPR}, \text{SVR} \geq \rho$ for several consecutive windows; this is incorporated into the gate as an additional rule.

IX. Discussion and Limitations

A. Advantages

The tensorial formalization provides: (i) a unified semantics for data, computation, and verification; (ii) algebraizable and auditable validation; (iii) platform independence; (iv) a natural extension to AGI and RL.

B. Cost and scalability

The cost of maintaining $\mathcal{R}$ grows with $IJKM$. Strategies: metrics *streaming*, compression (quantization, sketching), and sharding by (i,k). Reductions and validation are formulated as *map-reduce* operations over slices of $\mathcal{R}$.

C. Persistence and reproducibility

Each run logs $\left(D_i^{(v)}, P_j^{(v')}, M_k^{(v'')}, R_{i,j,k,:}\right)$ with seeds, hashes, and signatures. This enables algebraizable non-regression tests by comparing ΔR and $\|\Delta W\|_F$.

D. Alignment and safety

Including safety and fairness metrics as dimensions imposes composite *gates*. In RL, constraints are integrated via Lagrange or *shielding* (safe policies), and $\overline{R}_{\text{viol}}$ is monitored.

E. Limitations

(i) Learning curve for teams; (ii) definition and measurement of ethical metrics; (iii) computational cost for high-order tensors; (iv) need for interoperability standards for $\mathcal{R}$.

X. Conclusions and Future Work

We presented a tensorial MLOps framework that unifies representation and validation of AI, with algorithmic realization and an AGI+RL case study. The central piece is the global metrics tensor $\mathcal{R}$ and the predicate-based validation $\mathcal{V}$ that governs promotion and deployment.

Future directions: (i) federated learning with distributed $\mathcal{R}$; (ii) probabilistic validation (conformal for gates); (iii) multi-agent tensor graphs; (iv) integration with immutable ledgers for auditing.

XI. AGI Application Case: Environmental Protection

A. Narrative for the general public

Imagine a *general intelligent assistant* (AGI) that watches our forests day and night. This AGI receives multispectral satellite images, meteorological data, and real-time measurements from sensors installed on watchtowers and ground stations. Its mission is to detect wildfires even before smoke is visible, assessing the risk of spread and issuing alerts to authorities.

We can think of each type of data as a different thread:

- Red thread: satellite images (capture temperature and vegetation changes).
- Blue thread: meteorological data (wind, humidity, ambient temperature).
- Green thread: on-the-ground sensors (detect heat and smoke particles).

The tensorial AGI is the weaver that combines these threads to produce a robust fabric: the alert decision, mathematically validated before being sent.

B. Mathematical formalization

Let the set of datasets be:

$$\{D_1, D_2, D_3\} = \{\text{satellite}, \text{meteorology}, \text{sensors}\}$$

where:

$$D_1 \in \mathbb{R}^{N_1 \times H \times W \times C_s}, \quad D_2 \in \mathbb{R}^{N_2 \times f_m}, \quad D_3 \in \mathbb{R}^{N_3 \times f_t}.$$

Here C_s is the number of spectral channels, f_m the number of meteorological variables, and f_t the number of sensor measurements.

We define two pipelines:

$$P_1 = \text{unimodal processing of satellite images}$$

$$P_2 = \text{multimodal fusion of images, meteorology, and sensors}$$

and two models:

- M_1: CNN optimized for satellite images.
- M_2: multimodal network with cross-attention to integrate all sources.

C. Metrics tensor and constraints

Let $\mathcal{R} \in \mathbb{R}^{I \times J \times K \times M}$ be the metrics tensor, where we measure:

$$\text{acc} \quad (\text{accuracy}), \quad \text{lat} \quad (\text{latency}), \quad \text{tpr} \quad (\text{true positive rate for critical e}$$

The deployment constraint is:

$$\text{acc} \geq 0.90 \quad \wedge \quad \text{lat} \leq 5\,\text{s} \quad \wedge \quad \text{tpr} \geq 0.95$$

and the global validation is:

$$\mathcal{V}(j,k) = \bigwedge_{i=1}^{I} C_{j,k}\left(R_{i,j,k,:}\right)$$

Only if $\mathcal{V}(j,k) = \text{true}$ does the AGI deploy that pipeline–model pair in the monitoring system.

D. Pseudocode of the tensorial flow

```
# Datasets
datasets = [satellite_images, weather_data,
    ground_sensors]

# Pipelines
pipelines = [image_only_pipeline, fusion_pipeline]

# Models
models = [cnn_satellite, multimodal_attention]

# Metrics
metrics = ["acc", "lat", "tpr"]

# Evaluate metrics tensor
R = evaluate_and_log_R(datasets, pipelines, models,
    metrics)

# Constraints
constraints = {'acc': (">=", 0.90), 'lat': ("<=",
    5.0), 'tpr': (">=", 0.95)}
```

```
# Validation and deployment
for j, pipeline in enumerate(pipelines):
    for k, model in enumerate(models):
        if validate_constraints(R[:, j, k, :],
    constraints):
            deploy_model(model, pipeline, domain="
    environment")
        else:
            log_rejection(model, pipeline, domain="
    environment")
```

E. Social and environmental impact

This approach enables:

- Detecting fire outbreaks before they spread uncontrollably.
- Drastically reducing response times through automatic latency validation.
- Ensuring that only reliable models are deployed thanks to tensorial evaluation.
- Maintaining a complete performance history ($\mathcal{R}$) for audits and continuous improvements.

The integration of tensorial AGI into environmental monitoring represents a decisive advance to protect ecosystems and human communities.