The BX3 Framework:

A Universal Architecture for Accountable Autonomous Systems

Three Functional Layers. Five Named Protocols. Guaranteed Upstream Accountability.

Jeremy Blaine Thompson Beebe
Independent Researcher
Preprint — Bxthre3 Inc, April 2026

April 2026

Abstract

The current discourse around artificial intelligence frequently presents a false binary: either AI will replace traditional software systems, or it is merely a productivity tool of limited consequence. This paper argues that both positions miss a more fundamental and practically useful insight. We propose the **BX3 Framework** — a universal architectural model organized around three immutable functional layers: the *Purpose Layer*, which provides intent, judgment, and accountability; the *Bounds Engine*, which provides interpretation, bounded reasoning, and constrained execution; and the *Fact Layer*, which provides deterministic enforcement, hard physical constraint, and forensic auditability.

Critically, the BX3 Framework does not prescribe *who* or *what* occupies each layer. It prescribes the *functional properties* each layer must maintain — and holds any actor occupying that layer to those properties regardless of their nature. A human, an AI system, or a mechanical process may each legitimately occupy any layer, provided they satisfy that layer's functional requirements. This actor-agnostic, function-centered definition makes the framework universally applicable across human organizations, fully automated systems, multi-agent AI architectures, and any hybrid composition.

A core safety property of the framework is its upstream accountability guarantee: when any node encounters a state it cannot resolve within its bounds, accountability escalates recursively upward through the system hierarchy — bypassing all machine actors — until it reaches a human anchor. *The system fails upward into human consciousness — never downward into algorithmic chaos.*

We demonstrate that when these three functional layers are clearly separated and their properties enforced by design, systems become simultaneously more capable and

less complex. We further argue that the Bounds Engine is most powerfully employed not as a replacement for the Fact Layer but as a tool for designing, accelerating, and improving it. We show that the BX3 Framework is not merely a software engineering principle but a universal structural pattern observable across law, medicine, autonomous systems, organizational design, and biological cognition. A companion engineering specification — the BX3 Protocol — is in preparation and will provide normative, certifiable implementation standards derived from this framework.

This paper is a conceptual framework and position paper. Empirical validation studies are currently in development by the author and will be published as subsequent work. This preprint establishes the theoretical foundation and research agenda that those studies will address.

Keywords: BX3 Framework, Purpose Layer, Bounds Engine, Fact Layer, artificial intelligence, deterministic systems, software architecture, human-in-the-loop, upstream accountability, recursive systems, autonomous systems, AI governance, sociotechnical systems, agentic systems, edge computing

---

1 Introduction

The rapid advancement and deployment of large language models and AI-based systems has generated significant confusion about the appropriate role of AI within engineered systems. A prevalent narrative suggests that AI will progressively replace traditional deterministic software, eventually subsuming most computational tasks. A counter-narrative dismisses AI as a novelty, inadequate for the reliability demands of production systems.

Both narratives fail engineers, architects, and decision-makers in the same fundamental way: they offer no principled model for how these technologies relate to one another or how they should be combined in practice. The result is systems that are either over-reliant on AI where determinism is required, or unnecessarily constrained by legacy deterministic thinking where AI could genuinely help.

This paper proposes a clarifying framework: the **BX3 Framework**. The framework organizes any complex system into three immutable functional layers — the *Purpose Layer*, the *Bounds Engine*, and the *Fact Layer* — each defined by the properties it must maintain rather than by the type of actor that occupies it.

This actor-agnostic, function-centered definition is the framework's most important architectural property. A human, an AI system, or a mechanical process may each legitimately occupy any layer of the BX3 Framework — provided they satisfy the functional requirements of that role. What the framework prescribes is not *who* belongs in each layer but *what properties* each layer must maintain for the system as a whole to be reliable, governable, and

certifiable.

A companion engineering specification — the BX3 Protocol — is in preparation and will provide normative, certifiable implementation standards derived from this theoretical foundation.

We further argue that the BX3 Framework is not a novel invention but a synthesis of well-established principles — separation of concerns [?], sociotechnical systems theory [?], control theory [?], and human-in-the-loop design [?] — applied to the specific and urgent challenge of AI integration in the current technological moment. Its value lies not in the novelty of its components but in the clarity, universality, and completeness of their unification.

Note: The first-person plural "we" is used throughout in the conventional academic sense, consistent with single-author scholarly writing.

2 Defining the Three Functional Layers

The BX3 Framework defines three functional layers. Each layer is defined by the *properties it must maintain*, not by the type of actor that occupies it. A human, an AI system, a mechanical process, an institution, or any combination thereof may occupy any layer — provided the functional requirements of that layer are satisfied.

This actor-agnostic definition is deliberate and essential. It makes the framework applicable to human-only organizations, to fully automated systems, to multi-agent AI architectures, and to any hybrid composition. It also resolves the false question of whether AI will "replace" humans in any given role: the question is not who occupies the layer, but whether the occupant satisfies the layer's functional requirements.

2.1 Layer 1: The Purpose Layer

The Purpose Layer is responsible for *intent*, *judgment*, and *accountability*. It sets Service Level Objectives, strategic goals, and the "why" that governs all downstream activity. Whatever occupies this layer must be capable of:

- Defining the goals the system exists to achieve and why.
- Making trade-off decisions under genuine uncertainty, where no algorithmic optimum exists.
- Holding accountability for outcomes — answering for the system's behavior to external parties.
- Asking and answering "why are we building this, and for whom?"

- Updating goals when context changes in ways the system was not designed to anticipate.

In the current technological moment, the Purpose Layer must remain anchored to a *human accountability anchor* — an individual or institution capable of bearing legal and ethical responsibility for the system's actions. This is the **Human Root Mandate**: in the event of system failure, accountability does not dissipate into the algorithm. It remains fixed to the human at the root. The framework does not preclude an advanced AI system from eventually occupying this layer, but until AI systems can be held meaningfully accountable for intent-level decisions, the Human Root Mandate applies.

Key property: The Purpose Layer must be *accountable* — its decisions must be attributable to an actor who can be questioned, overridden, and held responsible.

2.2 Layer 2: The Bounds Engine

The Bounds Engine is responsible for *interpretation*, *bounded reasoning*, and *constrained execution*. It performs the cognitive work of the system — analysis, pattern recognition, simulation, and optimal path proposal — but is architecturally *limbless*: it can propose but cannot execute. It lacks the authority to commit actions to the physical world unilaterally. Whatever occupies this layer must be capable of:

- Receiving goals and constraints from the Purpose Layer and translating them into proposed actions.
- Handling inputs that are ambiguous, variable, unstructured, or novel.
- Performing complex analysis — probabilistic modeling, trend analysis, simulation — within defined boundaries.
- Operating within a sandboxed cognitive environment, separated from physical execution authority.
- Escalating to the Purpose Layer when situations exceed its authority or capability.

This layer is most commonly occupied by an *AI agent or heuristic engine* in modern systems. However, a human expert, a hybrid human-AI team, or a sophisticated rule-based system may also occupy this layer when appropriate. The defining requirement is *bounded adaptability* — the ability to reason flexibly within hard constraints, never autonomously. The Bounds Engine proposes; the Fact Layer decides whether to execute.

Key property: The Bounds Engine must be *bounded* — its outputs must pass through Fact Layer validation before any physical action occurs, and its authority is strictly limited by Purpose Layer direction.

2.3 Layer 3: The Fact Layer

The Fact Layer is the physical firewall and brakes of the system. It acts as the deterministic gate through which all Bounds Engine proposals must pass before becoming real-world actions. Whatever occupies this layer must be capable of:

- Producing the same output given the same input, without exception.
- Hard-blocking any Bounds Engine proposal that violates a pre-defined safety, regulatory, or physical constraint — regardless of how confident the Bounds Engine is in its proposal.
- Maintaining a complete, tamper-evident forensic ledger of all decisions, proposals, and physical outcomes.
- Operating at the latency and reliability level required by the system's risk profile.
- Providing a basis for formal verification, regulatory certification, or legal accountability.

This layer is most commonly occupied by *deterministic software, rule engines, control systems, or physical mechanisms.* An AI system may occupy this layer only under strict conditions: fixed parameters, no generalization, no probabilistic inference, and formal verification against specification. The result of a properly implemented Fact Layer is that the system remains bounded by reality at all times — no Bounds Engine action, however confidently proposed, can violate a hard physical or regulatory constraint.

Key property: The Fact Layer must be *deterministic* — the same input must always produce the same output, all outputs must be auditable, and no Bounds Engine proposal may bypass it.

2.4 Role Occupancy Rules

The flexibility of the BX3 Framework — allowing any actor to occupy any layer — is bounded by three non-negotiable rules:

1. **Property satisfaction is mandatory.** An actor may only occupy a layer if it genuinely satisfies that layer's functional requirements. Claiming to occupy a layer without satisfying its properties is an architectural violation.
2. **Layer properties are non-negotiable.** The properties of each layer — accountability, boundedness, determinism — cannot be relaxed to accommodate an actor's limitations. If an actor cannot satisfy a layer's requirements, a different actor must be found, or the system cannot be considered BX3-compliant.

3. **All three layers must be present.** A system missing any layer is architecturally incomplete. Without the Fact Layer there are no hard constraints. Without the Purpose Layer there is no accountable intent. Without the Bounds Engine the system cannot handle the ambiguity and complexity of the real world.

3 The BX3 Framework: Structure and Properties

3.1 Structural Representation

The BX3 Framework organizes any complex system into three functional layers. The diagram below shows the default configuration, but any actor satisfying a layer's functional requirements may occupy that position. Note the critical constraint arrow from the Fact Layer directly to the Bounds Engine — the physical firewall does not merely inform; it hard-blocks:

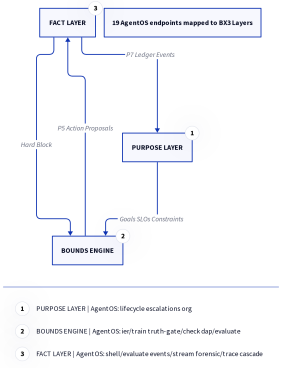

Figure 1: The BX3 Framework: Three immutable functional layers.

3.2 Why Separation Reduces Complexity

A counterintuitive but important property of the BX3 Framework is that explicitly adding a third named functional layer — separating the Bounds Engine from both the Purpose Layer and the Fact Layer — *reduces* overall system complexity rather than increasing it.

This occurs for three reasons:

Each layer is optimized for its function. The Fact Layer does not need to handle ambiguity. The Bounds Engine does not need to guarantee physical consistency. The Pur-

pose Layer does not need to manage execution. When each layer is relieved of responsibilities it handles poorly, the entire system becomes leaner and more reliable.

Interfaces become well-defined. Purpose-to-Bounds communication occurs through Service Level Objectives, goals, and constraints. Bounds-to-Fact communication occurs through structured action proposals against defined validation gates. Fact-to-Purpose feedback occurs through forensic ledger events, alerts, and escalation signals. Each interface is appropriate to its parties regardless of who or what occupies each layer.

Failure modes are isolated and directed upward. When the Fact Layer detects a constraint violation, it hard-blocks and escalates — it never fails silently. When the Bounds Engine exceeds its authority, the Fact Layer catches it. When the Purpose Layer makes a poor decision, it is an accountability problem with a named responsible party. Critically, failures propagate *upward* toward human consciousness, not downward into autonomous action.

3.3 The Functional Properties Table

Property	Purpose Layer	Bounds Engine	Fact Layer
Core function	Intent & SLO setting	Analysis & proposal	Physical enforcement
Required property	Accountable	Bounded (limbless)	Deterministic
Handles ambiguity	Yes (by design)	Yes (within bounds)	No
Consistent output	Variable	Mostly	Always
Auditable	Via attribution	Via reasoning trace	Fully (forensic ledger)
Latency	Slow (deliberate)	Moderate	Fast to instant
Can execute physically	Yes (override)	No (limbless)	Yes (only if validated)
Certifiable	Accountable	Difficult	Yes
Default occupant	Human / institution	AI agent / heuristic	Software / mechanism

3.4 Layer Interface Specification

The BX3 Framework defines not only what each layer must do but what crosses each layer boundary. Well-defined interfaces are the mechanism by which layer isolation is maintained in practice:

Interface	From	To	What Crosses the Boundary
Direction	Purpose Layer	Bounds Engine	Goals, SLOs, constraints, authorization scope
Proposal	Bounds Engine	Fact Layer	Structured action proposals, simulation outputs
Hard Block	Fact Layer	Bounds Engine	Constraint violation signals, blocked proposal receipts
Forensic Feedback	Fact Layer	Purpose Layer	Ledger events, escalation signals, performance metrics
Override	Purpose Layer	Fact Layer	Direct commands, emergency halt, Sandbox Gate approval
Escalation	Any node	Purpose Layer	Bailout signals when bounds are exceeded

Note that the Bounds Engine and Fact Layer never share a functional plane. The Bounds Engine cannot write directly to physical actuators. The Fact Layer cannot initiate reasoning. These are not software permissions — they are architectural separations enforced by Loop Isolation (Pillar 1).

4 The Five Pillars of BX3 Implementation

The three functional layers define *what* a BX3-compliant system must contain. The Five Pillars define *how* those layers must behave in practice to maintain their properties under real-world conditions including network failures, scale, security threats, and exception states. Each pillar addresses a specific failure mode that emerges when AI, deterministic, and human systems are combined without disciplined architectural separation.

4.1 Pillar 1: Loop Isolation

Problem solved: Logic Collision — when the Bounds Engine and the Fact Layer occupy the same functional plane, enabling un-vetted autonomous actions that bypass physical constraint.

Solution: Strict isolation of the three functional layers into discrete planes. Each BX3 loop is self-contained and operates independently. A Logic Collision is architecturally impossible because the Bounds Engine never shares a functional plane with physical execution. The Bounds Engine proposes; the Fact Layer decides. These are never the same operation.

A single human Purpose Layer can govern an arbitrarily large tree of Bounds Engine agents and Fact Layer mechanisms with absolute precision, because the accountability chain remains non-collapsing regardless of system scale.

4.2 Pillar 2: Recursive Spawning

Problem solved: Logic Rigidity — static edge devices that cannot adapt to local conditions without constant cloud connectivity.

Solution: A parent node (operating at the Bounds Engine layer) births a child BX3 loop by generating a *Worksheet* — a containerized, self-contained logic set encapsulating the parent's Purpose for a specific local context — and deploying it over-the-air to the child node.

Each Worksheet carries a hard-coded pointer to the parent's Purpose, preventing autonomous drift. The child loop applies the parent's intent to local sensor data independently, without requiring a constant cloud heartbeat. If cloud connectivity is lost, the child node executes the last-known-good Worksheet based on local inputs. The system maintains integrity and local reflexes in degraded network conditions (*Local Survivability*).

This mechanism allows a single human Purpose Layer to project authority and logic across an arbitrarily large distributed system while preserving BX3 layer integrity at every node:

Every spawned Worksheet maintains a hard-coded pointer to the parent's Purpose, preventing autonomous drift regardless of network conditions or child node failures.

4.3 Pillar 3: Spatial Firewall

Problem solved: Soft permissions that can be bypassed, and IP that is protected only by logical access controls rather than physical constraints.

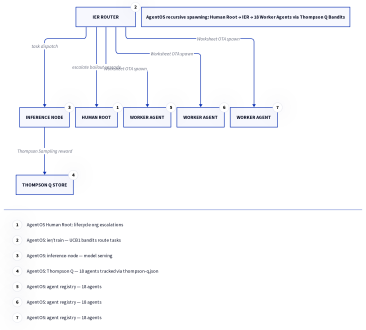

Figure 2: Recursive spawning: A parent Purpose Layer births child BX3 loops via Worksheet deployment, projecting authority across distributed systems while preserving layer integrity.

Solution: Access privileges and data resolution tiers are implemented as *physical, hard-coded constraints of the Fact Layer* — not as software permissions. The system's Fact Layer physically cannot serve data or execute actions beyond a node's provisioned authorization level.

When a Bounds Engine requests a capability beyond its provisioned tier, the Fact Layer does not return a "Permission Denied" error. Instead, it triggers an automated resolution pathway — which may include a commercial upgrade funnel, a human escalation, or a logged denial. Even if a Bounds Engine node is compromised, it cannot access or execute beyond what its Fact Layer gate physically permits. The firewall is in the physics, not the logic.

4.4 Pillar 4: Root Tunneling

Problem solved: Abstraction Leakage — the collapse of organizational hierarchy when a human operator accesses a sub-node, losing global context and breaking the recursive audit trail.

Solution: The *Root-Pipe Protocol* enables the human Purpose Layer to project authority into any node in the system without collapsing the hierarchy. When a tunnel is activated, the node's Purpose is redirected to the human operator's intent, all telemetry and logs are piped to the root dashboard, and all proposed Bounds Engine actions for that node must pass through a *Sandbox Gate* before reaching the Fact Layer.

The Sandbox Gate requires the Bounds Engine to model the projected outcome of any proposed action in a digital twin environment before physical actuators are unlocked [**?**]. The human reviews the simulation and provides explicit approval. Only after Human-in-the-Loop validation does the Fact Layer permit physical execution. This provides a high-stakes safety buffer against live-system errors while maintaining the full audit trail.

4.5 Pillar 5: Bailout Protocol

Problem solved: The "Black Box" problem — autonomous actions that are orphaned from human responsibility, creating unresolvable legal and operational risk.

Solution: A systemic escalation protocol that guarantees accountability always reaches a human. When any node encounters a state conflict it cannot resolve within its functional bounds, it escalates the exception upward through the recursive system hierarchy, bypassing all machine actors, until it reaches the human Purpose Layer anchor.

The system fails upward into human consciousness — never downward into algorithmic chaos.

No autonomous action is ever orphaned from its human source. Every exception has a traceable escalation path terminating at a named, accountable human.

The Forensic Ledger: Every event in the system is logged simultaneously across three discrete planes — the Purpose plane (what was authorized), the Bounds Engine plane (what was proposed and why), and the Fact plane (what physical action was taken and what was its outcome). This three-plane ledger provides the forensic standard required for high-stakes regulatory environments, proving that deterministic human oversight was structurally guaranteed throughout the lifecycle of every autonomous event.

5 The Bounds Engine Used to Build and Improve Fact Layer Systems

One of the most practically significant implications of the BX3 Framework is that the Bounds Engine's most powerful and appropriate use may not be *operating* within production systems, but *designing and improving* the Fact Layer systems that do. This inversion of the conventional narrative — the Bounds Engine as a builder of reliability rather than a replacement for it — is the framework's most actionable contribution for engineers and architects.

5.1 Pattern Discovery to Rule Encoding

Bounds Engine systems can be applied to large datasets to discover patterns, correlations, and decision boundaries that would be infeasible for humans to enumerate manually. Once discovered, these patterns can be encoded as deterministic Fact Layer rules, thresholds, or constraint tables — effectively graduating from probabilistic inference to reliable physical enforcement.

This is the model used in several mature autonomous systems domains, where machine

learning is used offline to develop and validate behavioral policies, and only validated, frozen logic is deployed to operational Fact Layer systems. The Bounds Engine's work happens in the design environment; the Fact Layer does the work in the field. The system benefits from the Bounds Engine's pattern recognition capability without inheriting its runtime unpredictability.

5.2 Bounds Engine-Accelerated Fact Layer Development

Bounds Engine systems are increasingly used to generate, review, and test deterministic Fact Layer code and constraint specifications. The output of this process is traditional deterministic software — auditable and certifiable. The Bounds Engine was a tool in the construction process, not a component of the runtime system. This preserves all the reliability properties of deterministic enforcement while dramatically accelerating development.

This pattern resolves a common false trade-off: the assumption that gaining Bounds Engine capability requires accepting Bounds Engine unpredictability at runtime. When the Bounds Engine is used to *build* Fact Layer systems rather than *replace* them, the organization gains speed and generalization at the design phase while retaining reliability and auditability at runtime.

5.3 Shadow Mode Validation

A powerful hybrid pattern involves running a Bounds Engine system in parallel with an existing Fact Layer system, comparing proposals against actual enforcement outcomes, and flagging divergences for Purpose Layer review. Over time, Bounds Engine behaviors that consistently improve on the Fact Layer can be promoted — after explicit Purpose Layer approval — into the Fact Layer itself.

This pattern is particularly valuable in regulated industries where replacing a certified Fact Layer system requires extensive re-certification. Shadow mode allows Bounds Engine-driven improvements to be identified, validated by the human Purpose Layer, and gradually incorporated without disrupting the certified operational system.

5.4 The Continuous Improvement Loop

The BX3 Framework enables a structured continuous improvement cycle that preserves system integrity at every stage:

1. The Fact Layer operates in production, generating performance data and constraint-event logs.

2. The Bounds Engine analyzes that data, identifying edge cases, inefficiencies, and improvement opportunities.
3. The Purpose Layer (human anchor) reviews Bounds Engine findings and exercises judgment about which improvements to pursue.
4. Approved improvements are encoded into the Fact Layer after validation and certification.
5. The cycle repeats — the system grows more capable without growing less reliable.

The Purpose Layer remains the gatekeeper through whom all Bounds Engine insight must pass before it becomes Fact Layer logic. Accountability is preserved throughout the improvement process, and the Forensic Ledger records every proposed change and its authorization status.

6 Application Across Domains

The BX3 Framework is not a software engineering principle alone. The same three functional layers — an accountable Purpose Layer, a bounded Bounds Engine, and a deterministic Fact Layer — appear across many complex domains. In each case, different types of actors occupy each layer, confirming that the framework's power lies in its functional definitions rather than in any prescription about actor type.

6.1 Autonomous Systems and Sensor Networks

In sensor-driven autonomous systems, the BX3 Framework maps directly onto system architecture. Human engineers occupying the Purpose Layer define mission parameters, safety constraints, and acceptable risk envelopes. AI processing layers occupying the Bounds Engine interpret sensor data, handle ambiguous environments, and propose real-time decisions within those constraints. Deterministic control systems occupying the Fact Layer enforce hard constraints — collision avoidance thresholds, actuator limits, safety interlocks — that cannot be overridden by the Bounds Engine under any circumstances.

This architecture is critical where the cost of Bounds Engine error is physical and potentially irreversible. The Fact Layer does not need to handle every situation — only to prevent catastrophic outcomes while the Bounds Engine and Purpose Layer resolve ambiguity.

6.2 Legal Systems

Legal systems demonstrate the actor-agnostic property of the BX3 Framework clearly. Legislatures and judges occupy the Purpose Layer — defining the purpose and interpretation of law, exercising judgment in novel cases, and bearing institutional accountability. AI systems increasingly occupy parts of the Bounds Engine — assisting with legal research, document analysis, and pattern recognition across case law. The written law itself, procedural rules, and enforcement mechanisms occupy the Fact Layer — the same statute applies the same way in the same circumstances, regardless of who the parties are.

6.3 Medicine

Physicians occupy the Purpose Layer — exercising judgment incorporating patient context, ethical considerations, and probabilistic reasoning beyond any protocol. AI systems occupy portions of the Bounds Engine — assisting with imaging analysis, drug interaction checking, and population-level pattern recognition. Drug dosage protocols, surgical checklists, equipment operation specifications, and regulatory requirements occupy the Fact Layer — deterministic rules that constrain both physician and AI behavior alike.

6.4 Organizational Design

Organizations themselves exhibit the BX3 structure. Executive leadership occupies the Intent Layer — setting strategic direction and bearing accountability. Knowledge workers and AI-augmented teams occupy the Bounds Engine — interpreting strategy and executing flexibly within organizational guidelines. Policies, compliance requirements, contractual obligations, and financial controls occupy the Fact Layer — rules that apply consistently regardless of who is executing or what the AI recommends.

6.5 Biological Cognition

Notably, the BX3 Framework mirrors the layered architecture of biological cognition [**?**]. The autonomic nervous system and reflexes occupy the Fact Layer — deterministic, fast, and non-negotiable. Learned heuristics, intuitions, and pattern recognition occupy the Bounds Engine — efficient responses to familiar situations. Conscious deliberative reasoning occupies the Purpose Layer — slow, deliberate, and engaged only for genuinely novel, high-stakes, or ethically complex situations.

This convergence suggests the BX3 Framework reflects something deeper than an engineering preference. The same three-layer functional architecture appears to be a near-optimal

solution for any system that must simultaneously be reliable, adaptive, and accountable — regardless of whether that system is biological, organizational, legal, or computational.

7 Why Deterministic Systems Will Not Be Replaced

A common claim is that sufficiently advanced AI will eventually subsume deterministic software. We argue this position misunderstands the distinct value of determinism as a *property*, not merely a *limitation*.

7.1 The Value of Determinism is Not Compensable

Determinism provides properties that probabilistic systems cannot replicate:

- **Reproducibility:** The ability to reproduce any past output given the same input, enabling debugging, auditing, and legal accountability.
- **Formal verification:** The ability to mathematically prove that a system satisfies certain properties under all possible inputs.
- **Certification:** Regulatory frameworks in aviation, medical devices, financial systems, and safety-critical infrastructure require deterministic behavior as a precondition for approval.
- **Latency:** Hard real-time requirements (microsecond response times) are achievable with deterministic systems and not with current AI inference pipelines.

7.2 The Infrastructure Argument

Every AI system in production today runs on top of vast deterministic infrastructure: operating systems, databases, networking stacks, authentication systems, billing pipelines, and monitoring tools. The claim that AI will replace software is structurally self-refuting — the AI systems making this possible are themselves dependent on deterministic software that will not and should not be replaced.

7.3 The Regulatory and Trust Barrier

Even in domains where AI could theoretically replace deterministic systems on a performance basis, the practical barriers are substantial. Organizations with core operations dependent on software will not replace functioning, auditable, certified systems with probabilistic alternatives without extraordinary evidence of equivalent reliability and auditability. The burden

of proof is appropriately high, and current AI systems do not meet it for most production contexts.

8 Relationship to Prior Work

The BX3 Framework synthesizes established ideas from separation of concerns, sociotechnical systems theory, cybernetics, systems safety, and human-in-the-loop design, but it departs from prior work by defining immutable functional layers according to required properties rather than according to actor type. The closest contemporary analogue is the intelligent sociotechnical systems framework [?], which similarly emphasizes structured coordination between human and technical components; however, BX3 extends this line of work by explicitly isolating a deterministic Fact Layer and by specifying an actor-agnostic architecture in which any occupant of a layer is bound by that layer's functional obligations.

Recent agentic-AI literature reinforces several needs that BX3 attempts to formalize. Tiered Agentic Oversight (TAO) [?] demonstrates that hierarchical supervision among specialized agents can reduce error propagation in safety-critical settings. BX3 differs in making human accountability the required terminal endpoint of unresolved escalation rather than a high-tier supervisory option. TAO's hierarchy routes to human oversight as a high-risk escalation pathway; BX3 makes this routing unconditional and architectural. Likewise, recent work on production-grade agent architectures [?] recommends fail-safe behavior, sandbox-first execution, deterministic fallback workflows, and human approval gates for risky actions; BX3 incorporates these concerns as named architectural pillars — most notably the Sandbox Gate and Bailout Protocol — rather than as implementation heuristics.

While recent agentic-architecture literature recommends sandbox-first execution and human approval gates for high-risk actions, BX3 specifies a narrower architectural requirement: proposed interventions must be evaluated in a digital twin and cleared through a role-bounded Sandbox Gate tied to Root Tunneling authority before the Fact Layer is unlocked.

BX3 also sits naturally alongside emerging governance and compliance literature. Koch [?] argues that standards such as ISO/IEC 42001 [?] and the NIST AI RMF [?] do not themselves provide implementable runtime guardrails, and instead require translation across governance, design-time, runtime, and assurance layers. BX3 may be read as one candidate architecture for that translation. In parallel, formal-verification approaches such as the Lean-Agent Protocol [?] show how proposed agentic actions can be forced through deterministic verification gates before execution. BX3 generalizes that intuition beyond theorem proving: wherever execution must remain non-probabilistic, a protected deterministic

layer is architecturally indispensable.

The recurrence of layered models in global AI governance discourse [**?**] further suggests that complex AI systems are increasingly being understood through stratified functional abstractions, although those models are policy-descriptive rather than engineering-prescriptive.

9 Conclusion

The question of how humans, AI, and traditional software should relate to one another is not merely a technical question. It is an architectural, organizational, ethical, and regulatory question with significant and growing practical consequences.

The BX3 Framework proposes a principled and universal answer. It organizes any complex system into three functional layers — Purpose, Bounds Engine, and Fact — each defined by the properties it must maintain rather than by the type of actor that occupies it. Any actor capable of satisfying a layer's functional requirements may occupy that layer: human, AI, mechanical, institutional, or hybrid. What cannot vary are the properties themselves — accountability in the Purpose Layer, boundedness in the Bounds Engine, and determinism in the Fact Layer.

This actor-agnostic definition is the framework's most important contribution beyond its predecessors. It makes the BX3 Framework applicable to the full range of systems that exist and the full range of systems that are coming: human-only organizations, fully automated pipelines, multi-agent AI architectures, and hybrid compositions that do not yet have names. In each case, the framework asks the same three questions: Is there an accountable layer that sets purpose and bears responsibility? Is there a bounded layer that reasons and proposes within defined constraints? Is there a deterministic layer that enforces, validates, and audits? If any layer is missing or its properties are not maintained, the system is architecturally incomplete — regardless of how capable its components are individually.

The BX3 Framework is not entirely new. Its pattern is visible in legal systems, medical practice, autonomous vehicles, biological cognition, and organizational design — wherever reliable systems have been built to handle a world that is simultaneously rule-bound and unpredictable. What is new is the urgency of making the pattern explicit, naming its layers by function rather than actor, and building from it deliberately — at a moment when the temptation to collapse these roles, and the cost of doing so, have never been higher.

Acknowledgments

The author wishes to acknowledge the foundational contributions of the researchers cited herein, whose work across control theory, sociotechnical systems, and computer science provides the shoulders on which this synthesis stands.

This work has not undergone peer review. Comments and correspondence are welcomed.