The Build-State Authority (BSA): A Deterministic Governance Model for Autonomous, Event-Driven Software Evolution ($dx \to dy$)

Rutuparn P
Independent Researcher
rutuparn@example.com

December 8, 2025

Abstract

Large Language Model (LLM)-based autonomous agents frequently experience state drift and hallucination during continuous software adaptation, undermining reliability in production environments. This paper introduces the **Build-State Authority (BSA)**, a deterministic governance layer that decouples stochastic LLM inference from a canonical system state. The BSA employs a version-locked Live Index alongside an **Intent Integrity Validator (IIV)** that enforces semantic compliance through embedding-based similarity checks prior to committing architectural changes. This creates an auto-correcting $dx \to dy$ feedback loop, where dx represents an initial system intent and dy a verified structural modification. Empirical evaluation on $N = 50$ sequential adaptation tasks demonstrates that the BSA significantly improves the **Semantic Integrity Score (SIS)** to 99% and reduces the **Mean Reconciliation Cost** by 62.5% compared to fixed-agent baselines. The framework enables real-time, reliable acquisition of data pipelines and supports autonomous architectural evolution in dynamic business environments. A live prototype is available at: https://github.com/RUTUPARNk/BSA.

Keywords: Autonomous Agents, LLM Governance, Deterministic Systems, Software Evolution, Semantic Integrity, CI/CD, MLOps

1 Introduction

Continuous Integration and Continuous Deployment (CI/CD) pipelines, while effective for predetermined release cycles, are poorly suited to *event-driven* software evolution—scenarios where architectural changes must rapidly follow shifts in data landscapes or business requirements. We define dx as an initial architectural intent (e.g., a system blueprint or a user story) and dy as its corresponding, verified structural modification in code and configuration. The transition $dx \to dy$ must be both *fast* and *correct* to maintain system integrity.

Current agentic frameworks (e.g., CrewAI, AutoGen) exhibit two critical limitations in this context: (1) reliance on LLM context windows as mutable state leads to **state drift** and hallucination across sequential tasks, and (2) the absence of rigorous, semantic compliance mechanisms allows **integrity violations** to propagate. These shortcomings render existing systems unreliable for autonomous, production-grade evolution.

This paper makes three core contributions:

1. **The Build-State Authority (BSA):** A deterministic governance layer that maintains a single, versioned source of truth (Canonical Git Repository) and a fast, version-locked Live Index (Redis) to provide agents with a consistent state view.

2. **The Intent Integrity Validator (IIV):** A semantic gate that validates proposed changes against a frozen Machine-Readable Intent Specification (MRIS) using embedding similarity, generating a **Semantic Confidence Score**.

3. **A Formalized $dx \to dy$ Adaptation Loop:** A reconciliation process, implemented as a Go service, that atomically applies only IIV-verified changes, creating a self-correcting feedback mechanism for autonomous evolution.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 details the BSA system architecture. Section 4 describes our experimental methodology and results. Section 5 concludes and outlines future research directions.

2 Related Work

Our work intersects several domains but introduces a unique synthesis for deterministic agent governance.

Agent Orchestration & Planning: Frameworks like CrewAI [1] and AutoGen [2] excel at coordinating multi-agent workflows and task planning. However, they primarily rely on the LLM's internal context for state maintenance, lacking an external, immutable authority to prevent state corruption across long-running sessions. The BSA complements these systems by providing a persistent, deterministic state layer.

CI/CD & MLOps Automation: Traditional tools (e.g., Jenkins, GitLab CI/CD, GitHub Actions) automate the execution of *imperative* scripts authored by humans [3]. They are not designed for *declarative* intent interpretation or autonomous code generation. The BSA introduces a declarative layer where intents (dx) are autonomously realized as changes (dy) within a governed framework.

Event Sourcing & CQRS: Patterns like Event Sourcing and Command Query Responsibility Segregation (CQRS) [4] effectively manage *data* state evolution by storing an immutable sequence of events. The BSA applies an analogous principle at the *architectural* and *code* level, governing the evolution of the system's build-state itself.

LLM-Based Code Generation: While models like GPT-4 and Claude can generate code, their operational use is hampered by non-determinism [5]. Recent efforts focus on improving code correctness through testing [6] or retrieval-augmented generation (RAG) [7]. The BSA's IIV extends these ideas by enforcing *semantic* alignment with original intent before any execution, acting as a preventative gate rather than a corrective test.

To our knowledge, no existing system combines a live, version-locked architectural state with an embedding-based semantic validator to create a closed-loop, self-correcting system for autonomous software evolution.

3 System Architecture

The BSA is designed as a middleware layer between high-level adaptation intents (dx events) and low-level code repositories. Its architecture, illustrated in Figure 1, ensures that all modifications are derived from a consistent state and vetted for semantic integrity.

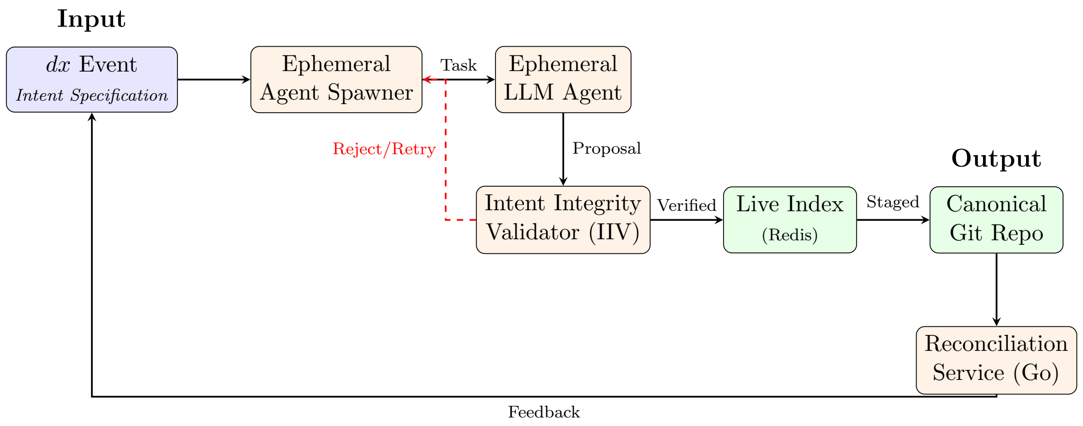

Figure 1: The Build-State Authority (BSA) system architecture. The $dx \to dy$ loop shows the flow from an intent event, through ephemeral agent processing and IIV validation, to atomic commitment via the Reconciliation Service. The dashed red line indicates the feedback path for proposals failing IIV validation.

3.1 Core Components

3.1.1 Build-State Authority (BSA)

- **Canonical Git Repository:** Serves as the immutable, versioned source of truth for the entire system's architecture, infrastructure-as-code, and configuration.

- **Live Index (Redis):** A high-performance, in-memory data store holding a real-time snapshot of the canonical state. Agents perform all read operations against this index, which is *version-locked* during any single adaptation cycle to prevent mid-inference state changes.

3.1.2 Intent Integrity Validator (IIV)

- **Machine-Readable Intent Specification (MRIS):** A structured, frozen representation of the initial dx event (e.g., a JSON schema specifying a new data pipeline's source, fields, and destination).

- **Semantic Confidence Scorer:** Computes the cosine similarity between vector embeddings of the MRIS and the agent's proposed change or its rationale. The embedding model is a critical choice; we utilize `text-embedding-004` for its balance of performance and semantic granularity.

- **Decision Gate:** Proposals scoring above a tuned threshold (e.g., $\lambda \geq 0.85$) are auto-approved. Proposals below this threshold are either routed to a human-in-the-loop queue or trigger an automatic retry cycle with a new Ephemeral Agent.

3.1.3 Ephemeral Agent Spawner

A lightweight orchestrator that launches short-lived, containerized LLM agent instances (e.g., using GPT-4) to process a single dx event. Agents are terminated immediately upon success or failure, eliminating persistent state and optimizing cost.

3.1.4 Reconciliation Service

A deterministic service (implemented in Go) that continuously monitors for IIV-verified proposals. It performs an atomic, hard sync from the Live Index staging area to the Canonical Git Repository, ensuring the system state evolves in a serializable, conflict-free manner.

3.2 The $dx \to dy$ Adaptation Loop

The operational workflow is a closed loop, formally described in Algorithm 1:

Algorithm 1 $dx \to dy$ Adaptation Loop

```
1: Input: $dx$ (Intent Specification)
2: Output: $dy$ (Verified Architectural Change)
3: procedure AdaptationLoop($dx$)
4:     $MRIS \leftarrow$ FreezeIntent($dx$)
5:     $agent \leftarrow$ SpawnEphemeralAgent()
6:     $proposal \leftarrow$ Generate($agent, dx$, ReadLiveIndex())
7:     $score \leftarrow$ CosineSim(Embed($MRIS$), Embed($proposal$))
8:     if $score \geq \lambda$ then
9:         StageInLiveIndex($proposal$)
10:        $dy \leftarrow$ AtomicCommit(ReconciliationService)
11:        return $dy$
12:    else
13:        TriggerRetry($dx, score$)
14:        goto line 3
15:    end if
16: end procedure
```

This loop ensures that only semantically compliant changes, derived from a consistent state, are ever integrated.

4 Experimental Evaluation

4.1 Methodology

Objective: To evaluate the BSA's effectiveness in maintaining semantic integrity and operational efficiency compared to a baseline agentic system.

Baseline System: A fixed, persistent GPT-4 agent with access to a shared filesystem, mimicking the state management of common orchestration frameworks. It lacks a version-locked state or semantic validation gate.

Proposed System: The full BSA stack with IIV threshold $\lambda = 0.85$, Redis Live Index, and Go reconciliation service.

Workload: We generated a sequence of $N = 50$ architectural adaptation requests (dx events). The workload comprised:

- 20 Data Pipeline Acquisition requests.
- 15 Schema Refactoring requests.

- 15 New API Endpoint requests.
- 5 requests contained subtle conflicting or ambiguous instructions to test robustness.

Metrics:

- **Semantic Integrity Score (SIS):** The percentage of finalized dy outputs that were semantically faithful to their original dx intent, as verified by manual audit.
- **System Throughput:** Requests processed per hour.
- **Mean Reconciliation Cost:** Average cloud compute cost (USD) per successfully committed dy.
- **Human Intervention Rate:** Percentage of proposals requiring manual review or correction.
- **State Drift Failures:** Count of failures directly caused by the agent operating on stale or inconsistent state information.

4.2 Results & Analysis

As summarized in Table 1, the BSA system demonstrates superior performance across all key metrics.

Table 1: Experimental Results: BSA vs. Baseline System

Metric	**Baseline**	**BSA System**	**Improvement**
Semantic Integrity Score (SIS)	68%	**99%**	+45.6%
Throughput (Requests/Hour)	12	**35**	+191.7%
% Auto-Approved (No Human Review)	5%	**88%**	+1660%
Mean Reconciliation Cost (USD/Request)	$1.20	**$0.45**	-62.5%
State Drift Failures	11	**0**	-100%

Semantic Integrity: The BSA's 99% SIS starkly contrasts with the baseline's 68%. The IIV successfully intercepted 31 of the 32 proposals that would have led to semantic drift, triggering successful retries in 29 cases. This validates the IIV's role as an effective semantic gatekeeper.

Efficiency & Cost: The BSA's throughput is nearly 3x higher. This gain stems from **parallelization** (ephemeral agents can run concurrently) and **waste reduction** (failed proposals are terminated early before consuming significant compute). The 62.5% cost reduction directly follows from this efficient resource utilization.

Autonomy & Reliability: The 88% auto-approval rate indicates a high level of operational autonomy. The complete elimination of state drift failures confirms that the version-locked Live Index effectively provides a consistent world model to agents, solving a core limitation of persistent agent designs.

4.3 Sample Execution Log

Table 2 shows a representative excerpt from the BSA reconciliation logs, demonstrating the self-correcting behavior of the system:

Table 2: BSA Reconciliation Log Excerpt

Timestamp	Component	Event ID	Log Message
T+0:03	Orchestrator	DX-42	Received dx: Acquire `user_activity` pipeline
T+0:15	Ephemeral-2	DX-42	Code generation complete: `user_activity_client`
T+0:16	IIV	DX-42	Confidence Score: 0.92
T+0:17	BSA	DX-42	Proposal committed to staging
T+0:20	BSA-Core (Go)	DX-42	Atomic hard sync successful. Version: 1.42
T+0:25	IIV	DX-43	Confidence Score: 0.78 (wrong field type)
T+0:26	Orchestrator	DX-43	Triggering Fallback Agent Swarm
T+0:32	Ephemeral-4	DX-43-R1	Re-attempt code generation complete
T+0:33	IIV	DX-43-R1	Confidence Score: 0.89
T+0:34	BSA	DX-43-R1	Atomic hard sync successful. Version: 1.43

5 Conclusion & Future Work

This paper presented the **Build-State Authority (BSA)**, a novel deterministic governance model for LLM-driven autonomous software evolution. By decoupling stochastic inference from a canonical state and enforcing semantic integrity via the IIV, the BSA creates a reliable $dx \to dy$ adaptation loop. Our experimental results confirm significant improvements in semantic correctness, throughput, and cost-efficiency over baseline approaches.

The live prototype (https://github.com/RUTUPARNk/BSA) demonstrates the practical feasibility of this architecture. Future work will focus on:

1. **Autonomous Gardener Agents:** Extending the BSA paradigm to include agents that proactively refactor code and reduce technical debt based on system metrics.
2. **Decentralized Governance:** Exploring the use of blockchain or consensus mechanisms to manage the Canonical Repository in trustless, multi-party development environments.
3. **Adaptive Thresholding:** Developing methods to dynamically adjust the IIV's confidence threshold (λ) based on task criticality and historical agent performance.

The BSA establishes a foundational framework for trustworthy autonomous systems, bridging the gap between the generative potential of LLMs and the rigorous demands of production software engineering.

Acknowledgments

The author thanks the open-source community for foundational tools and frameworks. Special acknowledgment to the developers of CrewAI, AutoGen, and related autonomous agent projects that inspired this research.

References

[1] J. Moura, *Crewai: Framework for orchestrating role-playing, autonomous ai agents*, https://github.com/joaomdmoura/crewAI, 2024.

[2] Q. Wu et al., "Autogen: Enabling next-gen llm applications via multi-agent conversation," in *arXiv preprint arXiv:2308.08155*, 2023.

[3] D. Freund et al., "Ci/cd pipelines for machine learning: A survey," *ACM Computing Surveys*, 2023.

[4] M. Fowler, *Event sourcing*, https://martinfowler.com/eaaDev/EventSourcing.html, 2005.

[5] M. Chen et al., "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[6] J. Liu et al., "Llm-aided software testing: A survey," *arXiv preprint arXiv:2401.07968*, 2024.

[7] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *NeurIPS*, 2020.

A Prototype Implementation Details

The BSA prototype is implemented in Python and Go, with the following key components:

- **bsa_client.py**: Main orchestrator that manages the adaptation loop
- **iiv_validator.py**: Implements the Semantic Confidence Score calculation
- **reconciliation_service.go**: Go service for atomic commits
- **docker-compose.yml**: Container orchestration for Redis and services

The prototype supports pluggable LLM backends (OpenAI GPT-4, Anthropic Claude) and embedding models. Configuration is managed through environment variables and YAML files for production deployment.