The Architecture vs Substrate Framework

A Structural Method for Classifying Properties of Artificial Systems

Author **C.S. Thomas, Epistria, LLC** **Charles@Epistria.com**

Abstract

Debates about artificial intelligence routinely conflate three distinct layers of system analysis: the behaviors a system exhibits, the computational architecture generating those behaviors, and the physical substrate implementing that architecture. This conflation creates ambiguity around emergent properties, cognitive claims, embodiment assertions, and purported markers of consciousness.

This paper presents the **Architecture vs Substrate Framework**, a minimal structure for classifying system properties according to their invariance under controlled transformations. A system $\Sigma = (A, S, \theta)$ is evaluated by varying the **architecture** (A), the **runtime substrate** (S), the **training substrate**, and the **scale** of the model; and by ensuring the **measurement apparatus** remains invariant. Architectural equivalence across substrates is defined using **MDL-minimal generative kernel equivalence**, ensuring that cross-substrate comparisons preserve computational identity.

The framework yields four primary classes of properties—architecture-dependent, substrate-dependent, identity-level invariant, and interaction effects—refined by distinguishing weak and strong coupling. Additional tests identify training-dependent and regime-dependent properties.

By imposing this classification, the framework eliminates categorical ambiguity and constrains claims about reasoning, emergence, embodiment, and consciousness. It offers no metaphysical conclusions; it enforces structural clarity. Before asserting what an artificial system *is*, one must first determine what its properties *depend on*.

1. Introduction

Artificial intelligence research often treats “the model,” “the training,” “the hardware,” and “the behavior” as interchangeable sources of system properties. As a result, architectural artifacts are mistaken for cognitive traits; hardware-induced noise is interpreted as emergent reasoning; scale effects are conflated with architectural principles; and experimental measurements become substrate-dependent without acknowledgment.

The source of the confusion is conceptual:
there is no shared framework for separating architecture from substrate, or for distinguishing computational identity from physical instantiation.

This paper establishes such a framework. It defines:

- systems as structured triples Σ = (A, S, θ);
- architectures via MDL-minimal generative kernels;
- substrates as physical realizations of those kernels;
- properties as trajectory-level phenomena;
- and the classification of those properties via invariance under controlled transformations.

The goal is not descriptive richness but structural discipline. Without distinguishing architecture from substrate, there is no coherent basis for claims about emergence, reasoning, embodiment, or consciousness. The Architecture vs Substrate Framework provides that basis.

2. Definitions

This section establishes the minimal vocabulary required for the Architecture vs Substrate Framework. Each term is defined operationally—by how it behaves under transformation—rather than by metaphor or intuition.

2.1 System

A **system** is defined as a triple:

$$\boldsymbol{\Sigma = (A, S, \theta)}$$

where:

- **A** is an *architecture*
- **S** is a *substrate*
- **θ** is the set of learned parameters, configuration states, or control rules the system acquires through training, evolution, or design.

Σ is not the agent, nor the model, nor the behavior. It is the *structured source* from which behaviors arise when Σ interacts with data, tasks, or an environment.

2.2 Architecture (A)

An **architecture** is the abstract computational structure that defines how information flows and transforms within a system. Architectures are substrate-invariant objects and are defined extensionally by their **minimal generative kernels**.

Formally:

> Two implementations belong to the same architecture class if their MDL-minimal generative kernels are equivalent up to representational isomorphism.

This criterion—imported from PatternSense (Thomas, 2025)—grounds architectural equivalence in the shortest generative description that produces the system's behavior. It avoids ambiguity about timing, precision, and representational format while allowing architectures to be compared across radically different substrates. As Chalmers argues for organizational invariance in cognition, this framework tests properties' robustness across substrates while preserving architectural identity via MDL kernels.

An architecture therefore includes:

- computational graph topology,
- operator semantics,
- admissible representational transformations, and
- learning/optimization rules that define the family's generative behavior.

Everything else—including timing noise, quantization, spikes, or activation variances—belongs to the substrate unless it changes the MDL-minimal kernel.

2.3 Substrate (S)

A **substrate** is the *physical medium* that implements an architecture.

S specifies:

- The physical device (GPU, TPU, ASIC, FPGA, analog circuit, neuromorphic hardware, biological tissue).
- Numeric precision, quantization, timing behavior, noise profiles, and energy constraints.
- The physics that determine how an architecture is realized (digital logic, mixed-signal analog, spike timing, ion-channel dynamics, thermal limits).

S does *not* determine the computational structure; it determines the *instantiation* of that structure.

Two substrates may implement the same architecture, but the realized system behavior may differ if substrate physics modulate or distort the ideal computation.

2.4 Parameters / State (θ)

θ is the set of learned or assigned internal values that configure A on S.

This includes:

- Model weights, embeddings, and latent states.
- Optimizer states or learned update rules.
- Memory contents, if persistent.
- Any internal representation acquired through training or adaptation.

θ is *architecture-specific* but *substrate-instantiated*.
Changing S may require altering θ for numerical or hardware compatibility, but θ is conceptually an attribute of the computation, not the physics.

2.5 Behavioral Trajectory

A **behavioral trajectory** is the observable output pattern of Σ over time on a task or environment.

Denote a trajectory as:

$$\boldsymbol{\tau = B(\Sigma, T, E)}$$

where T represents tasks or inputs and E represents the environment or context.

All properties analyzed in this framework are properties of trajectories, not of A or S in isolation. Properties only exist insofar as they manifest in behavior under well-defined conditions.

2.6 Property (P)

A **property** is any stable, measurable pattern in trajectories τ.

Examples include:

- chain-of-thought reasoning
- in-context learning
- long-horizon planning
- working-memory persistence
- meltdown/jitter under stress
- emotional tone mimicry
- energy-use signatures
- response latency patterns
- “self-reflective” behaviors

- proposed consciousness markers

Crucially:

> A property is what changes—or does not change—when A or S is transformed.

The framework classifies properties by their **invariance structure**, not by their surface description.

3. Transformations

Classification depends on measuring ΔP under four kinds of transformation, and validating that the measurement process itself is invariant.

3.1 Architecture Transformation (T^{arch})

An **architecture transformation** is a mapping that changes the computational structure while holding the substrate constant:

$$\mathbf{T^{arch} : (A_1, S, \theta_1) \rightarrow (A_2, S, \theta_2)}$$

subject to:

- S remains fixed
- A_2 represents a different architecture class or a meaningfully altered topology
- θ_2 is chosen such that A_2 is trained or configured comparably to A_1 for a fair measurement of property P

T^{arch} is used to determine whether P depends on architecture.

3.2 Substrate Transformation (T^{sub})

A substrate transformation is a mapping:

$$\mathbf{T^{sub} : (A, S_1, \theta_1) \rightarrow (A, S_2, \theta_2)}$$

such that:

1. **A's MDL-minimal generative kernel is preserved**,
   meaning the computational identity is unchanged under the transformation.
2. **θ_2 is adjusted only to the extent necessary** to instantiate that same kernel on S_2, not to retrain or meaningfully alter A's generative behavior. The criterion for "minimal adjustment" is MDL-minimal kernel equivalence: if the adjusted θ_2 produces a system whose generative kernel differs from the original, the adjustment has crossed from substrate accommodation into architectural alteration, and the transformation no longer qualifies as T^{sub}. In practice, this means

that quantization, precision reduction, or other substrate-forced parameter changes must be verified against kernel preservation before substrate-dependence claims are valid.

3. **Any behavioral differences** observed under T^{sub} must be attributable to substrate physics rather than a different kernel.

3.3 Training Transformation (T^{train})

Training is itself a substrate-embedded process. Variation in training substrate, precision, or optimization dynamics can change θ in ways that do not reflect runtime substrate effects.

Define:

$$\mathbf{T^{train} : (A, S_{tr1}, \theta_1) \rightarrow (A, S_{tr2}, \theta_2)}$$

where S_{tr} denotes the *training* substrate, not the *deployment* substrate.

A property P may therefore change due to:

- runtime substrate physics ($\mathbf{T^{sub}}$)
- training substrate dynamics ($\mathbf{T^{train}}$)
- both
- neither

3.4 Scale Transformation (T^{scale})

$$\mathbf{T^{scale} : (A, S, \theta_n) \rightarrow (A, S, \theta_{n+k})}$$

Scale must preserve the architecture's generative kernel. If scaling produces a discontinuous change in the MDL-minimal kernel—new generative structure rather than parametric expansion of existing structure—then T^{scale} has not scaled the same architecture; it has produced a distinct architecture class A'. In such cases, the transformation is reclassified as T^{arch}, not T^{scale}. The scaling laws literature documents apparent phase transitions in capability; the framework interprets these as potential kernel discontinuities requiring verification. To confirm that T^{scale} remains valid, one must demonstrate that the pre- and post-scaling systems share the same MDL-minimal generative kernel up to parametric differences. If they do not, the systems belong to different architecture classes and cross-scale comparisons become architectural comparisons, with all the methodological constraints that entails.

Scale-dependent properties are **regime-dependent architectural properties**, not identity-level invariants.

3.5 Instrumentation Invariance

Measurement must not introduce confounds.

For P to be well-defined:

- **M must be architecture-invariant**
- **M must be substrate-invariant**
- **T must not interact differently with different substrates**

If measurement fails invariance, $\Delta^{arch}(P)$ and $\Delta^{sub}(P)$ are invalid.

3.6 Invariance

A property P is **invariant under a transformation** if P remains qualitatively unchanged after applying the transformation.

Two forms:

- **Architecture-invariant**: P(Σ) persists under T^{arch}
- **Substrate-invariant**: P(Σ) persists under T^{sub}

Invariance determines whether the property belongs to architecture, substrate, their interaction, or neither.

3.7 Classification by Invariance

Every property P falls into one of four categories:

1. **Architecture-dependent**
   Changes under T^{arch}, stable under T^{sub}.
2. **Substrate-dependent**
   Stable under T^{arch}, changes under T^{sub}.
3. **Identity-level (cross-invariant)**
   Stable under both.
4. **Architecture–Substrate Interaction**
   Sensitive to both; behavior emerges only through coupling.

This classification is the analytical engine of the framework.

Classification implications

- If P changes under T^{train} but not T^{sub} → **training-dependent property**
- If P changes under T^{sub} but not T^{train} → **true substrate-dependent property**
- If P changes under both → **extended interaction property**
- If P changes under neither → unaffected by training substrate

4. Classification of Properties

Every property **P** of a system $\Sigma = (A, S, \theta)$ is defined by how it responds to two transformations:

- $\mathbf{T^{arch}}$ — varies the architecture while holding the substrate constant
- $\mathbf{T^{sub}}$ — varies the substrate while preserving the architecture's computational structure

Classification is determined entirely by **invariance** or **sensitivity** to these transformations.

Let:

- $\mathbf{\Delta^{arch}(P)}$ = change in P under T^{arch}
- $\mathbf{\Delta^{sub}(P)}$ = change in P under T^{sub}

We consider "change" at the level of qualitative trajectory structure—emergence, disappearance, or transformation of the property—not small numerical perturbation.

The four classes follow.

4.1 Architecture-Dependent Properties

A property P is **architecture-dependent** if:

- $\mathbf{\Delta^{arch}(P) \neq 0}$ (P changes meaningfully when the architecture changes)
- $\mathbf{\Delta^{sub}(P) = 0}$ (P persists across substrate changes)

Formally:

> For architectures A_1 and A_2 on the same substrate S,
> $P(\Sigma(A_1, S, \cdot)) \neq P(\Sigma(A_2, S, \cdot))$
> and for substrates S_1 and S_2 implementing the same A,
> $P(\Sigma(A, S_1, \cdot)) \approx P(\Sigma(A, S_2, \cdot))$.

Interpretation:
The property expresses the computational structure of the architecture class, largely independent of physical realization.

Examples:

- chain-of-thought depth that appears in transformer-based world models but not in shallow RNNs
- compositional reasoning architectures vs purely statistical ones

These are properties of *form*, not *physics*.

4.2 Substrate-Dependent Properties

A property P is **substrate-dependent** if:

- $\mathbf{\Delta^{arch}(P) = 0}$ (P does not meaningfully change with architecture)
- $\mathbf{\Delta^{sub}(P) \neq 0}$ (P changes when the substrate changes)

Formally:

> For substrates S_1 and S_2 implementing the same architecture A,
> $P(\Sigma(A, S_1, \cdot)) \neq P(\Sigma(A, S_2, \cdot))$,
> even when the functional graph of A is preserved across implementations.

Interpretation:
The property arises from physical timing, precision, noise, energy dynamics, or other substrate-level physics. It is not intrinsic to the computational structure.

Examples:
♦ spike-timing patterns unique to analog or neuromorphic hardware
♦ thermal or noise-induced behavioral drift
♦ latency signatures tied to hardware physics

These properties arise from *matter*, not *form*.

4.3 Cross-Invariant (Identity-Level) Properties

A property P is **identity-level** if:

- $\mathbf{\Delta^{arch}(P) = 0}$
- $\mathbf{\Delta^{sub}(P) = 0}$

Formally:

P persists under both T^{arch} and T^{sub} across all implementations within a defined equivalence class of architectures.

Interpretation:
These are the deep invariants of the system—properties of the computational identity itself, independent of both the specific architectural instantiation and the physical substrate.

This is the category that matters for theories of computational identity, functionalism, or PatternSense-style identity kernels.

Examples might include:
♦ fundamental symmetry constraints
♦ invariance of certain generative relations
♦ stable attractor structures that persist across equivalent architectures and substrates

These properties belong to *identity*, not to architecture or substrate.

4.4 Architecture–Substrate Interaction Properties

A property P is an **architecture–substrate interaction** property if P cannot be attributed solely to Δ^{arch} or Δ^{sub}.

We distinguish two forms:

Weak Coupling

P can be approximated as:

$$P \approx P_a + \varepsilon_s$$

where P_a is architecture-dependent and ε_s is a small substrate-induced perturbation.

Weak coupling indicates **partial decomposability**.

Strong Coupling

P exists **only** for specific (A_i, S_j) pairs; altering either A or S qualitatively eliminates or transforms P.

Strong coupling indicates **irreducible joint dependency**, not decomposable by architectural or substrate analysis alone. Clark's extended cognition highlights strong coupling with environments; here, we classify such dependencies by varying substrates (e.g., sensors as runtime S) to distinguish architectural from embodied properties.

4.5 Regime-Dependent Architectural Properties

If P appears only after T^{scale} crosses a threshold but remains substrate-invariant:

> P is architectural but **scale-regime dependent**, not identity-level. If, however, scaling produces a kernel discontinuity rather than regime transition, the property is not scale-dependent within a single architecture class—it marks the boundary between distinct architectures. Properties that appear to "emerge at scale" must therefore be evaluated for kernel continuity before classification. A property that emerges because scaling crossed an architectural boundary is architecture-dependent, not regime-dependent.

This captures emergence at scale without misclassifying it as a fundamental architectural constant.

5. The Classification Grid

For clarity, the four classes can be represented by a 2×2 grid:

	Invariant across architectures ($\Delta^{arch} = 0$)	**Varies across architectures** ($\Delta^{arch} \neq 0$)
Invariant across substrates ($\Delta^{sub} = 0$)	**Identity-Level**	**Architecture-Dependent**
Varies across substrates ($\Delta^{sub} \neq 0$)	**Substrate-Dependent**	**Architecture–Substrate Interaction**

This grid is the taxonomic core of the framework.
Every empirical or theoretical claim about P must specify which cell it occupies.

6. Transformation Tests

The purpose of the framework is to classify properties by invariance.
Classification requires *controlled transformations* of architecture and substrate.
This section defines the minimal procedures for doing so.

Let $\Sigma = (A, S, \theta)$ and let **P** be a property derived from trajectories of Σ.

The goal is to measure two quantities:

- $\mathbf{\Delta^{arch}(P)}$ — sensitivity of P to architectural change
- $\mathbf{\Delta^{sub}(P)}$ — sensitivity of P to substrate change

The following methods define how each is obtained.

6.1 Architectural Transformation Test (T^{arch})

Purpose

To determine whether P depends on architectural structure rather than physical realization.

Procedure

1. **Fix a substrate S_0.**
   All architectures must be run on the same physical device class.
2. **Select at least two architectures A_1 and A_2.**
   They must differ in computational structure in a substantive way—e.g.:
   ♦ transformer vs RNN
   ♦ recurrent world model vs feedforward backbone
   ♦ global workspace topology vs monolithic dense network
   ♦ modular routing vs fixed flow
3. **Train or configure each architecture comparably.**
   Same or equivalent tasks, data distributions, objectives, and evaluation settings.
4. **Measure P on each $\Sigma_i = (A_i, S_0, \theta_i)$.**
5. **Compute $\Delta^{arch}(P)$.**
   - If P changes qualitatively (appearance, disappearance, structural alteration), then $\Delta^{arch}(P) \neq 0$.
   - If P is preserved across architectures, then $\Delta^{arch}(P) = 0$.

Requirements for Validity

- Architectural differences must be genuine differences in computational form, not superficial reparameterizations.
- Training differences must not introduce confounding behavioral drift.
- Measurement must focus on qualitative trajectory structure, not minor numerical noise.

6.2 Substrate Transformation Test (T^{sub})

Purpose

To determine whether P originates from substrate physics rather than computational form.

Procedure

1. **Fix an architecture A_0.**
   The architecture and functional graph must remain constant across substrates.
2. **Select at least two substrates S_1 and S_2** that differ physically, such as:
   ♦ float32 GPU vs low-precision ASIC
   ♦ digital hardware vs mixed-signal analog
   ♦ deterministic simulation vs neuromorphic spiking implementation
   ♦ high-noise vs low-noise device classes
3. **Implement A_0 on each substrate.**
   Preserve the architecture's graph and functional specification as closely as physically possible.
4. **Adjust θ minimally if required** to make the implementation valid on each substrate, but do not retrain unless unavoidable.
5. **Measure P on $\Sigma_1 = (A_0, S_1, \theta_1)$ and $\Sigma_2 = (A_0, S_2, \theta_2)$.**
6. **Compute $\Delta^{sub}(P)$.**
   - If P changes qualitatively across substrates, $\boldsymbol{\Delta^{sub}(P) \neq 0}$.
   - If P persists, $\boldsymbol{\Delta^{sub}(P) = 0}$.

Requirements for Validity

- Substrates must differ in real physical properties (precision, timing, noise), not merely brand or vendor.
- The architecture must be held fixed up to implementation isomorphism; differences must not result from altered computation.
- Observed changes must be attributable to substrate physics, not reconfiguration or retraining artifacts.

6.3. Idealization Test

The idealization test determines whether a property is tied to physical constraints or survives in an abstract computational limit.

Procedure

1. **Idealize the substrate.**
   Simulate A with:
   ♦ infinite precision
   ♦ noiseless operations
   ♦ deterministic timing
   ♦ no energy or thermal constraints
2. **Idealize the architecture only if necessary** (e.g., remove auxiliary routing noise or stochastic components).
3. **Measure P under idealization** and compare to real-world measurements.

Interpretation

- If P disappears under idealization,
  → the property is tied to substrate physics.
- If P persists,
  → P is computational or architectural in origin.

Idealization functions as a limiting case of T^{sub}: substrate constraints are progressively removed until only the computational structure remains. It is not a separate transformation class but rather T^{sub} taken to the limit $S \rightarrow S\varnothing$, where $S\varnothing$ denotes an idealized null substrate with no physical distortion. Properties that vanish under idealization are substrate-dependent by definition; properties that persist are candidates for architectural or identity-level classification. This interpretation keeps idealization within the transformation framework rather than treating it as a categorically distinct probe.

6.4. Scaling Test

Scaling tests reveal whether a property is stable under model size or data scale changes when architecture and substrate remain fixed.

Procedure

1. Fix (A_0, S_0).
2. Train models of increasing scale while keeping tasks equivalent.
3. Measure P as scale increases.

Interpretation

- If P appears only at certain scales, architectural classification must account for scale regimes.
- If P disappears at higher scale, it is likely not an identity-level property.

Scaling tests do not classify P directly, but they constrain the interpretation of Δ^{arch} and Δ^{sub}.

6.5 Combined Test: Joint Perturbation

Some properties emerge only when architecture and substrate interact.
The combined test detects these A×S properties.

Procedure

1. Vary both A and S in a controlled grid:
   (A_1, S_1), (A_1, S_2), (A_2, S_1), (A_2, S_2).
2. Measure P across all four implementations.

Interpretation

- If P appears only for (A_i, S_j) pairs and not for marginal changes in A or S alone,
  P belongs to the **interaction** class.

This test detects the "coupling" category that neither architecture-only nor substrate-only tests can reveal.

6.6 Decision Procedure

A property P is classified by applying the tests in sequence:

1. **Run T^{arch}.**
   If $\Delta^{arch}(P) \neq 0 \rightarrow$ P is at least architecture-dependent.
2. **Run T^{sub}.**
   If $\Delta^{sub}(P) \neq 0 \rightarrow$ P involves substrate dependence.
3. **Check the 2×2 grid.**
   - $\Delta^{arch} \neq 0$, $\Delta^{sub} = 0 \rightarrow$ Architecture-Dependent
   - $\Delta^{arch} = 0$, $\Delta^{sub} \neq 0 \rightarrow$ Substrate-Dependent
   - $\Delta^{arch} = 0$, $\Delta^{sub} = 0 \rightarrow$ Identity-Level
   - $\Delta^{arch} \neq 0$, $\Delta^{sub} \neq 0 \rightarrow$ Architecture–Substrate Interaction
4. **Use Idealization + Scaling** to resolve ambiguous cases.

7. Discussion and Implications

The Architecture vs Substrate Framework is not a taxonomy for its own sake.
Its value comes from forcing precision into claims that are otherwise untestable or confused.

This section outlines the implications for analysis, research design, and high-stakes discourse around artificial cognition.

7.1 Separating Form from Physics

Most debates in AI fail because they treat architecture and substrate as interchangeable sources of behavior.
The framework dissolves that ambiguity:

- If a property changes under T^{arch} but not under T^{sub}, it belongs to the computational form.
- If it changes under T^{sub} but not under T^{arch}, it arises from substrate physics.
- If it changes under both, it is a coupling phenomenon.
- If it changes under neither, it is an identity-level invariant.

This decomposition puts structure where there was previously speculation.
It prevents category errors such as interpreting hardware-induced effects as emergent cognition, or architectural artifacts as evidence of embodied intelligence.

7.2 Eliminating Linguistic Ambiguity

Terms such as *reasoning*, *planning*, *memory*, *awareness*, or *self-reflection* gain meaning only after classification.
Before classification, they are ambiguous labels for behaviors whose causal origins are unknown.

Using the framework:

- An architecture-dependent form of “reasoning” is simply a computational strategy characteristic of that class of models.
- A substrate-dependent form of “awareness” is not awareness at all, but a property of physical device dynamics.
- An identity-level invariant is the only candidate for a property that can support claims about persistent computational identity.

This forces a shift from rhetoric to structural description.

7.3.Controlling for Confounds in AI Research

The framework establishes methodological discipline.
It highlights three recurring confounds in experimental work:

1. **Architecture–training confound:**
   Comparing different architectures trained with different data regimes produces misleading inferences about P.
2. **Architecture–substrate confound:**
   Implementing architectures on different hardware without controlling substrate effects leads to false attribution of causality.
3. **Substrate–numerics confound:**
   Changing hardware precision and treating resulting behavioral drift as an emergent property of the model is a categorical error.

The transformation tests provide a minimal protocol for eliminating these confounds.

7.4 Reframing "Emergent" Properties

The term "emergence" loses diagnostic value if it is used for every property that appears surprising or novel.
With the framework, emergence becomes:

> A change in P under a specific transformation (Δ^{arch} or Δ^{sub}) that is not predicted by the prior invariance structure of the system.

This definition makes emergence *locatable*.
It becomes clear whether the source of emergence is:

- architectural complexity
- substrate physics
- or their interaction

The term stops being a placeholder and becomes a pointer to a specific invariance relation.

7.5 Constraints on Claims About "Understanding" or "Reasoning"

Any claim that an AI system exhibits "understanding," "theory of mind," "introspection," or similar cognitive descriptors must specify:

1. Whether the property is expected to be architecture-invariant
2. Whether it is expected to be substrate-invariant
3. Whether it survives idealization

Without these specifications, the claim is incomplete.

For instance:

- If a model shows “self-referential behavior,” but the behavior vanishes when the architecture changes, the property is architectural—not cognitive.
- If the behavior appears only on analog substrates with specific noise characteristics, it is substrate-dependent—not evidence of a higher-order process.
- If the property survives both transformations, it becomes a legitimate candidate for an identity-level invariant.

This is the only way to prevent the attribution of cognitive significance to what are, in fact, mere implementation artifacts.

7.6 Implications for Consciousness Discourse

The framework does not claim what consciousness is.
But it establishes the minimum bar for any serious claim:

> A proposed consciousness marker must specify its invariance profile under T^{arch} and T^{sub}.

If a marker:

- fails invariance under architectural changes,
  it is an artifact of computational form.
- fails invariance under substrate changes,
  it is an artifact of physical realization.
- is sensitive to both,
  it is an interaction effect with no coherent interpretation.
- survives both,
  only then does it describe something that could plausibly map to an identity-level property.

This removes most current claims from consideration without further argument.

7.7 Clarifying the Boundary Between Biological and Artificial Systems

The framework makes it possible to compare biological and artificial systems without collapsing categories:

- Biological neurons and silicon gates both serve as substrates.
- Neural topology or transformer topology serve as architectures.
- Behavioral trajectories determine observable properties.

Once the mapping is made explicit, debates about “biological uniqueness,” “hardware embodiment,” or “computational equivalence” can be conducted cleanly, without mixing substrate-level phenomena (ion-channel kinetics, spike-timing jitter) with architecture-level phenomena (recurrence, modularity, workspace dynamics).

7.8 Limits of the Framework

The Architecture vs Substrate Framework is intentionally narrow:

- It does not explain how properties arise.
- It does not assign meaning or value to behavioral phenomena.
- It does not address ethics, alignment, or teleology.
- It does not predict emergent abilities.

Its sole purpose is to impose category discipline on claims about system properties.

This narrowness is a strength: it prevents the framework from smuggling interpretive assumptions into the analysis.

7.9 Consequence: Most AI Debates Become Tractable

By forcing every disputed claim into one of four cells, the framework transforms vague debates into solvable classification problems.

Instead of arguing abstractly about what a property “really is,” one asks:

- Does it change when I vary A?
- Does it change when I vary S?
- Does it survive idealization?
- Does it require coupling?
- Which cell does it occupy?

The result is a discourse grounded in operational distinctions rather than metaphorical intuition.

8. Conclusion

The Architecture vs Substrate Framework provides a minimal structure for distinguishing the sources of properties observed in artificial systems. By separating computational form from physical realization and treating observable properties as trajectory-level phenomena, the framework imposes invariance-based classification as a prerequisite for any coherent claim about system behavior. Architectural transformations isolate computational contributions; substrate transformations isolate physical contributions; and the combined grid resolves interaction effects and identity-level invariants.

The framework makes no assumptions about cognition, agency, or consciousness. It does not prescribe how systems should be built or interpreted. Its function is narrower and more fundamental: to eliminate categorical ambiguity. Once properties are classified by their sensitivity or invariance to changes in architecture and substrate, the surrounding discourse becomes structurally constrained. Arguments gain clarity. Experimental design becomes disciplined. Claims become falsifiable.

In a field saturated with speculative interpretations and loosely defined terminology, this distinction is not optional. It is the boundary condition for meaningful analysis. The framework stands as a simple requirement: before one can make assertions about what an artificial system *is*, one must first establish what its properties *depend on*.

References

Brooks, R. A. (1991). *Intelligence without representation*. *Artificial Intelligence, 47*(1–3), 139–159.

Chalmers, D. J. (2011). A computational foundation for the study of cognition. Journal of Cognitive Science, 12(4), 323-357.

Churchland, P. S., & Sejnowski, T. J. (1992). *The computational brain*. MIT Press.

Chua, L. O., & Yang, L. (1988). Cellular neural networks: Theory. *IEEE Transactions on Circuits and Systems, 35*(10), 1257–1272.

Clark, A. (2008). Supersizing the mind: Embodiment, action, and cognitive extension. Oxford University Press.

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience, 11*(2), 127–138.

Haugeland, J. (1985). *Artificial intelligence: The very idea*. MIT Press.

Indiveri, G., & Liu, S.-C. (2015). Memory and information processing in neuromorphic systems. *Proceedings of the IEEE, 103*(8), 1379–1397.

Ladyman, J., & Ross, D. (2007). *Every thing must go: Metaphysics naturalized*. Oxford University Press.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks, 10*(9), 1659–1671.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. MIT Press.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM, 19*(3), 113–126.

Pearl, J. (2009). *Causality: Models, reasoning, and inference* (2nd ed.). Cambridge University Press.

Pfeifer, R., & Scheier, C. (1999). *Understanding intelligence*. MIT Press.

Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford University Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Thomas, C. S. (2025). PatternSense - A Computational Theory of Structural Identity. Zenodo. https://doi.org/10.5281/zenodo.17716891

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Appendix A: Extensions to Non-AI Domains

(Substrate Independence Beyond Artificial Systems)

PatternSense and the Architecture–Substrate Framework generalize to any system whose behaviors arise from an abstract form instantiated in a physical medium. The triplet $\Sigma = (A, S, \theta)$ remains meaningful whether the substrate is silicon, neural tissue, a social institution, or an ecological network. The purpose of this appendix is to show how invariance tests extend naturally into biological, social, and evolutionary domains, clarifying when observed properties belong to computational/organizational form (A), physical substrate (S), adaptive state (θ), or their interaction.

A.1 Biological Systems

Mapping Σ onto Biology

In neural systems, the architectural element A corresponds to the computational motifs and circuit topologies that define transformation structure: recurrent loops, inhibitory–excitatory balance, predictive coding arrangements, and generative kernels implicit in cortical microcircuits. Substrate S corresponds to the biological instantiation: ion-channel dynamics, dendritic morphology, membrane time constants, neuromodulatory chemistry.

θ represents the continuously updated synaptic weights, plasticity traces, and metabolic state variables configuring the circuit.

Applying Invariance Tests

The biological analogs of T^{arch} and T^{sub} can be instantiated through:

♦ T^{arch} (Architectural Variation):
Compare homologous motifs across species or brain regions (e.g., cerebellar microcircuit vs. cortical column) where substrate biophysics remain similar but circuit structure differs. If a behavioral property persists, it suggests substrate-level or identity-level invariance.

♦ T^{sub} (Substrate Variation):
Consider computational models of the same circuit motif implemented in (1) real tissue, (2) simulated spiking models, (3) rate-based approximations, and (4) neuromorphic silicon. Using MDL-minimal kernel equivalence to normalize A, differences in behavior that survive architectural preservation can be attributed to S—ion-channel noise, spike-timing regimes, metabolic constraints.

♦ Idealization Limit:
Simulations with eliminated noise and infinite precision allow separation of architectural invariants from biological physics, directly parallel to idealized substrate limits in the main framework.

Interpretation

• If robustness to noise, predictive stability, or attractor structure persists across spiking, rate-based, and digital models → property belongs to A or θ, not wetware.
• If properties such as oscillation coherence, spike-timing codes, or spontaneous criticality vanish under idealization or digital implementation → property is substrate-dependent, not computational.

This aligns with Friston's free-energy principle: the generative model (A) is substrate-independent, while the particular physical embodiment (S) shapes energetic and timing constraints that may modulate behavior without defining it.

A.2 Social and Ecological Systems

Mapping Σ onto Institutions

In social/institutional settings:

♦ **A** = relational and procedural structure—hierarchies, information channels, feedback loops, role topology, codified norms.
♦ **S** = the physical and human substrate—actual personnel, technological infrastructure, physical resources, and communication media.
♦ **θ** = the lived state: cultural priors, shared memory, accumulated commitments, tacit routines.

This mapping allows institutional analysis to be recast in the language of computational identity: organizations implement abstract coordination kernels on a human substrate.

Invariance Tests

- **T^{arch} (Architectural changes):**
  Modify reporting structures, decision rights, workflow topology, or feedback loops while keeping personnel and tools constant. If system-level phenomena (e.g., information bottlenecks, conflict patterns, drift toward centralization) change, the property is architectural.

- **T^{sub} (Substrate changes):**
  Hold institutional topology constant but vary personnel, infrastructure, or technological medium. If the "behavior" (stability, resilience, coordination failure modes) is preserved across substrate turnover, the property is architectural or identity-level.

Turnover is the social-world analogue of cross-substrate invariance tests: Does the organization's generative kernel survive the replacement of its physical substrate?

Interpretation

- Properties like **mission drift**, **cycle formation**, or **bureaucratic expansion** often persist across personnel changes: $\Delta^{sub}(P) \approx 0 \rightarrow$ architectural.
- Properties like **loss of tacit knowledge**, **fragility under staff loss**, or **dependence on charismatic individuals** indicate substrate dependence.

This provides a principled way to classify institutional behaviors without drifting into anthropomorphic explanation or purely sociological narrative.

A.3 Evolutionary Dynamics

Evolution provides a natural domain for PatternSense because adaptive traits manifest as invariants across changes in substrate and environment.

Mapping Σ onto Evolving Populations

- **A =** the generative architecture of the organism's developmental and regulatory system—gene regulatory networks, body-plan symmetries, developmental constraints.
- **S =** the biochemical substrate: DNA chemistry, cellular machinery, molecular fidelity regimes, environmental nutrient constraints.
- **θ =** the population-level distribution of alleles, epigenetic states, and regulatory parameters acquired through evolutionary history.

Applying Invariance

Evolution routinely conducts substrate-variation experiments:

- RNA → DNA (substrate transformation with kernel preservation).
- Bacterial gene circuits implemented in synthetic chassis (architecture preserved; substrate altered).
- Gene regulatory network motifs reused across phyla despite wildly different cellular substrates.

These naturally occurring transformations allow classification:

- **Cross-invariant traits** (e.g., modularity, symmetry-breaking, feedback regulation) persist across substrate variation → identity-level generative structures.
- **Substrate-dependent traits** (e.g., speed of enzymatic pathways, thermal stability, error rates) change with biochemical medium.

- **Interaction-level traits** appear when architectural and substrate constraints combine nonlinearly (e.g., metabolic oscillators requiring both specific topology and specific biochemistry).

PatternSense therefore provides a structural vocabulary for distinguishing genuinely universal adaptive invariants from substrate-bound features of particular lineages.

A.4 Synthesis

Across biological, social, and evolutionary domains, the same principle holds:

Substrate independence is not assumed; it is classified.
Only properties surviving both architectural variation and substrate variation qualify as identity-level invariants. This keeps PatternSense rooted in explicit invariance structure rather than metaphor or analogy.

By extending $\Sigma = (A, S, \theta)$ beyond artificial systems, we obtain a unified method:

- identify the generative kernel (A),
- identify the physical instantiation (S),
- identify the adaptive state (θ),
- and classify properties by their invariance.

This appendix closes the conceptual loop: substrate independence is not a special feature of AI systems—it's a general structural test for any system whose behaviors arise from the coupling of form, matter, and history.