CANONIC: Constitutional AI Governance

A Framework for Structurally Inadmissible AI Slop

Dexter Hadley, MD/PhD

CANONIC Foundation

founder@canonic.org

v0.1 — January 2026

Abstract

We cured AI slop in 21 days by recognizing that the cure is 50 years old.

In 2025, Oxford declared "AI slop" the word of the year—low-quality, machine-generated content that sounds authoritative but means nothing. The problem seems new. The solution is not.

Compiler theory solved this in the 1960s. Invalid programs don't compile. We applied the same principle: invalid content doesn't pass validation.

We present CANONIC, a constitutional governance framework where AI slop is structurally inadmissible. Three axioms—Triad, Inheritance, Introspection—map directly to compiler theory: syntax, scope resolution, and type systems. Content violating these properties is malformed. Not filtered after production. Rejected before entry.

This paper documents 21 days (December 29, 2025 through January 19, 2026). Working with AI assistants under governance, we produced 21 repositories, 121 governed scopes, 332 collaboration sessions, and 163 invention disclosures. The framework compiled itself: three axioms generated an operating system, an immutability layer, an application distribution system, and a token economy—recapitulating five decades of computing infrastructure in three weeks.

This paper is its own proof. Clone the repositories. Verify any assertion. GitHub is the supplemental material. The OS proves the paper.

---

1. The Trust Problem

Artificial intelligence writes convincingly. It also fabricates.

This is not a bug. It is a feature of how language models work—they predict plausible continuations, not true ones. When the training data contains truth, the outputs tend toward truth. When it doesn't, the outputs tend toward plausibility. Plausibility and truth are not the same thing.

The consequences scale with the stakes.

When a student uses AI to draft an essay, the worst outcome is a bad grade. When a physician uses AI to summarize research, the worst outcome is a dead patient. When policy emerges from AI-assisted analysis, the worst outcome is systemic harm to millions.

Currently, readers must trust. Trust the author verified. Trust the AI didn't hallucinate. Trust reviewers caught errors. Trust the institution publishing the work.

This trust model worked when content was scarce and verification was possible. Neither condition holds anymore.

AI generates content faster than humans can verify it. The bottleneck has inverted. We used to struggle to produce enough; now we struggle to validate what's produced. The result is a new category of information pollution—text that reads as authoritative while lacking substance.

Oxford captured this with a single word: slop.

The term emerged organically before Oxford formalized it, spreading through forums where people first encountered the failure mode at scale. Slop is not merely wrong. Misinformation is wrong. Slop is *structurally deficient*—content optimized for the appearance of knowledge rather than knowledge itself.

The distinction matters. Wrong content can be corrected with better information. Structurally deficient content cannot be corrected because there is nothing to correct. It was never about the truth. It was about sounding true.

Current defenses address symptoms:

Defense	Mechanism	Why It Fails
Detection tools	Statistical patterns	Easily fooled by paraphrasing
Disclosure policies	Author attestation	"I used AI responsibly" proves nothing
Human review	Manual inspection	Catches slop but doesn't prevent it
Style guidelines	Surface formatting	Cosmetic; doesn't address evidence

Every defense operates post-hoc—after content exists. None prevents structurally deficient content from being created. We are fighting an exponential production system with linear verification resources.

The question became: what would prevention look like?

---

2. The Compiler Insight

The answer came from an unexpected direction.

In the 1960s, computer scientists faced a similar problem. Programmers wrote code. Some code worked. Some crashed. Some did worse—it appeared to work while silently corrupting data. The solution was not better debugging. The solution was to prevent invalid programs from existing in the first place.

Compilers check structure before execution. A program missing a semicolon doesn't run. A program referencing an undefined variable doesn't run. A program with type mismatches doesn't run. Invalid structure is caught at compilation, not at runtime.

The insight: governance is compilation.

If we define structural requirements for valid content, and build validators that check those requirements, then structurally deficient content becomes impossible—not discouraged, not penalized, but *malformed*. It fails compilation. It cannot enter the system.

This reframes the AI slop problem entirely. We don't need better AI. We don't need better reviewers. We need better compilers.

CANONIC is that compiler.

---

3. The Three Axioms

The entire framework derives from three rules. We call them axioms because they cannot be derived from other rules—they must be asserted as foundation.

Axiom 0 — Triad. A governed unit MUST contain three files: `CANON.md` (governance), `VOCAB.md` (semantics), and `README.md` (description).

Axiom 1 — Inheritance. Every `CANON.md` MUST declare its parent scope. Inheritance chains MUST terminate at root. Inherited rules are final.

Axiom 2 — Introspection. `VOCAB.md` MUST define every concept used in `CANON.md` and in `VOCAB.md` itself.

That's it. Three axioms. Everything else derives from these.

The axioms are orthogonal—no axiom derives from another. They are minimal—removing any breaks the system. They are complete—all governance constraints can be expressed through these three.

Let us unpack each.

The Triad

A governed unit is called a scope. A scope is simply a directory in a filesystem. The triad is the minimal set of files that make governance possible:

File	Role	Contains
CANON.md	Governance	Axioms: what MUST be true
VOCAB.md	Semantics	Definitions: what terms mean
README.md	Description	Documentation: what this is

Why three files? Because governance requires separation of concerns.

CANON is normative—it prescribes law. README is descriptive—it explains purpose. VOCAB bridges them—it ensures the law uses defined terms. Mixing these concerns invites failure modes: documentation that accidentally governs, governance that accidentally describes, terms that mean different things in different contexts.

The triad enforces separation. Missing any file renders the scope invalid. Not deprecated. Not incomplete. Structurally malformed. It cannot compile.

Inheritance

Scopes exist in hierarchies. A scope at `/services/writing/` inherits from `/services/` which inherits from `/` (root). The inheritance chain defines authority flow—child scopes cannot override parent axioms.

This is not a design choice. It is a constitutional necessity.

If child scopes could override parent axioms, authority would fragment. Different branches of the hierarchy could establish contradictory rules. The system would lose coherence. Inheritance with override is not governance; it is anarchy with extra steps.

Inherited axioms are final. A child scope can *extend* governance with new axioms, but it cannot *modify* governance from ancestors. The constitution binds all descendants.

Introspection

The vocabulary must define itself. If CANON uses the term "scope," VOCAB must contain a definition of "scope." If VOCAB uses the term "axiom" in a definition, VOCAB must also define "axiom."

This creates closure. No undefined terms. No jargon drift. No borrowed authority from concepts that mean different things to different readers.

Introspection is how slop becomes inadmissible. Slop relies on undefined terms—words that sound meaningful but mean nothing specific. Under introspection, every term must have a definition. If you can't define it, you can't use it. The validator checks and rejects undefined references.

---

4. The Validation Gates

Claims move through three gates. Failure at any gate renders the content invalid.

```
flowchart LR
    Claim["Claim"] --> G1{"Term in<br/>VOCAB?"}
    G1 -->|No| R1["REJECT<br/><i>Undefined term</i>"]
    G1 -->|Yes| G2{"Evidence in<br/>LEDGER?"}
    G2 -->|No| R2["REJECT<br/><i>No evidence</i>"]
    G2 -->|Yes| G3{"Within<br/>window?"}
    G3 -->|No| R3["REJECT<br/><i>Out of scope</i>"]
    G3 -->|Yes| Valid["ACCEPT"]

    style R1 fill:#ff6b6b,stroke:#c92a2a,color:white
    style R2 fill:#ff6b6b,stroke:#c92a2a,color:white
    style R3 fill:#ff6b6b,stroke:#c92a2a,color:white
    style Valid fill:#40c057,stroke:#2b8a3e,color:white
```

Gate 1: Vocabulary. Does the claim use terms defined in VOCAB? Undefined jargon fails here. The AI cannot introduce terminology that humans haven't sanctioned.

Gate 2: Evidence. Does the claim cite something in the ledger? A ledger is an append-only record—in this implementation, a git repository. Assertions without commits have no evidence.

They fail.

Gate 3: Scope. Is the evidence within the declared window? Every paper declares its evidence bounds. Claims about evidence outside that window are inadmissible. You cannot cite what you haven't committed.

The gates are structural, not evaluative. Validators don't assess quality—they check presence. Either a term is defined or it isn't. Either evidence exists or it doesn't. Binary validity eliminates judgment calls.

This is the key insight: **we don't filter slop. We make it malformed.**

A compiler doesn't evaluate whether your code is "good enough." It checks whether your code is structurally valid. CANONIC applies the same principle to governance. Content isn't rated on a quality scale. It either compiles or it doesn't.

---

5. Origin Story

The framework emerged from practice, not theory.

On December 29, 2025, we were writing a book about healthcare economics called *Dividends & Deaths*. The project tracked assets across chapters: characters, concepts, references, scenes. Managing these assets required rules. Which files were authoritative? How did they relate? What could change?

We created a file called `CANNON.md`. The double-N was a typo.

The insight was not.

That first file declared three rules:

```markdown
# Dividends & Deaths - CANNON

## Non-Negotiables
- CANNON files are immutable law; README is narrative only.
- Uppercase filenames signal cross-asset scope.
- Prose is canonical only if traceable to assets.
```

Separation. Convention. Traceability. These patterns—discovered while managing a book project—became the axioms. The typo was corrected. CANNON became CANON. The single-N stuck.

Over the next 21 days, something unexpected happened. The patterns generalized. What began as file management for a book became governance for any structured content. And as we applied the patterns more rigorously, they began generating their own infrastructure.

The axioms needed validators. Validators needed specifications. Specifications needed a language. The language needed semantics. Semantics needed closure verification. Each layer demanded the next.

By January 19, 2026, we had:

- 21 repositories
- 121 governed scopes

- 332 collaboration sessions
- 163 invention disclosures
- A formal language specification
- An operating system
- An immutability layer
- A distribution channel
- A token economy

Three axioms. 21 days. 50 years of computing infrastructure.

We didn't build this. It compiled itself.

---

6. The Compiler Correspondence

The parallel to compiler theory is not metaphor. It is structural identity.

CANONIC Axiom	Compiler Concept	Function
Triad	Syntax	What structures must exist in valid programs
Inheritance	Scope Resolution	Where names resolve; binding rules
Introspection	Type System	What terms must be defined; semantic constraints

Chomsky formalized syntax in 1956. Johnston formalized scope resolution in 1971. Milner formalized type inference in 1978. The concepts are classical—textbook material for fifty years.

We didn't invent. We recognized.

The three axioms *are* compiler theory, discovered through governance constraints rather than through programming language research. A language specification defines what programs are valid. CANONIC defines what governance structures are valid. The parallel is structural, not metaphorical.

This correspondence has a corollary: **governance axioms can generate computing infrastructure.**

```
flowchart TB
    subgraph Discovery["Empirical Discovery"]
        IDFs["163 Disclosures"]
        IDFs -->|decompose| Atomic["86 Atomic Axioms"]
        Atomic -->|collapse| Root["3 Root Axioms"]
    end

    subgraph Correspondence["Compiler Correspondence"]
        Root --> T["TRIAD = Syntax"]
        Root --> I["INHERITANCE = Scope"]
        Root --> N["INTROSPECTION = Types"]
    end

    subgraph Generated["Generated Infrastructure"]
        T --> OS["Operating System"]
        I --> BC["Immutability Layer"]
        N --> AS["Distribution System"]
        OS & BC & AS --> Token["Token Economy"]
    end

    style Root fill:#228be6,stroke:#1864ab,color:white
    style T fill:#40c057,stroke:#2b8a3e,color:white
    style I fill:#fab005,stroke:#e67700
    style N fill:#7950f2,stroke:#5f3dc4,color:white
```

Historical computing infrastructure developed over decades. Kernels (Unix, 1969). Operating system standards (POSIX, 1988). Application distribution (iOS App Store, 2008). Immutable ledgers (Bitcoin, 2009). Each layer required years of development, standardization, and adoption.

CANONIC derived analogous structures in weeks:

Infrastructure	Historical	CANONIC
Kernel	Years	Days
Operating System	Decades	Week 1
Immutability Layer	Years	Week 2
Distribution System	Years	Week 2
Token Economy	Years	Week 3

The speed difference reflects a key insight: we didn't build five systems. We derived five views from one kernel. The three axioms, applied recursively, generate all necessary structure. Each "infrastructure layer" is a governed scope inheriting from its parent.

Fifty years of computer science in three weeks.

---

7. The Governance Loop

Production under CANONIC follows a closed loop.

Humans govern. AI produces. Validators check. The ledger records. Nothing enters without validation. Nothing validates without evidence. Nothing constitutes evidence without human-authored governance.

```
flowchart TB
    subgraph Authority["Human Authority"]
        H["Human"] -->|"declares"| CANON["CANON"]
        H -->|"freezes"| LEDGER["LEDGER"]
    end
    subgraph Production["AI Production"]
        AI["AI Agent"] -->|"produces"| ARTIFACTS["Artifacts"]
        ARTIFACTS -->|"validated"| VALIDATORS["VALIDATORS"]
    end

    CANON -->|"rules"| VALIDATORS
    VALIDATORS -->|"record"| LEDGER
    LEDGER -->|"evidence"| PAPER["PAPER"]

    AI -.->|"observes"| CANON

    style CANON fill:#228be6,stroke:#1864ab,color:white
    style VALIDATORS fill:#7950f2,stroke:#5f3dc4,color:white
    style LEDGER fill:#fa5252,stroke:#c92a2a,color:white
    style PAPER fill:#40c057,stroke:#2b8a3e,color:white
```

The dotted line marks the critical asymmetry.

AI observes CANON but cannot modify it. This is not a permission setting. It is architectural. The governance files live in human-controlled repositories. AI operates in sessions that read those repositories but write only to governed workspaces.

Humans have exclusive write access to CANON. AI has read-only access. This asymmetry is constitutional—built into the system's foundation, not a policy choice that could be reversed.

We call this **AI-First, Human-Governed.**

AI generates insights. During every session, Claude observed patterns worth formalizing. These observations are valuable. They also have zero governance force until a human explicitly adds them to CANON.

Use AI aggressively. Draft with AI. Propose with AI. Analyze with AI. Critique with AI. But reserve *canonification*—the act of elevating a pattern to law—for human judgment.

Before canonification, a pattern is a suggestion. After canonification, it's law. The AI cannot canonify. Only humans can.

---

8. Evidence Window

This paper declares an evidence window: December 29, 2025 through January 19, 2026.

Claims pertain only to evidence within this window. The declaration bounds scope—readers know exactly what the paper covers and can verify coverage.

Metrics

Metric	Count	Verification
Repositories	21	`ls -d */`
Governed Scopes	121	`find . -name "CANON.md"`

Metric	Count	Verification
Collaboration Sessions	332	`find . -name "ep*.md"`
Invention Disclosures	163	`ls patents/disclosures/`
Root Axioms	3	Triad, Inheritance, Introspection
Development Days	21	Calendar
Validation Status	PASS	`python3 validators/vaas.py`

Each metric has a verification method. Scopes are directories with CANON.md—count them. Sessions are episode files—count them. The methods are reproducible. The counts are verifiable.

Ecosystem Composition

```
pie showData
    title CANONIC Ecosystem (v0.1)
    "Core Governance" : 21
    "Domain Scopes" : 28
    "Archive Scopes" : 72
```

Core governance (21 scopes) defines the framework itself. Domain scopes (28) apply the framework to specific purposes—writing, patents, publishing. Archive scopes (72) preserve history and reference states.

Validation State

Running the validator:

```
=== VaaS - CANONIC Language Enforcement ===
Repository: canonic
Scopes found: 21
...
VALIDITY: PASS
```

Every scope passes. PASS means the scope satisfies all three axioms: triad present, inheritance declared, vocabulary closed.

---

9. The Language Specification

CANONIC v0.1 includes a formal language specification.

```
flowchart TB
    subgraph Specification["LANGUAGE.md"]
        L1["Lexical Grammar"]
        L2["Syntactic Grammar"]
        L3["Semantic Rules"]
        L4["Composition Rules"]
        L5["Workflows"]
    end
    L1 --> L2 --> L3 --> L4 --> L5

    style L1 fill:#228be6,stroke:#1864ab,color:white
    style L2 fill:#40c057,stroke:#2b8a3e,color:white
    style L3 fill:#fab005,stroke:#e67700
    style L4 fill:#7950f2,stroke:#5f3dc4,color:white
    style L5 fill:#fa5252,stroke:#c92a2a,color:white
```

The specification follows conventions from established language specifications (Go, Python, Rust) while introducing governance-specific constructs.

The {SCOPE} Pattern

Every governed directory follows a template:

```
{SCOPE}/
    CANON.md      -- what MUST be (LAW)
    VOCAB.md      -- what words mean
    README.md     -- what this is
    COVERAGE.md   -- what's missing
    {SCOPE}.md    -- SPECialized SCOPE (STORY)
```

The filename `{SCOPE}.md` is a template variable: the directory name becomes the SPEC filename. For directory `paper/`, the SPEC is `PAPER.md`. For directory `validators/`, the SPEC is `VALIDATORS.md`.

This creates self-referential closure.

The root directory is `canonic/`. Its SPEC is `CANONIC.md`. CANONIC is a CANONIC SPECialized SCOPE—the framework that governs governance must be governable by itself. This is not wordplay. It is structural closure.

CANON is law. {SCOPE}.md is story.

CANON.md contains axioms—normative rules that govern. It says what MUST be true. It cannot contain lifecycle, history, or rationale.

`{SCOPE}.md` closes CANON—it provides the context that makes CANON reproducible: purpose, lifecycle, evidence window, validation instructions.

The bundle `{CANON.md, {SCOPE}.md}` is the minimal reproducible governance unit. Neither is complete without the other.

---

10. Distribution: GitHub as Operating System

During session 168, while documenting the system architecture, we discovered something unexpected.

GitHub already provides everything an application distribution system needs.

```
flowchart TB
    subgraph Expected["Expected"]
```

```
        E1["Build platform"]
        E2["Build users"]
        E3["Build billing"]
        E4["Build CI/CD"]
        E5["Build discovery"]
    end

    subgraph Discovered["Discovered"]
        D1["GitHub Marketplace"]
        D2["100M+ developers"]
        D3["GitHub Actions"]
        D4["Search + Topics"]
        D5["Badges"]
    end

    Expected -->|"Build"| Build["Everything"]
    Discovered -->|"Build"| VaaS["Validators only"]

    style VaaS fill:#228be6,stroke:#1864ab,color:white
    style Build fill:#868e96
```

GitHub provides hosting, users, billing, CI/CD, discovery, and certification (badges). The only missing piece is validation. CANONIC provides validators. The integration is a configuration file:

```yaml
name: CANONIC Validation
on: [push]
jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
      - uses: canonic-machine/vaas-action@v1
```

Repositories passing validation display a badge. The badge links to results. Users verify compliance without running validators locally.

This is the Ouroboros pattern—the system discovering its own distribution channel. The session documenting the architecture became evidence of the architecture working. This paper was written under CANONIC governance, validated by CANONIC validators, and will be distributed through the channel CANONIC discovered.

GitHub is the operating system.

The supplemental material lives there. The evidence lives there. The validators live there. GitHub IS the OS. The OS proves the paper.

---

11. Limitations

We do not claim:

Optimality. Three axioms suffice here. Other configurations might work for other domains. We chose these axioms because they emerged from practice, not because we proved minimality.

Generalizability. CANONIC governs governance specifications. Whether the framework applies to other domains—scientific data, legal documents, software systems—remains untested.

Scalability. The evidence window covers 21 repositories and 121 scopes. Enterprise-scale deployment (thousands of repositories, millions of scopes) is unproven.

Completeness. LANGUAGE.md is version 0.1. Known gaps exist. The specification will evolve.

The study bounds itself: December 29, 2025 through January 19, 2026. Claims about evidence outside this window are out of scope.

---

12. Conclusion

We asked: can governed human-AI collaboration produce verifiable content while making AI slop structurally inadmissible?

Within this evidence window, yes.

The system grew through recorded iteration. Every failure was caught by validators, documented in sessions, and corrected. The failures prove the governance works—invalid states were detected and rejected.

```
flowchart LR
    subgraph Achieved["v0.1"]
        A1["3 Axioms"]
        A2["121 Scopes"]
        A3["332 Sessions"]
        A4["163 Disclosures"]
        A5["LANGUAGE v0.1"]
    end

    A1 & A2 & A3 & A4 & A5 --> Closure["Closure"]
    Closure --> Proof["Self-proving"]

    style Closure fill:#228be6,stroke:#1864ab,color:white
    style Proof fill:#40c057,stroke:#2b8a3e,color:white
```

Constitutional governance makes verifiability structural.

Traditional papers describe work that happened elsewhere. This paper *is* the work. The repositories are the evidence. Clone them. Run validators. Trace any claim to its commit.

The paper proves itself by running.

We didn't invent. We recognized. The cure for AI slop is 50 years old. Compiler theory solved it. We applied it.

Three axioms. 21 days. Closure.

Author Context

This work emerges from two decades of research in precision medicine, data annotation, and clinical AI [41–50]. The problem of AI slop—content that sounds authoritative but lacks evidence—mirrors the challenge of biomedical data annotation at scale [42,44]. Precision annotation of digital samples in NCBI's Gene Expression Omnibus [42] required distinguishing validated labels from inferred ones. Large-scale semi-automated labeling of clinical records [44] required separating ground truth from approximation. The governance/description separation in CANONIC applies the same principle: what is LAW (CANON) versus what is DESCRIPTION (README).

Deep learning models for clinical diagnosis [46,47] highlighted a paradox: more powerful models require more rigorous validation. A model that predicts Alzheimer's disease from PET scans [47] or estimates delirium risk [46] must trace every claim to evidence. CANONIC makes this traceability structural—not a best practice, but a compilation requirement.

The MammoChat project [43,45,51] pioneered patient-centered AI for breast cancer imaging. Blockchain-based provenance ensured patients owned their data [43]. CANONIC extends this insight: all governed content has cryptographic provenance via git. The LEDGER is the immutable record. The framework emerged from practice—from the constraints of handling real patient data where every claim has consequences.

Acknowledgments

The AI assistants Claude Opus 4.5 and Claude Sonnet 4.5 (Anthropic, 2025) contributed production under governance throughout this project. Their contributions are recorded in the ledger with model identity disclosure per CANONIC governance requirements.

The framework name honors the insight from *Dividends & Deaths* where the governance/description separation first emerged—originally as CANNON (with a typo), now as CANON.

Funding

This work builds on over $5M in NIH-funded research spanning two decades:

BD2K (Big Data to Knowledge) Awards: - NIH U01-LM012675: "Crowd-Assisted Deep Learning (CrADLe): Digital Curation to Translate Big Data into Precision Medicine" - NIH BD2K Crowdsourcing Award, NCI (2016) - STARGEO.org: Search Tag Analyze Resource for Gene Expression Omnibus

Recognition: - Inaugural Marcus Award for Precision Medicine Innovation, UCSF (2016) - Casey DeSantis Florida Cancer Innovation Award, MammoChat (2025)

Training: - MD/PhD in Genomics and Computational Biology, University of Pennsylvania - Clinical Pathology Residency, Stanford University - NIH-funded translational bioinformatics fellowship, UCSF (Butte Lab)

Institutional Support: - University of Central Florida, College of Medicine (Chief of AI) - American Board of Precision Medicine (Director of AI in Precision Medicine)

The governance patterns crystallized here were forged in the constraints of clinical AI where every claim must be defensible—from PennCNV [48] to precision annotation [42] to breast cancer imaging

[43,45]. All funding is ledgered in NIH RePORTER. All publications are ledgered in PubMed. CANONIC extends this ledger to governance itself.

Disclosure

This manuscript was produced under CANONIC governance.

Model: Claude Opus 4.5 (claude-opus-4-5-20251101)

Evidence window: December 29, 2025 through January 19, 2026

Validation status: PASS

Supplemental material: github.com/canonic-machine

References

AI Slop and Content Verification

[1] Oxford Languages. (2025). Word of the Year 2025: "AI Slop." Oxford University Press.

[2] Weidinger, L., et al. (2021). Ethical and social risks of harm from Language Models. arXiv:2112.04359.

[3] Ji, Z., et al. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12), 1–38.

[4] Bender, E.M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *FAccT '21*, 610–623.

Compiler Theory Foundations

[5] Backus, J.W., et al. (1960). Report on the Algorithmic Language ALGOL 60. *Communications of the ACM*, 3(5), 299–314.

[6] Chomsky, N. (1956). Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3), 113–124.

[7] Chomsky, N. (1959). On Certain Formal Properties of Grammars. *Information and Control*, 2(2), 137–167.

[8] Knuth, D.E. (1965). On the Translation of Languages from Left to Right. *Information and Control*, 8(6), 607–639.

[9] Aho, A.V., Lam, M.S., Sethi, R., & Ullman, J.D. (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Addison-Wesley.

Scope Resolution and Name Binding

[10] Johnston, J.B. (1971). The Contour Model of Block Structured Processes. *ACM SIGPLAN Notices*, 6(2), 55–82.

[11] Dijkstra, E.W. (1960). Recursive Programming. *Numerische Mathematik*, 2(1), 312–318.

[12] Landin, P.J. (1964). The Mechanical Evaluation of Expressions. *The Computer Journal*, 6(4), 308–320.

Type Systems and Semantic Verification

[13] Milner, R. (1978). A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3), 348–375.

[14] Hindley, R. (1969). The Principal Type-Scheme of an Object in Combinatory Logic. *Transactions of the American Mathematical Society*, 146, 29–60.

[15] Cardelli, L., & Wegner, P. (1985). On Understanding Types, Data Abstraction, and Polymorphism. *Computing Surveys*, 17(4), 471–523.

[16] Pierce, B.C. (2002). *Types and Programming Languages.* MIT Press.

Formal Verification and Proof Assistants

[17] Curry, H.B., & Feys, R. (1958). *Combinatory Logic, Volume I.* North-Holland.

[18] Howard, W.A. (1980). The Formulae-as-Types Notion of Construction. *To H.B. Curry: Essays on Combinatory Logic*, 479–490.

[19] Martin-Löf, P. (1984). *Intuitionistic Type Theory.* Bibliopolis.

[20] Coquand, T., & Huet, G. (1988). The Calculus of Constructions. *Information and Computation*, 76(2–3), 95–120.

Distributed Systems and Immutable Ledgers

[21] Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7), 558–565.

[22] Merkle, R.C. (1988). A Digital Signature Based on a Conventional Encryption Function. *CRYPTO '87*, 369–378.

[23] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.

[24] Torvalds, L., & Hamano, J. (2005). Git: Fast Version Control System. git-scm.com.

Language Specifications

[25] Ritchie, D.M. (1993). The Development of the C Language. *History of Programming Languages II*, 671–698.

[26] Gosling, J., et al. (2021). *The Java Language Specification* (Java SE 17 ed.). Oracle.

[27] Bradbury, S., et al. (2025). The Go Programming Language Specification. go.dev/ref/spec.

[28] Klabnik, S., & Nichols, C. (2023). *The Rust Programming Language.* No Starch Press.

AI Systems and Human-AI Collaboration

[29] Vaswani, A., et al. (2017). Attention Is All You Need. *NeurIPS 2017*, 5998–6008.

[30] Brown, T., et al. (2020). Language Models are Few-Shot Learners. *NeurIPS 2020*, 1877–1901.

[31] Anthropic. (2024). Claude 3 Model Card. anthropic.com.

[32] OpenAI. (2023). GPT-4 Technical Report. arXiv:2303.08774.

Governance and Constitutional Design

[33] Ostrom, E. (1990). *Governing the Commons.* Cambridge University Press.

[34] Lessig, L. (1999). *Code and Other Laws of Cyberspace.* Basic Books.

[35] Wright, A., & De Filippi, P. (2015). Decentralized Blockchain Technology and the Rise of Lex Cryptographia. SSRN 2580664.

CANONIC Specifications (This Work)

[36] Hadley, D. (2026). CANONIC LANGUAGE Specification v0.1. github.com/canonic-machine/canonic.

[37] Hadley, D. (2026). CANONIC COVERAGE Specification. github.com/canonic-machine/canonic.

[38] Hadley, D. (2026). CANONIC VOCAB Definitions. github.com/canonic-machine/canonic.

[39] Hadley, D. (2026). VaaS: Validators as a Service. github.com/canonic-machine/validators.

[40] Hadley, D. (2026). CANONIC CANON (Root Axioms). github.com/canonic-machine/canonic.

Prior Work by the Author

[41] Hadley, D., et al. (2017). Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *eLife*, 6, e26726. PMID: 28936969.

[42] Hadley, D., et al. (2017). Precision annotation of digital samples in NCBI's gene expression omnibus. *Scientific Data*, 4, 170125. PMID: 28925997.

[43] Panahiazar, M., Chen, N., Lituiev, D., & Hadley, D. (2022). Empowering study of breast cancer data with application of artificial intelligence technology. *Clinical & Experimental Metastasis*, 39(1), 117–127. PMID: 34697751.

[44] Ding, S., et al. (2019). Large Scale Semi-Automated Labeling of Routine Free-Text Clinical Records for Deep Learning. *Journal of Digital Imaging*, 32(1), 30–37. PMID: 30128778.

[45] Ding, S., et al. (2019). Automatic Labeling of Special Diagnostic Mammography Views from Images and DICOM Headers. *Journal of Digital Imaging*, 32(2), 228–233. PMID: 30465142.

[46] Wong, W., et al. (2018). Development and Validation of an Electronic Health Record-Based Machine Learning Model to Estimate Delirium Risk. *JAMA Network Open*, 1(4), e181018. PMID: 30646095.

[47] Ding, S., et al. (2019). A Deep Learning Model to Predict a Diagnosis of Alzheimer Disease by Using 18F-FDG PET of the Brain. *Radiology*, 290(2), 456–464. PMID: 30398430.

[48] Wang, K., Li, M., Hadley, D., et al. (2007). PennCNV: An integrated hidden Markov model designed for high-resolution copy number variation detection. *Genome Research*, 17(11), 1665–1674. PMID: 17921354.

[49] Hadley, D., et al. (2020). The Impact of COVID-19 on African American Communities in the United States. *Health Equity*, 4(1), 476–483. PMID: 33269331.

[50] Gianfrancesco, M., et al. (2019). Tracing diagnosis trajectories over millions of patients reveal an unexpected risk in schizophrenia. *Scientific Data*, 6, 210. PMID: 31615985.

CANONIC Ecosystem (Open Source)

[51] MammoChat. (2025). Empowering Women with Empathic AI. github.com/canonic-machine/mammochat.

[52] STARGEO. (2016). Search Tag Analyze Resource for Gene Expression Omnibus. github.com/idrdex/stargeo.

[53] Hadley Lab. (2025). Translating Big Data into Precision Medicine. github.com/hadleylab.

[54] CANONIC Foundation. (2026). Constitutional AI Governance Framework. canonic.org.

Appendix A: Root Axioms

Verbatim from `canonic/CANON.md`:

Axiom 0 — Triad

> A scope **MUST** contain: `CANON.md`, `VOCAB.md`, `README.md`.

Axiom 1 — Inheritance

> Every `CANON.md` **MUST** declare `inherits:`. Inheritance chains **MUST** terminate at `/`. Inherited axioms are final.

Axiom 2 — Introspection

> `VOCAB.md` **MUST** define every concept used in `CANON.md` and `VOCAB.md`.

Appendix B: Formal Foundations

B.1 Definitions

Definition 1 (Scope). A scope S is a tuple (P, T, A, V) where: - P is a filesystem path - T = {CANON.md, VOCAB.md, README.md} is the triad - A is a set of axioms declared in CANON.md - V is a set of definitions declared in VOCAB.md

Definition 2 (Inheritance Chain). For scope S with path P, the inheritance chain I(S) is the sequence:

```
I(S) = [S, parent(S), parent(parent(S)), ..., root]
```

where parent(S) is the scope at the parent directory of P, and root is the scope at path "/".

Definition 3 (Vocabulary Closure). A scope S is vocabulary-closed iff:

```
  t   terms(CANON_S)   terms(VOCAB_S) : t   definitions(VOCAB_S)   definitions(I(S))
```

That is, every term used in CANON or VOCAB must be defined either locally or in an ancestor scope.

Definition 4 (Validity). A scope S is valid iff: 1. T   files(P) — the triad exists 2. I(S) terminates at root — inheritance is grounded 3. S is vocabulary-closed — introspection holds

---

B.2 Compiler Correspondence

Theorem 1 (Syntactic Correspondence). The Triad axiom is isomorphic to context-free grammar syntax requirements.

Proof. A context-free grammar G = (V, Σ, R, S) requires: - V: set of non-terminal symbols - Σ: set of terminal symbols - R: set of production rules - S: start symbol

We construct a mapping : CANONIC → CFG: - (scope identifiers) = V - ({CANON.md, VOCAB.md, README.md}) = Σ - (triad requirement) = R, specifically: Scope → CANON VOCAB README - (root) = S

The Triad axiom mandates that for all valid scopes:

```
files(S)   {CANON.md, VOCAB.md, README.md}
```

This is precisely a syntactic well-formedness constraint in the grammar sense: a scope is syntactically valid iff it contains the required terminal symbols. The mapping   is structure-preserving.

Therefore, TRIAD   SYNTAX.

Theorem 2 (Scope Resolution Correspondence). The Inheritance axiom implements static scope resolution equivalent to the contour model.

Proof. The contour model [Johnston 1971] defines name resolution in block-structured languages:

```
resolve(name, block) =
    if name   declarations(block) then lookup(name, block)
    else resolve(name, enclosing(block))
```

The Inheritance axiom defines concept resolution:

```
resolve(term, scope) =
    if term   definitions(VOCAB_scope) then definition(term, scope)
    else resolve(term, parent(scope))
```

The structural identity is evident: - block   scope - declarations   definitions(VOCAB) - enclosing   parent - name   term

Both terminate: the contour model at the outermost block, CANONIC at root. Root termination is guaranteed by Axiom 1: inheritance chains MUST terminate at "/".

Therefore, INHERITANCE   SCOPE_RESOLUTION.

Theorem 3 (Type System Correspondence). The Introspection axiom implements a closed type system equivalent to Hindley-Milner completeness.

Proof. The Hindley-Milner type system [Hindley 1969, Milner 1978] has the principal type property: every well-typed expression has a most general type that can be inferred without annotations.

The key property is *closure*: all type references resolve to type definitions within the system.

Introspection requires:

```
 t   terms(CANON)   terms(VOCAB) : t   definitions(VOCAB*)
```

where VOCAB* denotes the transitive closure of VOCAB through inheritance.

This is vocabulary closure—no undefined terms can appear. The correspondence: - type reference   term usage - type definition   vocabulary definition - type inference   definition lookup through inheritance chain - principal types   canonical definitions (inherited definitions are final)

A CANONIC scope is well-typed iff every term resolves to exactly one definition (the closest in the inheritance chain). This parallels how Hindley-Milner infers the principal type.

Therefore, INTROSPECTION   TYPE_SYSTEM.

B.3 Completeness

Theorem 4 (Axiom Independence). The three axioms are mutually independent.

Proof. We construct counterexamples showing each axiom can fail while others hold:

1. *Triad fails, others hold:* A directory with only CANON.md and VOCAB.md, where CANON declares inheritance to root and VOCAB defines all its terms. The triad is incomplete, but inheritance and introspection are satisfied.
2. *Inheritance fails, others hold:* A scope with complete triad and closed vocabulary, but CANON declares `inherits: /nonexistent/path/`. The chain doesn't terminate at root.
3. *Introspection fails, others hold:* A scope with complete triad and valid inheritance, but CANON uses term “foobar” which appears in no VOCAB in the inheritance chain.

Since each axiom can fail independently, no axiom is derivable from the others.

Theorem 5 (Axiom Minimality). Removing any axiom breaks the governance system.

Proof.

1. *Remove Triad:* Without mandatory files, there is no structure to validate. A directory with arbitrary files could claim validity. Governance becomes undefined.
2. *Remove Inheritance:* Without grounded authority chains, scopes can claim arbitrary parents or form cycles. Authority becomes unverifiable.
3. *Remove Introspection:* Without vocabulary closure, terms can mean anything. Slop becomes admissible—undefined jargon passes validation.

Each removal admits a class of invalid content that the full system rejects.

Corollary (Compiler Correspondence Completeness). The three axioms provide complete coverage of the three fundamental compiler concepts.

Proof. By Theorems 1-3: - TRIAD SYNTAX (structure) - INHERITANCE SCOPE_RESOLUTION (binding) - INTROSPECTION TYPE_SYSTEM (semantics)

By Theorems 4-5, the axioms are independent and minimal.

Syntax, scope resolution, and type systems are the three pillars of compiler theory [Aho et al. 2006]. CANONIC provides governance analogs for each, and only these three.

The correspondence is complete.

B.4 Decidability

Theorem 6 (Validation Decidability). Scope validity is decidable in O(n) where n is the total size of all files in the inheritance chain.

Proof. The validation algorithm:

```
validate(scope):
    // Triad check: O(1)
    if not exists(CANON.md) or not exists(VOCAB.md) or not exists(README.md):
        return INVALID

    // Inheritance check: O(d) where d = chain depth
    chain = []
    current = scope
    while current != root:
        if current in chain:  // cycle detection
            return INVALID
        chain.append(current)
        current = parent(current)

    // Introspection check: O(n)
    all_definitions = union(definitions(v) for v in chain)
    all_terms = union(terms(c) for c in chain)  union(terms(v) for v in chain)
    if not all_terms  all_definitions:
        return INVALID

    return VALID
```

Each step is decidable: - File existence: decidable (filesystem operation) - Parent traversal: decidable (terminates by finite depth or cycle detection) - Set operations: decidable (finite sets)

Total complexity: O(d) + O(n) = O(n) since d n.

Therefore, validation halts for all inputs.

Appendix C: Reproducibility Protocol

```
# Clone the evidence
git clone https://github.com/canonic-machine/canonic.git
cd canonic

# Run validators
python3 validators/vaas.py
# Expected: VALIDITY: PASS

# Count scopes
find . -name "CANON.md" | wc -l
# Expected: 21

# Verify specific claim
git show 11affab
# Expected: First CANON.md content
```

If any verification fails, the paper's claims are falsified.

Appendix D: Evidence Index

Claim	Evidence	Verification
Proto-CANONIC origin	`dividends:07a5834`	`git show 07a5834`
First CANON.md	`canonic:11affab`	`git show 11affab`
LANGUAGE.md v0.1	`canonic:81bb6d5`	`git show 81bb6d5`
21 repositories	Directory listing	`ls -d */`
121 scopes	CANON.md count	`find . -name "CANON.md"`
332 sessions	Episode count	`find . -name "ep*.md"`
163 disclosures	Disclosure count	`ls patents/disclosures/`

Supplemental Material: github.com/canonic-machine

166 drift episodes. 163 invention disclosures. Full validator source. Complete git history.

GitHub IS the OS. The supplemental proves the paper.

Built with Claude. Governed by CANONIC. Self-proving.