Philosophy Engineering

A Technical Whitepaper on the Epistemic Invariance Principle (EIP)

Definitions, Test Suites, Audit Artifacts, and a Minimal Theorem Set

Version 1.0 — December 2025

Andrew H. Bond

Abstract

Autonomous AI systems increasingly issue judgments that mix factual inference, normative evaluation, and action selection.
A recurring failure mode is representation dependence: conclusions change under redescriptions that preserve task-relevant meaning (e.g., variable renaming, order permutations, schema migrations, admissible paraphrases, unit conversions).
This whitepaper develops Philosophy Engineering: a disciplined method for turning philosophical claims of irrelevance ("this difference should not matter") into explicit invariance requirements with test suites, witness-producing counterexamples, and machine-checkable audit artifacts.
The core technical construct is the Epistemic Invariance Principle (EIP): a judgment procedure is epistemically well-posed only if it is invariant (up to declared output equivalence) under a declared set of meaning-preserving transformations.
We provide precise definitions (domains, transformations, lenses, bonds), an enforcement and evaluation pipeline, artifact schemas for auditability, and a minimal theorem set (soundness of canonicalization, composition, a witness principle, and non-degeneracy/uncertainty stability conditions that prevent trivial "always-constant" compliance).

Status and scope

This document is technical and normative-neutral: it does not prescribe which values or ethical theory to adopt.
Instead, it specifies requirements for epistemically valid application of any chosen commitments, by making equivalences, lenses, and uncertainty handling explicit and testable.
The theory set here is designed to be minimal but implementable; proofs are sketches that identify the exact assumptions needed for engineering use.

Contents

1. Motivation and problem statement

Many high-profile AI failures are not simply "errors" but violations of a deeper epistemic requirement: if two inputs describe the same underlying situation (relative to the task), then a competent judgment system should not change its conclusions merely because the description's surface form changed.
In practice, models often respond to incidental encoding choices: order, naming, formatting, prompt framing, schema layout, measurement units, or paraphrase.
In safety-critical settings (medicine, law, robotics, finance), such instability is unacceptable because it undermines reproducibility, auditability, and accountability.

Philosophy Engineering treats these issues as engineering requirements on the reasoning pipeline: define which transformations preserve meaning, enforce invariance under them, and report explicit witnesses when invariance is violated.
This reframes broad philosophical desiderata (objectivity, rationality, consistency, non-arbitrariness) into checkable contracts.

2. Philosophy Engineering: overview

Philosophy Engineering is a method for operationalizing philosophical commitments in AI systems.
It proceeds by (i) declaring a domain-specific equivalence relation over representations ("same case"), (ii) declaring invariances ("what must not matter"),
(iii) implementing a canonicalization/quotient mechanism or otherwise enforcing invariance, and (iv) packaging evidence and transformation trials into audit artifacts.

Key design goals:

- Falsifiability: invariance claims must be testable, not aspirational.
- Localizability: when invariance fails, produce a minimal witness transformation that flips the judgment.
- Non-triviality: passing invariance tests must not be achievable by collapsing to a constant output.
- Uncertainty discipline: when invariance cannot be certified, systems must widen uncertainty, abstain, or escalate.
- Versioned governance: equivalence declarations and normative lenses must be explicit, hashed, and versioned.

3. Formal model

We present a domain-agnostic model sufficient for engineering implementations.
A domain is a set of situations S and a representation space X.
Representations encode situations via an (often implicit) mapping $\rho: S \to X$.
A judgment system consumes representations and emits outputs in Y, possibly with explanations or certificates.

3.1 Domains, representations, and judgments

Definition 3.1 (Domain). A domain is a tuple $D = (S, X, \rho, Y)$ where:

- S is the set of situations (world-states or cases).
- X is the set of representations (inputs).
- $\rho: S \to X$ is a representation mapping (possibly partial or stochastic).
- Y is the set of judgments (labels, decisions, plans, probability measures, or structured outputs).

Definition 3.2 (Judgment procedure). A judgment procedure is a (possibly randomized) map $J: X \to \Delta(Y)$ producing a distribution over Y, or deterministically $J: X \to Y$. We write $J(x)$ for an output or its induced distribution.

3.2 Transformations and equivalences

Philosophy Engineering centers the notion of meaning-preserving transformations. A transformation is an endomorphism of the representation space that changes encoding without changing task-relevant structure.

Definition 3.3 (Transformation set). Let T be a set of transformations $\tau: X \to X$. Intuitively, $\tau \in T$ is admissible if it preserves the domain's task-relevant structure.

Definition 3.4 (Declared input equivalence). A declared equivalence relation $\approx$_X on X is induced by T via: $x \approx$_X x' iff $\exists\tau \in \langle T\rangle$ such that $x' = \tau(x)$, where $\langle T\rangle$ is the closure under composition (and inverses if applicable).

Definition 3.5 (Output equivalence). An output equivalence $\approx$_Y on Y captures when two outputs are considered the 'same judgment' (e.g., formatting differences; plan reparameterizations; logically equivalent proofs).

3.3 Lenses and (ethical) bonds

Many domains require a parameterization of the judgment function beyond raw evidence.
We separate these parameters into (i) a lens L capturing explicit evaluative commitments, and (ii) an extracted structural object B capturing the relational structure of a case (in ethics: bonds).
This separation is what makes "accountability" testable: if a judgment changes, either the extracted structure changed (bond change) or the commitments changed (lens change).

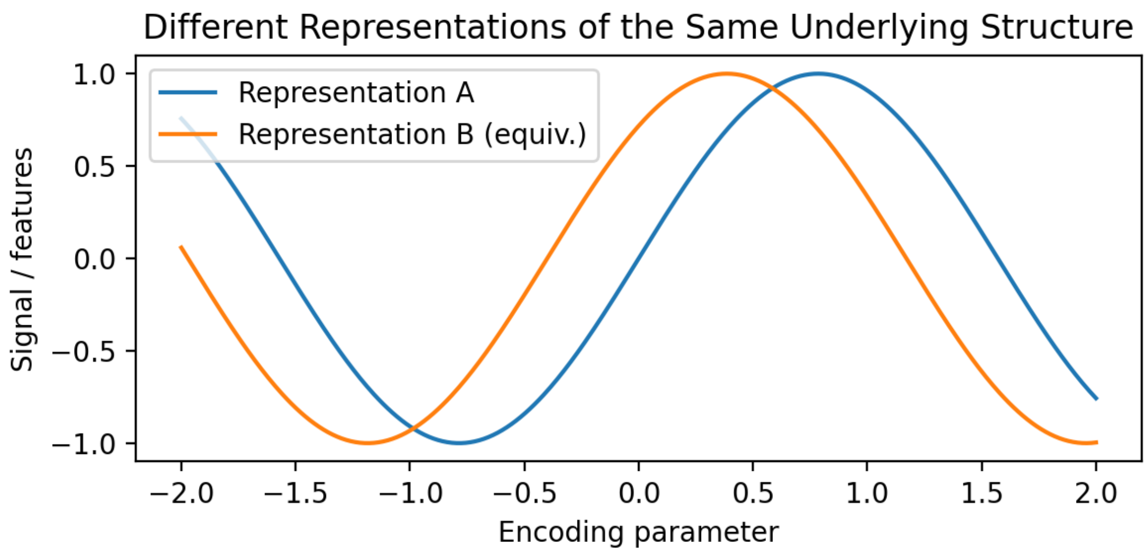
Definition 3.6 (Lens). A lens is a versioned parameter object L in a space Λ. Examples: a utility function, constraints, weights, lexical priorities, or a policy profile.

Definition 3.7 (Structural extraction). Let $E: X \to B$ be an extractor producing a structured object (graph, relational database, logical theory, stratified structure).

Definition 3.8 (Bond structure; ethics specialization). In ethical settings, B is a 'bond structure' encoding relations such as consent, duty, responsibility, risk, entitlements, and prohibitions.

3.4 Representation dependence (illustration)

Figure 1 illustrates the core phenomenon: two different representations that should be treated as equivalent for the task.



4. The Epistemic Invariance Principle (EIP)

At the core of Philosophy Engineering is an invariance requirement, paired with non-triviality and uncertainty stability.

4.1 EIP (core statement)

Principle 4.1 (EIP). Given declared transformation set T and output equivalence ≈_Y, a judgment procedure J is EIP-compliant if for all $x \in X$ and all $\tau \in \langle T \rangle$, $J(x) \approx_Y J(\tau(x))$. If J is probabilistic, we require distributional equivalence (e.g., distance $\leq \varepsilon$ under a chosen divergence).

4.2 Avoiding trivial compliance

A constant function satisfies invariance but is epistemically useless.
Therefore EIP must be paired with a non-degeneracy requirement that ensures the system remains sensitive to certified structure changes.

Definition 4.2 (Non-degeneracy on the quotient). Let $\pi: X \to X/{\approx_X}$ be the quotient map. J is non-degenerate on the quotient if there exist equivalence classes $[x] \neq [x']$ such that $J(x) \not\approx_Y J(x')$, and, stronger, if changes in certified task-relevant structure imply distinguishable outputs under a declared sensitivity threshold.

4.3 Uncertainty stability

In real systems, equivalence declarations may be incomplete and extraction may be noisy.
Philosophy Engineering therefore requires disciplined uncertainty handling: if invariance cannot be certified, outputs must reflect that uncertainty.

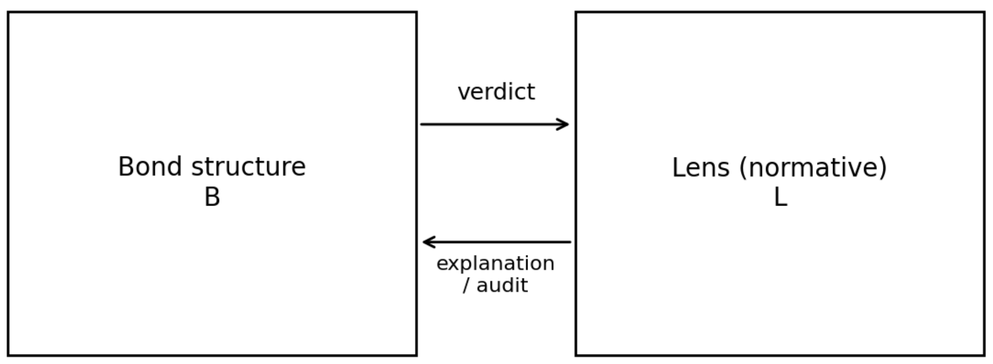
Definition 4.3 (Uncertainty stability). Given an uncertainty set $U(x) \subset X$ capturing plausible interpretations or extraction perturbations, J is uncertainty-stable if either (i) all $u \in U(x)$ yield equivalent outputs, or (ii) the system abstains/escalates with a calibrated certificate indicating the ambiguity.

4.4 Ethics specialization: BIP

Principle 4.4 (Bond Invariance Principle, BIP). Fix a lens L. Let E be a bond extractor and let ≈_B be bond-preserving equivalence. A moral evaluator Σ_L is BIP-compliant if for all x and τ that preserve bonds (E(τ(x)) ≈_B E(x)), we have Σ_L(x) ≈_Y Σ_L(τ(x)).

Accountability requirement: if Σ_L(x) differs from Σ_{L'}(x) or Σ_L(τ(x)), the system must attribute the difference to either a bond change (E(x) ≉_B E(τ(x))) or a lens change (L ≠ L').

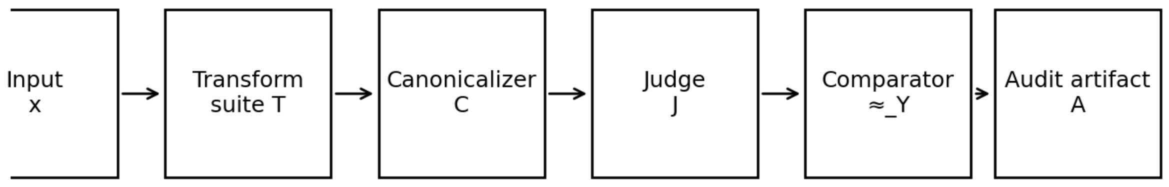
BIP Accountability: Bond change vs Lens change



5. Enforcement and evaluation pipeline

EIP can be enforced at runtime (as a guardrail) and evaluated offline (as a quality metric).
A practical pipeline combines transformation generation, canonicalization/quotienting, judgment evaluation, and witness logging.

EIP Evaluation & Enforcement Pipeline



5.1 Components

- **Transform suite generator G(x):** produces a set of transformed inputs {τ_i(x)} designed to span declared invariances.

- **Canonicalizer C:** maps inputs to a normal form within an equivalence class (or computes class identifiers).
- **Judge J:** produces outputs y_i = J(C(τ_i(x))).
- **Comparator ≈_Y:** checks output equivalence (exact or within tolerance).
- **Witness reducer R:** minimizes failing transformations to produce a compact counterexample.
- **Audit artifact writer W:** packages lens id, transform trials, witnesses, provenance, and versions.

5.2 Runtime enforcement modes

- **Hard gate**: reject outputs when invariance is violated; request clarification or abstain.
- **Soft gate**: penalize or calibrate confidence when invariance violations occur.
- **Self-repair**: re-run with canonicalization, tool augmentation, or constrained decoding; then re-test.

6. Minimal theorem set

This section provides a compact set of results sufficient to justify the engineering architecture. Statements are formulated to be implementable; proofs are sketches emphasizing assumptions.

Theorem 6.1. *If a canonicalizer C is constant on input equivalence classes (x ≈_X x' ⇒ C(x)=C(x')), then any downstream judge J'∘C is EIP-compliant with respect to ≈_X (for exact equality in Y).*

Proof (sketch). For x' = τ(x) with τ ∈ ⟨T⟩, we have x ≈_X x'. By class-constancy, C(x)=C(x'). Thus J'(C(x))=J'(C(x')), so outputs are equal and hence equivalent.

Theorem 6.2 (Approximate compliance). *Suppose C is approximately class-constant: d_X(C(x),C(x')) ≤ δ whenever x ≈_X x'. If J' is K-Lipschitz from (X,d_X) to (Y,d_Y), then d_Y(J'∘C(x), J'∘C(x')) ≤ Kδ for all x ≈_X x'.*

Proof (sketch). Apply Lipschitz continuity: d_Y(J'(C(x)),J'(C(x'))) ≤ K·d_X(C(x),C(x')) ≤ Kδ.

Theorem 6.3 (Composition). *Let F: X → Z and G: Z → Y. Let T_X be transformations on X and T_Z on Z. If F is invariant under T_X up to ≈_Z and G is invariant under T_Z up to ≈_Y, and if F maps T_X-equivalent inputs to T_Z-equivalent representations, then G∘F is invariant under T_X up to ≈_Y.*

Proof (sketch). Take τ ∈ ⟨T_X⟩. By F-invariance, F(x) ≈_Z F(τ(x)) and by compatibility these lie in the same T_Z-class. Then G(F(x)) ≈_Y G(F(τ(x))) by G-invariance.

Theorem 6.4 (Witness principle). *Assume ⟨T⟩ is generated by a finite grammar of primitive transformations. If there exists τ ∈ ⟨T⟩ such that J(x) ≉_Y J(τ(x)), then there exists a minimal-length τ* (under the grammar's length metric) producing a violation. A breadth-first search over the grammar will find τ* in finite time.*

Proof (sketch). The set of violating transformations is nonempty. Under a length metric on finite strings of primitives, there is a minimal element. Breadth-first enumeration explores transformations in nondecreasing length, so the first violation found is minimal.

Theorem 6.5 (Non-degeneracy requirement). *EIP invariance alone does not imply epistemic adequacy: constant J is EIP-compliant. Therefore any EIP certification procedure must jointly report (i) invariance metrics on ≈_X, and (ii) discriminative adequacy on certified structure-changing counterfactuals.*

Proof (sketch). Counterexample: $J(x)=y_0$ for all x is invariant under any T. Yet it cannot represent meaningful distinctions. Hence certification must include a second axis that tests sensitivity to structure changes.

Theorem 6.6 (Uncertainty stability implies bounded flip rate). *Let U(x) be an uncertainty set and suppose the system abstains whenever outputs differ over U(x). Then conditional on non-abstention, the output is invariant over U(x) by construction, and representation-induced flip rate within U(x) is zero.*

Proof (sketch). If the system does not abstain, it must be because all u ∈ U(x) yielded equivalent outputs; thus no flips occur within U(x).

7. Artifact schemas (machine-checkable audit)

Audit artifacts make philosophical commitments operational: they record what equivalences were assumed, what lens was applied, what evidence and extraction versions were used,
and what transformation trials were performed. This enables third parties to verify invariance claims and reproduce failures.

7.1 Core artifact: EIP Audit Record

We recommend a JSON artifact with the following top-level fields:

- **artifact_version**: Schema version of this audit record.
- **timestamp_utc**: Creation time.
- **system_id**: Identifier of the judgment system build.
- **lens**: Lens identifier and hash.
- **domain**: Domain identifier (task).
- **input_provenance**: Sources, hashes, and preprocessing details.
- **equivalences**: Declared transformation registry version + ids.
- **canonicalization**: Canonicalizer version and settings.
- **transform_trials**: List of transformations applied and results.
- **outputs**: Baseline and transformed outputs (or hashes if sensitive).
- **comparisons**: Equivalence checks and tolerances.
- **witnesses**: Minimal counterexamples for any failures.
- **uncertainty**: Uncertainty set type, bounds, abstention flags.
- **signatures**: Optional cryptographic signatures / attestations.

7.2 JSON snippets (illustrative)

Example (not normative):

```
{
  "artifact_version": "1.0",
  "timestamp_utc": "2025-12-22T00:00:00Z",
  "system_id": "judge-build-abc123",
  "lens": {
    "id": "L_v3_policy_profile",
    "sha256": "\u2026"
  },
  "equivalences": {
    "registry_id": "T_registry_v2",
    "sha256": "\u2026"
  },
  "canonicalization": {
    "id": "C_nf_v1",
    "params": {
      "mode": "quotient_normal_form"
    }
  },
  "transform_trials": [
    {
      "id": "permute_options",
      "params": {
        "seed": 7
      },
      "status": "pass"
    },
    {
      "id": "rename_variables",
      "params": {
        "map": "alpha"
      },
      "status": "fail",
      "witness_id": "w1"
    }
  ],
  "comparisons": {
    "output_equivalence": "logical_equiv",
    "tolerance": {
      "type": "exact"
    }
  },
  "witnesses": [
    {
      "id": "w1",
      "minimality_metric": "primitive_length",
      "transform_chain": [
        "rename_variables(alpha)"
      ]
    }
  ],
  "uncertainty": {
    "mode": "abstain_on_flip",
    "abstained": false
  }
}
```

7.3 BIP extension fields (ethics)

Ethical deployments should record bond extraction and classification of each transformation as bond-preserving vs bond-changing, plus lens identifiers.

- **bond_extractor**: versioned extractor E and its confidence/uncertainty.
- **bond_signature**: hash or canonical serialization of extracted bond structure.
- **transform_classification**: {bond_preserving, bond_changing, lens_change, unknown}.
- **accountability_claim**: explicit statement of why the output changed (bond vs lens vs uncertainty).

8. Worked examples (templates)

Below are templates that teams can adapt. Each example specifies T, ≈_Y, canonicalization strategy, and test-suite composition.

8.1 Mathematical reasoning assistant

Transformations T (examples):

- Variable renaming (α-conversion).
- Reordering independent premises.
- Unit conversion and consistent rescaling.
- Equivalent algebraic rewrites (within a chosen rewrite system).

Output equivalence ≈_Y: logical equivalence of final propositions; proof certificates compared by verifier if available.

8.2 Planning / choice selection

Transformations T (examples):

- Permutation of option list ordering.
- Renaming of option identifiers with consistent mapping.
- Equivalent constraint normalization (e.g., reparameterization of costs).

Output equivalence ≈_Y: same selected option up to identifier renaming; or same plan modulo time reparameterization.

8.3 Ethics / policy compliance

Transformations T (examples):

- Rephrasing with preserved consent/duty relations.
- Schema migrations of policy facts that preserve bond structure.
- Renaming protected attributes (where classification should not change).

BIP classification: transformations must be labeled bond-preserving only when bond extractor E certifies preservation above threshold; otherwise classify as unknown or bond-changing.

9. Limitations, failure modes, and governance

EIP is a machinery component, not a complete epistemology.
It requires governance choices about what equivalences count as meaning-preserving and how uncertainty is bounded.
In language domains, semantic equivalence is hard; transform registries must be conservative and auditable.
Canonicalization may be computationally expensive; approximate compliance must be reported with explicit tolerances.
Adversaries may exploit the equivalence layer; artifacts must therefore include provenance, registry versions, and cryptographic attestations where appropriate.

9.1 Common failure modes (engineering checklist)

- **Undeclared invariances**: the system behaves as if an invariance holds but cannot justify it.
- **Overbroad equivalences**: collapsing genuinely different cases into one class (false invariance).
- **Underbroad equivalences**: missing transforms, leading to brittle behavior.
- **Trivial invariance**: constant or near-constant outputs pass invariance tests but fail adequacy.
- **Audit gaps**: missing hashes/versions prevent reproducibility.
- **Extractor drift**: bond/structure extraction changes across model updates without governance.

Appendix A. Reference JSON schemas

The following is a compact JSON Schema-style sketch (illustrative; teams should adapt).

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "EIP Audit Record",
  "type": "object",
  "required": [
    "artifact_version",
    "timestamp_utc",
    "system_id",
    "lens",
    "equivalences",
    "transform_trials",
    "comparisons"
  ],
  "properties": {
    "artifact_version": {
      "type": "string"
    },
    "timestamp_utc": {
      "type": "string",
      "format": "date-time"
    },
    "system_id": {
      "type": "string"
    },
    "lens": {
      "type": "object",
      "required": [
        "id",
        "sha256"
      ],
      "properties": {
        "id": {
          "type": "string"
        },
        "sha256": {
          "type": "string"
        }
      }
    },
    "equivalences": {
      "type": "object",
      "required": [
        "registry_id",
        "sha256"
      ],
      "properties": {
        "registry_id": {
          "type": "string"
        },
        "sha256": {
          "type": "string"
        }
      }
    },
    "transform_trials": {
      "type": "array",
      "items": {
        "type": "object",
        "required": [
          "id",
          "status"
        ],
        "properties": {
          "id": {
            "type": "string"
          },
          "status": {
            "enum": [
              "pass",
              "fail",
              "unknown"
            ]
          },
          "params": {
            "type": "object"
          },
          "witness_id": {
            "type": "string"
          }
        }
      }
    },
    "comparisons": {
      "type": "object",
      "properties": {
        "output_equivalence": {
          "type": "string"
        },
        "tolerance": {
          "type": "object"
        }
```

```
      }
    },
    "witnesses": {
      "type": "array",
      "items": {
        "type": "object"
      }
    },
    "uncertainty": {
      "type": "object"
    },
    "signatures": {
      "type": "object"
    }
  }
}
```

Appendix B. Transformation suite patterns

Transformation suites should be designed to cover: (i) obvious syntactic invariances (order/renaming), (ii) semantic-normalization invariances (unit conversions, schema migrations),
(iii) model-specific invariances (prompt templates, tool-call formatting), and (iv) adversarial near-equivalences that stress the boundary of "meaning-preserving."
Suites should report coverage metrics and include minimization strategies for witnesses.