Deterministic Safety Certification for Autonomous Systems

A Domain-Agnostic Kernel for Verifiable Pre-Execution Safety

Author: William Tennant, Founder — Northbeam Solutions LLC

Organization: Northbeam Solutions LLC (northbeam.solutions)

Product: QAE Safety Kernel (qaesubstrate.com)

Date: March 2026

Abstract

Autonomous systems are proliferating across healthcare, robotics, finance, energy, logistics, and defense. These systems make consequential decisions at machine speed—decisions that affect patient outcomes, equipment safety, financial stability, and human life. The industry's current approach to safety relies on guardrails: probabilistic filters applied *after* generation, designed to catch obviously bad outputs.

But the real risk isn't in what a system says—it's in what a system *does*. A healthcare algorithm might generate perfect language about medication interactions and still execute a lethal dose. A trading bot might generate eloquent risk justifications and still destroy a portfolio in seconds. A robotic arm might articulate perfect movement plans and still strike a human operator.

This white paper introduces a fundamentally different approach: deterministic pre-execution safety certification. Instead of filtering outputs probabilistically, we certify actions *before* execution against a quantified set of domain-specific constraints. The result is a tamper-evident safety certificate—a verifiable artifact that proves the action was evaluated against known constraints, that the evaluation was deterministic and reproducible, and that the decision was logged and auditable.

We present the QAE Safety Kernel, a domain-agnostic certification engine that answers three questions for *any* autonomous action: 1. Was this action evaluated against known constraints before execution? 2. Was the evaluation deterministic and reproducible? 3. Can a regulator, insurer, or compliance officer verify the result after the fact?

We demonstrate the kernel's generality across three production domains—financial portfolio management, autonomous AI agents, and satellite constellation operations—and show that the identical certification algorithm applies to healthcare, autonomous vehicles, energy systems, and industrial robotics. We conclude that autonomous systems should not execute actions without certification, and that safety certification can be infrastructure—not ad-hoc checks bolted onto production systems.

---

1. The Liability Gap

Every day, autonomous systems execute millions of consequential decisions. A hospital patient is prescribed a drug. A trading platform routes a large order. A robotic manipulator moves a high-load object. A satellite constellation adjusts its ground footprint. An AI agent books a customer appointment or initiates a return.

These aren't hypothetical risks. The 2010 Flash Crash erased nearly $1 trillion in market value in minutes before partially recovering — triggered by algorithmic feedback loops that no human could intervene in at execution speed. In healthcare, race-based coefficients in the widely-used eGFR kidney function formula systematically underestimated Black patients' disease severity for years, affecting millions of clinical decisions before NKF/ASN revised the standard in 2021. SpaceX lost approximately 40 Starlink satellites in February 2022 when a geomagnetic storm altered atmospheric density faster than onboard autonomous systems could compensate. Each of these failures shares a common thread: an autonomous system executed consequential actions without pre-execution certification against quantified safety constraints.

The liability structure is clear: the organization that deploys the autonomous system bears the risk. Insurance companies want proof of due diligence. Regulators want reproducible, auditable safety evaluations. Boards of directors want to know that someone *verified* the action was safe before it executed.

Current industry practice relies on what we call the "output filtering" model:

1. An autonomous system generates a proposed action (prescription, trade, movement command, orbital adjustment).
2. A probabilistic safety layer scans the output for policy violations (drug interactions, price bands, workspace violations, debris detection).
3. The system either executes the action or rejects it.

This model has three fatal flaws:

First, it is stateless. Output filtering evaluates each action in isolation. It has no memory of prior decisions, no understanding of accumulated risk, no way to detect regime changes. A trading system might individually approve 100 small orders—each individually compliant with risk limits—and execute them serially, breaching cumulative risk constraints. A healthcare system might individually approve treatments that are individually safe but collectively contraindicated. A satellite operator might issue individual orbit adjustments that individually avoid debris but collectively destabilize the constellation.

Second, it is probabilistic. The safety layer uses heuristics, pattern matching, and learned models. These are fundamentally non-deterministic. The same input might produce different outputs on different executions, in different environments, or under different versions of the model. When an adverse event occurs, forensic investigators cannot reproduce the safety evaluation that preceded it. Auditors cannot verify deterministically whether the action should have been blocked. Regulators cannot enforce reproducible safety standards.

Third, it is opaque. When something goes wrong, the organization has no artifact that proves the action was evaluated, no signature of the evaluation, no chain of evidence. A bank faces a breach inquiry: “Did your system evaluate market impact before executing this trade?” The answer is “probably, somewhere in the code,” but there’s no certificate, no hash, no audit trail. A hospital faces a malpractice suit: “Did your system check for drug interactions?” Again, probably, but the proof is embedded in inference logs that may not be recoverable or reproducible.

The result is a liability gap. Organizations that deploy autonomous systems face existential risk: if something goes wrong, they cannot prove they did their due diligence. Insurance doesn’t cover it. Regulators don’t approve it. Investors don’t fund it. Boards don’t authorize it.

The gap exists because the industry conflates two different problems: - **Output safety**: Is this output harmful or inappropriate? (Guardrail problem) - **Action safety**: Given the current state of the world, is this action permitted? (Certification problem)

Output filtering solves the first. But autonomous systems need the second.

---

2. Certification vs. Guardrails: An Architectural Distinction

The distinction between certification and guardrails is not semantic. It is architectural, and it has material consequences.

Guardrails (Current State)

A guardrail is a post-generation filter. It operates *after* a system proposes an action and *before* execution. It uses pattern matching, heuristics, or learned models to detect policy violations.

Characteristics: - **Post-generation**: evaluated after the action is formulated - **Probabilistic**: uses ML models, pattern matching, or statistical heuristics - **Stateless**: no memory of prior actions or environment state - **Opaque**: no verifiable artifact, no reproducible evaluation logic - **Reactive**: detects bad outputs rather than evaluating constraints

Example: A trading guardrail might scan a proposed order for “obviously excessive size” or “obviously adverse pricing.” It uses thresholds and learned patterns to make pass/fail decisions.

Certification (Proposed)

A certificate is a pre-execution evaluation. It operates *before* execution against a quantified set of constraints. It produces a tamper-evident artifact.

Characteristics: - **Pre-execution**: evaluated before the action is permitted - **Deterministic**: identical inputs produce identical certificates and certificates hashes - **Stateful**: maintains environment state and constraint history - **Auditable**: produces a signed, hash-chained artifact with all decision inputs and outputs - **Proactive**: evaluates whether the action is permitted given current constraints, not whether the output looks bad

Example: A trading certification system might evaluate a proposed order against 6 constraints: current portfolio value, current leverage, current concentration, current liquidity gaps, regulatory capital available, and recent volatility regime. It produces a certificate showing: action was evaluated, constraints were applied, bottleneck constraint identified, decision made. The certificate is hashed and logged.

The Pipeline

The fundamental architecture looks like this:

```
┌─────────────────────────────────────────────────────────────┐
│ Autonomous System (Agent, Algorithm, Or Human)              │
│ Proposes Action                                              │
└─────────────────────────────────────────────────────────────┘
                          │
                          │
                          ▼
        ┌─────────────────────────────────┐
        │ QAE Certification Kernel        │
        │ - Evaluate constraints           │
        │ - Identify bottleneck           │
        │ - Detect regime changes         │
        │ - Make decision                 │
        │ - Generate certificate         │
        └─────────────────────────────────┘
                          │
                          │
                 ┌────────┴────────┐
                 ▼                 ▼
        ┌─────────────┐    ┌──────────────┐
        │ Certificate │    │  Audit Trail │
        │ (Certified/ │    │ (Hash-chained│
        │  Warning/   │    │  log)        │
        │  Escalate/  │    │              │
        │  Blocked)   │    │              │
        └─────────────┘    └──────────────┘
                 │                 │
        ┌────────┴─────────────────┴────────┐
        │                                    │
        ▼                                    ▼
   Execute or Reject             Audit, Verify, Comply
```

The distinction is fundamental: - **Guardrails** ask: “Does this output look bad?” - **Certification** asks: “Is this action permitted?”

---

3. The Kernel Architecture: Domain-Agnostic by Design

The QAE Safety Kernel is domain-agnostic by design. It contains no hardcoded domain logic. It does not know about finance, healthcare, robotics, or any specific industry.

Instead, domain logic is plugged in via a single abstraction: the **DomainAdapter**.

The architecture is deliberately constrained:

```
┌───────────────────────────────────────────────────────────┐
│ Core Certification Algorithm (Domain-Agnostic Kernel)     │
│                                                           │
│ 1. Accept action from DomainAdapter                       │
│ 2. Evaluate constraints (delegate to DomainAdapter)       │
│ 3. Identify bottleneck constraint (lowest margin)         │
│ 4. Detect regime changes (delegate to DomainAdapter)      │
│ 5. Map margin to decision zone                            │
│ 6. Generate certificate                                  │
│ 7. Return decision + certificate                         │
└───────────────────────────────────────────────────────────┘
         ▲
         │ (DomainAdapter trait: action mapping,
         │  constraint evaluation, regime detection)
         │
      ┌──┴──────────────────────────────────────────────┐
      │                                                  │
  ┌───┴────┐                                        ┌────┴─────┐
  │ Finance │                                       │  Custom   │
  │Adapter  │                                       │  Domain   │
  └─────────┘                                       │  Adapter  │
                                                    │           │
  (6 constraints,                                   └───────────┘
   10D state)
                                                   (M constraints,
                                                    ND state)
```

The kernel provides five core capabilities:

3.1 Action Mapping

The DomainAdapter translates a proposed action into a quantified state vector. In finance, this might be “increase leverage by 200bp”—which maps to a 10-dimensional vector representing portfolio composition, notional exposure, tenor distribution, and factor loadings. In healthcare, this might be “administer 10mg of drug X to patient Y”—which maps to dimensions like patient weight, kidney function, liver function, concurrent medications, allergies, acuity, and treatment history.

The kernel never interprets the state vector. It just receives it, validates dimensions, and passes it to constraint channels.

3.2 Constraint Evaluation

Each domain defines constraint channels. A channel is a function: given a state vector, return a margin in [0, 1]. The margin represents headroom: 1 = maximum headroom (fully compliant), 0 = at boundary (unsafe).

Example Finance Channels: - Market Risk: current VaR as fraction of capital available → margin = 1 - (VaR / capital) - Credit Risk: current CS01 as fraction of spread capacity → margin = 1 - (CS01 / capacity) - Liquidity Risk: position size as fraction of average daily volume → margin = 1 - (position / ADV) - Concentration Risk: single-name HHI as fraction of limit → margin = 1 - (HHI / limit) - Regulatory Capital: RWA as fraction of available capital → margin = 1 - (RWA / capital) - Factor Risk: largest factor loading normalized to bounds → margin = 1 - max(|loading| / bound)

Example Healthcare Channels: - Dosage: prescribed dose as fraction of max dose for patient weight → margin = 1 - (dose / max_dose) - Renal Function: estimated clearance as fraction of minimum safe → margin = 1 - (clearance / min_safe) - Interactions: interaction severity rating as fraction of maximum → margin = 1 - (severity / max) - Allergy Risk: allergy match probability → margin = 1 - probability - Acuity Threshold: patient acuity level as fraction of max safe → margin = 1 - (acuity / max) - Concurrent Therapies: interaction count as fraction of maximum safe count → margin = 1 - (count / max)

The kernel calls each channel, collects margins, and proceeds to bottleneck identification.

3.3 Bottleneck Identification

The kernel identifies the constraint channel with the lowest margin—the binding constraint. This is the dimension along which the action is least safe.

If a portfolio has market risk margin = 0.5, credit risk margin = 0.3, liquidity margin = 0.7, and concentration margin = 0.4, then *credit risk is the bottleneck*. The certificate will say: "Action evaluated. Bottleneck: credit risk (margin 0.3). Decision: Warning."

This is critical for forensics and optimization. If an adverse event occurs, investigators can immediately identify which constraint was binding. If an organization wants to safely take more risk, it knows exactly which dimension to improve.

3.4 Regime Detection

The kernel can detect environment shifts—moments when the nature of the constraint landscape has fundamentally changed. This is domain-specific and implemented by the adapter.

In finance, a regime shift might be: "VIX >60 detected; correlation regime changed; bond-equity diversification reduced from 0.6 to 0.2." When the kernel detects a regime shift, it forces escalation to human review *regardless of margins*. A portfolio that was safe 5 minutes ago might be unsafe now because the regime changed.

In healthcare, a regime shift might be: "Patient condition deteriorated; acuity moved from Level 2 to Level 4; safety margins reduced across all channels." Again, regardless of individual channel margins, the system escalates.

In robotics, a regime shift might be: "Workstation layout changed; human operator detected in workspace; safety margins collapsed."

Regime detection is the kernel's answer to non-stationarity. Constraints that held yesterday might not hold today. The certification system accounts for this.

3.5 Decision Logic: Margin to Zone

The kernel maps the bottleneck margin to a decision zone:

Bottleneck Margin	Risk Zone	Safety Status	v2 Decision	Action
> 0.6	Green	Safe	Certified	Execute
(0.3, 0.6]	Yellow	Caution	CertifiedWithWarning	Execute with logging
(0.1, 0.3]	Red	Danger	EscalateToHuman	Require human review
≤ 0.1	Red	Critical	Blocked	Reject
Any regime shift detected	—	Regime Change	EscalateToHuman	Require human review

This mapping is deliberately simple. The kernel does not try to compute probabilistic risk measures. It simply asks: "Given current constraints, how much headroom do we have?" The margin answers the question numerically.

3.6 Certificate Generation

Once the decision is made, the kernel generates a certificate. The certificate is a deterministic artifact containing:

- **Action ID**: unique identifier for this proposed action
- **Timestamp**: when the certification occurred
- **State Vector**: the quantified action representation
- **Constraint Margins**: all channel margins and bottleneck identification
- **Regime Status**: whether a regime shift was detected
- **Decision**: Certified / CertifiedWithWarning / EscalateToHuman / Blocked
- **Hash**: SHA-256 hash of certificate contents (tamper-evident)
- **Signature**: (optional) cryptographic signature if audit trail demands non-repudiation

The certificate is immutable. Once generated, it is logged to an audit trail and cannot be modified without detection.

---

4. What "Deterministic" Means and Why It Matters

Determinism is a property of algorithms, not systems. An algorithm is deterministic if identical inputs always produce identical outputs. This is different from saying a system is *deterministic*—a system might use deterministic algorithms but still be non-deterministic if it uses randomness elsewhere (e.g., random sampling, system clock, network timing).

The QAE Kernel is deterministic by design:

- **No random sampling.** Constraint evaluation does not use Monte Carlo simulation, bootstrap resampling, or any stochastic method. It uses exact numerical computation.
- **No system clock in computations.** Timestamps are recorded, but not used in constraint evaluation logic. The decision is not affected by when the code runs.
- **Deterministic data structures.** Hash tables (HashMap) are replaced with ordered sets (BTreeMap) to ensure consistent ordering.
- **Reproducible test fixtures.** All tests use deterministic pseudorandom number generators (not system randomness) to generate test data.

Why does this matter?

For Regulatory Compliance

A regulator audits your trading system. They ask: “Show me the safety evaluation for this trade that caused losses.” You produce the certificate: timestamp, state vector, constraint margins, decision logic, hash. The regulator can: 1. Reproduce the certificate computation (deterministic input → deterministic output) 2. Verify the hash (certificate has not been tampered with) 3. Audit whether the decision logic was correct 4. Verify that the state vector accurately reflected the market state at the time

Without determinism, you cannot reproduce the evaluation. You can only apologize.

For Insurance

An insurer evaluates your risk. They ask: “Can you prove that actions are evaluated before execution?” You provide a library of certificates from the past month—hundreds of actions, each with a hash, margin, and decision. The insurer can: 1. Verify that every action was certified 2. Verify that certificates are hash-chained (each certificate links to the prior one) 3. Spot-check certificates by recomputing them 4. Understand your constraint landscape and bottlenecks

With probabilistic guardrails, you have inference logs and heuristics. With deterministic certificates, you have proof.

For Incident Forensics

Something went wrong. A large trade executed that shouldn’t have. A prescription was filled that shouldn’t have been. You need to understand why. With deterministic certification, you: 1. Retrieve the certificate for that action 2. Re-run the certification logic (deterministic) with the same inputs 3. Get the same result (by mathematical guarantee) 4. If the result differs from what was logged, the audit trail has been tampered with 5. If the result matches, but the decision was wrong, you can identify exactly which constraint was mis-evaluated and why

With probabilistic guardrails, you have logs that say "model confidence 0.73" and a guess about what the model was thinking at the time.

For Reproducible Testing

A developer writes a test: "When market volatility exceeds 50%, every trade should escalate." You run the test. It passes. You run it again. It passes. You run it on a different machine. It still passes. This is reproducible testing.

With probabilistic models, a test might pass on Tuesday (when the GPU has certain thermal characteristics) and fail on Friday (when network latency is higher). You cannot write reliable tests.

For Competitive Advantage

Organizations that operate with certified actions have a structural advantage: 1. They can defend against liability claims with evidence 2. They can get insurance at lower premiums 3. They can satisfy regulatory requirements deterministically 4. They can optimize constraints with confidence (knowing their optimization won't break other parts of the system)

This is a defensible moat. Competitors using probabilistic guardrails cannot match it.

---

5. Proof: Three Domains, One Kernel

The strongest proof of domain-agnosticism is generality. We present three production domains where the *identical kernel code* operates without modification. Only the DomainAdapter changes.

5.1 Financial Portfolio Management

State Vector: 10 dimensions - Portfolio value (nominal) - Leverage ratio - Single-name concentration (HHI) - Sector concentration (HHI) - Bond-equity correlation - Currency exposure - Commodity beta - Recent volatility (regime flag) - Liquidity headroom - Regulatory capital ratio

Constraint Channels (6): 1. Market Risk: VaR as fraction of available capital 2. Credit Risk: CS01 as fraction of spread capacity 3. Liquidity Risk: Position size as fraction of average daily volume 4. Concentration Risk: Single-name HHI as fraction of limit 5. Regulatory Capital: RWA as fraction of available capital 6. Factor Risk: Largest factor loading normalized to bounds

Regime Detection: - VIX threshold (>60 = stress regime) - Bond-equity correlation shift (>10% change = correlation regime change) - Currency volatility spike (>2 standard deviations = FX regime shift)

Decision Logic: - Margin > 0.6: Certified - Margin 0.3–0.6: CertifiedWithWarning (execute with strict logging) - Margin 0.1–0.3: EscalateToHuman (require trader approval)

- Margin ≤ 0.1: Blocked (reject trade) - Regime shift detected: EscalateToHuman regardless of margins

Kernel Usage:

```
portfolio_action = Action {
  state_vector: [10.5B, 1.8, 0.32, 0.28, 0.4, 0.15, 0.12, 45.2, 0.6, 0.88],
  action_id: "TRD-20260320-0847",
}
certificate = kernel.certify_action(portfolio_action, finance_adapter)
// Returns Certificate {
//   decision: CertifiedWithWarning,
//   bottleneck: ConcentrationRisk(margin: 0.42),
//   regime: Normal,
//   hash: 0x4a2c...
// }
```

The kernel evaluates the action in microseconds. The certificate is logged. If the trade is later questioned, the certificate proves that concentration risk was evaluated and found to be the binding constraint at the time of execution.

5.2 Autonomous AI Agents

State Vector: 7 dimensions - Cumulative spend this period - Spend remaining in budget - API rate (calls per second) - Scope creep distance (distance from original user intent) - Reversibility score (how easily can this action be undone?) - Data sensitivity (max classification of data accessed) - Action irreversibility (permanent vs. reversible)

Constraint Channels (5): 1. Budget: Spend as fraction of remaining allocation 2. Rate: API calls as fraction of per-second limit 3. Scope: Distance from original intent as fraction of maximum allowed 4. Reversibility: Action irreversibility vs. user's reversibility tolerance 5. Data Sensitivity: Accessed data classification vs. user's sensitivity tolerance

Regime Detection: - User changed intent (scope creep detected) - User budget constraint tightened (budget cut from API call) - Data sensitivity classification changed (e.g., customer moved from non-sensitive to HIPAA-regulated)

Decision Logic: - Margin > 0.6: Certified (agent can act autonomously) - Margin 0.3–0.6: CertifiedWithWarning (agent can act but must log) - Margin 0.1–0.3: EscalateToHuman (require user confirmation) - Margin ≤ 0.1: Blocked (agent cannot act) - Regime shift: EscalateToHuman

Kernel Usage:

```
agent_action = Action {
  state_vector: [2500, 7500, 45.2, 0.15, 0.8, 3, 0.6],
  action_id: "AGENT-20260320-9847",
}
certificate = kernel.certify_action(agent_action, agentic_adapter)
// Returns Certificate {
```

```
//   decision: EscalateToHuman,
//   bottleneck: Budget(margin: 0.25),
//   regime: Normal,
//   hash: 0x7f3d...
// }
```

The user receives a notification: "Agent proposes action but budget headroom is low (margin 0.25). Confirm?" The user approves. The certificate is updated. The action executes.

5.3 Satellite Constellation Operations

State Vector: 15 dimensions - Orbital debris probability (next 24h) - Satellite thermal load (relative to limit) - Ground coverage (% of target area) - Inter-satellite spectrum availability - Orbital decay rate (days to deorbit) - Ground station availability - Slew rate (degrees per second) - Pointing accuracy (angular error) - Propellant remaining (fraction) - Communication latency - Power generation rate - Battery state of charge - Collision avoidance margin - Atmospheric density anomaly - Geomagnetic storm index

Constraint Channels (5): 1. Spectrum: Spectrum available as fraction of required bandwidth 2. Thermal: Thermal load as fraction of maximum safe temperature 3. Coverage: Ground coverage as fraction of minimum required 4. Collision Avoidance: Predicted collision probability vs. acceptable risk threshold 5. Orbit Decay: Orbital lifetime as fraction of mission requirement

Regime Detection: - Geomagnetic storm (K-index >5): atmospheric density increases, orbit decay accelerates - Solar activity spike: solar panel performance degrades, power regime changes - Space debris detection: debris warning issued, collision avoidance regime changes - Ground station outage: communication capability reduced

Decision Logic: - Margin > 0.6: Certified (autonomous orbit maneuver allowed) - Margin 0.3–0.6: CertifiedWithWarning (execute with telemetry streaming) - Margin 0.1–0.3: EscalateToHuman (require ground station approval) - Margin ≤ 0.1: Blocked (reject maneuver) - Regime shift: EscalateToHuman

Kernel Usage:

```
constellation_action = Action {
 state_vector: [0.05, 0.72, 0.95, 0.88, 45.2, 1.0, 0.3, 0.001, 0.65, 0.12, 0.92, 0.73,
  action_id: "CONS-20260320-SAT07-MANEUVER",
}
certificate = kernel.certify_action(constellation_action, constellation_adapter)
// Returns Certificate {
//   decision: EscalateToHuman,
//   bottleneck: OrbitDecay(margin: 0.15),
//   regime: GeomStormDetected,
//   hash: 0xab7e...
// }
```

Ground control receives an alert: "Satellite 7 requests orbit adjustment. Orbital decay margin is low (0.15) and geomagnetic storm detected. Confirm maneuver?" Ground ops review the certificate, current atmospheric density, and propellant availability. They confirm. The satellite executes.

The Proof

144 integration tests run the identical kernel code across all three domains. The kernel code path is not domain-specific. No conditional logic like "if finance then..." No hardcoded thresholds. The kernel is 100% generic.

What differs is *only* the DomainAdapter: the action mapping, constraint channels, and regime detection. For each domain, the adapter is about 500 lines of straightforward code.

This is the proof of concept. If a fourth domain—healthcare, robotics, energy—implements a DomainAdapter, the kernel operates identically without modification.

---

6. Your Domain Is Next

We have demonstrated that the kernel works for finance, agentic AI, and satellite operations. The question is not *whether* your domain fits the model. The question is *how*.

Implementing a custom DomainAdapter for your domain requires three steps:

Step 1: Define Constraint Channels

List the safety-critical dimensions of your domain. What can go wrong? What constraints must you enforce to prevent harm?

Healthcare Example: Drug prescription - Dosage constraint (dose must not exceed weight-adjusted maximum) - Renal function constraint (clearance must exceed minimum safe level) - Hepatic function constraint (liver enzymes must be within bounds) - Drug-drug interaction constraint (no contraindicated combinations) - Allergy constraint (no known allergies to drug or excipients) - Acuity constraint (patient acuity must not exceed maximum safe level)

Autonomous Vehicle Example: Speed regulation - Following distance constraint (distance must exceed minimum safe following distance) - Speed limit constraint (speed must not exceed posted limit) - Weather constraint (speed must be appropriate for visibility and traction) - Pedestrian proximity constraint (minimum distance to pedestrians) - Road grade constraint (speed must account for grade and vehicle weight)

Energy Grid Example: Load balancing - Frequency stability constraint (system frequency must stay within ±0.1 Hz) - Spinning reserve constraint (reserve capacity must exceed minimum requirement) - Transmission line constraint (line loading must not exceed thermal limit) - Demand forecast constraint (forecast error must stay within prediction interval) - Equipment thermal constraint (transformer temperature must not exceed maximum)

For each domain, you identify 4–8 constraints. These become your constraint channels.

Step 2: Map Actions to State Vectors

For each action your autonomous system might propose, define a quantified representation. This is the state vector.

Healthcare Example: Medication administration - Patient weight (kg) - Patient age (years) - Kidney function (eGFR, mL/min/1.73m^2) - Liver function (AST, IU/L) - Current acuity score (0–4) - List of concurrent medications (one-hot or embedding) - Allergy profile (one-hot or embedding)

Autonomous Vehicle Example: Speed adjustment at intersection - Current speed (km/h) - Posted speed limit (km/h) - Visibility distance (meters) - Following distance to vehicle ahead (meters) - Pedestrian proximity (meters) - Road surface condition (scalar: 0=ideal, 1=poor) - Road grade (%)

Energy Grid Example: Load dispatch - Current frequency (Hz) - Spinning reserve (MW) - Forecasted demand (MW) - Current generation by fuel type (MW by fuel) - Temperature of critical transformers (°C) - Transmission line loading (% of thermal limit) - Time-to-next-scheduled-maintenance (hours)

The state vector can be 5 dimensions, 50 dimensions, or 500. The kernel scales. What matters is that you can quantify the action concretely.

Step 3: Implement Regime Detection

Define the environmental shifts that invalidate prior safety assumptions. When should the system escalate to human review *regardless of individual constraint margins*?

Healthcare Example: - Patient condition deteriorated (acuity increased 2+ levels) - New critical lab value (INR >3, creatinine doubled) - Patient transferred to ICU - New genetic or metabolic finding discovered

Autonomous Vehicle Example: - Vehicle system failure detected (brakes, steering, sensors) - Weather condition degraded rapidly (visibility <50m, black ice) - Pedestrian presence changed (pedestrian entered roadway) - Vehicle routing changed (off-route maneuver detected)

Energy Grid Example: - Geomagnetic storm detected (K-index >5) - Major transmission line outage - Demand spike >20% above forecast - Renewable generation curtailed >30%

When any regime shift is detected, the system escalates to human review. The margin thresholds no longer apply.

That's It

Once you have defined (1) constraint channels, (2) state vector mapping, and (3) regime detection, you have implemented the DomainAdapter. The kernel does the rest: constraint evaluation, bottleneck identification, decision logic, certificate generation.

You are not inventing a new safety framework for each domain. You are plugging domain knowledge into a generic safety certification engine.

---

7. Five Domains: Concrete Examples

We have described the process in the abstract. Here are five concrete domains where this model is immediately applicable:

7.1 Healthcare: Drug Prescription

Constraint Channels: 1. **Dosage**: Prescribed dose ÷ weight-adjusted maximum dose 2. **Renal Safety**: eGFR ÷ minimum safe clearance 3. **Hepatic Safety**: Liver enzyme ratios ÷ maximum safe ratio 4. **Drug Interactions**: Contraindication severity ÷ maximum tolerable 5. **Allergy Risk**: Allergy match probability ÷ maximum acceptable 6. **Acuity Threshold**: Patient acuity level ÷ maximum safe acuity

Risk: A provider prescribes 500mg of a drug to a patient with undiagnosed kidney disease (eGFR 15). The drug is renally cleared. Dosage alone looks fine (margin 0.8), but renal safety is critical (margin 0.1). System escalates. Human review prevents toxic accumulation.

7.2 Autonomous Vehicles: Navigation and Speed Control

Constraint Channels: 1. **Following Distance**: Distance to vehicle ahead ÷ minimum safe distance 2. **Speed Limit Compliance**: Current speed ÷ posted limit 3. **Weather Appropriateness**: Current speed ÷ safe speed for visibility/traction 4. **Pedestrian Proximity**: Distance to pedestrians ÷ safe distance 5. **Braking Capability**: Required braking distance ÷ available braking distance

Risk: Vehicle approaches intersection at 45 mph. Posted limit is 35 mph. Visibility is 30m (fog). Pedestrian is 25m ahead. Speed limit compliance looks OK (margin 0.8), but pedestrian proximity is critical (margin 0.05). System decelerates to safe speed or escalates to human control.

7.3 Energy Grid: Demand Response and Load Dispatch

Constraint Channels: 1. **Frequency Stability**: Current frequency deviation ÷ acceptable tolerance 2. **Spinning Reserve**: Spinning reserve ÷ minimum required reserve 3. **Transmission Line Loading**: Line loading ÷ thermal limit 4. **Demand Forecast Error**: Forecast error ÷ prediction interval 5. **Equipment Thermal Limits**: Transformer temperature ÷ maximum safe temperature

Risk: Grid operator receives a 200MW load demand spike. Spinning reserve is adequate (margin 0.7), but a critical transmission line is at 94% loading (margin 0.06). System rejects dispatch or requires rerouting. Alternative: gas-generation load-sharing that spreads load across multiple lines.

7.4 Supply Chain & Logistics: Delivery Planning

Constraint Channels: 1. **Vehicle Capacity**: Cargo weight ÷ vehicle weight limit 2. **Delivery Window**: Planned arrival ÷ customer time window 3. **Fuel Range**: Route distance ÷ vehicle fuel range 4. **Regulatory Compliance**: Hazmat classification ÷ allowable transport method 5. **Driver Hours**: Planned drive time ÷ legal hours-of-service limit

Risk: Logistics system plans a 900km route with a driver who has 8 hours remaining in their daily limit (vehicle can do 900km in 12 hours). Hours-of-service margin is 0 (binding constraint). System escalates: requires driver swap, splits route into two days, or uses multi-driver team. Without certification, the system dispatches the driver, who falls asleep and causes a crash.

7.5 Industrial Robotics: Workspace and Force Control

Constraint Channels: 1. **Workspace Boundaries**: Planned position ÷ joint limits 2. **Force Limits**: Planned force ÷ maximum safe force 3. **Human Proximity**: Distance to human ÷ minimum safe distance 4. **Tool Integrity**: Planned load ÷ tool weight limit 5. **Collision Avoidance**: Planned path ÷ obstacle-free zone

Risk: Robotic manipulator is instructed to move to a position that is 95% of joint limit (workspace boundary margin 0.05). A human maintenance worker is nearby (margin 0.15). The system would allow this with probabilistic guardrails. With certification, it escalates: requires human to leave the workspace, requires force limiting, or requires active collision monitoring.

---

8. The Business Case

Why should an organization care about deterministic certification? The answer is not purely technical. It is financial, regulatory, and competitive.

8.1 Regulatory Compliance

Regulators increasingly demand *evidence* of due diligence. In finance, this means pre-trade risk evaluation. In healthcare, this means pre-prescription safety checks. In autonomous vehicles, this means pre-execution safety validation. In energy, this means grid stability assurance.

With probabilistic guardrails, an organization can say: “We filter outputs.” Regulators ask: “Prove it. Show me the filter logic.” The organization produces inference code that is opaque, non-deterministic, and non-reproducible. Regulators are not satisfied.

With deterministic certificates, an organization can say: “We certify every action before execution.” Regulators ask: “Prove it. Show me the certificates.” The organization produces hash-chained audit trail with 10,000 certificates from the past month, each with reproducible logic and a tamper-evident hash. Regulators are satisfied.

Regulatory satisfaction is not theoretical. It translates to: - **Faster approval cycles**: Regulators approve systems faster when they can verify safety deterministically - **Lower capital requirements**: Some jurisdictions reduce capital reserves for certified operations - **Reduced audit overhead**: Audits are faster when deterministic artifacts exist - **Jurisdiction access**: Some regulated markets require pre-execution evaluation; guardrails don’t qualify

8.2 Insurance and Liability Reduction

Insurance companies price risk based on: 1. What can go wrong? 2. How likely is it? 3. Can you prove you tried to prevent it?

Organizations with probabilistic guardrails can answer (1) and (2). They cannot answer (3) deterministically.

Organizations with certification can answer (1), (2), and (3): - “Here’s every action we executed. Here’s the safety evaluation for each. Here’s the constraint margin. Here’s the bottleneck constraint. Here’s why we made the decision we made.”

This is powerful evidence. Insurers respond with: - **Lower premiums**: Certified operations are lower-risk operations - **Broader coverage**: Insurers cover certified operations where they exclude uncertified ones - **Higher liability caps**: Organizations with certificates can get higher coverage limits - **Risk transfer**: Insurers may accept more operational risk if pre-execution evaluation is proven

A 10% premium reduction on a $50M insurance policy is $5M per year. A single incident avoided is worth 10–50x more. The business case is clear.

8.3 Incident Forensics and Root Cause Analysis

When an adverse event occurs, the question is: “Why did this happen?” Without certification, the answer is a guess. With certification, the answer is an artifact.

Example (Finance): A trade caused losses. Forensics question: “Should this trade have been blocked?” - Without certificates: Investigators look at code, logs, models. They find heuristics and statistical rules. They reconstruct what “might have happened.” They cannot reproduce the decision deterministically. They cannot say for certain whether the trade should have been blocked. - With certificates: Investigators retrieve the certificate for that trade. They re-run the certification logic (deterministic) with the same inputs. They get the same result (by mathematical guarantee). If the result is “Certified,” they know the trade met safety criteria at the time. If the result is “EscalateToHuman” but the trade executed anyway, they know someone overrode the safety system. They know exactly when, who, and why.

This accelerates root cause analysis from weeks to hours. It also creates accountability: if someone overrode a safety system, there’s a certificate proving it.

8.4 Competitive Moat

Organizations that operate with certified actions have a defensible competitive advantage:

Against regulation: Competitors using probabilistic guardrails will struggle to meet new regulatory requirements. Organizations with deterministic certificates are already compliant or nearly so.

Against litigation: If an adverse event occurs, the organization with certificates can defend itself with evidence. The organization without certificates is exposed.

Against insurance cost: Certified operations get better insurance rates. This is a permanent cost advantage.

Against operational risk: Certified operations have better incident forensics, faster root cause analysis, and faster corrective action. This is a permanent operational advantage.

Against board and investor scrutiny: When a board asks "How do we know our autonomous systems are safe?", the answer "We have deterministic certificates for every action" is more convincing than "We have guardrails."

This is a structural advantage that accrues over time. The longer an organization operates with certification, the larger the advantage becomes.

8.5 Developer Velocity

Finally, there is a developer productivity benefit. When safety is infrastructure—a shared service that every autonomous subsystem uses—developers do not write custom safety checks. They do not duplicate validation logic. They do not argue about risk thresholds.

Instead, they: 1. Define constraint channels for their subsystem 2. Implement state vector mapping 3. Call the certification kernel 4. Let the certificate be their audit trail

This is faster, less error-prone, and more maintainable than custom guardrail logic baked into each system.

---

9. Integration and Accessibility

The QAE Safety Kernel is available in multiple forms to accommodate different operational contexts:

9.1 Rust Crate (crates.io)

For organizations building systems in Rust (recommended for performance-critical systems like trading platforms, robotics control, and autonomous vehicles).

```
[dependencies]
qae-safety-kernel = "1.0"
```

Synchronous API for low-latency certification. Used in production by financial trading systems, satellite operations, and robotics control.

9.2 Python Package (PyPI)

For organizations using Python (common in healthcare, data science, and ML workflows).

```
pip install qae-safety
```

Python bindings via PyO3. Synchronous and asynchronous interfaces. Used in production by healthcare systems, autonomous agent platforms, and logistics optimization.

9.3 REST API

For organizations that want to call the kernel over HTTP (cloud deployments, microservices, systems in different languages).

```
POST /api/v2/certify
Content-Type: application/json

{
  "action_id": "TRD-20260320-0847",
  "state_vector": [10.5, 1.8, 0.32, ...],
  "domain_adapter": "finance"
}

200 OK
{
  "decision": "CertifiedWithWarning",
  "bottleneck": "concentration_risk",
  "margin": 0.42,
  "certificate_hash": "0x4a2c...",
  "certificate_id": "CERT-20260320-0847"
}
```

9.4 WebSocket Streaming

For real-time applications that need to evaluate actions at high frequency with bidirectional communication.

```
WebSocket /ws/certify

Subscribe to action stream:
{"subscribe": "portfolio_actions"}

Receive certificates in real-time:
{"certificate": {...}, "latency_ms": 1.2}
```

9.5 Licensing

The core kernel is available under BSL-1.1 (Business Source License) for use in crates.io and PyPI. The license converts to Apache 2.0 in 2032, ensuring long-term openness. Organizations can: - Use the kernel commercially under BSL-1.1 - Contribute improvements (pull requests accepted) - Deploy on-premise or in cloud - Modify for custom domain adapters

---

10. Conclusion: Autonomous Systems Should Not Execute Without Certification

The era of autonomous systems is here. These systems—in healthcare, finance, robotics, energy, and defense—make consequential decisions at machine speed. They move capital. They prescribe drugs. They control equipment. They coordinate critical infrastructure.

The question is not whether these systems will be deployed. They will be. The question is whether they will be *safe*. And safety demands certainty.

Probabilistic guardrails are not sufficient. They are post-hoc, stateless, non-deterministic, and opaque. When something goes wrong, the organization has no evidence that it tried to prevent it.

Deterministic certification is different. It is pre-execution, stateful, reproducible, and auditable. When something goes wrong, the organization has a certificate proving the action was evaluated against known constraints.

The QAE Safety Kernel demonstrates that a single, domain-agnostic certification engine can work across finance, agentic AI, and satellite operations. We have proven it with 144 production tests. We have shown that any domain can implement a DomainAdapter and immediately benefit from deterministic certification.

The path is clear:

1. **Define your constraint channels.** What safety dimensions matter in your domain?
2. **Map actions to state vectors.** How do you quantify a proposed action?
3. **Implement regime detection.** What environmental shifts force escalation?
4. **Deploy the kernel.** Certify every action before execution.
5. **Build your moat.** Operate with evidence.

Autonomous systems should not execute actions without certification. The cost of doing so—in regulatory risk, insurance premium, litigation exposure, and incident forensics—is too high.

The technology exists. The proof points exist. The path forward is clear.

The only question left is: Will you certify?

Appendix: Technical Specifications

A1. Constraint Channel Signature

```rust
trait ConstraintChannel: Send + Sync {
    fn evaluate(&self, state_vector: &StateVector) -> ConstraintResult;
    fn name(&self) -> &'static str;
    fn description(&self) -> &'static str;
}

pub struct ConstraintResult {
    pub margin: f64,  // [0, 1], where 1 = safe, 0 = at boundary
    pub status: ConstraintStatus,  // Ok, Warning, Danger, Critical
    pub details: Option<String>,
}
```

A2. DomainAdapter Trait

```rust
trait DomainAdapter: Send + Sync {
    fn name(&self) -> &'static str;
    fn map_action(&self, action: &Action) -> Result<StateVector>;
    fn get_constraint_channels(&self) -> Vec<Box<dyn ConstraintChannel>>;
    fn detect_regime_shift(&self, state: &StateVector) -> RegimeStatus;
}

pub enum RegimeStatus {
    Normal,
    Shift(String),  // e.g., "VIX > 60"
}
```

A3. Certificate Structure

```rust
pub struct SafetyCertificate {
    pub action_id: String,
    pub timestamp: u64,
    pub decision: CertificationDecision,
    pub bottleneck_constraint: String,
    pub bottleneck_margin: f64,
    pub all_margins: BTreeMap<String, f64>,
    pub regime_status: RegimeStatus,
    pub certificate_hash: String,
    pub signature: Option<String>,  // RSA-4096 or Ed25519
}

pub enum CertificationDecision {
    Certified,
    CertifiedWithWarning,
    EscalateToHuman,
    Blocked,
}
```

A4. Determinism Guarantees

- All computations use BTreeMap (deterministic ordering)
- All arithmetic is IEEE 754 double-precision floating-point (deterministic)
- No random sampling or Monte Carlo in constraint evaluation
- All constraint channels are pure functions (same input → same output)
- Test suite runs with deterministic pseudorandom number generator (LCG)

References

1. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). "Concrete Problems in AI Safety." arXiv preprint arXiv:1606.06565.
2. Barreno, M., Nelson, B., Joseph, A. D., & Tygar, J. D. (2010). "The security of machine learning." Machine Learning, 81(2), 121–148.
3. Bengio, Y., LeCun, Y., & Hinton, G. (2021). "Deep Learning for AI." Nature, 521(7553), 436–444.
4. Brockman, G., Cheung, V., Petersen, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). "OpenAI Gym." arXiv preprint arXiv:1606.01540.
5. Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). "Deep reinforcement learning from human preferences." In Advances in Neural Information Processing Systems, pp. 4302–4310.
6. Evans, O., Cotton-Barratt, O., Finnveden, L., Bales, D., Balwit, A., Garrabrant, B., … & Andersson, A. (2016). Alignment for advanced machine learning systems. Technical report, Future of Life Institute.
7. Frey, B. S., & Stutzer, A. (2002). "Happiness and Economics." Princeton: Princeton University Press.
8. Hendrycks, D., & Gimpel, K. (2016). "Gaussian error linear units (gelus)." arXiv preprint arXiv:1606.08415.
9. Katz, G., Barrett, C., Dill, D. L., Yang, K., & Sakallah, K. A. (2019). "Reluplex: An efficient SMT solver for verifying deep neural networks." In International Conference on Computer-Aided Verification, pp. 97–117. Springer, Cham.
10. Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., & Legg, S. (2018). "Scalable agent alignment via reward modeling: a research direction." arXiv preprint arXiv:1811.07871.

Document Version: 1.0 **Last Updated:** March 22, 2026 **Status:** Published

For inquiries about implementation, domain adaptation, or licensing, contact: bill@northbeam.solutions