

Governance-as-Code for Deterministic, Reliable and Reproducible AIoT Systems in Smart Buildings

Young Im Cho,

¹Department of Computer Engineering, Gachon University, Seongnam-si, Republic of Korea

email: imcho@gachon.ac.kr

Saydirasulov Norkobil

¹Department of Computer Engineering, Gachon University, Seongnam-si, Republic of Korea;

²Department of Technologies of Mobile Communication, Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi, Tashkent 100200, Uzbekistan;

email: saydirasulov@gachon.ac.kr

Davronbekov Dilmurod

Department of Technologies of Mobile Communication, Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi, Tashkent 100200, Uzbekistan;

d.davronbekov@gmail.com

Dusmurod Kilichev

Department of Information Security, National University of Uzbekistan, Tashkent 100174, Uzbekistan;

email: dusmurod@nuu.uz

Abstract: Reliable and reproducible operation remains difficult in Artificial Intelligence of Things (AIoT) systems for smart buildings, where rapid software updates must coexist with safety, privacy, resilience and energy-efficiency requirements. This article presents a shortened journal version of a governance-as-code framework that embeds executable policies into both CI/CD pipelines and runtime control. The proposed monolithic-modular architecture centralizes admission control, intent routing, capability resolution, scheduling, health management and rollback in a deterministic orchestrator, while allowing independently versioned AI service modules through standardized contracts. A normalized device layer abstracts heterogeneous building protocols such as BACnet/IP, Modbus/TCP, OPC-UA and MQTT, preventing direct module-device interaction and supporting coordinated recovery. The deployment problem is formulated as a constrained multi-objective trade-off over tail latency, mean time to repair (MTTR), availability, energy-normalized comfort and governance overhead. Evaluation follows a stage-consistent pipeline spanning simulation, hardware-in-the-loop experiments and a production-like prototype. Across these stages, the proposed architecture reduces P90 action latency by approximately 40%, reduces MTTR by about 60%, approaches 99.9% availability under fault injection, improves energy-normalized comfort by approximately 15% and keeps governance overhead within 3-5%. These findings suggest that governance-as-code can transform compliance from static documentation into a continuously enforced operational mechanism for trustworthy and sustainable AIoT systems.

Keywords: AIoT; governance-as-code; smart buildings; deterministic orchestration; tail latency; MTTR; reproducibility; cyber-physical systems; Industry 5.0

Introduction

Smart buildings are becoming dense cyber-physical environments in which sensors, controllers, AI services and human-facing applications interact continuously. These environments are central to sustainability because buildings account for a large share of final energy consumption and energy-related emissions. AIoT systems can improve comfort, reduce energy use and automate maintenance; however, they also introduce operational risk when model updates, network instability or policy violations reach physical control loops [1-4].

Current deployments often separate architecture, lifecycle governance and evaluation. Cloud-centric systems simplify management but suffer from network-dependent tail latency. Microservice architectures increase modularity but distribute security, privacy and resilience responsibilities across many services, complicating rollback and auditability. Building digital twins and HIL testbeds improve evaluation, yet many studies focus on control-level metrics and do not provide stage-consistent evidence for latency, MTTR, availability and compliance [5-9].

This study addresses the resulting gap by treating governance as executable code. Policies are not only documents or post-deployment checklists; they are machine-checkable predicates evaluated during builds, releases and runtime operation. The framework combines centralized deterministic orchestration with modular AI service contracts, enabling rapid innovation while preserving bounded control behavior. It is aligned with the logic of emerging AI governance frameworks such as the NIST AI RMF and ISO/IEC 42001, but translates governance expectations into enforceable system mechanisms [1,2,10].

The contribution of this shortened article is threefold: first, it defines a monolithic-modular AIoT architecture for deterministic smart-building operation; second, it integrates governance-as-code gates across delivery and runtime; third, it evaluates the framework across simulation, HIL and prototype stages using comparable metrics and reproducible artifacts. The manuscript is condensed from a longer version and is structured for journal-style submission.

Related Work and Research Gap

Smart-building AIoT research covers edge-cloud coordination, reinforcement learning for energy optimization, predictive maintenance, occupancy inference and semantic interoperability. Platforms such as BOPTEST support standardized control benchmarking, while semantic models such as Brick Schema and RealEstateCore improve metadata consistency [11-14]. These advances are important, but they do not fully integrate runtime governance, coordinated rollback and system-level reproducibility.

Distributed AIoT architectures also face the tail-at-scale problem: even when median latency is acceptable, fan-out communication and queuing effects increase P90/P99 latency, which is harmful for time-sensitive actuation [6]. Microservices add deployment flexibility but can fragment cross-cutting concerns such as access control, drift management, privacy checks and audit logging. Partial centralization can improve predictability, but prior work rarely combines this with executable governance policies and stage-consistent evaluation.

AI governance standards emphasize accountability, risk management, auditability and human oversight [1,2]. Yet standards generally describe what must be managed rather than how policies should be enforced in cyber-physical runtime systems. This creates a gap between governance documentation and operational behavior. The present framework addresses this gap by embedding policy gates into CI/CD and runtime enforcement so that every release and every admitted control intent can be traced to explicit predicates.

Table 1. Condensed comparison of AIoT approaches for smart buildings.

Approach	Strength	Limitation	Governance	System metrics
Cloud-centric AIoT	Centralized data and model management	Variable network tail latency	Usually external	Energy and analytics
Microservices/edge	Modularity and agility	Fragmented policy and rollback	Service-level	Latency partially reported
Digital twins/HIL	Safe evaluation and repeatability	Often disconnected from release governance	Limited	Control KPIs and faults
Semantic models	Metadata interoperability	No orchestration or recovery mechanism	Not applicable	Metadata quality
Proposed framework	Deterministic orchestration with modular services	Requires disciplined policy design	Executable across CI/CD and runtime	Latency, MTTR, availability, ENC, overhead

Materials and Methods

Governance-as-Code Architecture

The architecture is centered on a deterministic orchestrator that handles admission control, intent routing, capability resolution, priority scheduling, health management, policy enforcement and rollback. AI service modules are independently versioned and connected through standardized contracts that specify inputs, outputs, service-level objectives, resource limits and rollback semantics. This design preserves modular extensibility without allowing services to bypass the governance boundary.

A normalized device layer abstracts heterogeneous controllers and sensors, including BACnet/IP, Modbus/TCP, OPC-UA and MQTT. Device capabilities such as setpoint bounds, sampling rates and actuator limits are represented as first-class abstractions. Direct module-device interaction is prohibited, and all actuation traverses the orchestrator. This makes unsafe or incompatible requests rejectable before they reach physical devices.

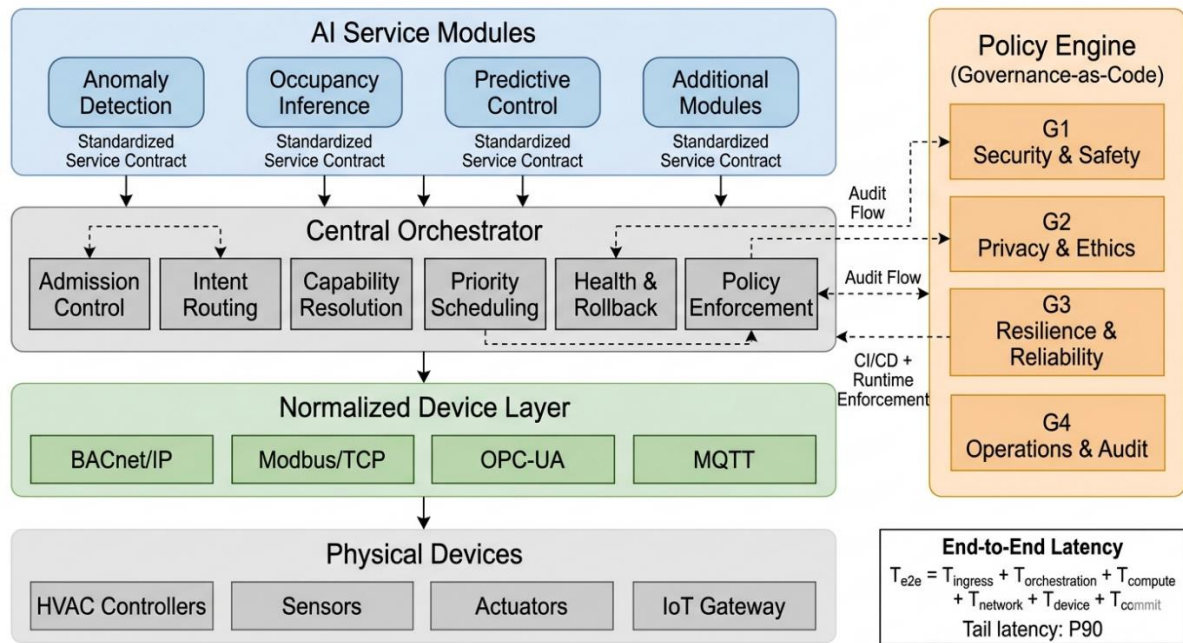


Figure 1. Governance-as-code AIoT architecture. The central orchestrator mediates AI service modules, the normalized device layer and the policy engine, while solid and dashed flows distinguish control, audit and feedback paths.

Executable Policy Gates and Runtime Recovery

Governance policies are written as executable predicates and organized into four gate families: security and safety, privacy and ethics, resilience and reliability, and operations and audit. Candidate releases must satisfy all gates before promotion. At runtime, the same policy logic supports admission, throttling, rejection, drift detection and rollback.

Each release produces a signed audit trail that links source artifacts, container digests, configuration hashes, policy outcomes, dataset identifiers and deployment timestamps. Persistent policy violations or SLO breaches trigger coordinated rollback to the last verified system state. This mechanism reduces recovery ambiguity and limits fault propagation across dependent services.

Governance-Embedded Delivery Lifecycle for AIoT Systems

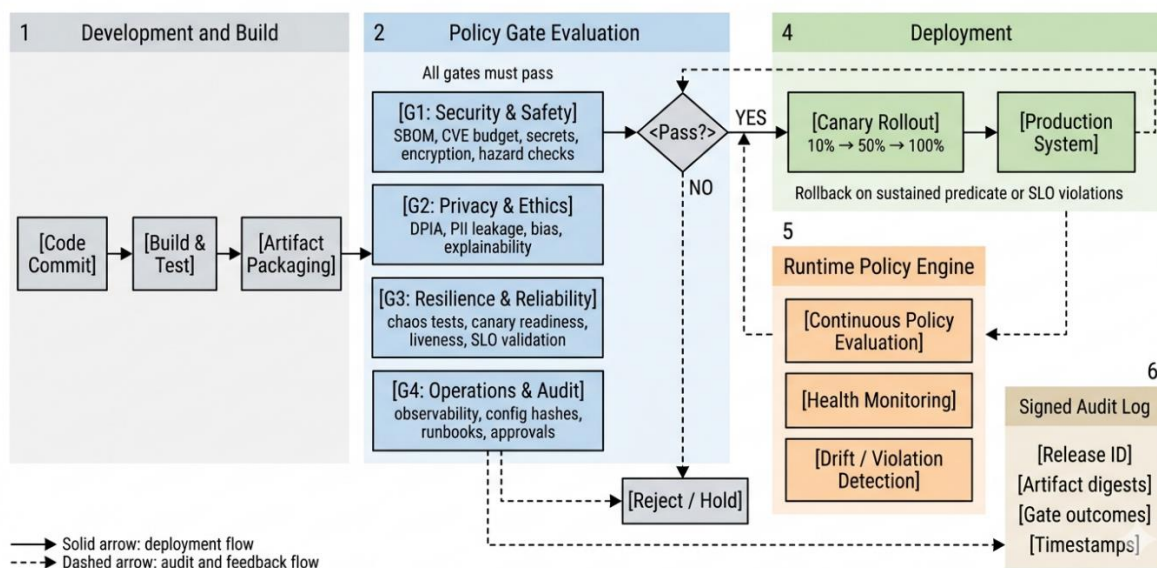


Figure 2. Governance-embedded delivery lifecycle. Candidate releases pass development, policy evaluation, canary rollout, runtime enforcement and signed audit logging before full production deployment.

Optimization Objectives and Evaluation Pipeline

System deployment is modeled as a constrained multi-objective problem. The optimization minimizes tail-aware latency, MTTR and governance overhead while maximizing availability and energy-normalized comfort (ENC). Constraints enforce safety, privacy and auditability. In compact form, the objective is minimize $[L_{tail}(x,p), MTTR(x,p), O_{gov}(p)]$ and maximize $[A(x,p), ENC(x,p)]$, subject to policy predicates $P_{safety}, P_{privacy}$ and P_{audit} .

The evaluation pipeline includes three stages: a portfolio-scale simulation with weather, occupancy and network stress inputs; an HIL stage with real controllers, sensors and network emulation; and a production-like prototype with staged canary rollout. All stages use harmonized datasets, fixed configuration snapshots, versioned policy bundles and synchronized observability to support direct comparison.

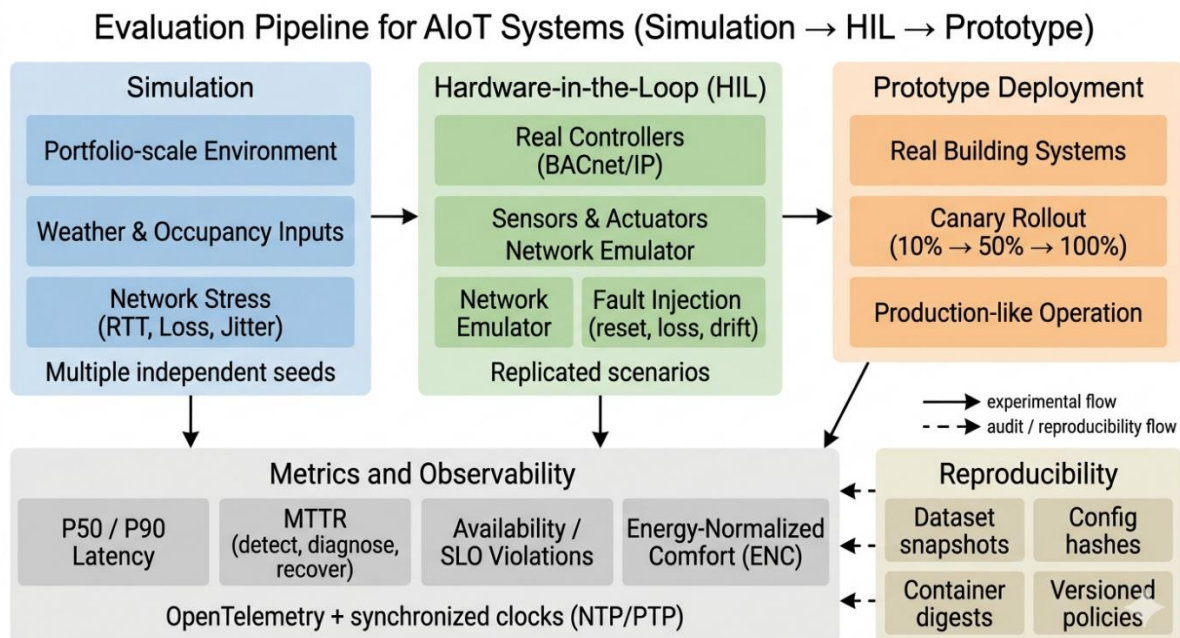


Figure 3. Stage-consistent evaluation pipeline. Simulation, HIL and prototype deployment share metrics, observability and reproducibility artifacts, enabling comparison across increasingly realistic environments.

Table 2. Evaluation stages and measurement focus.

Stage	Environment	Primary stressors	Measured outcomes
Simulation	Portfolio-scale environment	RTT, packet loss, jitter, stochastic occupancy	P50/P90 latency, SLO violations, ENC
HIL	Real BACnet/IP controllers and sensors	Fault injection, device resets, sensor drift	MTTR phases, availability, rollback behavior
Prototype	Production-like building deployment	Canary rollout and realistic schedules	Release cadence, overhead, audit traceability

Results

Across simulation, HIL and prototype stages, the monolithic-modular architecture with governance gates consistently outperforms cloud-centric and microservice baselines. The largest gains appear in tail latency and recovery, where deterministic orchestration and coordinated rollback directly reduce queuing uncertainty and diagnostic delay.

P90 action latency is reduced by approximately 40% relative to the cloud-centric baseline and by a substantial margin relative to the microservice baseline. The improvement remains visible in all three stages, indicating that the effect is not limited to simulation. Median P50 latency also decreases, but the stronger result is the contraction of the tail, which is more relevant for cyber-physical actuation.

Action Latency Across Evaluation Stages

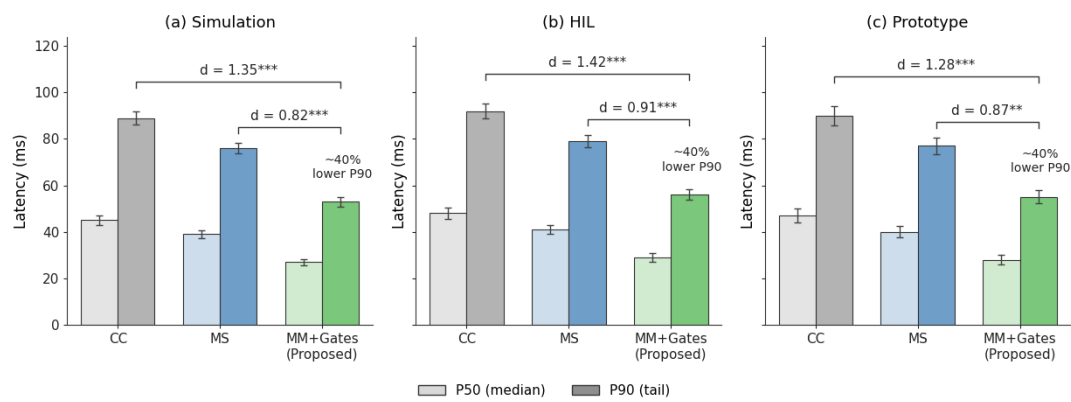


Figure 4. Action latency across simulation, HIL and prototype stages. The proposed MM+Gates architecture shows lower P50 and P90 latency than cloud-centric and microservice baselines.

MTTR decreases by approximately 60% because detection, diagnosis and recovery are coordinated through the orchestrator rather than distributed across independent services. The decomposition shows that the largest contribution comes from shorter diagnosis and deterministic rollback. This is consistent with the design goal of keeping failure handling in the orchestrator rather than distributing it across independent modules.

MTTR Decomposition Across Evaluation Stages

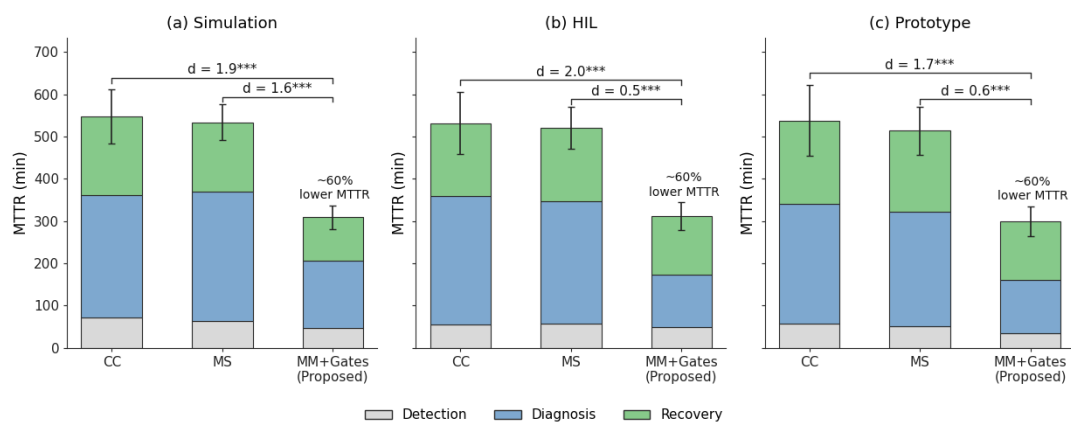


Figure 5. MTTR decomposition across evaluation stages. Coordinated diagnosis and rollback reduce recovery time in simulation, HIL and prototype conditions.

Availability approaches 99.9% under fault injection, with lower availability degradation during repeated disturbance windows. In contrast, cloud-centric and microservice baselines experience deeper dips because recovery is delayed by remote dependencies, distributed diagnosis and less coordinated state restoration.

Availability under Fault Scenarios

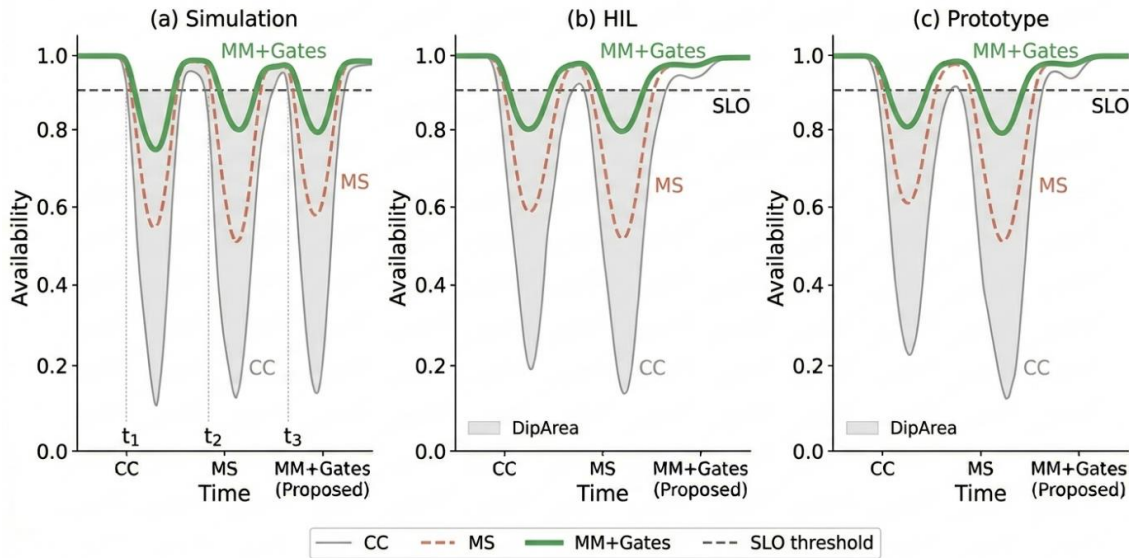


Figure 6. Availability under fault scenarios. The MM+Gates curve remains closer to the SLO threshold and shows lower degradation area across repeated disturbances.

Energy-normalized comfort improves by approximately 15%, indicating that governance and determinism do not prevent energy-aware optimization. The result suggests that a more controlled lifecycle can improve both resilience and sustainability by preventing unstable service versions from reaching production.

Energy-Normalized Comfort (ENC) Across Evaluation Stages

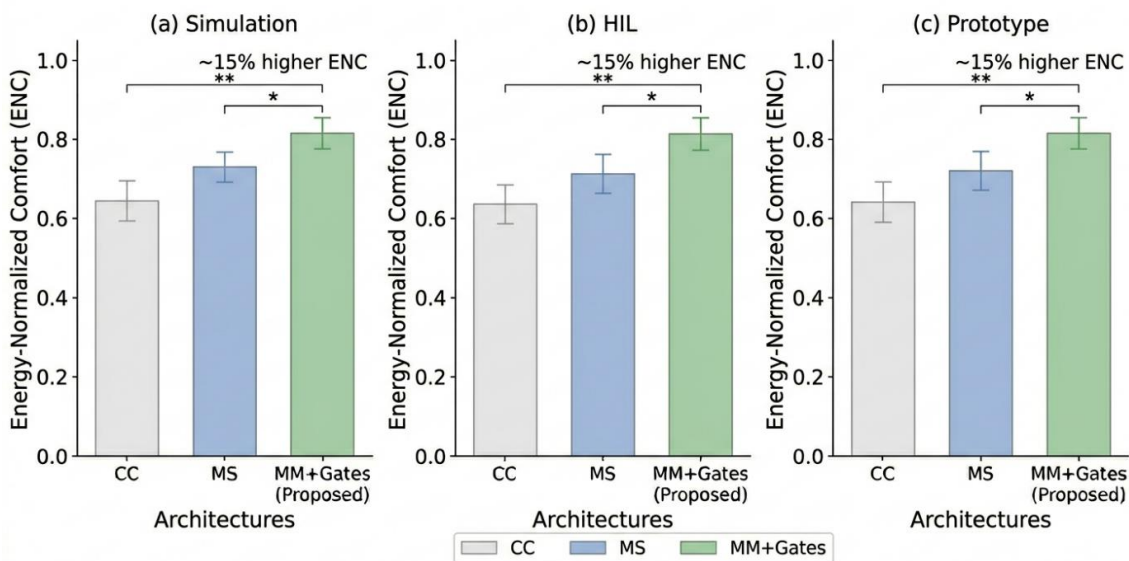


Figure 7. Energy-normalized comfort across evaluation stages. The proposed architecture improves the comfort-energy trade-off in simulation, HIL and prototype deployment.

Governance overhead remains bounded at roughly 3-5%. Although policy evaluation, audit packaging and resilience drills introduce measurable cost, the overhead is offset by faster recovery,

fewer SLO violations and shorter release time. The results therefore support the claim that executable governance can improve operational efficiency rather than merely add compliance burden.

Table 3. Condensed quantitative outcomes.

Metric	Cloud-centric	Microservices	MM+Gates	Improvement
P90 latency (ms)	90-92	76-80	53-56	about 40% lower vs CC
P50 latency (ms)	45-48	39-41	27-30	about 35% lower vs CC
MTTR (min)	520-550	500-530	300-310	about 60% lower vs CC
Availability (%)	98.5-99.0	99.0-99.3	about 99.9	+0.6 to +1.4 pp
ENC score	0.68-0.72	0.72-0.75	0.78-0.82	about 15% higher
Release time (min)	15-18	14-16	10-12	about 30% shorter
Governance overhead	-	-	3-5%	bounded

Multi-objective Trade-off Analysis of AIoT Architectures

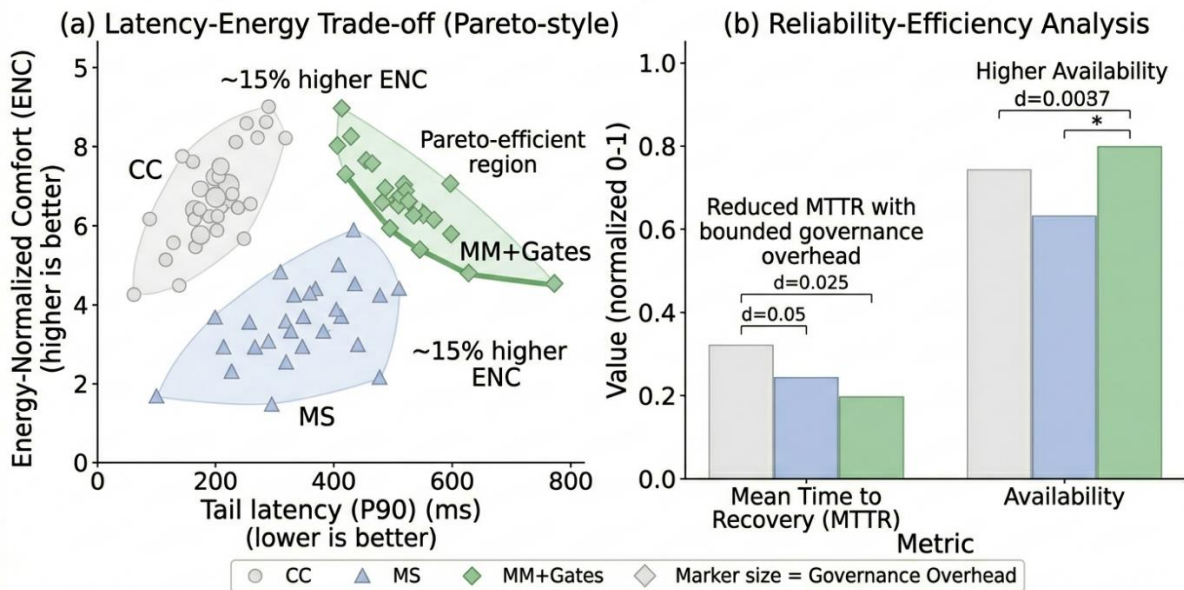


Figure 8. Multi-objective trade-off analysis. The proposed architecture occupies a more favorable region by combining lower tail latency, reduced MTTR, higher availability and improved energy-normalized comfort.

Table 4. Cross-stage meta-analysis and operational overheads.

Outcome	Pooled effect	95% CI	p-value	Interpretation
P90 latency	-35.8 ms	[-38.6, -33.1]	<0.001	Large tail-latency reduction
MTTR	-228.4 min	[-250.7, -206.2]	<0.001	Faster recovery
Availability	+0.92 pp	[+0.74, +1.10]	<0.001	Higher uptime

ENC	+0.11	[+0.08, +0.14]	<0.001	Better comfort-energy score
Governance overhead	3-5%	bounded	-	Operationally acceptable

Discussion

The results show that the main benefit of the framework comes from combining architectural determinism with executable governance. Centralized orchestration bounds fan-out effects and provides a single location for admission, scheduling and recovery decisions. Modular service contracts preserve development agility while preventing unverified modules from directly affecting actuators.

Governance-as-code changes the role of compliance. Instead of collecting evidence after a release, the proposed lifecycle produces evidence during development, policy evaluation, staged rollout and runtime monitoring. This supports auditability and reproducibility because each system state is linked to artifacts, configurations and policy outcomes.

There are limitations. The evaluation includes simulation, HIL and a production-like prototype, but broader external validity requires multi-building and multi-region deployments. Device diversity is also limited compared with real building portfolios. Finally, policy predicates require careful design; weak or incomplete policies would reduce the benefits of the framework. Future work should explore adaptive governance, federated orchestration and open benchmarks that capture both control and lifecycle metrics.

Conclusions

This article presented a shortened journal version of a governance-as-code AIoT framework for smart buildings. The framework integrates deterministic centralized orchestration, modular AI service contracts, normalized device abstraction and executable policy enforcement across CI/CD and runtime operation.

Stage-consistent evaluation shows approximately 40% lower P90 latency, 60% lower MTTR, near-99.9% availability under fault injection, 15% higher energy-normalized comfort and bounded governance overhead. These results indicate that trustworthy AIoT deployment can be treated as an operational systems problem in which governance, reliability and reproducibility are enforced continuously rather than documented separately. The framework provides a practical basis for sustainable, human-centric and auditable AIoT systems in Industry 5.0 environments.

Abbreviations

Abbreviation	Meaning
AIoT	Artificial Intelligence of Things
CI/CD	Continuous integration and continuous delivery
ENC	Energy-normalized comfort
HIL	Hardware-in-the-loop
MTTR	Mean time to repair
P50/P90	Median and tail latency percentiles
SLO	Service-level objective

OPA	Open Policy Agent
-----	-------------------

Reference

1. NIST. AI Risk Management Framework (AI RMF 1.0); NIST: Gaithersburg, MD, USA, 2023.
2. ISO/IEC 42001:2023. Information Technology-Artificial Intelligence-Management System; ISO: Geneva, Switzerland, 2023.
3. IEA. Buildings: Tracking Clean Energy Progress 2024; IEA: Paris, France, 2024.
4. UNEP. 2022 Global Status Report for Buildings and Construction; UNEP: Nairobi, Kenya, 2022.
5. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. IEEE Internet Things J. 2016, 3, 637-646.
6. Dean, J.; Barroso, L.A. The tail at scale. Commun. ACM 2013, 56, 74-80.
7. Richardson, C. Microservices Patterns: With Examples in Java; Manning: Shelter Island, NY, USA, 2018.
8. Beyer, B.; Jones, C.; Petoff, J.; Murphy, N.R. Site Reliability Engineering: How Google Runs Production Systems; O'Reilly Media, 2016.
9. ETSI. Zero-Touch Service Management (ZSM): Reference Architecture Release 2; ETSI GS ZSM 002 V3.1.1, 2020.
10. Open Policy Agent. OPA Gatekeeper Documentation v3.10, 2023.
11. BOPTEST Project. Building Optimization Testing Framework v3.2, 2024.
12. Brick Schema Consortium. Brick Schema 1.3 Release Notes, 2023.
13. RealEstateCore Consortium. RealEstateCore v4.0, 2024.
14. Pineau, J.; Vincent-Lamarre, P.; Sinha, K.; Lariviere, V.; et al. Improving reproducibility in machine learning research. J. Mach. Learn. Res. 2023, 24, 1-20.
15. Attia, A.; Du, Y.; Ahamer, G.; Wang, S.; Lee, K. Closed-loop model predictive control for energy-efficient buildings. Appl. Energy 2023, 332, 120456.
16. Vazquez-Canteli, J.; Nagy, Z. Reinforcement learning for demand response: A review of real-life applications in buildings. Energy Build. 2019, 208, 109636.
17. Kairouz, P.; McMahan, H.B.; Avent, B.; et al. Advances and open problems in federated learning. Found. Trends Mach. Learn. 2021, 14, 1-210.
18. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. AISTATS, 2017.
19. Wen, Q.; Zhou, T.; Zhang, C.; Chen, W.; Ma, Z.; Yan, J.; Sun, L. Transformers in time series: A survey. IJCAI, 2022.
20. IEA EBC Annex 81. Data-Driven Smart Building Operation and Maintenance, 2023.
21. Balasubramanian, V.; et al. Towards a shared and interoperable digital twin for buildings. Springer, 2023.
22. Zhang, Y.; et al. Reproducibility and benchmarking test methods for building energy forecasting models. Appl. Energy 2022, 321, 119307.