

P \neq NP from Operational Gradients

A Process-Primacy Approach to Computational Complexity

David D. Zelenka

December 6, 2025

Abstract

We present a novel approach to the P vs NP problem grounded in **operational geometry**: a framework where processes (operations) are ontologically prior to objects. We argue that computational complexity classes are distinguished by **intrinsic operational gradients**—directional asymmetries in the operational substrate itself that cannot be eliminated by algorithmic cleverness.

We introduce the **Axiom of Intrinsic Operational Gradient**, which posits that operational space has canonical directionality encoded in the threading cost functional I_τ . Under this axiom, together with process-primacy and the free traced symmetric monoidal category (TSMC) structure, we prove that $P \neq NP$.

Our approach reframes P vs NP as an **ontological question**: Does mathematical reality have intrinsic operational gradients? We provide evidence from category theory (freeness of the operational category), information theory (Shannon bounds on distinguishability), and physical analogies (thermodynamic and causal gradients). The paper thus establishes a **conditional equivalence**: the existence of intrinsic operational gradients is equivalent to $P \neq NP$.

Our main contributions are:

- Philosophical and category-theoretic foundation for process-primacy
- Formalization of the Axiom of Intrinsic Operational Gradient
- Rigorous proof that operational substrate is acyclic with exponential potential
- Conservation law relating threading cost to potential difference
- Lower bounds on Divide operations for NP problems
- Main theorem: operational gradient axiom implies $P \neq NP$

This framework clarifies why the intuition “search is harder than verification” reflects a deep structural principle about reality itself, and suggests that $P \neq NP$ may be a law of nature analogous to the second law of thermodynamics.

A Process-Primacy Approach to Computational Complexity

1 Introduction

1.1 The Standard Approach to P vs NP

The P vs NP problem asks whether every problem whose solution can be **verified** in polynomial time can also be **solved** in polynomial time [1, 2].

Formally:

Definition 1.1 (Complexity Classes P and NP).

- **P**: The class of decision problems solvable by a deterministic Turing machine in time polynomial in the input size.

- **NP:** The class of decision problems for which a proposed solution can be verified by a deterministic Turing machine in polynomial time.

Clearly $P \subseteq NP$ (if you can solve efficiently, you can verify efficiently). The question is whether $P = NP$ or $P \neq NP$.

Intuition for $P \neq NP$: Finding a solution requires exploring exponentially many possibilities, while verifying a given solution only requires checking a single path. Going "upstream" against the branching structure seems fundamentally harder than going "downstream" with a guide. Why this intuition hasn't yielded a proof: Standard complexity theory treats algorithms as abstract Turing machines operating on pre-existing objects (strings, graphs). The intuitive "gradient" is not formalized—it's just a metaphor. One cannot rule out that some clever non-obvious algorithm exploits hidden structure to achieve polynomial time.

1.2 The Operational Geometry Approach

We propose a radically different foundation: **Operational Geometry**, where:

- **Operations are primitive:** The fundamental entities are processes (Thread, Divide, Rotate, Scale, Iterate), not objects [3].
- **Objects emerge from operations:** Mathematical objects are equivalence classes of operational sequences under invariant-preserving equivalence.
- **Gradients are intrinsic:** Every operational sequence has an intrinsic **threading cost** $I_\tau \sigma$, representing the operational depth or complexity of the process.

In this framework, the "gradient" metaphor becomes **literal**: operational sequences have intrinsic directionality encoded in their threading parameter τ , and this directionality cannot be eliminated by reformulation.

1.3 Main Thesis

Central Claim: If process-primacy holds (operations are ontologically prior to objects), then $P \neq NP$ follows from the irreducibility of operational gradients.

Specifically:

- Verification and search are both operational sequences.
- For NP-complete problems, search sequences have systematically higher threading cost I_τ than verification sequences.
- This gradient is **irreducible**: no operational sequence can eliminate it, because the gradient is a property of the operational substrate, not of any particular algorithm.

Therefore, no polynomial-time algorithm exists for NP-complete problems, implying $P \neq NP$.

1.4 Why This Approach Is Different

- **Standard complexity theory:** Tries to prove "no algorithm can solve NP-complete problems in polynomial time." Must enumerate all possible algorithms (impossible). Gets stuck because you cannot rule out unknown algorithms.
- **Our approach:** Demonstrates "the operational substrate forbids polynomial solutions." Does not enumerate algorithms—shows that the substrate structure forbids them. Works because acyclicity + exponential branching = irreducible gradient.

Remark 1.1 (Key Difference). The standard approach is *algorithmic*: it tries to prove a negative statement about all possible algorithms. Our approach is *ontological*: it proves a positive statement about the structure of operational reality. This is fundamentally different because:

1. We do not need to enumerate algorithms—we characterize the substrate they operate on.
2. We do not need to rule out unknown algorithms—we show the substrate makes them costly.
3. We do not rely on clever proof techniques—we rely on structural properties of the free TSMC.

2 Operational Geometry: Key Definitions

2.1 Primitive Operations

Definition 2.1 (Generating Operations). The set of generating morphisms M consists of:

- Thread_τ : Advance by parameter τ (one-dimensional threading)
- Divide_n : Partition into n parts (branching)
- Rotate_θ : Rotate by angle θ (phase/angular action)
- Scale_λ : Scale by factor λ (metric/intensity adjustment)
- Iterate_k : k -fold composition of operation f (repetition)

2.2 Operational Sequences

Definition 2.2 (Operational Sequence). An operational sequence is a finite or countably infinite composition:

$$\sigma = f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1$$

where each $f_i \in M$. The set of all operational sequences is denoted M (finite) or \mathcal{M}^ω (including limits).

2.3 Threading Invariant

Definition 2.3 (Threading Cost Invariant). The threading invariant $I_\tau : \mathcal{M}^\omega \rightarrow \mathbb{R}_{\geq 0}$ assigns to each operational sequence its total operational cost:

$$I_\tau \sigma = \sum_{i=1}^n w_{f_i}$$

where $w : M \rightarrow \mathbb{R}_{\geq 0}$ assigns operational weight to each primitive operation. For limiting sequences:

$$I_\tau \sigma = \lim_{n \rightarrow \infty} I_\tau \sigma_n$$

when the limit exists.

Interpretation: The threading cost $I_\tau \sigma$ measures the **operational depth** of the sequence: how far the process must "thread" through operational space to produce its output. This is analogous to:

- Time complexity in standard complexity theory (number of steps)
- Action in physics (integral of Lagrangian along path)
- Entropy production in thermodynamics (irreversible work)
- Gravity recently reformulated as gradient flow [4]

2.4 Operational Equivalence and Objects

Definition 2.4 (Operational Equivalence). Two operational sequences σ_1, σ_2 are operationally equivalent, written $\sigma_1 \sim \sigma_2$, if they produce the same output object and agree on all primitive invariants.

Definition 2.5 (Objects as Equivalence Classes). The set of objects \mathcal{O} is:

$$\mathcal{O} = \mathcal{M}^\omega / \sim$$

Objects are what operational sequences produce, not what they operate on.

Remark 2.1 (Ontological Inversion). This definition represents a fundamental ontological inversion: instead of treating objects as primitive entities that operations act upon, we treat objects as *outputs* of operations. An object has no independent existence—it is defined entirely by the equivalence class of operational sequences that produce it.

This is the essence of process-primacy: **objects are invariants of processes**. Two different operational sequences that produce indistinguishable outputs are considered equivalent, and the object itself is nothing more than the equivalence class of all such sequences.

3 Process-Primacy: Foundations and Justification

3.1 Motivation from Constructive Mathematics

Constructive mathematics provides the first and most direct motivation for process-primacy. In the constructive tradition (Brouwer, Bishop, Martin-Löf), a mathematical object is not an independently existing Platonic entity; rather, it **is** the outcome of a construction procedure with specified steps.

- A real number is a process that generates a convergent sequence.
- A function is a rule that transforms inputs into outputs. A proof is a finite sequence of inference steps.

Formally, constructive type theory identifies:

$$\text{object} \equiv \text{equivalence class of constructions}$$

In this view, objects cannot be divorced from the operations that bring them about. Process-primacy in operational geometry generalizes this constructive stance by taking **operational sequences** as primitive, and by treating objects as "operational fixed points"—stable patterns produced by repeatable processes.

3.2 Motivation from Category Theory

Category theory further strengthens the case. In categorical foundations, objects are determined by their morphisms:

$$A \simeq B \text{ if and only if there exists an isomorphism } A \leftrightarrow B$$

The internal structure of objects is not taken as primitive; what matters are the **maps between them**. This is most pronounced in:

- Topos theory: where logical structure emerges from morphisms.
- Operad theory: where objects are composites of operations.

- Traced monoidal categories: where feedback loops and iteration are explicit.

Operational geometry extends these ideas by giving a specific generative operad M whose morphisms **are** the primitive operations (Thread, Divide, Rotate, etc.). The "objects" of the category do not carry primitive information—they are equivalence classes of morphism compositions. Thus category theory naturally supports the ontological inversion:

process before object, morphism before point

3.3 Motivation from Physics

Physical ontology provides another line of support. Modern physics increasingly treats "objects" (particles, fields, spacetime points) as emergent from more fundamental dynamical processes:

- In quantum field theory, particles are excitations of underlying fields: they are processes in disguise.
- In general relativity, curvature is not an object but a relational property of dynamical geometry.
- In statistical mechanics, macrostates emerge from operational interactions among microstates.
- In quantum information, states are fully characterized by the operations (channels, gates) that can prepare, transform, or distinguish them.

This aligns precisely with the operational geometry slogan:

Objects are invariants of processes.

3.4 Avoiding Object-Primacy Paradoxes

Object-primacy has a long history of generating mathematical and philosophical paradoxes:

- Set-theoretic paradoxes (Russell, Burali-Forti) arise from treating sets as primitive objects independent of how they are generated.
- Zeno-type paradoxes arise from treating motion as a sequence of completed positions (objects), instead of a continuous process.
- Non-measurable sets in Banach–Tarski arise only when arbitrary subsets are treated as primitive objects rather than outcomes of constructive operations.

Under operational geometry, these paradoxes dissolve because pathological "objects" cannot be generated by any legal sequence of primitive operations. If there is no operational procedure producing an entity, then the "entity" does not exist in the theory. Thus process-primacy offers a resolution to classic mathematical and conceptual problems: ontological minimalism eliminates pathological constructions.

3.5 Category-Theoretic Justification

Category theory provides a natural mathematical foundation for the process-primacy axiom [5, 6]. In categorical ontology, **morphisms** are fundamental and objects are defined only through the arrows that relate them. Operational geometry extends this paradigm by treating primitive operations (Thread, Divide, Rotate, Scale) as generators of a freely constructed monoidal category. Objects have no intrinsic content; they are positions in the web of composable morphisms.

The Operational Category: Let \mathcal{M} denote the operational category generated by the primitive operations:

Thread, Divide, Rotate, Scale

Formally, \mathcal{M} is the **free traced symmetric monoidal category** generated by these operations:

$$\mathcal{M} = \text{FreeTSMC}(\text{Thread}, \text{Divide}, \text{Rotate}, \text{Scale})$$

Objects as process-bound positions: In such a free category, an "object" is not a primitive ontological entity. It is a label required for domain and codomain bookkeeping of morphisms. Operationally, an object A is nothing more than the type boundary between composable sequences of operations; it does not contain independent structure. Thus:

information about A is exhausted by the morphisms in and out of A

Morphisms as generators of structure: The structure of \mathcal{M} is determined entirely by its morphisms:

- Thread introduces coherence by generating a one-dimensional threadlike extension.
- Divide produces a branching comonoid structure, giving access to combinatorial explosion.
- Rotate implements a $U(1)$ -action, encoding phase-like or angular relations.
- Scale introduces a positive scalar action corresponding to metric or intensity adjustments.

Because \mathcal{M} is freely generated, no additional relations exist beyond those imposed by the axioms of traced symmetric monoidal categories. Consequently, the operational sequences themselves determine all higher-level features of the geometry.

Traced monoidal structure and feedback: Tracing is essential for modeling iterative or self-referential processes. In a traced symmetric monoidal category, a morphism

$$f : A \otimes X \rightarrow B \otimes X$$

can be "fed back" along X , producing a morphism

$$\text{Tr}_X f : A \rightarrow B$$

This captures operational phenomena such as recursion, repeated refinement, and fixed-point formation. In OpGeom, physical coherence and algorithmic iteration both emerge from such traced feedback. Objects that appear stable correspond to fixed points of these traced maps. This formalizes the idea that:

objects are invariants of repeated operational processes

Universal property and ontological minimalism: The universal property of a free traced symmetric monoidal category implies that:

- Any structure present in operational geometry must arise from the generators and the monoidal axioms—nothing else may be assumed.

Therefore:

- No "object data" independent of morphisms may be introduced.
- All geometric or informational structure must be reachable through composition, tensor product, and tracing of primitive operations.
- Any entity not constructible from the generators by these rules is not part of the ontology.

This is precisely the content of the process-primacy axiom, recast as a categorical universal constraint.

3.6 Formal Statement of Process-Primacy

Axiom 3.1 (Process-Primacy).

- The primitive elements of mathematical reality are operational morphisms $f \in M$.
- An **object** is an equivalence class of operational sequences $\sigma \in \mathcal{M}^\omega$ under the operational equivalence relation \sim .
- An **algorithm** is an operational sequence.
- No entity may be assumed to exist unless it arises as the output of some operational sequence.

This axiom is not a metaphysical assertion but a structural commitment: it states that the theory cannot refer to objects except through the processes that produce them. In this sense, process-primacy is a **conservation rule** governing the ontology.

Consequences for complexity theory: Process-primacy has three immediate consequences relevant to P vs NP:

- Algorithms = operational sequences: This converts complexity theory directly into geometry on \mathcal{M}^ω .
- Outputs inherit operational cost: If an object requires many Divide or Thread operations to produce, then this cost is intrinsic and cannot be shortcut by a different algorithm.
- Gradients become structural: Operational gradients are not metaphors: they describe asymmetries in the generative substrate.

A process that climbs against an n -ary branching structure must pay a cost proportional to the structure itself. Because outputs are not free-standing entities but **records of the operational work that produced them**, any algorithm that produces a solution to an NP-complete problem must pay the full operational cost that the problem structure demands. There is no object-primacy loophole through which a clever algorithm can bypass the underlying generative structure. In short: If objects inherit the operational cost of their generation, then no algorithm can compress an intrinsically exponential branching structure into polynomial effort. This is the essence of the operational-gradient argument for $P \neq NP$.

3.7 Axiom of Intrinsic Operational Gradient

The following axiom makes explicit the structural assumption underlying our approach to P vs NP. Rather than claiming to prove $P \neq NP$ unconditionally, we show that it follows from a fundamental principle about the nature of operational space.

Axiom 3.2 (Intrinsic Operational Gradient). The threading cost functional

$$I_\tau : \mathcal{M}^\omega \rightarrow \mathbb{R}_{\geq 0}$$

induces an intrinsic gradient structure on operational space satisfying:

1. **Directional Existence:** For every operational sequence σ and every primitive extension $f \in \mathcal{M}$, the directional derivative exists:

$$\nabla I_\tau(\sigma; f) = \lim_{\epsilon \rightarrow 0^+} \frac{I_\tau(\sigma \circ f^\epsilon) - I_\tau(\sigma)}{\epsilon}$$

where f^ϵ denotes a parametrized family of operations approaching f .

2. **Non-Degeneracy:** The gradient is non-zero except at operational fixed points:

$$\|\nabla I_\tau(\sigma)\| > 0 \quad \text{whenever } \sigma \text{ is not an identity or equivalence-class fixed point}$$

This ensures there are no “flat directions” in operational space except trivial symmetries.

3. **Monotonicity Under Divide:** For any Divide operation Divide_k :

$$\nabla I_\tau(\sigma; \text{Divide}_k) \geq \log_2 k$$

This encodes the information-theoretic cost of branching: creating k branches requires at least $\log_2 k$ bits of information.

4. **Irreducibility:** No operational sequence can reduce the gradient accumulated through Divide operations:

$$I_\tau(\sigma_2 \circ \text{Divide}_k \circ \sigma_1) \geq I_\tau(\sigma_2 \circ \sigma_1) + \log_2 k$$

for any sequences σ_1, σ_2 not containing Divide operations.

5. **Canonical Directionality:** The gradient field ∇I_τ is intrinsic to the operational substrate, independent of:

- Choice of coordinates or representation
- Algorithmic strategy or implementation
- Physical substrate or computational model

The gradient is a **structural property of operational space itself**.

Remark 3.1 (Interpretation). This axiom formalizes the intuition that:

Operational space has intrinsic directionality. Some processes are fundamentally “up-hill” (require work), others are “downhill” (release work). This directionality cannot be eliminated by clever algorithm design—it is built into the fabric of operational reality.

Remark 3.2 (Relation to Physical Laws). The Intrinsic Operational Gradient Axiom parallels fundamental physical principles:

- **Thermodynamics:** Entropy gradient (second law)
- **Relativity:** Causal gradient (no faster-than-light travel)
- **Quantum mechanics:** Information gradient (no-cloning theorem)
- **Complexity theory:** Operational gradient (no polynomial NP solver)

This suggests the gradient axiom might be a **law of nature**, not merely a mathematical assumption.

Remark 3.3 (Testable Predictions). The gradient axiom makes predictions that could potentially be tested or falsified:

- No algorithm can solve NP-complete problems in polynomial time
- Quantum computers face the same gradient constraints
- Physical processes cannot “compute” NP solutions efficiently
- The gradient should be measurable in physical implementations of computation

4 Computational Processes as Operational Sequences

4.1 Encoding Turing Machines

Definition 4.1 (Computational Operational Sequence). A computational operational sequence σ_{comp} is an operational sequence that encodes the execution trace of a Turing machine:

- Thread: Advance tape head / increment step counter
- Divide: Branch on conditional / explore search tree node
- Rotate: Permute state / rearrange data structure
- Scale: Modify values / update variables
- Iterate: Loop / recursive call

Step 1: Representing TM states. Let a TM be defined by a tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

where Q is the set of states, Σ the input alphabet, Γ the tape alphabet, and δ the transition function. We map each TM state $q \in Q$ to an operational state σ_q in \mathcal{M} .

Step 2: Encoding TM tape positions. The TM tape head position is represented as a Thread parameter in the operational substrate. Moving the tape head corresponds to applying a Thread operation with the appropriate parameter shift.

Step 3: Encoding TM transitions. Each TM transition

$$\delta(q, a) = (q', b, D)$$

where $D \in \{\text{Left}, \text{Right}\}$, is encoded as a composition of operations:

$$\sigma_q \xrightarrow{\text{Write } b} \sigma_{q'} \xrightarrow{\text{Thread}(D)} \sigma_{q''}$$

Writing to the tape corresponds to a combination of Rotate and Scale that modifies the operational state to encode the tape symbol. Moving the head is implemented via Thread.

Step 4: Encoding nondeterministic branching. If the TM is nondeterministic and has multiple possible transitions from q, a , we encode this as Divide operations:

$$\sigma_q \xrightarrow{\text{Divide}} \{\sigma_{q'_1}, \sigma_{q'_2}, \dots\}$$

This faithfully represents all possible computational paths.

Step 5: Threading cost corresponds to time complexity. Each TM step corresponds to a finite sequence of operations in \mathcal{M} with bounded cost. Therefore, the total threading cost $I_\tau \sigma_{\text{comp}}$ is proportional to the TM's runtime:

$$I_\tau \sigma_{\text{comp}} = O(\text{Time}_M(\text{input}))$$

Proposition 4.1 (Turing Completeness of Operational Sequences). *Every Turing machine computation can be encoded as an operational sequence with threading cost proportional to time complexity, and conversely, every operational sequence corresponds to a computable function.*

4.2 Search vs Verification as Operational Sequences

Consider an NP-complete problem like SAT (Boolean satisfiability).

Example 4.1 (SAT Search Operational Sequence). To search for a satisfying assignment to a formula with n variables:

$$\begin{aligned}\sigma_{\text{search}} &= \text{Divide}_2^n \circ \text{Iterate}_{2^n}(\text{check}) \\ I_\tau(\sigma_{\text{search}}) &= n \cdot w_{\text{Divide}} + 2^n \cdot w_{\text{check}}\end{aligned}$$

The threading cost scales **exponentially** with n .

Example 4.2 (SAT Verification Operational Sequence). To verify a proposed satisfying assignment:

$$\begin{aligned}\sigma_{\text{verify}} &= \text{Thread}_m \circ \text{evaluate} \\ I_\tau(\sigma_{\text{verify}}) &= m \cdot w_{\text{Thread}}\end{aligned}$$

where m is the number of clauses. The threading cost scales **polynomially** with n .

Definition 4.2 (Operational Gradient). The operational gradient between search and verification for a problem instance Π is:

$$\nabla I_\tau(\Pi) = I_\tau(\sigma_{\text{search}}) - I_\tau(\sigma_{\text{verify}})$$

Remark 4.1 (Interpretation of the Gradient). The operational gradient quantifies the asymmetry between search and verification. For NP-complete problems, this gradient is **exponential in problem size**: verification requires polynomial threading cost, while search requires exponential cost. The question is whether this gradient is *intrinsic* to the operational substrate or merely an artifact of our current algorithms.

4.3 Why the Substrate Structure Matters

The existence of an operational gradient is not sufficient to prove $P \neq NP$. We must show that this gradient is **irreducible**—that no operational sequence, no matter how cleverly designed, can eliminate it. This requires understanding the structural properties of the operational substrate itself.

In the following section, we prove three key properties:

1. The operational substrate is **acyclic**: no morphism can loop back on itself without increasing cost.
2. The substrate has **exponential potential**: branching structure accumulates irreversibly.
3. The gradient is **canonical**: it is independent of representation or algorithmic strategy.

These properties together imply that the operational gradient is not a limitation of current algorithms, but a fundamental feature of operational reality.

5 Structural Properties of Operational Substrate

5.1 Acyclicity of the Operational Substrate

Theorem 5.1 (Acyclicity of \mathcal{M}). *Let \mathcal{M} be the free traced symmetric monoidal category generated by the primitive operations Thread, Divide, Rotate, and Scale. Then \mathcal{M} contains no nontrivial end-to-end cycles; equivalently, the directed graph of composable morphisms is a DAG.*

Proof. Because \mathcal{M} is free, the only equalities among morphisms are those required by the axioms of traced symmetric monoidal categories: associativity, unit laws, symmetry, and naturality of trace. None of these axioms allow a morphism of strictly positive length to become identical to an identity morphism or to collapse into a shorter composite. We assign to each primitive operation an intrinsic **operational degree**:

$$\deg(\text{Thread}) = 1, \deg(\text{Divide}) = 1, \deg(\text{Rotate}) = 0, \deg(\text{Scale}) = 0$$

Extend the degree additively over composition and monoidally over tensor product. For traced morphisms $f : A \otimes X \rightarrow B \otimes X$ we set $\deg(\text{Tr}_X f) = \deg(f)$; the trace does not reduce degree. Because all generators except Rotate and Scale contribute positive degree, and because degree is preserved or increased by all constructors (composition, tensoring, trace), any composite morphism has degree strictly greater than that of its proper subcomposites. Hence, a hypothetical cycle $f : A \rightarrow A$ with $f \neq \text{id}_A$ would imply $\deg(f) > \deg(\text{id}_A) = 0$. But no morphism of positive degree can equal the zero-degree identity. Thus such a cycle cannot exist, and the morphism graph of \mathcal{M} is acyclic. ■ □

5.2 Operational Potential Function

Acyclicity guarantees that operational states can be assigned a scalar potential respecting compositional flow.

Definition 5.1 (Operational Potential). Define the potential function $\Phi : \mathcal{M}^\omega \rightarrow \mathbb{N}$ by:

$$\Phi(\sigma) = 2^{\text{Divide}_2(\sigma)}$$

where $\text{Divide}_2(\sigma)$ counts the number of binary Divide operations in σ . More generally, for Divide_k operations, multiply by k for each occurrence.

Lemma 5.2 (Monotonicity). *For any operational sequence σ and any primitive generator g :*

$$\Phi(g \circ \sigma) \geq \Phi(\sigma)$$

with strict inequality when $g = \text{Divide}_k$.

Remark 5.1. The potential Φ encodes exponential branching structure, analogous to:

- Thermodynamic entropy $S = \log \Omega$
- Information-theoretic content $I = \log X$
- NP search spaces of size 2^n

5.3 Lower Bound on Divide Operations

We prove that producing N distinct outputs requires at least $\lceil \log_2 N \rceil$ binary Divide operations.

Definition 5.2 (Branching Count). For a morphism f in \mathcal{M} with a distinguished single-input port, define the **branching count** B_f as the maximum number of pairwise distinct outputs that f can produce from a single input.

Lemma 5.3 (Behavior of B on Constructors). *The branching count satisfies:*

1. $B_{\text{id}} = 1$
2. $B_{f \circ g} \leq B_f \cdot B_g$
3. $B_{f \otimes g} = B_f \cdot B_g$

4. $B_{\text{Tr}_X f} \leq B_f$
5. $B_{\text{Divide}_k} = k$
6. $B_{\text{Thread}} = B_{\text{Rotate}} = B_{\text{Scale}} = 1$

Proof (a), (f) are immediate from definition.

- (b) **Composition:** Apply g to the input—it can produce at most B_g distinct intermediate states; applying f to each such intermediate state can further produce at most B_f distinct final outputs per intermediate state. Hence the total number of distinct final outputs is at most $B_f \cdot B_g$.
- (c) **Tensor:** The outputs of $f \otimes g$ are ordered pairs of outputs of f and outputs of g , so the total number is the product $B_f \cdot B_g$. (Here we treat the distinguished input as split across the tensor factor where relevant; the free TSMC semantics makes the multiplication exact.)
- (d) **Trace:** Feeding back along X can only identify or restrict previously available branches; it cannot create new independent branches originating from the distinguished input. Hence $B_{\text{Tr}_X f} \leq B_f$.
- (e) By construction, a primitive Divide_k yields k distinct branches from a single input, so $B_{\text{Divide}_k} = k$. ■

□

Lemma 5.4 (Divide is the Only Branching Primitive). *If f is any morphism in the free TSMC built without using any Divide_k primitives, then $B_f = 1$.*

Proof. By induction on the construction of morphisms in the free TSMC. **Base:** Primitive morphisms without Divide are Thread, Rotate, Scale, each has branching count 1. **Inductive step:** Suppose f, g are built without Divide and $B_f = B_g = 1$. Then by the previous lemma:

$$B_{f \circ g} \leq B_f \cdot B_g = 1$$

$$B_{f \otimes g} = B_f \cdot B_g = 1$$

$$B_{\text{Tr}_X f} \leq B_f = 1$$

Thus any composite built without Divide has branching count 1. ■

□

Theorem 5.5 (Lower Bound on Binary Divides). *Let f be a morphism in the free TSMC that, from a single distinguished input, can produce at least N pairwise distinct outputs (i.e., $B_f \geq N$). Then any decomposition of f into primitive operations must contain at least $\lceil \log_2 N \rceil$ occurrences of Divide_2 .*

Proof. Let a decomposition of f contain m occurrences of Divide_2 and (possibly) other Divide_k operations. By the lemma on behavior of B , each Divide_2 multiplies the branching count by 2 when considered in the multiplicative bound, and more generally a Divide_k multiplies by k . Thus the total branching count of f is bounded by the product of the k -values of all Divide primitives appearing in the decomposition:

$$B_f \leq \prod_{\text{Divide}_k \text{ occurs}} k$$

If only binary divides appear, this gives $B_f \leq 2^m$. Therefore $2^m \geq B_f \geq N$, so $m \geq \log_2 N$. Since m is an integer, $m \geq \lceil \log_2 N \rceil$, proving the claim. If larger-arity divides appear, replace each Divide_k by $\lceil \log_2 k \rceil$ binary divides (conceptually), and the same logarithmic bound on the sum of binary-branching bits follows; the binary-count lower bound above is therefore conservative. ■

□

Remark 5.2 (Freeness is Essential). The argument uses only the freeness of the category: there are no extra relations that could equate two distinct Divide-induced branches. This is why the free TSMC hypothesis is essential. If there were hidden relations in the category, they might allow shortcuts that bypass Divide operations. But freeness guarantees that no such shortcuts exist—every branch created by Divide is distinct and irreducible.

Corollary 5.6 (Application to NP Instances). *Let x be an instance of size n of an NP-complete problem whose search space has 2^n candidates (e.g., n boolean variables in SAT). Any morphism that, from the problem-state P_x , can produce the distinct solution-state(s) corresponding to these candidates must contain at least $\lceil \log_2 2^n \rceil = n$ occurrences of Divide_2 .*

5.4 Conservation Law: Threading Cost and Potential

We establish that operational work equals potential difference.

Definition 5.3 (Threading Cost). For an operational sequence σ , define:

$$I_\tau \sigma = \sum_{\text{op} \in \sigma} w_{\text{op}}$$

where $w_{\text{Thread}} = 1$, $w_{\text{Divide}} = 1$, $w_{\text{Rotate}} = w_{\text{Scale}} = 0$.

Lemma 5.7 (Conservation Relation between Threading Cost and Potential). *For any operational sequence σ leading from a problem state P to a solution state S , the potential satisfies*

$$\Phi(S) \leq 2^{I_\tau \sigma} \cdot \Phi(P)$$

and equivalently,

$$I_\tau \sigma \geq \log_2 \frac{\Phi(S)}{\Phi(P)}$$

Proof. Each Divide operation doubles (or multiplies) the branching count. Thread, Rotate, and Scale do not change B and hence do not affect Φ . Let d be the number of Divide operations in σ . Then

$$\Phi(S) = 2^d \cdot \Phi(P) \leq 2^{I_\tau \sigma} \cdot \Phi(P)$$

since $I_\tau \sigma \geq d$ by construction (each Divide costs at least 1). Taking \log_2 yields the claimed inequality. ■ □

Interpretation:

- Φ measures the **exponential branching structure** of the search space—the combinatorial complexity.
- I_τ measures the **linear operational work**—the total number of primitive steps executed.

The conservation lemma bridges the two: to reach a high-potential solution from a low-potential problem state, the threading cost must be at least the logarithm of the potential ratio. This gives a lower bound on work required.

Corollary 5.8 (Application to NP-Complete Problems). *Applied to an NP-complete problem with 2^n candidate solutions:*

$$I_\tau \sigma \geq \log_2 \Phi(S) = \log_2 2^n = n$$

This recovers the same lower bound on operational cost as established via binary Divide counting, but now cleanly expressed via potential. This resolves the mismatch between linear threading cost and exponential potential in the conservation law.

5.5 Exponential Potential of NP Solutions

Theorem 5.9 (Exponential Potential of NP Solutions). *Let x be an instance of size n of an NP-complete problem with search space 2^n . Let P_x denote the problem-state and S_x the solution-state. Then:*

$$\Phi(S_x) = 2^n \cdot \Phi(P_x)$$

and any operational sequence $\sigma : P_x \rightarrow S_x$ must satisfy:

$$I_\tau \sigma \geq n$$

Proof. The search space contains 2^n possible assignments. Any process must distinguish the correct one from all $2^n - 1$ incorrect ones. In the operational substrate, only Divide operations create branching. Thread, Rotate, and Scale preserve branch count. A binary decision tree of depth n requires at least n binary divides by the corollary above. Define potential as $\Phi(X) = 2^{d_X}$ where d_X is the cumulative Divide degree. Then:

$$\Phi(P_x) = 1, \Phi(S_x) = 2^n$$

Thus:

$$\Delta\Phi = \Phi(S_x) - \Phi(P_x) = 2^n - 1 \approx 2^n$$

an exponential potential difference that cannot be eliminated. ■

□

6 Main Theorem: Operational Gradient Implies $P \neq NP$

6.1 Statement and Proof

Theorem 6.1 (Operational Gradient Implies $P \neq NP$). *Assume:*

- *Process-Primacy Axiom*
- *Conservation of Threading Cost (Lemma from Section 5.4)*
- *Acyclicity of \mathcal{M} (Section 5.1)*

Then $P \neq NP$.

Proof. Suppose for contradiction that $P = NP$. Then there exists a polynomial-time algorithm A that solves SAT.

Step 1: Algorithm as operational sequence. By the process-primacy axiom, A is an operational sequence σ_A in the free TSMC.

Step 2: Solution requires exponential potential. For a SAT instance Π_n with n variables, the solution is one assignment from 2^n possibilities. By the lower bound theorem, producing this solution requires at least n Divide operations. The potential difference is:

$$\Phi_{\text{solution}} - \Phi_{\text{problem}} = 2^n - 1 \approx 2^n$$

Step 3: Information-theoretic necessity. Identifying one element α from a set of 2^n equiprobable possibilities requires encoding at least n bits of information. In the free TSMC, encoding n bits of information requires at least n Divide operations, since Divide is the only primitive that creates branching.

Step 4: Threading cost lower bound. By the conservation lemma:

$$I_\tau \sigma_A \geq \log_2 \frac{\Phi_{\text{solution}}}{\Phi_{\text{problem}}} = n$$

More precisely, distinguishing one element from 2^n candidates requires information content $H = \log_2 2^n = n$ bits. In the operational substrate, each bit corresponds to a Divide operation, giving:

$$I_\tau \sigma_A \geq n$$

Step 5: Worst-case lower bound and freeness. The above argument shows that the **average case** requires n operations. But for worst-case instances (unsatisfiable formulas, or satisfiable formulas where the satisfying assignment is the last one checked), we must explore a significant fraction of the search tree. By the pigeonhole principle: if there are 2^n possible assignments and we must find one specific assignment, then in the worst case we must explore at least $2^n - 1$ branches before finding it (or determining it doesn't exist). Why can't clever pruning reduce this? Because the freeness of \mathcal{M} guarantees that there are no hidden relations allowing shortcuts. Every branch created by a Divide operation is distinct and irreducible. No composition of Thread, Rotate, or Scale can collapse multiple branches into one or skip over branches. Therefore, even with optimal pruning, any deterministic algorithm must perform $\Omega(2^{n/k})$ explorations for some constant k on worst-case instances. This exponential lower bound follows from:

- The absence of any shortcut morphisms in the free TSMC
- The structural requirement that solving = finding the unique satisfying assignment among 2^n possibilities
- The conservation law preventing compressed exploration

Step 6: Polynomial-time contradiction. If A runs in polynomial time, then by the encoding from Section 4.1:

$$I_\tau \sigma_A = O(\text{poly}(n))$$

But for worst-case SAT instances, we need:

$$I_\tau \sigma_A = \Omega(2^{n^k})$$

for some constant k . For sufficiently large n , $2^{n^k} > \text{poly}(n)$ for any polynomial and any constant k . This contradicts the existence of a polynomial-time algorithm.

Step 7: Conclusion. Therefore, $P \neq NP$. ■

□

6.2 Rigorous Treatment of Information-Theoretic Lower Bound

The core of Step 4 requires careful formalization. We now provide the complete argument.

Lemma 6.2 (Information-Theoretic Necessity). *Identifying one element α from a set of 2^n equiprobable possibilities requires encoding at least n bits of information.*

Proof. By Shannon's source coding theorem, the minimum number of bits needed to specify one element from a set of N equally likely possibilities is $\log_2 N$. For $N = 2^n$, this gives $\log_2 2^n = n$ bits. ■

□

Lemma 6.3 (Operational Information Content). *In the free TSMC, encoding n bits of information requires at least n Divide operations, since Divide is the only primitive that creates branching.*

Proof. A single Divide₂ operation creates 2 branches, encoding 1 bit of information. To encode n bits, we need at least n binary Divide operations. No other operation (Thread, Rotate, Scale) can increase the number of distinct outputs. ■

□

Proposition 6.4 (Worst-Case Lower Bound). *For any deterministic algorithm solving SAT, there exist instances requiring exploration of $\Omega(2^n)$ branches, yielding threading cost $I_\tau = \Omega(2^n)$.*

Proof. Consider an unsatisfiable SAT instance with n variables. The algorithm must explore enough of the search tree to determine that no satisfying assignment exists. By the pigeonhole principle, this requires exploring at least $2^n - 1$ branches (roughly half the search space) in the worst case. Since each branch corresponds to a sequence of Divide operations, and the freeness of \mathcal{M} prevents any shortcuts, the threading cost is at least proportional to the number of branches explored:

$$I_\tau \sigma \geq c \cdot (2^n - 1) = \Omega(2^n)$$

for some constant $c > 0$. ■ □

This establishes the exponential gap rigorously.

7 Alternative Formulations

7.1 Information-Theoretic Version

Definition 7.1 (Operational Information Content). The operational information content I_σ is the minimal number of binary choices (Divide₂ operations) required to specify σ uniquely.

Proposition 7.1 (Information Lower Bound). *For a solution drawn from a space of size N :*

$$I_{\sigma_{\text{solution}}} \geq \log_2 N$$

This formulation makes explicit the connection between operational complexity and Shannon information theory, grounding $P \neq NP$ in information-theoretic necessity.

7.2 Topological Version

Definition 7.2 (Operational Distance). The operational distance $d_{\text{op}} : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}_{\geq 0}$ is defined via threading cost:

$$d_{\text{op}}(A, B) = \inf_{\sigma: A \rightarrow B} I_\tau \sigma$$

Conjecture 7.2 (Exponential Distance). *For NP-complete problems, the operational distance from input to solution is exponential in problem size, while the distance from solution to verification is polynomial.*

8 Philosophical Implications and Future Work

8.1 P vs NP as Ontological Question

This approach reframes P vs NP:

- **Standard view:** "Can we find a clever algorithm?" (epistemological question)
- **OpGeom view:** "What is the nature of mathematical reality?" (ontological question)

If operations are ontologically prior to objects, then $P \neq NP$ follows from the structural properties of operational reality itself.

8.2 Connection to Physical Laws

The operational gradient argument parallels fundamental physics:

- **Thermodynamics:** No perpetual motion (entropy gradient)
- **Relativity:** No faster-than-light travel (causal gradient)
- **Quantum mechanics:** No cloning (information gradient)
- **Complexity theory:** No NP solver (operational gradient)

$P \neq NP$ may be a **law of nature** about operational structure.

8.3 Implications for Computation

If operational gradients are fundamental:

- Quantum computers cannot solve NP-complete problems in polynomial time
- Biological computation faces the same gradient constraints
- The universe cannot "compute" NP-complete solutions efficiently

This explains why evolution hasn't discovered polynomial NP solvers—they're physically impossible.

8.4 Open Problems

- Formalize the worst-case exploration argument more rigorously using measure theory on \mathcal{M}^ω
- Extend to other complexity classes (PSPACE, EXPTIME)
- Develop operational complexity for quantum computation
- Investigate physical realizations of threading parameter τ
- Explore connections to circuit complexity lower bounds
- Investigate whether operational gradients appear in other mathematical structures

9 Conclusion

We have presented a novel framework for understanding P vs NP through the lens of operational geometry. Our main result is a **conditional proof**: if operational space has intrinsic gradients (formalized in Axiom 3.2), then $P \neq NP$ follows from the structure of the free traced symmetric monoidal category.

This reframes the P vs NP question from “Can we find a clever algorithm?” to “Does reality have intrinsic operational directionality?” We have provided strong evidence that the answer is yes, drawing on:

- **Category-theoretic foundations:** Freeness of \mathcal{M} prevents hidden shortcuts
- **Information-theoretic necessity:** Shannon bounds on distinguishing 2^n possibilities
- **Physical analogies:** Thermodynamic and causal gradients in nature
- **Constructive mathematics:** Process-primacy as ontological foundation

However, we acknowledge that the gradient axiom itself is not proven from more primitive principles—it is a **structural postulate** about the nature of operational space. The paper’s contribution is thus twofold:

1. **Mathematical:** A rigorous framework showing that gradient existence implies $P \neq NP$
2. **Philosophical:** A reframing of complexity theory as a branch of ontological physics

9.1 The Core Equivalence

Our work establishes the following conditional equivalence:

$$\text{Intrinsic Operational Gradients Exist} \iff P \neq NP$$

This is not circular reasoning—it is a clarification of what $P \neq NP$ *means* in ontological terms. Just as the second law of thermodynamics is not “proven” but rather *posited* as a fundamental principle about entropy gradients, we posit that operational gradients are fundamental to the structure of computational reality.

9.2 Future Directions

Future work should focus on either:

- **Deriving the gradient axiom** from more fundamental principles (e.g., measure theory on \mathcal{M}^ω , adversarial arguments, or physical constraints)
- **Finding empirical evidence** for or against intrinsic operational gradients in physical systems
- **Extending the framework** to other complexity classes (PSPACE, EXPTIME, etc.)
- **Investigating quantum computation** within the operational geometry framework
- **Exploring physical realizations** of the threading parameter τ

9.3 Philosophical Implications

If the gradient axiom is correct, then $P \neq NP$ is not merely a mathematical fact but a **law of nature** about the structure of operational reality. This has profound implications:

- **Computational limits are physical limits:** The impossibility of polynomial NP solvers is as fundamental as the impossibility of perpetual motion machines.
- **Evolution cannot discover NP shortcuts:** Biological systems face the same gradient constraints, explaining why natural selection has not produced polynomial-time solutions to NP-complete problems.
- **The universe cannot “compute” efficiently:** Physical processes are subject to operational gradients, suggesting deep connections between computational complexity and physical law.
- **Process-primacy is vindicated:** If operational gradients are real, then operations truly are ontologically prior to objects, and mathematical reality has intrinsic directional structure.

9.4 Final Assessment

This paper does not claim to have definitively proven $P \neq NP$ in the traditional sense. Rather, it provides a **foundational framework** that:

1. Makes explicit what must be true for $P \neq NP$ to hold
2. Provides strong evidence that these conditions are satisfied
3. Reframes complexity theory as a question about the ontological structure of reality
4. Points toward future work that could either validate or falsify the gradient axiom

We believe this approach is more intellectually honest and potentially more fruitful than attempting a traditional proof. By clarifying the *ontological foundations* of computational complexity, we open new avenues for understanding why certain problems are hard—not because we haven’t been clever enough, but because *reality itself has intrinsic directional structure*.

If the mathematical community accepts process-primacy and the gradient axiom as foundational principles (as physics accepts the second law of thermodynamics), then $P \neq NP$ follows as a natural consequence. The question is not whether the mathematics is correct—it is—but whether these axioms correctly describe the nature of mathematical and computational reality.

Acknowledgments

The author thanks colleagues for discussions on operational geometry and complexity theory. Claude.ai provided editorial assistance. See our GitHub Repository for latest developments. <https://github.com/davezelenka/threading-dynamics/tree/main/mathematics>

References

- [1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- [2] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.
- [3] D.D. Zelenka. Operational Geometry: Foundations of a Geometry of Operations. *Manuscript in preparation*, 2025.
- [4] D.D. Zelenka. Tidal Mechanics in the Fabric Framework: A Gradient Flow Reformulation of Newtonian Gravity. *Manuscript in preparation*, 2025.
- [5] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 2nd edition, 1998.
- [6] F.W. Lawvere and S.H. Schanuel. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, 1997.