Toward a Requirements Framework for Trustworthy AI in Enterprise Cloud Deployments

Kranthi Kumar Manchikanti

kmanchikanti@ieee.org

Abstract. As AI systems—including large language models (LLMs)—are deployed in enterprise cloud environments, the gap between formal requirements specifications and production realities presents a challenge for safety assurance. Drawing on multi-cloud enterprise deployment experience across major public cloud platforms, this paper identifies four requirements challenges that become acute in cloud-native AI: multi-tenancy isolation, model drift, cross-jurisdictional compliance, and pipeline provenance. We introduce the concept of *requirements freshness*—the conditions under which a prior validation judgment remains authoritative—and propose CART (Cloud-AI Requirements Traceability), comprising a domain-specific language for operational invariants that compiles to signal temporal logic, a reference architecture for pipeline-aware traceability, jurisdictional parameterization, and runtime verification hooks. We present a four-dimensional evaluation protocol and illustrate CART's applicability through worked examples spanning classical ML (fairness, drift) and LLM-based systems (groundedness, provider-drift).

Keywords: AI requirements engineering · cloud-native AI · LLM governance · runtime verification · domain-specific languages

1 Introduction

Deploying AI in enterprise cloud environments introduces requirements challenges fundamentally different from traditional software or standalone AI. Formal RE frameworks have advanced specifications for safety, fairness, and robustness [1,2,3], but typically assume a static deployment context—a trained model in a well-defined environment with stable distributions. The rise of LLM-based enterprise systems has intensified this gap: foundation models are updated by providers on schedules outside the deployer's control, retrieval pipelines introduce new leakage surfaces, and generative outputs lack the closed-form evaluability of classifier predictions [12].

Enterprise AI on major public cloud ML platforms involves multi-layered architectures where requirements span model, data pipelines, infrastructure, and organizational processes. A fraud detection model's fairness requirement (e.g.,

demographic parity across protected attributes) must account for weekly retraining on data that may introduce proxy features, multi-region serving against jurisdiction-specific definitions of protected classes under fair-lending and AI transparency regulations, and monitoring by platform teams lacking such domain context. An LLM-based clinical assistant adds prompt templates, RAG indices, tool invocations, and provider-side model updates as additional surfaces requiring requirements coverage.

This paper identifies four challenge categories drawn from multi-cloud enterprise deployment experience across major public cloud platforms, and proposes CART, which unlike prior work treating the model atomically [4,6,10], decomposes requirements across the deployment pipeline with monitoring predicates grounded in signal temporal logic (STL) [1]. Our contribution is fourfold: (i) identification of *requirements freshness* as a missing construct in formal RE for AI; (ii) a compact DSL for operational invariants that compiles to STL; (iii) a pipeline-aware reference architecture with jurisdictional parameterization; (iv) runtime verification hooks and a four-dimensional evaluation protocol applicable to both classical ML and LLM-based systems.

2 Requirements Challenges in Cloud-Native AI

We identify four challenge categories arising from the interaction between AI's stochastic, evolving behavior and cloud infrastructure's shared, distributed, multi-jurisdictional nature—a combination absent in traditional software.

Multi-Tenancy and Data Isolation. Cloud AI serves multiple tenants on shared infrastructure. Beyond functional correctness, requirements must specify that tenant A's data cannot influence tenant B's predictions. AI introduces *statistical* isolation risks—shared feature stores, cached embeddings, and batch scheduling create cross-tenant leakage without explicit data transfer [4]. These risks are routinely mitigated by per-tenant isolation, but the guarantee is typically stated informally and enforced by scattered controls—configuration changes and library updates can silently invalidate it over time. LLM-based systems extend this surface: shared base models with tenant-specific RAG indices can leak through retrieval results, cached completions, or few-shot exemplars that span tenant boundaries.

Model Drift and Continuous Validation. AI models degrade as distributions shift—a compliant model at deployment may violate requirements within weeks due to covariate shift outside the test distribution. Current frameworks lack constructs for *requirements freshness*—conditions under which a validation judgment remains valid. For LLM-based systems, drift has an additional provider-driven dimension: a foundation model update (e.g., a provider upgrading an underlying model endpoint) can invalidate all prior validation without any code change by the deployer. This is unique to learning-enabled components consumed as a service.

Cross-Jurisdictional Compliance. Global enterprise AI must satisfy varying, potentially conflicting regulations. Specifications must be parameterizable

by jurisdiction and verifiable against multiple overlays simultaneously. Beyond legal mandates, normative considerations—social and ethical constraints [7]—add dimensions that static specifications cannot accommodate. The EU AI Act's specific provisions for general-purpose AI models [15] introduce LLM-specific documentation and evaluation obligations that differ from sector-specific regulations elsewhere, while the NIST AI Risk Management Framework [16] provides a complementary voluntary risk-based approach adopted across U.S. regulated sectors.

Pipeline Integrity and Provenance. An AI system's behavior depends on its entire pipeline: ingestion, feature engineering, training, registry, deployment, and inference. A non-discrimination requirement must trace through every stage where protected attributes could leak via proxies. For LLM systems, the pipeline expands to include prompt templates, system prompts, retrieval indices, tool definitions, and provider model identifiers—each a stage where requirements can be violated. A substantial share of production ML failures originates in pipeline components rather than model logic [4].

3 The CART Framework

CART extends formal RE with four constructs for cloud-native AI that operate as a single pipeline: DSL invariants (Sect. 3.1) are decomposed across pipeline stages (Sect. 3.2, Fig. 1), parameterized for jurisdiction (Sect. 3.3), and enforced at runtime via verification hooks (Sect. 3.4).

3.1 Operational Invariants and the CART DSL

CART introduces *operational invariants*—properties that hold continuously, not just at deployment—expressed in a compact DSL that compiles to STL [1] for runtime monitoring. An invariant declares a requirement R, a signal s measurable on production data, a threshold-comparison relation, a sliding window w, and a bounded time τ within which revalidation must occur when the trigger condition holds. We define the translation of a DSL invariant into an STL formula as:

$$[\![\,\text{invariant}(R, s, \mathit{rel}, \theta, w, \tau)\,]\!] = \Box_{[0,T]}\big(s(w)\ \mathit{rel}\ \theta \rightarrow \Diamond_{[0,\tau]}\ \mathit{revalidate}(R)\big) \quad (1)$$

where $\mathit{rel} \in \{<, \leq, >, \geq, \neq\}$ and s ranges over signals such as dpg, KL-divergence, $\mathit{grounded\_rate}$, or $\mathit{provider\_id}$. The DSL intentionally does not express properties requiring model-internal access, global pipeline state, or cross-tenant reasoning—these are delegated to pipeline-aware traceability (Sect. 3.2) and RV hooks (Sect. 3.4). Three worked examples follow.

Example 1 (Fairness, classical ML). For fairness requirement R_f:

$$\Box_{[0,T]}\big(\mathit{dpg}(w) \geq \theta\ \rightarrow\ \Diamond_{[0,\tau_f]}\,\mathit{revalidate}(R_f)\big) \quad (2)$$

Illustrative parameterization: θ=0.1, w=7d, τ_f=48h. When the demographic parity gap exceeds θ, a bounded liveness obligation triggers revalidation within τ_f.

Example 2 (Drift, any model type). For accuracy requirement R_a:

$$\Box_{[0,T]}\big(KL(P_{prod}\|P_{ref}) > \delta \;\rightarrow\; \Diamond_{[0,\tau]}\, revalidate(R_a)\big) \tag{3}$$

Illustrative parameterization: δ=0.1, τ=48h.

Example 3 (Groundedness, LLM systems). For a RAG-based LLM assistant with groundedness requirement R_g:

$$\Box_{[0,T]}\big(grounded\_rate(w) < \gamma \;\rightarrow\; \Diamond_{[0,\tau_g]}\, revalidate(R_g)\big) \tag{4}$$

where $grounded\_rate(w)$ is measured over window w via an evaluation method of the deployer's choice (e.g., reference-free groundedness metrics [13] or τ-bench-style [14] trajectory evaluation). The choice of measurement instrument is orthogonal to the specification pattern. A complementary provider-drift invariant captures foundation model updates outside the deployer's control:

$$\Box_{[0,T]}\big(provider\_id \neq provider\_id_0 \rightarrow \Diamond_{[0,\tau_p]}\, revalidate(R_g)\big) \tag{5}$$

triggered when the underlying foundation model identifier changes.

The bounded liveness operator $\Diamond_{[0,\tau]}$ captures *requirements freshness*—absent from existing RE frameworks. CART deliberately operates at the pipeline boundary: the DSL excludes predicates over model-internal state (weights, gradients, attention), since cloud deployments consume foundation models as services and pipeline components contribute disproportionately to production ML failures [4]. Full formal semantics of $revalidate(\cdot)$ within an established specification language is future work.

3.2 Pipeline-Aware Traceability

CART decomposes requirements across pipeline stages where violations *originate*, without inspecting model internals. Each requirement maps to components with designated verification points. Figure 1 shows the CART reference architecture: six canonical classical ML pipeline stages (Ingestion through Inference) extended by an LLM artifact substrate (prompt templates, RAG indices, tool registries, and provider bindings) that is consumed at inference time. A fairness requirement traces to both feature engineering (proxy detection) and training (bias mitigation); a groundedness requirement traces to retrieval (index coverage) and prompt construction (prompt integrity); a provider-drift invariant (Eq. 5) attaches to the provider binding. This builds on the safety case approach of Borg et al. [3], extending it from model-level reasoning to full pipeline decomposition.

3.3 Jurisdictional Parameterization

CART supports parameterized specifications instantiated with jurisdiction-specific constraints. A transparency requirement refines into EU AI Act documentation for European deployments, Colorado AI Act consumer-disclosure obligations for

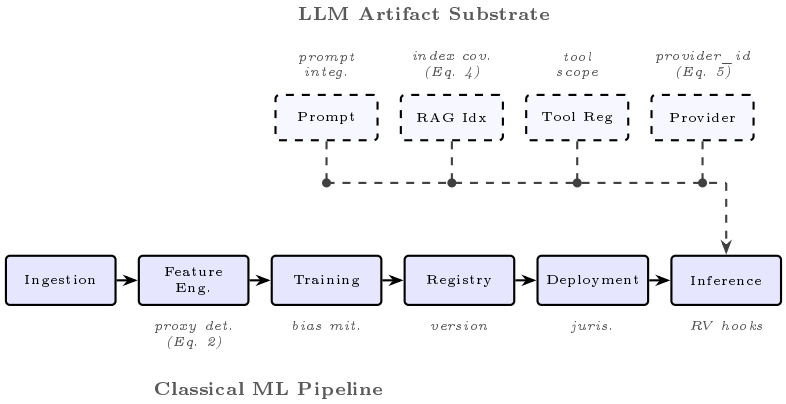

Fig. 1. CART reference architecture. The *Classical ML Pipeline* (bottom) comprises stages connected by temporal flow (solid arrows). The *LLM Artifact Substrate* (top)—Prompt templates, RAG indices, Tool registries, and Provider bindings—feeds into inference via a dashed dependency bus; junction dots mark where each artifact's trace joins the bus. Italic labels denote verification points anchoring CART DSL predicates; equation references locate concrete invariants instantiated in Sect. 3.1.

high-risk systems deployed in Colorado, and NIST AI RMF [16] Govern-function controls for U.S. federal deployments. For LLM systems, the EU AI Act's general-purpose AI provisions [15] add additional parameterization axes (systemic risk thresholds, downstream documentation obligations).

3.4 Runtime Verification Hooks

CART specifies integration patterns for attaching runtime monitors [9,11] to deployed systems. These hooks operationalize the predicates compiled from Eq. 1, enabling continuous property assessment. For classical ML, hooks integrate with cloud telemetry systems; for LLM systems, hooks integrate with evaluation frameworks (e.g., OpenAI Evals, Inspect AI, τ-bench harnesses [14]) to sample groundedness or trajectory-level properties on production traffic. This composition with existing monitoring infrastructure is consistent with an emerging industry framework for safe agent deployment that emphasizes continuous classifier-based monitoring, threat intelligence pipelines, and tiered deployment safeguards [17].

4 Evaluation Approach

We define a four-dimensional protocol for evaluating CART deployments: *coverage* (fraction of declared requirements with ≥ 1 operational invariant), *decomposition depth* (mean pipeline stages per requirement's verification point

set), *detection latency* (time between trigger condition becoming true and invariant firing, bounded by window w), and *revalidation specificity* (fraction of triggered revalidations identifying genuine drift vs. false positives). The first two are specification-time measurements; the latter two are runtime measurements. The protocol applies uniformly to classical ML and LLM systems: Eq. 2 and Eq. 4 differ in signal definition but are measured along the same dimensions. Full empirical instantiation is reserved for the extended version.

CART's constructs were distilled from production deployments across multiple public cloud environments. In fraud detection deployments, fairness violations traced to feature engineering—where geographic proxies for protected attributes emerge through feature selection—motivated pipeline-aware invariants of the form in Eq. 2 that require revalidation on feature pipeline updates, not only model retraining. Insurance underwriting assistants deployed across jurisdictions motivated jurisdictional parameterization; drift events observed in multi-region deployments informed Eq. 3. Provider-side model updates in enterprise LLM deployments motivated Eq. 5; groundedness invariants following Eq. 4 are under development using τ-bench-style [14] evaluation, complementing concurrent work on agent-trajectory safety predicates [18].

Limitations and Open Challenges. CART inherits STL monitoring's assumptions: properties requiring unbounded liveness or second-order quantification cannot be compiled via Eq. 1, and runtime monitors incur observation cost at high traffic. The four evaluation dimensions are defined but not yet empirically calibrated; the deployment vignettes are illustrative rather than controlled case studies, and may not generalize to vision-language or hybrid systems. Open challenges include formal semantics of $\mathit{revalidate}(\cdot)$, completeness criteria for pipeline traceability, and extension to agentic systems where trajectory-level properties do not reduce to scalar signals.

5 Related Work

Maler and Ničković [1] introduced STL monitoring underpinning CART's invariants; Seshia et al. [2] articulated a verified AI agenda; Borg et al. [3] demonstrated safety cases for ML (SMIRK), which CART generalizes from model level to full pipeline in cloud architectures. Vogelsang and Borg [5] and Kuwajima et al. [6] identified RE and engineering challenges for ML motivating pipeline-level decomposition; Yaman et al. [7] formalized normative requirements for autonomous agents. MLOps tooling [8] and ML testing surveys [4,10] address related concerns without formal grounding; runtime verification [9,11] provides CART's monitoring foundations. CART differs from both lines by unifying formal STL monitoring with pipeline-aware traceability and jurisdictional parameterization. For LLM-based systems, Bommasani et al. [12] characterized foundation model risks; RAGAS [13] and τ-bench [14] provide measurement instruments for CART's LLM-facing invariants. Regulatory and industry frameworks—the EU AI Act [15], NIST AI RMF [16], and an emerging vendor-published safety framework [17]—motivate jurisdictional parameterization and runtime monitoring extensions.

6 Conclusion

CART bridges formal RE and production AI through a DSL compiling to STL, a pipeline-aware reference architecture, and an evaluation protocol applicable to both classical ML and LLM-based systems; future work targets formalization of requirements freshness semantics and extension to agentic systems.

References

1. Maler, O., Ničković, D.: Monitoring temporal properties of continuous signals. In: FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer (2004). https://doi.org/10.1007/978-3-540-30206-3_12
2. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7), 46–55 (2022)
3. Borg, M., et al.: Ergo, SMIRK is safe: a safety case for an ML component in a pedestrian AEB system. Softw. Qual. J. **31**(2), 335–403 (2023)
4. Sculley, D., et al.: Hidden technical debt in machine learning systems. NeurIPS, pp. 2503–2511 (2015)
5. Vogelsang, A., Borg, M.: Requirements engineering for machine learning: perspectives from data scientists. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 245–251 (2019)
6. Kuwajima, H., Yasuoka, H., Nakae, T.: Engineering problems in machine learning systems. Mach. Learn. **109**, 1103–1126 (2020)
7. Yaman, S.G., et al.: Specification and validation of normative rules for autonomous agents. FASE 2023, LNCS vol. 13991, pp. 241–248. Springer (2023)
8. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine learning operations (MLOps): overview, definition, and architecture. IEEE Access **11**, 31866–31879 (2023)
9. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. **78**(5), 293–303 (2009)
10. Riccio, V., et al.: Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. **25**, 5193–5254 (2020)
11. Havelund, K., Roşu, G.: Runtime verification – 17 years later. In: RV 2018. LNCS, vol. 11237, pp. 3–17. Springer (2018)
12. Bommasani, R., et al.: On the opportunities and risks of foundation models. arXiv:2108.07258 (2021)
13. Es, S., et al.: RAGAS: automated evaluation of retrieval-augmented generation. arXiv:2309.15217 (2023)
14. Yao, S., et al.: τ-bench: a benchmark for tool-agent-user interaction in real-world domains. arXiv:2406.12045 (2024)
15. European Parliament and Council: Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L 2024/1689 (12 July 2024)
16. Tabassi, E.: Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, National Institute of Standards and Technology, Gaithersburg, MD (2023). https://doi.org/10.6028/NIST.AI.100-1
17. Anthropic: Our framework for developing safe and trustworthy agents (2025). https://www.anthropic.com/news/our-framework-for-developing-safe-and-trustworthy-agents
18. Manchikanti, K., Lacerda, P.: SAFE: A framework for responsible agentic systems with evaluation as control signal. Zenodo (2026). https://doi.org/10.5281/zenodo.19697951