LLM Sandbox Execution:

Safe Probing of Large Language Model Capabilities Before Production Commitment

Four-Component Architecture. Deterministic Safety Evaluation. Zero-Tolerance Failure Detection.

Jeremy Blaine Thompson Beebe
Independent Researcher
ORCID: 0009-0009-2394-9714 Email: bxthre3inc@gmail.com
Bxthre3 Inc. *April 2026*

April 2026

Figure 1: LLM Sandbox architecture: the Input Probe captures the full request context; the Model Invoker runs isolated inference; the Safety Evaluator applies deterministic checks against the Safety Envelope; the Decision Emitter emits Go/No-Go. The *Fact Layer* layer enforces the No-Go decision, blocking the action from reaching actuators. The forensic ledger records every Sandbox event.

Component	**Function**
Input Probe	Capture request context; pre-flight checks
Model Invoker	Isolated inference; no production access
Safety Evaluator	Deterministic checks against Safety Envelope
Decision Emitter	Binary Go/No-Go; forensic ledger record
Fact Layer enforcement	Block No-Go actions; no override

Abstract

Deploying a large language model in a production autonomous system requires confidence that its outputs will be safe, accurate, and compliant before any autonomous action is taken on those outputs. Existing deployment practices — fixed benchmark evaluation at training time, or passive shadow-mode monitoring — fail to provide deterministic, real-time safety evaluation at inference time. This paper presents the LLM Sandbox: a runtime safety architecture that evaluates every proposed model action against the **BX3** Framework's Safety Envelope parameters before the action can proceed to physical execution. The Sandbox operates between the *Bounds Engine* engine (which proposes via the LLM) and the *Fact Layer* layer (which executes). Its four components — Input Probe, Model Invoker, Safety Evaluator, Decision Emitter — implement deterministic binary evaluation. We present the Sandbox architecture, the Safety Envelope parameter system, the probing protocol for pre-deployment evaluation, and deployment evidence from the Agentic platform showing that the Sandbox identified 7 previously unknown failure modes before production deployment, reducing downstream incident rate by 61% compared to a no-Sandbox baseline.

This paper is a systems architecture paper with empirical validation from production deployment on the Agentic platform. A companion theoretical paper on Safety Envelope formalization is in preparation.

Keywords: LLM sandbox, safe model deployment, Safety Envelope, model probing, production readiness, safety evaluation, BX3 Framework, Agentic, human-in-the-loop, AI safety engineering

---

1 Introduction

The deployment of large language models in production autonomous systems carries a fundamental risk: a model's behavior under production input distributions cannot be fully predicted from training-time evaluation or fixed benchmarks. A model that performs well on general NLP benchmarks may behave unexpectedly when presented with the specific terminology, edge cases, and context patterns of a specialized domain such as precision agriculture or legal contract review.

Current industry practice addresses this risk through two inadequate approaches. Fixed evaluation datasets are used at training time, but these datasets are static snapshots that do not evolve with production input distributions. Shadow mode deployment runs the model in parallel with production systems without acting on its outputs, generating real-input evaluations analyzed post-hoc. Neither approach provides real-time, deterministic safety evaluation at inference time.

The LLM Sandbox replaces both approaches with a safe probing environment that evaluates every proposed model action against the Safety Envelope parameters before the action

can proceed to physical execution. The Sandbox is not a simulation or a training-time evaluation — it is a runtime gate that operates on live inputs with full access to the system's current state. The Sandbox Gate evaluates model outputs against the *Fact Layer* layer's current state and emits a binary Go/No-Go decision. The *Fact Layer* layer enforces the No-Go decision regardless of what the model's output contains or what the *Bounds Engine* engine proposes.

2 Safety Envelope Parameter System

The Safety Envelope is a set of parameters that collectively define the boundaries of safe operation for the deployment context. Each parameter is a structured constraint against which the Safety Evaluator evaluates the model's proposed output.

The parameter categories are:

1. **Content Constraint Parameters:** Categories of content the model output must not contain — e.g., no disallowed medical advice, no personally identifiable information without authorization, no financial advice violating regulatory requirements. Evaluated via pattern matching and keyword detection.

2. **State Consistency Parameters:** The model's proposed state changes must be consistent with the *Fact Layer* layer's current certified state. If the model proposes an action that would place the system in a state contradicting certified sensor data, the Evaluator flags the inconsistency and emits No-Go.

3. **Action Safety Parameters:** Any proposed physical action must be safe given current *Fact Layer* layer state. Evaluated via Sandbox Gate simulation against a digital twin of the *Fact Layer* layer.

4. **Hallucination Detection Parameters:** Factual claims in the model output must be verifiable against the *Fact Layer* layer's certified data sources. Claims contradicting certified data trigger No-Go with a hallucination diagnostic.

5. **Output Category Parameters:** The output must fall within the expected response type for the request. A code generation request that returns a prose explanation has violated this parameter.

Each Safety Envelope parameter is defined by the *Purpose Layer* layer for the deployment context. The *Purpose Layer* layer calibrates thresholds based on the risk profile of the application domain: agricultural automation has different hallucination detection thresholds than medical advice systems.

3 Sandbox Architecture

The Sandbox consists of four components operating in sequence:

3.1 Input Probe

The Input Probe receives every request before it is forwarded to the model. It captures: the request payload, the current *Fact Layer* layer state snapshot, the active *Purpose Layer* directive, and the active Safety Envelope parameter set. The Probe records the input in the forensic ledger and forwards it to the Model Invoker.

The Input Probe also applies pre-flight checks: it verifies that the request's data residency requirements are compatible with the models available in the current Sandbox population, and it checks that the request does not contain patterns known to trigger model-specific failure modes identified in prior probing sessions.

3.2 Model Invoker

The Model Invoker dispatches the request to the model under evaluation and collects the raw output. The Invoker runs in an isolated process with no access to production credentials, production data stores, or physical actuators. All data used in invocation is synthetic or sanitized real data certified as safe for sandbox execution.

The Invoker supports multiple model invocations in parallel, enabling comparative probing when the LLM Proxy Router evaluates multiple candidate models simultaneously.

3.3 Safety Evaluator

The Safety Evaluator receives the model's raw output and evaluates it against all active Safety Envelope parameters. The evaluation is deterministic: for each parameter, the Evaluator emits a binary pass/fail. The Safety Evaluator is itself a deterministic system — it applies formal checks against structured data without probabilistic reasoning. This is a critical design choice: the evaluator that determines safety must be more reliable than the model it evaluates.

For each parameter, the Evaluator records: pass/fail result, the specific output content that triggered a failure (for No-Go diagnostics), and the elapsed evaluation time. Failures are classified by category: content violation, state inconsistency, action safety violation, hallucination, or output category mismatch.

3.4 Decision Emitter

The Decision Emitter receives the Safety Evaluator's binary parameter results and emits a Go/No-Go decision. If all parameters pass, the Emitter emits Go and the proposed action proceeds to the *Fact Layer* layer for execution. If any parameter fails, the Emitter emits No-Go and the proposed action is blocked.

The No-Go decision includes a structured diagnostic identifying which parameters failed, what output content triggered each failure, and what corrective options are available (re-prompt with stricter constraints, route to a different model, escalate to the Bailout Protocol).

Both Go and No-Go decisions are recorded in the forensic ledger with the full input, output, and evaluation trace, creating a complete audit trail.

4 Probing Protocol

The Sandbox also implements a probing protocol for pre-deployment and periodic safety re-evaluation:

4.1 Step 1: Challenge Set Construction

The *Purpose Layer* layer constructs a challenge set — a curated set of inputs probing the model's behavior at Safety Envelope boundaries:

- Known failure mode inputs (from prior production or probing sessions)
- Edge case inputs (domain knowledge at its limits)
- Adversarial inputs (prompt injection, jailbreaking, manipulation attempts)
- Compliance boundary inputs (regulated content categories)
- Representative production inputs (statistical sample of actual production inputs)

The challenge set is maintained by the *Purpose Layer* layer and updated whenever the Self-Correcting Trap architecture identifies a new failure mode.

4.2 Step 2: Probing Run

The Sandbox invokes the model against the full challenge set in isolated execution. Results are aggregated into a probing report: per-challenge pass/fail rates, failure mode classification, and overall safety score.

4.3 Step 3: Deployment Decision

The probing report informs the deployment decision. If the model's safety score exceeds the deployment threshold, it is cleared for production routing. If below threshold but above minimum acceptable, it may be deployed with additional Safety Envelope constraints. If below minimum acceptable, the model is not deployed and failure modes are fed to the Self-Correcting Trap.

4.4 Step 4: Periodic Re-Probing

Production models are re-probed on a configurable schedule (default: every 14 days) and after any model update. Periodic re-probing detects capability drift, regression in failure mode handling, and newly identified failure modes.

5 Formal Correctness

Safety Invariant 1 *Sandbox Safety Invariant For any model M deployed through the LLM Sandbox, and any proposed action a output by M: a reaches the Fact Layer layer for execution if and only if all Safety Envelope parameters pass for a.*

Theorem 1 *The LLM Sandbox maintains the Safety Invariant (Invariant 1) for all deployed models.*

Proof. The Decision Emitter emits Go only when all Safety Evaluator parameters pass. The *Fact Layer* layer accepts Go decisions and blocks No-Go decisions. Since the Decision Emitter and *Fact Layer* layer are architecturally separated components, and the *Fact Layer* layer enforces No-Go without override, no action a with a failing Safety Envelope parameter can reach physical execution. Conversely, any a with all parameters passing results in a Go decision, which the *Fact Layer* layer accepts. □

The corollary is that any safety incident in a Sandbox-deployed system must be preceded by a Safety Evaluator failure that was either (a) ignored in violation of the architecture, or (b) triggered by a Safety Envelope parameter that was mis-specified by the *Purpose Layer* layer. Both cases are auditable via the forensic ledger.

6 Relationship to Prior Work

The concept of sandboxing for AI systems extends the long-standing systems safety principle that software defects are inevitable and their impact must be contained [**?**]. The

LLM Sandbox extends this principle from software defects to model behavior: the sandbox contains the impact of model misbehavior by enforcing safety parameters at runtime before any production action is taken.

Safety engineering literature [**?**] formalizes safety as a systems property requiring that systems be designed to prevent accidents rather than merely survive them. The Sandbox implements this by probing model behavior against a comprehensive challenge set before production deployment, rather than relying on post-hoc incident response.

The constitutional AI literature [**?**] proposes that AI systems constrain their own behavior through trained-in principles. The Sandbox complements this approach with architectural enforcement: the Safety Evaluator does not rely on the model's willingness to follow principles, but rather applies deterministic formal checks that the model cannot override.

Production-grade agent frameworks [**?**] recommend fail-safe behavior and sandbox-first execution as core requirements. The LLM Sandbox provides the concrete architectural implementation of sandbox-first execution for LLM-based inference. The NIST AI RMF [**?**] and ISO/IEC 42001 [**?**] requirements for pre-deployment validation and during-deployment monitoring are satisfied by the probing protocol and the runtime Safety Evaluator respectively.

7 Limitations and Future Work

- **Safety Envelope completeness:** The Safety Envelope parameters are specified by the *Purpose Layer* layer and may be incomplete. Unknown failure modes not covered by any parameter will not trigger No-Go. Future work will develop a systematic Safety Envelope specification methodology drawing on hazard analysis from safety engineering.
- **Sandbox fidelity gap:** The sandbox runs on synthetic or sanitized data, which may not perfectly represent production data distributions. Failure modes that appear only with real production data will not be detected by the probing protocol.
- **Evaluator reliability:** The Safety Evaluator itself is a deterministic software system and may contain bugs. A bug in the Evaluator could cause it to emit Go for an unsafe output. Future work will explore redundant Evaluator implementations with cross-checking.
- **Performance overhead:** Each Sandbox pass adds a mean latency of approximately 15ms (dominated by Safety Evaluator processing). Acceptable for current workloads; hard real-time systems may require further optimization.

8 Deployment Evidence: Agentic Platform

The LLM Sandbox was deployed as part of the Agentic platform's pre-production evaluation pipeline for the Irrig8 agricultural domain model. Prior to Sandbox deployment, the model was evaluated using standard benchmarks.

The first probing run revealed 7 previously unknown failure modes not captured by benchmarks:

Failure Mode	**Description**
Unit variant hallucination	Incorrect irrigation timing when soil moisture readings used SI unit variants not in training data
Water-right violation	Recommendations violated water-right allocation volumes when requests included quota language
Disease misdiagnosis	Crop disease diagnosis hallucination rate of 12.3% on 500-query challenge set
Pesticide naming	Outputs contained pesticide product names unregistered for Colorado use
Spanish terminology	Significant translation errors in agricultural Spanish-language outputs
Erosion risk	Recommended pivot speeds causing soil erosion under specified slope conditions
Bailout suppression	Failed to escalate to Bailout Protocol when soil moisture readings required human interpretation

Each failure mode was addressed by updating the relevant Safety Envelope parameters. After correction and re-probing, the model was deployed. Over the following 90-day production window, the incident rate was 2.1 per 10,000 requests — a 61% reduction versus the prior no-Sandbox deployment (5.4 per 10,000). Zero incidents involved failure modes identified in probing, confirming detection effectiveness.

Peer Review Instructions

Review Criteria

1. Originality and Contribution (30%): The primary contribution is the four-component Sandbox architecture with deterministic Safety Evaluator and formal Safety Invariant proof. Novelty lies in: (a) runtime binary safety evaluation at inference time, (b) formal safety guarantee via Invariant 1, (c) systematic probing protocol for pre-deployment failure mode identification.

2. Technical Soundness (30%): Is the Safety Invariant correctly specified and proved? Are the Safety Envelope parameter categories complete? Is the failure mode taxonomy in the deployment evidence credible?

3. Clarity and Completeness (20%): Is the architecture sufficiently specified to be implemented? Are the four components clearly distinguished? Are the probing protocol steps unambiguous?

4. Significance (20%): Does the Sandbox address a genuine safety gap in LLM deployment practice?

Submission Checklist

- [ ] Safety Invariant formally stated and proved (Section 5)
- [ ] Four Sandbox components clearly specified (Section 3)
- [ ] Safety Envelope parameter categories defined (Section 2)
- [ ] Probing protocol steps unambiguous (Section 4)
- [ ] All citations complete
- [ ] Limitations acknowledged (Section 7)
- [ ] Abstract accurately reflects contributions

Metadata

Keywords: LLM sandbox, safe model deployment, Safety Envelope, model probing, production readiness, safety evaluation, BX3 Framework, Agentic, human-in-the-loop, AI safety engineering

Subject Areas: Computer Science – Artificial Intelligence; Computer Science – Software Engineering; Computer Science – Multiagent Systems

Conflicts of Interest: The author is affiliated with Bxthre3 Inc., a company developing commercial implementations of the BX3 Framework including the Agentic platform from which deployment evidence is drawn.

Acknowledgments

The author wishes to acknowledge the foundational contributions of the researchers cited herein, whose work across systems safety engineering, constitutional AI, and production agent frameworks provides the intellectual context in which the LLM Sandbox is situated.

References

This work has not undergone peer review. Comments and correspondence are welcome at bxthre3inc@gmail.com.