CORE: A Constitution-Oriented Runtime Environment for Governed Autonomy

Dariusz Newecki
https://github.com/DariuszNewecki/CORE
Belgium
(Unaffiliated)

February 2026

Abstract

Modern software systems increasingly include autonomous subsystems that can mutate artifacts, execute multi-step workflows, and act across heterogeneous environments. Governance mechanisms for such systems frequently evolve incrementally and implicitly, leading to scattered enforcement logic, fragile rule interactions, and unclear authority boundaries.

This paper describes CORE (Constitution-Oriented Runtime Environment), an open-source architecture for "governed autonomy": a system where normative constraints are first-class, machine-evaluable artifacts, and where enforcement is designed to be reproducible, phase-bounded, and resistant to accidental authority expansion. CORE separates normative law (Mind) from execution (Body) and decision-making (Will), and introduces an explicit Infrastructure category for purely coordinative components that must cross-cut layers without becoming an exemption space.

We present the CORE governance model, its dynamic rule execution approach, and an implementation snapshot that integrates (i) a policy repository (".intent"), (ii) a PostgreSQL-backed knowledge graph, and (iii) a Qdrant vector store for semantic indexing. We also discuss observed failure modes that motivated constitutional "resets" (e.g., bootstrap wiring hardening, session detachment, and infrastructure authority boundaries), and outline an evaluation strategy and limitations.

1 Introduction

Autonomous development agents, self-healing workflows, and policy-driven runtime guards create a new class of software systems: systems that can act and mutate under programmatic control. In such systems, a governance model is not optional infrastructure; it is a correctness and safety boundary.

In practice, governance tends to be implemented as a patchwork of conventions, linters, CI gates, ad hoc scripts, and human processes. Over time, authority becomes implicit: components gain power because they are "needed to make things work," not because their authority is defined. This failure mode is amplified in agentic systems, where the ability to change code and policies becomes part of the runtime.

CORE emerged from repeated attempts to govern these systems, and from accumulated failure cases where implicit authority and unclear boundaries produced non-auditable, non-reproducible behavior [1].

2 Problem Statement

We target systems with the following properties:

- A substantial portion of behavior is controlled by rules (policies, constraints, safety conditions).
- The system can execute multi-step workflows and may modify artifacts (code, documents, configuration).
- Multiple subsystems are involved (e.g., code introspection, knowledge graphs, vector stores, LLM providers).
- Governance must remain inspectable and reproducible over time.

The central problem is not merely enforcing rules, but enforcing *authority boundaries*:

- Where does law live?
- When is a rule evaluated?
- What evidence is allowed for evaluation?
- Which components are permitted to coordinate across layers?

3 Design Overview

CORE is organized into three primary layers and one explicitly bounded cross-cutting category:

- **Mind**: normative law and governance artifacts (the ".intent" repository).
- **Body**: deterministic execution components (services, sync pipelines, IO).
- **Will**: decision-making and orchestration components (agents, strategy).
- **Infrastructure**: coordinative components with mechanical authority only.

A key principle is that Infrastructure is defined and bounded explicitly, rather than treated as an implied exemption. Infrastructure may coordinate resources (sessions, caches, paths), but may not decide what is correct, important, or semantically meaningful.

3.1 Core primitives and governance semantics

CORE treats governance artifacts as machine-evaluable rules, rather than as human-interpretive guidance. Rules are designed to be:

- **Phase-bounded**: each rule belongs to an explicit evaluation phase.
- **Evidence-limited**: evaluation depends only on evidence permitted within that phase.
- **Reproducible**: evaluation must be reproducible, or outcomes become indeterminate.
- **Non-interpretive**: rules are checked, not debated.

4 Governance Architecture

4.1 The ".intent" repository

CORE stores constitutional and policy documents in a dedicated directory tree (".intent"). The repository is indexed and validated at boot. Policies

expose executable rules which are extracted into a normalized representation used by the runtime evaluator.

4.2 Dynamic rule execution

CORE supports "dynamic rule execution": executable rules declared in policy documents are executed by registered engines (e.g., AST-based gates, glob/path gates, knowledge gates). This eliminates the need for each rule to be expressed as a custom Python class, and makes the rule set inspectable.

In a representative implementation snapshot, the audit pipeline executes a set of executable rules, tracking executed rule IDs, skipping known stubs, and reporting coverage metrics. This makes constitutional coverage a measurable system property rather than a promise.

4.3 Conflict semantics

CORE treats conflicts between rules of equal authority as governance defects. Conflicts are not resolved by precedence or ordering. When detected, the system must block progression for affected scopes until law is corrected.

5 Infrastructure: Mechanical Coordination Without Authority

In practice, certain components must cross-cut layers to coordinate resources (database sessions, repo paths, secrets). CORE formalizes this as "Infrastructure" and restricts it.

Example infrastructure components include:

- A minimal bootstrap registry that holds a repo path and session factory.
- A configuration service that reads runtime settings from a database and delegates secret decryption.
- A session manager that provides per-event-loop engine/session factories.

The critical constraint is that these components provide coordination only. They must not decide which configurations are correct, interpret semantic meaning, or implement domain logic.

6 Implementation Snapshot

This paper reports an implementation snapshot that includes:

- A developer workflow that runs ID assignment, docstring completion, formatting, linting, knowledge sync to PostgreSQL, and vectorization.
- A knowledge graph (PostgreSQL) used as a structured system-of-record for discovered symbols and governance artifacts.
- A vector store (Qdrant) used for semantic retrieval over capabilities and constitutional/policy documents.
- A cognitive facade (Will-facing service) that wires configuration access, mind state access, and provider/client creation.

6.1 Bootstrap hardening and session detachment

Two recurring operational failure modes in async-heavy systems are (i) incorrect boot order leading to missing "root path" coordinates, and (ii) leaking database sessions into long-lived services.

CORE addresses these via (a) hardened bootstrap wiring (explicitly priming session factories and synchronizing repo coordinates), and (b) explicit "detach" operations after initialization to release DB-session references back to the pool.

7 Evaluation Plan

CORE is primarily an architectural and governance proposal, but it admits concrete evaluation dimensions:

- **Governance coverage**: fraction of declared executable rules that are deterministically evaluated per phase.
- **Reproducibility**: ability to reproduce audit outcomes given the same repo state and declared law.

- **Authority containment**: frequency and impact of discovered "authority leaks" (components acting outside declared limits).
- **Operational stability**: incidence of bootstrap and session-lifecycle failures under typical dev workflows.

Future work will include a structured benchmark suite that exercises common failure modes (bootstrap ordering, rule conflicts, evidence gaps, and autonomous modification attempts).

8 Limitations

CORE does not claim to solve:

- human governance processes (approval boards, organizational decision-making),
- semantic interpretation of policy intent (CORE enforces declared law),
- alignment or safety of LLM outputs by itself (CORE constrains where LLMs may act).

Additionally, dynamic rule execution is only as strong as the determinism and phase-correctness of its engines. Where evaluation cannot be made deterministic, the correct outcome is indeterminate and must not be treated as permission.

9 Reproducibility and Artifacts

The reference implementation is available as open-source code. For arXiv distribution, this submission includes the LaTeX source. The code repository and its documentation are linked from the author URL.

Acknowledgements

This project reflects iterative design work over multiple prototypes and refactors. The author thanks open-source maintainers of the underlying toolchain (Python, SQLAlchemy, Qdrant, and the broader ecosystem) for enabling practical experimentation.

References

[1] Dariusz Newecki. Core: Constitution-oriented runtime environment. https://github.com/DariuszNewecki/CORE, 2026. Accessed: 2026-02.