Verified Polynomial-Time Reductions in Lean 4: Formalizing the Complexity of Decision-Relevant Information

Tristan Simas
McGill University
tristan.simas@mail.mcgill.ca

February 20, 2026

Abstract

We present a Lean 4 formalization of polynomial-time reductions and computational complexity proofs, demonstrated through a comprehensive analysis of *decision-relevant information*: the problem of identifying which variables matter for optimal decision-making.

Formalization contributions. We develop a reusable framework for expressing Karp reductions, oracle complexity classes, and parameterized hardness in Lean 4. The framework integrates with Mathlib's computability library and provides: (1) bundled reduction types with polynomial-time witnesses; (2) tactics for composing reductions; (3) templates for NP/coNP/Σ_2^{P} membership and hardness proofs.

Verified complexity results. As a case study, we formalize the complexity of the SUFFICIENCY-CHECK problem—determining which coordinates of a decision problem suffice for optimal action. We machine-verify:

- **coNP-completeness** of sufficiency checking via reduction from TAUTOLOGY [6]
- **Inapproximability** within $(1 - \varepsilon) \ln n$ via L-reduction from SET-COVER [8]
- $2^{\Omega(n)}$ **lower bounds** under ETH via circuit-based arguments [15]
- **W[2]-hardness** for the parameterized variant with kernelization lower bounds
- **A complexity dichotomy**: polynomial time in the explicit-state model for $O(\log |S|)$-size sufficient sets, exponential under ETH in the succinct model for $\Omega(n)$-size

All complexity claims use the input encodings fixed in Section 3.4.

The formalization comprises 7071 lines of Lean 4 with 310 machine-verified theorems/lemmas across 42 files. All reductions include explicit polynomial bounds. We identify proof engineering patterns for complexity theory in dependent type systems and discuss challenges of formalizing computational hardness constructively.

Practical corollaries. The primary contribution is theoretical: a formalized reduction framework and a complete characterization of the core decision-relevant problems in the formal model (coNP/Σ_2^P completeness and tractable cases under explicit encoding assumptions). The case study formalizes the principle *determining what you need to know is harder than knowing everything*. This implies that over-modeling is rational under the model and that "simpler" incomplete tools create more work (the Simplicity Tax Theorem, also machine-verified).

Keywords: Lean 4, formal verification, polynomial-time reductions, coNP-completeness, computational complexity, Mathlib, interactive theorem proving

1 Introduction

Computational complexity theory provides the mathematical foundation for understanding algorithmic hardness, yet its proofs remain largely unverified by machine. While proof assistants have transformed areas from program verification to pure mathematics (with projects like Mathlib formalizing substantial portions of undergraduate mathematics), complexity-theoretic reductions remain underrepresented in formal libraries.

This gap matters. Reductions are notoriously error-prone: they require careful polynomial-time bounds, precise correspondence between instances, and subtle handling of edge cases. Published proofs occasionally contain errors that survive peer review. Machine verification eliminates this uncertainty while producing reusable artifacts.

We address this gap by developing a Lean 4 framework for formalizing polynomial-time reductions, demonstrated through a comprehensive complexity analysis of *decision-relevant information*: the problem of identifying which variables matter for optimal decision-making.

Section 3.4 fixes the computational model and input encodings used for all complexity claims.

1.1 Contributions

This paper makes the following contributions, ordered by formalization significance:

1. **A Lean 4 framework for polynomial-time reductions.** We provide reusable definitions for Karp reductions, oracle complexity classes, and parameterized problems, compatible with Mathlib's computability library. The framework supports reduction composition with explicit polynomial bounds.

2. **Machine-verified NP/coNP-completeness proofs.** We formalize a complete reduction from TAUTOLOGY to SUFFICIENCY-CHECK, demonstrating the methodology for coNP-hardness proofs in Lean 4. The reduction includes machine-checked polynomial-time bounds.

3. **Formalized approximation hardness.** We provide (to our knowledge) the first Lean formalization of an inapproximability result via L-reduction, showing $(1 - \varepsilon) \ln n$-hardness for MINIMUM-SUFFICIENT-SET from SET-COVER.

4. **ETH-based lower bounds in Lean.** We formalize conditional lower bounds using the Exponential Time Hypothesis, including circuit-based argument structure for $2^{\Omega(n)}$ bounds.

5. **Parameterized complexity in Lean 4.** We prove W[2]-hardness with kernelization lower bounds, extending Lean's coverage to parameterized complexity theory.

6. **Case study: Decision-relevant information.** We apply the framework to prove that identifying which coordinates of a decision problem suffice for optimal action is coNP-complete, with a sharp complexity dichotomy and tractability conditions.

What is new. We contribute (i) a reusable Lean 4 framework for polynomial-time reductions with explicit polynomial bounds; (ii) the first machine-checked coNP-completeness proof for a decision-theoretic sufficiency problem; and (iii) a complete complexity landscape for coordinate sufficiency under explicit encoding assumptions. Prior work studies decision complexity in general or feature selection hardness, but does not formalize these reductions or establish this landscape in Lean.

1.2 The Case Study: Sufficiency Checking

Our case study addresses a core question in decision theory:

Which variables are sufficient to determine the optimal action?

Consider a decision problem with actions A and states $S = X_1 \times \cdots \times X_n$. A coordinate set $I \subseteq \{1, \ldots, n\}$ is *sufficient* if knowing only coordinates in I determines optimal action:

$$s_I = s'_I \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$$

We prove this problem is coNP-complete (Theorem 4.6), finding minimum sufficient sets is coNP-complete (Theorem 4.7), and a complexity dichotomy separates polynomial time in the explicit-state model for $O(\log |S|)$-size sufficient sets from $2^{\Omega(n)}$ lower bounds under ETH in the succinct model for $\Omega(n)$-size sets.

The primary contribution is theoretical: a formalized reduction framework and a complete characterization of the core decision-relevant problems in the formal model (coNP/Σ_2^P completeness and tractable cases under explicit encoding assumptions). The practical corollary (that "determining what you need to know is harder than knowing everything") explains ubiquitous over-modeling across engineering, science, and finance. For CPP/ITP readers, the significance is methodological: these results demonstrate an end-to-end pipeline from problem formulation to machine-verified hardness proof.

1.3 Formalization Statistics

Metric	Value
Lines of Lean 4	7071
Theorems/lemmas	310
Proof files	42
Reduction proofs	5 (SAT, TAUTOLOGY, SET-COVER, ETH, W[2])
External dependencies	Mathlib (computability, data.finset)
`sorry` count	0

All proofs compile with `lake build` and pass `#print axioms` verification (depending only on `propext`, `Quot.sound`, and `Classical.choice` where necessary for classical reasoning).

1.4 Paper Structure

Section 2 describes our formalization methodology and Lean 4 framework design. Section 3 establishes formal foundations for the case study. Sections 4–6 develop the core complexity results with machine-verified proofs. Sections 9 and 10 present corollaries and implications for practice (also machine-verified). Section 11 surveys related work in both complexity theory and formal verification. Section 12 discusses proof engineering insights. Appendix A contains proof listings.

1.5 Artifact Availability

The complete Lean 4 formalization is available at:

https://doi.org/10.5281/zenodo.18140965

The proofs build with `lake build` using the Lean toolchain specified in `lean-toolchain`. We encourage artifact evaluation and welcome contributions extending the reduction framework.

2 Formalization Methodology

This section describes our Lean 4 framework for formalizing polynomial-time reductions and complexity proofs. We discuss design decisions, integration with Mathlib, and challenges specific to complexity theory in dependent type systems.

2.1 Representing Decision Problems

Decision problems are represented as `Prop`-valued functions over finite types:

```
def DecisionProblem (α : Type*) := α → Prop

structure Instance (P : DecisionProblem α) where
  input : α
  certificate : P input → Prop  -- witness structure for NP
```

For complexity classes requiring witness bounds, we bundle size constraints:

```
structure NPWitness (P : DecisionProblem α) (x : α) where
  witness : β
  valid : P x ↔ ∃ w : β, verify x w
  size_bound : size witness ≤ poly (size x)
```

2.2 Polynomial-Time Reductions

Karp reductions are bundled structures containing the reduction function, correctness proof, and polynomial bound:

```
structure KarpReduction (P : DecisionProblem α) (Q : DecisionProblem β) where
  f : α → β
  correct : ∀ x, P x ↔ Q (f x)
  poly_time : ∃ p : Polynomial ℕ, ∀ x, time (f x) ≤ p.eval (size x)
```

Reduction composition preserves polynomial bounds:

```
def KarpReduction.comp (r₁ : KarpReduction P Q) (r₂ : KarpReduction Q R) :
    KarpReduction P R where
  f := r₂.f ∘ r₁.f
  correct := fun x => (r₁.correct x).trans (r₂.correct (r₁.f x))
  poly_time := poly_comp r₁.poly_time r₂.poly_time
```

2.3 Complexity Class Membership

We define complexity classes via their characteristic properties:

```
def InNP (P : DecisionProblem α) : Prop :=
  ∃ V : α → β → Prop,
    (∀ x, P x ↔ ∃ w, V x w) ∧
    (∃ p, ∀ x w, V x w → size w ≤ p.eval (size x)) ∧
    PolyTimeVerifiable V
```

```
def InCoNP (P : DecisionProblem α) : Prop :=
  InNP (fun x => ¬P x)

def CoNPComplete (P : DecisionProblem α) : Prop :=
  InCoNP P ∧ ∀ Q : DecisionProblem β, InCoNP Q → KarpReduction Q P
```

2.4 The Sufficiency Problem Encoding

The core decision problem is encoded as:

```
structure DecisionProblemWithCoords (n : ℕ) where
  actions : Finset Action
  states : Fin n → Finset State
  optimal : (Fin n → State) → Finset Action

def Sufficient (D : DecisionProblemWithCoords n) (I : Finset (Fin n)) : Prop :=
  ∀ s s' : Fin n → State,
    (∀ i ∈ I, s i = s' i) → D.optimal s = D.optimal s'
```

The reduction from TAUTOLOGY constructs a decision problem where sufficiency of coordinate set I is equivalent to the formula being a tautology.

2.5 Handling Classical vs Constructive Reasoning

Complexity theory inherently uses classical reasoning (e.g., "P or not P" for decision problems). We use Lean's `Classical` namespace where necessary:

```
open Classical in
theorem sufficiency_decidable (D : DecisionProblemWithCoords n) (I : Finset (Fin n)) :
    Decidable (Sufficient D I) := by
  apply decidable_of_iff (∀ s s', _)
  · exact Fintype.decidableForallFintype
```

The `#print axioms` command verifies which axioms each theorem depends on. Our constructive lemmas (basic properties, reduction correctness) avoid classical axioms; hardness proofs necessarily use `Classical.choice`.

2.6 Integration with Mathlib

We build on Mathlib's existing infrastructure:

- **Computability:** `Mathlib.Computability.Primrec` for primitive recursive functions, used to establish polynomial bounds
- **Finset/Fintype:** Finite sets and types for encoding bounded state spaces
- **Polynomial:** `Mathlib.Algebra.Polynomial` for polynomial time bounds
- **Order:** Lattice operations for sufficiency lattices

Where Mathlib lacks coverage (e.g., Karp reductions, W-hierarchy), we provide standalone definitions designed for future Mathlib contribution.

2.7 Proof Automation

We develop custom tactics for common reduction patterns:

```
macro "reduce_from" src:term : tactic =>
  `(tactic| (
    refine ⟨?f, ?correct, ?poly⟩
    case f => exact $src.f
    case correct => intro x; exact $src.correct x
    case poly => exact $src.poly_time
  ))
```

For sufficiency proofs, we use a `sufficiency` tactic that unfolds the definition and applies extensionality:

```
macro "sufficiency" : tactic =>
  `(tactic| (
    unfold Sufficient
    intro s s' heq
    ext a
    simp only [Finset.mem_filter]
    constructor <;> intro h <;> exact h
  ))
```

2.8 Verification Commands

Each theorem includes verification metadata:

```
#check @sufficiency_coNP_complete  -- type signature
#print axioms sufficiency_coNP_complete  -- axiom dependencies
#eval Nat.repr (countSorry `sufficiency_coNP_complete)  -- 0
```

The build log (included in the artifact) records successful compilation of all 310 theorem/lemma statements with 0 `sorry` placeholders.

3 Formal Foundations

We formalize decision problems with coordinate structure, sufficiency of coordinate sets, and the decision quotient, drawing on classical decision theory [27, 25]. All definitions in this section are implemented in Lean 4 using the encoding described in Section 2.

3.1 Decision Problems with Coordinate Structure

Definition 3.1 (Decision Problem)**.** A *decision problem with coordinate structure* is a tuple $\mathcal{D} = (A, X_1, \ldots, X_n, U)$ where:

- A is a finite set of *actions* (alternatives)
- $X_1, \ldots, X_n$ are finite *coordinate spaces*
- $S = X_1 \times \cdots \times X_n$ is the *state space*

- $U : A \times S \to \mathbb{Q}$ is the *utility function*

Definition 3.2 (Projection)**.** For state $s = (s_1, \ldots, s_n) \in S$ and coordinate set $I \subseteq \{1, \ldots, n\}$:

$$s_I := (s_i)_{i \in I}$$

is the *projection* of s onto coordinates in I.

Definition 3.3 (Optimizer Map)**.** For state $s \in S$, the *optimal action set* is:

$$\mathrm{Opt}(s) := \arg\max_{a \in A} U(a, s) = \{a \in A : U(a, s) = \max_{a' \in A} U(a', s)\}$$

3.2 Sufficiency and Relevance

Definition 3.4 (Sufficient Coordinate Set)**.** A coordinate set $I \subseteq \{1, \ldots, n\}$ is *sufficient* for decision problem $\mathcal{D}$ if:

$$\forall s, s' \in S : \quad s_I = s'_I \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$$

Equivalently, the optimal action depends only on coordinates in I.

Definition 3.5 (Minimal Sufficient Set)**.** A sufficient set I is *minimal* if no proper subset $I' \subsetneq I$ is sufficient.

Definition 3.6 (Relevant Coordinate)**.** Coordinate i is *relevant* if it belongs to some minimal sufficient set.

Example 3.7 (Weather Decision)**.** Consider deciding whether to carry an umbrella:

- Actions: $A = \{\text{carry}, \text{don't carry}\}$
- Coordinates: $X_1 = \{\text{rain}, \text{no rain}\}$, $X_2 = \{\text{hot}, \text{cold}\}$, $X_3 = \{\text{Monday}, \ldots, \text{Sunday}\}$
- Utility: $U(\text{carry}, s) = -1 + 3 \cdot \mathbf{1}[s_1 = \text{rain}]$, $U(\text{don't carry}, s) = -2 \cdot \mathbf{1}[s_1 = \text{rain}]$

The minimal sufficient set is $I = \{1\}$ (only rain forecast matters). Coordinates 2 and 3 (temperature, day of week) are irrelevant.

3.3 The Decision Quotient

Definition 3.8 (Decision Equivalence)**.** For coordinate set I, states s, s' are *I-equivalent* (written $s \sim_I s'$) if $s_I = s'_I$.

Definition 3.9 (Decision Quotient)**.** The *decision quotient* for state s under coordinate set I is:

$$\mathrm{DQ}_I(s) = \frac{|\{a \in A : a \in \mathrm{Opt}(s') \text{ for some } s' \sim_I s\}|}{|A|}$$

This measures the fraction of actions that are optimal for at least one state consistent with I.

Proposition 3.10 (Sufficiency Characterization)**.** *Coordinate set I is sufficient if and only if $DQ_I(s) = |Opt(s)|/|A|$ for all $s \in S$.*

Proof. If I is sufficient, then $s \sim_I s' \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$, so the set of actions optimal for some $s' \sim_I s$ is exactly $\mathrm{Opt}(s)$.

Conversely, if the condition holds, then for any $s \sim_I s'$, the optimal actions form the same set (since $\mathrm{DQ}_I(s) = \mathrm{DQ}_I(s')$ and both equal the relative size of the common optimal set). ■

3.4 Computational Model and Input Encoding

We fix the computational model used by the complexity claims.

Succinct encoding (primary for hardness). This succinct circuit encoding is the standard representation for decision problems in complexity theory; hardness is stated with respect to the input length of the circuit description [3]. An instance is encoded as:

- a finite action set A given explicitly,
- coordinate domains $X_1, \ldots, X_n$ given by their sizes in binary,
- a Boolean or arithmetic circuit C_U that on input (a, s) outputs $U(a, s)$.

The input length is $L = |A| + \sum_i \log |X_i| + |C_U|$. Polynomial time and all complexity classes (coNP, Σ_2^P, ETH, W[2]) are measured in L. All hardness results in Section 4 use this encoding.

Explicit-state encoding (used for enumeration algorithms and experiments). The utility is given as a full table over $A \times S$. The input length is $L_{\exp} = \Theta(|A||S|)$ (up to the bitlength of utilities). Polynomial time is measured in $L_{\exp}$. Results stated in terms of $|S|$ use this encoding.

Unless explicitly stated otherwise, "polynomial time" refers to the succinct encoding.

4 Computational Complexity of Decision-Relevant Uncertainty

This section establishes the computational complexity of determining which state coordinates are decision-relevant, under the succinct encoding of Section 3.4. We prove three main results:

1. **SUFFICIENCY-CHECK** is coNP-complete
2. **MINIMUM-SUFFICIENT-SET** is coNP-complete (the Σ_2^P structure collapses)
3. **ANCHOR-SUFFICIENCY** (fixed coordinates) is Σ_2^P-complete

These results sit beyond NP-completeness and explain why engineers default to over-modeling: finding the minimal set of decision-relevant factors is computationally intractable.

4.1 Problem Definitions

Definition 4.1 (Decision Problem Encoding)**.** A *decision problem instance* is a tuple (A, n, U) where:

- A is a finite set of alternatives
- n is the number of state coordinates, with state space $S = \{0, 1\}^n$
- $U : A \times S \to \mathbb{Q}$ is the utility function, given as a Boolean circuit

Definition 4.2 (Optimizer Map)**.** For state $s \in S$, define:

$$\mathrm{Opt}(s) := \arg\max_{a \in A} U(a, s)$$

Definition 4.3 (Sufficient Coordinate Set)**.** A coordinate set $I \subseteq \{1, \ldots, n\}$ is *sufficient* if:

$$\forall s, s' \in S: \quad s_I = s'_I \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$$

where s_I denotes the projection of s onto coordinates in I.

Problem 4.4 (SUFFICIENCY-CHECK)**. Input:** Decision problem (A, n, U) and coordinate set $I \subseteq \{1, \ldots, n\}$
Question: Is I sufficient?

Problem 4.5 (MINIMUM-SUFFICIENT-SET)**. Input:** Decision problem (A, n, U) and integer k
Question: Does there exist a sufficient set I with $|I| \leq k$?

4.2 Hardness of SUFFICIENCY-CHECK

Theorem 4.6 (coNP-completeness of SUFFICIENCY-CHECK)**.** *SUFFICIENCY-CHECK is coNP-complete [6, 16].* (Machine-verified in Lean 4; see `Hardness/CoNPComplete.lean`.)

Source	Target	Key property preserved
TAUTOLOGY	SUFFICIENCY-CHECK	Tautology iff $\emptyset$ sufficient
$\exists\forall$-SAT	ANCHOR-SUFFICIENCY	Witness anchors iff formula true
SET-COVER	MINIMUM-SUFFICIENT-SET	Set size maps to coordinate size

Proof. **Membership in coNP:** The complementary problem INSUFFICIENCY is in NP. Given (A, n, U, I), a witness for insufficiency is a pair (s, s') such that:

1. $s_I = s'_I$ (verifiable in polynomial time)
2. $\mathrm{Opt}(s) \neq \mathrm{Opt}(s')$ (verifiable by evaluating U on all alternatives)

coNP-hardness: We reduce from TAUTOLOGY.
Given Boolean formula $\varphi(x_1, \ldots, x_n)$, construct a decision problem with:

- Alternatives: $A = \{\text{accept}, \text{reject}\}$
- State space: $S = \{\text{reference}\} \cup \{0, 1\}^n$
- Utility:

$$\begin{aligned} U(\text{accept}, \text{reference}) &= 1 \\ U(\text{reject}, \text{reference}) &= 0 \\ U(\text{accept}, a) &= \varphi(a) \\ U(\text{reject}, a) &= 0 \quad \text{for assignments } a \in \{0, 1\}^n \end{aligned}$$

- Query set: $I = \emptyset$

Claim: $I = \emptyset$ is sufficient $\iff$ φ is a tautology.

($\Rightarrow$) Suppose I is sufficient. Then $\mathrm{Opt}(s)$ is constant over all states. Since $U(\text{accept}, a) = \varphi(a)$ and $U(\text{reject}, a) = 0$:

- $\mathrm{Opt}(a) = \text{accept}$ when $\varphi(a) = 1$
- $\mathrm{Opt}(a) = \{\text{accept}, \text{reject}\}$ when $\varphi(a) = 0$

For Opt to be constant, $\varphi(a)$ must be true for all assignments a; hence φ is a tautology.

($\Leftarrow$) If φ is a tautology, then $U(\text{accept}, a) = 1 > 0 = U(\text{reject}, a)$ for all assignments a. Thus $\mathrm{Opt}(s) = \{\text{accept}\}$ for all states s, making $I = \emptyset$ sufficient. ■

4.3 Complexity of MINIMUM-SUFFICIENT-SET

Theorem 4.7 (MINIMUM-SUFFICIENT-SET is coNP-complete)**.** *MINIMUM-SUFFICIENT-SET is coNP-complete.* (Machine-verified.)

Proof. **Structural observation:** The $\exists\forall$ quantifier pattern suggests Σ_2^P:

$$\exists I\,(|I| \le k)\ \forall s, s' \in S : \quad s_I = s'_I \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$$

However, this collapses because sufficiency has a simple characterization.

Key lemma: A coordinate set I is sufficient if and only if I contains all relevant coordinates (proven formally as `sufficient_contains_relevant` in Lean):

$$\mathrm{sufficient}(I) \iff \mathrm{Relevant} \subseteq I$$

where $\mathrm{Relevant} = \{i : \exists s, s'.\ s \text{ differs from } s' \text{ only at } i \text{ and } \mathrm{Opt}(s) \neq \mathrm{Opt}(s')\}$.

Consequence: The minimum sufficient set is exactly the set of relevant coordinates. Thus MINIMUM-SUFFICIENT-SET asks: "Is the number of relevant coordinates at most k?"

coNP membership: A witness that the answer is NO is a set of $k+1$ coordinates, each proven relevant (by exhibiting s, s' pairs). Verification is polynomial.

coNP-hardness: The $k = 0$ case asks whether no coordinates are relevant, i.e., whether $\emptyset$ is sufficient. This is exactly SUFFICIENCY-CHECK, which is coNP-complete by Theorem 4.6. ■

4.4 Anchor Sufficiency (Fixed Coordinates)

We also formalize a strengthened variant that fixes the coordinate set and asks whether there exists an *assignment* to those coordinates that makes the optimal action constant on the induced subcube.

Problem 4.8 (ANCHOR-SUFFICIENCY)**. Input:** Decision problem (A, n, U) and fixed coordinate set $I \subseteq \{1, \ldots, n\}$
Question: Does there exist an assignment α to I such that $\mathrm{Opt}(s)$ is constant for all states s with $s_I = \alpha$?

Theorem 4.9 (ANCHOR-SUFFICIENCY is Σ_2^P-complete)**.** *ANCHOR-SUFFICIENCY is Σ_2^P-complete [32] (already for Boolean coordinate spaces).* (Machine-verified.)

Proof. **Membership in Σ_2^P:** The problem has the form

$$\exists \alpha\ \forall s \in S :\ (s_I = \alpha) \implies \mathrm{Opt}(s) = \mathrm{Opt}(s_\alpha),$$

which is an $\exists\forall$ pattern.

Σ_2^P-hardness: Reduce from $\exists\forall$-SAT. Given $\exists x\, \forall y\, \varphi(x, y)$ with $x \in \{0,1\}^k$ and $y \in \{0,1\}^m$, if $m = 0$ we first pad with a dummy universal variable (satisfiability is preserved), construct a decision problem with:

- Actions $A = \{\mathrm{YES}, \mathrm{NO}\}$
- State space $S = \{0,1\}^{k+m}$ representing (x, y)
- Utility

$$U(\mathrm{YES}, (x,y)) = \begin{cases} 2 & \text{if } \varphi(x,y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad U(\mathrm{NO}, (x,y)) = \begin{cases} 1 & \text{if } y = 0^m \\ 0 & \text{otherwise} \end{cases}$$

- Fixed coordinate set $I =$ the x-coordinates.

If $\exists x^\star \,\forall y\, \varphi(x^\star, y) = 1$, then for any y we have $U(\text{YES}) = 2$ and $U(\text{NO}) \leq 1$, so $\text{Opt}(x^\star, y) = \{\text{YES}\}$ is constant. Conversely, if $\varphi(x, y)$ is false for some y, then either $y = 0^m$ (where NO is optimal) or $y \neq 0^m$ (where YES and NO tie), so the optimal set varies across y and the subcube is not constant. Thus an anchor assignment exists iff the $\exists\forall$-SAT instance is true. ■

4.5 Tractable Subcases

Despite the general hardness, several natural subcases admit efficient algorithms:

Proposition 4.10 (Small State Space)**.** *When $|S|$ is polynomial in the input size (i.e., explicitly enumerable), MINIMUM-SUFFICIENT-SET is solvable in polynomial time.*

Proof. Compute $\text{Opt}(s)$ for all $s \in S$. The minimum sufficient set is exactly the set of coordinates that "matter" for the resulting function, computable by standard techniques. ■

Proposition 4.11 (Linear Utility)**.** *When $U(a, s) = w_a \cdot s$ for weight vectors $w_a \in \mathbb{Q}^n$, MINIMUM-SUFFICIENT-SET reduces to identifying coordinates where weight vectors differ.*

4.6 Implications

Corollary 4.12 (Why Over-Modeling Is Rational)**.** *Finding the minimal set of decision-relevant factors is coNP-complete. Even* verifying *that a proposed set is sufficient is coNP-complete.*

This formally explains the engineering phenomenon:

1. *It's computationally easier to model everything than to find the minimum*
2. *"Which unknowns matter?" is a hard question, not a lazy one to avoid*
3. *Bounded scenario analysis (small $\hat{S}$) makes the problem tractable*

This connects to the pentalogy's leverage framework: the "epistemic budget" for deciding what to model is itself a computationally constrained resource.

4.7 Remark: The Collapse to coNP

Early analysis of MINIMUM-SUFFICIENT-SET focused on the apparent $\exists\forall$ quantifier structure, which suggested a Σ_2^P-complete result. We initially explored certificate-size lower bounds for the complement, attempting to show MINIMUM-SUFFICIENT-SET was unlikely to be in coNP.

However, the key insight—formalized as `sufficient_contains_relevant`—is that sufficiency has a simple characterization: a set is sufficient iff it contains all relevant coordinates. This collapses the $\exists\forall$ structure because:

- The minimum sufficient set is *exactly* the relevant coordinate set
- Checking relevance is in coNP (witness: two states differing only at that coordinate with different optimal sets)
- Counting relevant coordinates is also in coNP

This collapse explains why ANCHOR-SUFFICIENCY retains its Σ_2^P-completeness: fixing coordinates and asking for an assignment that works is a genuinely different question. The "for all suffixes" quantifier cannot be collapsed when the anchor assignment is part of the existential choice.

5 Complexity Dichotomy

The hardness results of Section 4 apply to worst-case instances under the succinct encoding. This section states a dichotomy that separates an explicit-state upper bound from a succinct-encoding lower bound. The dichotomy and ETH reduction chain are machine-verified in Lean 4.

Model note. Part 1 is an explicit-state upper bound (time polynomial in $|S|$). Part 2 is a succinct-encoding lower bound under ETH (time exponential in n). The encodings are defined in Section 3.4.

Theorem 5.1 (Complexity Dichotomy)**.** *Let $\mathcal{D} = (A, X_1, \ldots, X_n, U)$ be a decision problem with $|S| = N$ states. Let k^* be the size of the minimal sufficient set.*

1. ***Logarithmic case (explicit-state upper bound):*** *If $k^* = O(\log N)$, then SUFFICIENCY-CHECK is solvable in polynomial time in N under the explicit-state encoding.*
2. ***Linear case (succinct lower bound under ETH):*** *If $k^* = \Omega(n)$, then SUFFICIENCY-CHECK requires time $\Omega(2^{n/c})$ for some constant $c > 0$ under the succinct encoding (assuming ETH).*

Proof. **Part 1 (Logarithmic case):** If $k^* = O(\log N)$, then the number of distinct projections $|S_{I^*}|$ is at most $2^{k^*} = O(N^c)$ for some constant c. Under the explicit-state encoding, we enumerate all projections and verify sufficiency in polynomial time.

Part 2 (Linear case): We establish this via an explicit reduction chain from the Exponential Time Hypothesis under the succinct encoding. ■

5.1 The ETH Reduction Chain

The lower bound in Part 2 of Theorem 5.1 follows from a chain of reductions originating in the Exponential Time Hypothesis. We make this chain explicit.

Definition 5.2 (Exponential Time Hypothesis (ETH))**.** There exists a constant $\delta > 0$ such that 3-SAT on n variables cannot be solved in time $O(2^{\delta n})$ [15].

The chain proceeds as follows:

1. **ETH $\Rightarrow$ 3-SAT requires $2^{\Omega(n)}$:** This is the definition of ETH.
2. **3-SAT $\leq_p$ TAUTOLOGY:** Given 3-SAT formula $\varphi(x_1, \ldots, x_n)$, define $\psi = \neg\varphi$. Then φ is satisfiable iff ψ is not a tautology. This is a linear-time reduction preserving the number of variables.
3. **TAUTOLOGY requires $2^{\Omega(n)}$ (under ETH):** By the contrapositive of step 2, if TAUTOLOGY is solvable in $o(2^{\delta n})$ time, then 3-SAT is solvable in $o(2^{\delta n})$ time, contradicting ETH.
4. **TAUTOLOGY $\leq_p$ SUFFICIENCY-CHECK:** This is Theorem 4.6. Given formula $\varphi(x_1, \ldots, x_n)$, we construct a decision problem where:
   - The empty set $I = \emptyset$ is sufficient iff φ is a tautology
   - When φ is not a tautology, all n coordinates are relevant

The reduction is polynomial-time and preserves the number of coordinates.

5. **SUFFICIENCY-CHECK requires $2^{\Omega(n)}$ (under ETH):** Combining steps 3 and 4: if SUFFICIENCY-CHECK is solvable in $o(2^{\delta n/c})$ time for some constant c, then TAUTOLOGY (and hence 3-SAT) is solvable in subexponential time, contradicting ETH.

Proposition 5.3 (Tight Constant)**.** *The reduction in Theorem 4.6 preserves the number of variables up to an additive constant: an n-variable formula yields an $(n+1)$-coordinate decision problem. Therefore, the constant c in the $2^{n/c}$ lower bound is asymptotically 1:*

$$\textit{SUFFICIENCY-CHECK requires time } \Omega(2^{\delta(n-1)}) = 2^{\Omega(n)} \textit{ under ETH}$$

where δ is the ETH constant for 3-SAT.

Proof. The TAUTOLOGY reduction (Theorem 4.6) constructs:

- State space $S = \{\text{ref}\} \cup \{0,1\}^n$ with $n+1$ coordinates (one extra for the reference state)
- Query set $I = \emptyset$

When φ has n variables, the constructed problem has $n+1$ coordinates. The asymptotic lower bound is $2^{\Omega(n)}$ with the same constant δ from ETH. ■

5.2 Phase Transition

Corollary 5.4 (Phase Transition)**.** *There is a sharp transition between tractable and intractable regimes at the logarithmic scale (with respect to the encodings in Section 3.4):*

- *If $k^* = O(\log N)$, SUFFICIENCY-CHECK is polynomial in N under the explicit-state encoding*
- *If $k^* = \Omega(n)$, SUFFICIENCY-CHECK is exponential in n under ETH in the succinct encoding*

For Boolean coordinate spaces ($N = 2^n$), the transition is between $k^* = O(\log n)$ (explicit-state tractable) and $k^* = \Omega(n)$ (succinct-encoding intractable).

Proof. The logarithmic case (Part 1 of Theorem 5.1) gives polynomial time when $k^* = O(\log N)$. More precisely, when $k^* \leq c \log N$ for constant c, the algorithm runs in time $O(N^c \cdot \text{poly}(n))$.

The linear case (Part 2) gives exponential time when $k^* = \Omega(n)$.

The transition point is where $2^{k^*} = N^{k^*/\log N}$ stops being polynomial in N, i.e., when $k^* = \omega(\log N)$. For Boolean coordinate spaces ($N = 2^n$), this corresponds to the gap between $k^* = O(\log n)$ and $k^* = \Omega(n)$. ■

Remark 5.5 (Sharpness of Dichotomy). Under ETH, the lower bound is asymptotically tight in the succinct encoding. The explicit-state upper bound is tight in the sense that it matches enumeration complexity in N.

This dichotomy explains why some domains admit tractable model selection (few relevant variables) while others require heuristics (many relevant variables). The ETH reduction chain makes precise what "hard" means: not merely coNP-complete, but requiring $2^{\Omega(n)}$ time under widely-believed complexity assumptions.

Remark 5.6 (Circuit Model Formalization). The ETH lower bound is stated in the succinct circuit encoding of Section 3.4, where the utility function $U : A \times S \to \mathbb{R}$ is represented by a Boolean circuit computing $\mathbf{1}[U(a, s) > \theta]$ for threshold comparisons. In this model:

- The input size is the circuit size m, not the state space size $|S| = 2^n$
- A 3-SAT formula with n variables and c clauses yields a circuit of size $O(n + c)$
- The reduction preserves instance size up to constant factors: $m_{\text{out}} \leq 3 \cdot m_{\text{in}}$

This linear size preservation is essential for ETH transfer. In the explicit enumeration model (where S is given as a list), the reduction would blow up the instance size exponentially, precluding ETH-based lower bounds. The circuit model is standard in fine-grained complexity and matches practical representations of decision problems.

6 Tractable Special Cases

We distinguish the encodings of Section 3.4. The tractability results below state the model assumption explicitly. All results are formalized in `DecisionQuotient/Tractability/`.

Theorem 6.1 (Tractable Subcases)**.** *SUFFICIENCY-CHECK is polynomial-time solvable in the following cases:*

1. ***Explicit-state encoding:*** *The input contains the full utility table over $A \times S$. SUFFICIENCY-CHECK runs in $O(|S|^2|A|)$ time; if $|A|$ is constant, $O(|S|^2)$.*
2. ***Separable utility (any encoding):*** *$U(a, s) = f(a) + g(s)$.*
3. ***Tree-structured utility with explicit local factors (succinct structured encoding):*** *There exists a rooted tree on coordinates and local functions u_i such that*
$$U(a, s) = \sum_i u_i\big(a, s_i, s_{\text{parent}(i)}\big),$$
with the root term depending only on (a, s_{root}) and all u_i given explicitly as part of the input.

6.1 Explicit-State Encoding

Proof of Part 1. Given the full table of $U(a, s)$, compute $\text{Opt}(s)$ for all $s \in S$ in $O(|S||A|)$ time. For SUFFICIENCY-CHECK on a given I, verify that for all pairs (s, s') with $s_I = s'_I$, we have $\text{Opt}(s) = \text{Opt}(s')$. This takes $O(|S|^2|A|)$ time by direct enumeration and is polynomial in the explicit input length. If $|A|$ is constant, the runtime is $O(|S|^2)$. ∎

6.2 Separable Utility

Proof of Part 2. If $U(a, s) = f(a) + g(s)$, then:
$$\text{Opt}(s) = \arg\max_{a \in A}[f(a) + g(s)] = \arg\max_{a \in A} f(a)$$
The optimal action is independent of the state! Thus $I = \emptyset$ is always sufficient. ∎

6.3 Tree-Structured Utility

Proof of Part 3. Assume the tree decomposition and explicit local tables as stated. For each node i and each value of its parent coordinate, compute the set of actions that are optimal for some assignment of the subtree rooted at i. This is a bottom-up dynamic program that combines local tables with child summaries; each table lookup is explicit in the input. A coordinate is relevant if and only if varying its value changes the resulting optimal action set. The total runtime is polynomial in n, $|A|$, and the size of the local tables. ∎

6.4 Practical Implications

These tractable cases correspond to common modeling scenarios:

- **Explicit-state encoding:** Small or fully enumerated state spaces
- **Separable utility:** Additive cost models, linear utility functions
- **Tree-structured utility:** Hierarchical decision processes, causal models with tree structure

For problems given in the succinct encoding without these structural restrictions, the hardness results of Section 4 apply, justifying heuristic approaches.

7 Implications for Practice: Why Over-Modeling Is Optimal

This section states corollaries for engineering practice. Within the formal model, the complexity results of Sections 4 and 5 transform engineering practice from art to mathematics. The observed behaviors (configuration over-specification, absence of automated minimization tools, heuristic model selection) are not failures of discipline but *provably optimal responses* to computational constraints under the stated cost model.

The conventional critique of over-modeling ("identify only the essential variables") is computationally unrealistic in the general case. It asks engineers to solve coNP-complete problems. A rational response is to include everything and pay linear maintenance costs, rather than attempt exponential minimization costs.

7.1 Configuration Simplification is SUFFICIENCY-CHECK

Real engineering problems reduce directly to the decision problems studied in this paper.

Theorem 7.1 (Configuration Simplification Reduces to SUFFICIENCY-CHECK)**.** *Given a software system with configuration parameters $P = \{p_1, \ldots, p_n\}$ and observed behaviors $B = \{b_1, \ldots, b_m\}$, the problem of determining whether parameter subset $I \subseteq P$ preserves all behaviors is equivalent to SUFFICIENCY-CHECK.*

Proof. Construct decision problem $\mathcal{D} = (A, X_1, \ldots, X_n, U)$ where:

- Actions $A = B$ (each behavior is an action)
- Coordinates X_i = domain of parameter p_i
- State space $S = X_1 \times \cdots \times X_n$
- Utility $U(b, s) = 1$ if behavior b occurs under configuration s, else $U(b, s) = 0$

Then $\mathrm{Opt}(s) = \{b \in B : b \text{ occurs under configuration } s\}$.

Coordinate set I is sufficient iff:

$$s_I = s'_I \implies \mathrm{Opt}(s) = \mathrm{Opt}(s')$$

This holds iff configurations agreeing on parameters in I exhibit identical behaviors.

Therefore, "does parameter subset I preserve all behaviors?" is exactly SUFFICIENCY-CHECK for the constructed decision problem. ■

Remark 7.2. This reduction is *parsimonious*: every instance of configuration simplification corresponds bijectively to an instance of SUFFICIENCY-CHECK. The problems are not merely related; they are identical up to encoding.

7.2 Computational Rationality of Over-Modeling

We now prove that over-specification is an optimal engineering strategy given the stated cost model and complexity constraints.

Theorem 7.3 (Rational Over-Modeling)**.** *Consider an engineer specifying a system configuration with n parameters. Let:*

- $C_{over}(k)$ *= cost of maintaining k extra parameters beyond minimal*
- $C_{find}(n)$ *= cost of finding minimal sufficient parameter set*
- C_{under} *= expected cost of production failures from underspecification*

When SUFFICIENCY-CHECK is coNP*-complete (Theorem 4.6):*

1. *Worst-case finding cost is exponential:* $C_{find}(n) = \Omega(2^n)$
2. *Maintenance cost is linear:* $C_{over}(k) = O(k)$
3. *For sufficiently large n, exponential cost dominates linear cost*

Therefore, there exists n_0 such that for all $n > n_0$, over-modeling minimizes total expected cost:

$$C_{over}(k) < C_{find}(n) + C_{under}$$

Over-modeling is economically optimal under the stated model and complexity constraints.

Proof. By Theorem 4.6, SUFFICIENCY-CHECK is coNP-complete. Under standard complexity assumptions ($\mathsf{P} \neq \mathsf{coNP}$), no polynomial-time algorithm exists for checking sufficiency.

Finding the minimal sufficient set requires checking sufficiency of multiple candidate sets. Exhaustive search examines:

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n \text{ candidate subsets}$$

Each check requires $\Omega(1)$ time (at minimum, reading the input). Therefore:

$$C_{\text{find}}(n) = \Omega(2^n)$$

Maintaining k extra parameters incurs:

- Documentation cost: $O(k)$ entries
- Testing cost: $O(k)$ test cases
- Migration cost: $O(k)$ parameters to update

Total maintenance cost is $C_{\text{over}}(k) = O(k)$.

For concrete threshold: when $n = 20$ parameters, exhaustive search requires $2^{20} \approx 10^6$ checks. Including $k = 5$ extra parameters costs $O(5)$ maintenance overhead but avoids 10^6 computational work.

Since 2^n grows faster than any polynomial in k or n, there exists n_0 such that for all $n > n_0$:

$$C_{\text{over}}(k) \ll C_{\text{find}}(n)$$

Adding underspecification risk C_{under} (production failures from missing parameters), which is unbounded in the model, makes over-specification strictly dominant. ■

Corollary 7.4 (Impossibility of Automated Configuration Minimization)**.** *Assuming* $\mathsf{P} \neq \mathsf{coNP}$*, there exists no polynomial-time algorithm that:*

1. *Takes an arbitrary configuration file with* n *parameters*
2. *Identifies the minimal sufficient parameter subset*
3. *Guarantees correctness (no false negatives)*

Proof. Such an algorithm would solve MINIMUM-SUFFICIENT-SET in polynomial time, contradicting Theorem 4.7 (assuming $\mathsf{P} \neq \mathsf{coNP}$). ■

Remark 7.5. Corollary 7.4 explains the observed absence of "config cleanup" tools in software engineering practice. Engineers who include extra parameters are not exhibiting poor discipline; they are adapting optimally to computational impossibility under the model. The problem is not lack of tooling effort; it is mathematical intractability.

7.3 Connection to Observed Practice

These theorems provide mathematical grounding for three widespread engineering behaviors:

1. Configuration files grow over time. Removing parameters requires solving coNP-complete problems. Engineers rationally choose linear maintenance cost over exponential minimization cost.

2. Heuristic model selection dominates. ML practitioners use AIC, BIC, cross-validation instead of optimal feature selection because optimal selection is intractable (Theorem 7.3).

3. "Include everything" is a legitimate strategy. When determining relevance costs $\Omega(2^n)$, including all n parameters costs $O(n)$. For large n, this is the rational choice.

These are not merely workarounds or approximations. They are optimal responses under the stated model. The complexity results provide a mathematical lens: over-modeling is not a failure; it is the rational strategy under the model.

8 Experimental Validation

We validate our theoretical complexity bounds through synthetic experiments on randomly generated decision problems. These runtime experiments are separate from the Lean 4 formalization; they confirm that the asymptotic bounds predicted by the formal proofs match observed behavior. All experiments use the straightforward $O(|S|^2 \cdot |A|)$ algorithm for SUFFICIENCY-CHECK.

Model note. These experiments assume the explicit-state encoding of Section 3.4 (the input includes all states and utilities). The ETH-based lower bounds elsewhere assume the succinct circuit encoding. The results are complementary: the experiments validate scaling in the explicit model, while the ETH reductions rule out subexponential algorithms in the succinct model.

8.1 Runtime Scaling with State Space Size

Our theory predicts $O(|S|^2)$ runtime scaling. Table 1 confirms this prediction; the normalized ratio $t/|S|^2$ remains constant as $|S|$ grows.

8.2 Runtime vs. Action Space Size

Fixing $|S| = 200$, we vary $|A|$. The runtime is approximately constant because set equality comparison (for $\text{Opt}(s) = \text{Opt}(s')$) dominates only for very large $|A|$.

$\|S\|$	Time (ms)	$t/\|S\|^2 \times 10^6$
50	0.96	383
100	3.21	321
200	12.86	322
400	53.29	333
800	211.77	331

Table 1: State space scaling. The normalized ratio is approximately constant, confirming $O(|S|^2)$ complexity.

$\|A\|$	Time (ms)
2	12.53
10	12.68
50	13.59
100	12.94

Table 2: Action space scaling. Runtime is nearly constant for moderate $|A|$.

8.3 Early Termination

A key practical observation: in our experiments, when I is *not* sufficient, the algorithm terminates early upon finding a counterexample pair (s, s') with $s \sim_I s'$ but $\mathrm{Opt}(s) \neq \mathrm{Opt}(s')$.

Table 3 shows dramatic speedups for insufficient sets versus sufficient ones (which require full traversal).

$\|S\|$	Sufficient (ms)	Insufficient (ms)	Speedup
100	3.26	0.013	256×
200	13.46	0.012	1,134×
400	53.98	0.012	4,383×
800	212.41	0.012	17,690×

Table 3: Early termination speedups. Insufficient sets (empty set $I = \emptyset$) are detected almost instantly because counterexamples are found early.

8.4 Implications

These experiments validate our theoretical predictions:

1. **Quadratic scaling**: The $O(|S|^2)$ bound is tight in practice. For $|S| = 1000$, expect ~ 300ms on commodity hardware.

2. **Action-independence**: For bounded $|A|$, the FPT result (Theorem 6.1) is reflected in the data: runtime is dominated by state-pair enumeration, not action comparison.

3. **Early termination**: Most *wrong* candidate sets are rejected almost instantly. This makes greedy search for minimal sufficient sets practical despite the worst-case coNP-hardness.

The experiments use $|A| = 5$ actions with deterministic optimal action per state (ensuring full traversal is required for sufficient sets). Code is available in the supplementary material.

9 Implications for Software Architecture

This section states corollaries for software architecture. The complexity results have direct implications for software engineering practice.

9.1 Why Over-Specification Is Rational

Software architects routinely specify more configuration parameters than strictly necessary. Our results show this is computationally rational:

Corollary 9.1 (Rational Over-Specification)**.** *Given a software system with n configuration parameters, checking whether a proposed subset suffices is* coNP*-complete. Finding the minimum such set is also* coNP*-complete.*

This explains why configuration files grow over time: removing "unnecessary" parameters requires solving a hard problem.

9.2 Connection to Leverage Theory

Paper 3 introduced leverage as the ratio of impact to effort. The decision quotient provides a complementary measure:

Definition 9.2 (Architectural Decision Quotient)**.** For a software system with configuration space S and behavior space B:

$$\mathrm{ADQ}(I) = \frac{|\{b \in B : b \text{ achievable with some } s \text{ where } s_I \text{ fixed}\}|}{|B|}$$

High ADQ means the configuration subset I leaves many behaviors achievable; it doesn't constrain the system much. Low ADQ means I strongly constrains behavior.

Proposition 9.3 (Leverage-ADQ Duality)**.** *High-leverage architectural decisions correspond to low-ADQ configuration subsets: they strongly constrain system behavior with minimal specification.*

9.3 Practical Recommendations

Based on our theoretical results:

1. **Accept over-modeling:** Don't penalize engineers for including "extra" parameters. The alternative (minimal modeling) is computationally hard.
2. **Use bounded scenarios:** When the scenario space is small (Proposition 3.10), minimal modeling becomes tractable.
3. **Exploit structure:** Tree-structured dependencies, bounded alternatives, and separable utilities admit efficient algorithms.
4. **Invest in heuristics:** For general problems, develop domain-specific heuristics rather than seeking optimal solutions.

9.4 Hardness Distribution: Right Place vs Wrong Place

A general principle emerges from the complexity results: problem hardness is conserved and is *distributed* across a system in qualitatively different ways.

Definition 9.4 (Hardness Distribution)**.** Let P be a problem with intrinsic hardness $H(P)$ (measured in computational steps, implementation effort, or error probability). A *solution architecture* S partitions this hardness into:

- $H_{\text{central}}(S)$: hardness paid once, at design time or in a shared component
- $H_{\text{distributed}}(S)$: hardness paid per use site

For n use sites, total realized hardness is:

$$H_{\text{total}}(S) = H_{\text{central}}(S) + n \cdot H_{\text{distributed}}(S)$$

Theorem 9.5 (Hardness Conservation)**.** *For any problem P with intrinsic hardness $H(P)$, any solution S satisfies:*

$$H_{central}(S) + H_{distributed}(S) \geq H(P)$$

Hardness cannot be eliminated, only redistributed.

Proof. By definition of intrinsic hardness: any correct solution must perform at least $H(P)$ units of work (computational, cognitive, or error-handling). This work is either centralized or distributed. ∎

Definition 9.6 (Hardness Efficiency)**.** The *hardness efficiency* of solution S with n use sites is:

$$\eta(S, n) = \frac{H_{\text{central}}(S)}{H_{\text{central}}(S) + n \cdot H_{\text{distributed}}(S)}$$

High η indicates centralized hardness (paid once); low η indicates distributed hardness (paid repeatedly).

Theorem 9.7 (Centralization Dominance)**.** *For $n > 1$ use sites, solutions with higher $H_{central}$ and lower $H_{distributed}$ yield:*

1. *Lower total realized hardness: $H_{total}(S_1) < H_{total}(S_2)$ when $H_{distributed}(S_1) < H_{distributed}(S_2)$*
2. *Fewer error sites: errors in centralized components affect 1 location; errors in distributed components affect n locations*
3. *Higher leverage (Paper 3): one unit of central effort affects n sites*

Proof. (1) follows from the total hardness formula. (2) follows from error site counting. (3) follows from Paper 3's leverage definition $L = \Delta\text{Effect}/\Delta\text{Effort}$. ∎

Corollary 9.8 (Right Hardness vs Wrong Hardness)**.** *A solution exhibits* hardness in the right place *when:*

- *Hardness is centralized (high $H_{central}$, low $H_{distributed}$)*
- *Hardness is paid at design/compile time rather than runtime*
- *Hardness is enforced by tooling (type checker, compiler) rather than convention*

A solution exhibits hardness in the wrong place *when:*

- *Hardness is distributed (low $H_{central}$, high $H_{distributed}$)*
- *Hardness is paid repeatedly at each use site*
- *Hardness relies on human discipline rather than mechanical enforcement*

Example: Type System Instantiation. Consider a capability C (e.g., provenance tracking) that requires hardness $H(C)$:

Approach	H_{central}	$H_{\text{distributed}}$
Native type system support	High (learning cost)	Low (type checker enforces)
Manual implementation	Low (no new concepts)	High (reimplement per site)

For n use sites, manual implementation costs $n \cdot H_{\text{distributed}}$, growing without bound. Native support costs H_{central} once, amortized across all uses. The "simpler" approach (manual) is only simpler at $n = 1$; for $n > H_{\text{central}}/H_{\text{distributed}}$, native support dominates.

Remark 9.9 (Connection to Decision Quotient). The decision quotient (Section 3) measures which coordinates are decision-relevant. Hardness distribution measures where the cost of *handling* those coordinates is paid. A high-axis system makes relevance explicit (central hardness); a low-axis system requires users to track relevance themselves (distributed hardness).

The next section develops the major practical consequence of this framework: the Simplicity Tax Theorem.

10 The Simplicity Tax Theorem

This section states a practical corollary of the complexity results. Sections 4–6 establish that identifying decision-relevant dimensions is coNP-complete. This section develops the consequence: what happens when engineers *ignore* this hardness and attempt to use "simple" tools for complex problems.

The answer is the *Simplicity Tax*: a per-site cost that cannot be avoided, only redistributed. This result refutes the intuition that "simpler is always better" whenever the tool is incomplete for the problem, and establishes the principle: **No Free Simplicity**.

10.1 The Conservation Law: No Free Simplicity

Definition 10.1 (Problem and Tool)**.** A *problem* P has a set of *required axes* $R(P)$ (the dimensions of variation that must be represented). A *tool* T has a set of *native axes* $A(T)$ (what it represents directly).

This terminology is grounded in Papers 1–2: "axes" correspond to Paper 1's axis framework (`requiredAxesOf`) and Paper 2's degrees of freedom.

Definition 10.2 (Expressive Gap and Simplicity Tax)**.** The *expressive gap* between tool T and problem P is:

$$\mathrm{Gap}(T, P) = R(P) \setminus A(T)$$

The *simplicity tax* is $|\mathrm{Gap}(T, P)|$: the number of axes the tool does not handle natively. This tax is paid at *every use site*.

Definition 10.3 (Complete vs. Incomplete Tools). Tool T is *complete* for problem P if $R(P) \subseteq A(T)$. Otherwise T is *incomplete* for P.

Theorem 10.4 (Simplicity Tax Conservation). *For any problem P with required axes $R(P)$ and any tool T:*

$$|Gap(T, P)| + |R(P) \cap A(T)| = |R(P)|$$

The required axes are partitioned into "covered natively" and "tax." You cannot reduce the total; only shift where it is paid.

Proof. Set partition: $R(P) = (R(P) \cap A(T)) \cup (R(P) \setminus A(T))$. The sets are disjoint. Cardinality follows. ■

This is analogous to conservation laws in physics: energy is conserved, only transformed. Complexity is conserved, only distributed.

Theorem 10.5 (Complete Tools Pay No Tax). *If T is complete for P, then $SimplicityTax(T, P) = 0$.*

Theorem 10.6 (Incomplete Tools Pay Positive Tax). *If T is incomplete for P, then $SimplicityTax(T, P) > 0$.*

Theorem 10.7 (Simplicity Tax Grows Linearly). *For n use sites with an incomplete tool:*

$$TotalExternalWork(T, P, n) = n \times SimplicityTax(T, P)$$

Total work grows linearly. There is no economy of scale for distributed complexity.

Theorem 10.8 (Complete Dominates Incomplete). *For any $n > 0$, a complete tool has strictly less total cost than an incomplete tool:*

$$TotalExternalWork(T_{complete}, P, n) < TotalExternalWork(T_{incomplete}, P, n)$$

Proof. Complete: $0 \times n = 0$. Incomplete: $k \times n$ for $k \geq 1$. For $n > 0$: $0 < kn$. ■

10.2 The Simplicity Preference Fallacy

Definition 10.9 (Simplicity Preference Fallacy). The *simplicity preference fallacy* is the cognitive error of preferring low H_{central} (learning cost) without accounting for $H_{\text{distributed}}$ (per-site cost).

This fallacy manifests as:

- "I prefer simple tools" (without asking: simple relative to what problem?)
- "YAGNI" applied to infrastructure (ignoring amortization across use sites)
- "Just write straightforward code" (ignoring that n sites pay the tax)
- "Abstractions are overhead" (treating central cost as total cost)

Theorem 10.10 (The Fallacy Theorem). *Let T_{simple} be incomplete for problem P and $T_{complex}$ be complete. For any $n > 0$:*

$$TotalExternalWork(T_{complex}, P, n) < TotalExternalWork(T_{simple}, P, n)$$

The "simpler" tool creates more total work, not less.

The fallacy persists because $H_{\text{distributed}}$ is invisible: it is paid by users, at runtime, across time, in maintenance. H_{central} is visible: it is paid by the designer, upfront, once. Humans overweight visible costs.

Theorem 10.11 (Amortization Threshold)**.** *There exists a threshold n^* such that for all $n > n^*$, the total cost of the "complex" tool (including learning) is strictly less than the "simple" tool:*

$$n^* = \frac{H_{central}(T_{complex})}{SimplicityTax(T_{simple}, P)}$$

Beyond n^* uses, the complex tool is cheaper even accounting for learning cost.

Remark 10.12 (On the Learning Cost Model)*.* This theorem models learning cost as H_{central}, a scalar. A more precise formalization treats learning cost as the rank of a *concept matroid* (the prerequisite concepts required to master the tool; see Conclusion, Future Work). Paper 1 established that type axes form matroids; we conjecture that concept axes admit similar structure. Critically, the matroid property ensures that *different minimal learning paths have equal cardinality*, making the scalar well-defined despite multiple valid trajectories. The qualitative result (amortization threshold exists) is robust to the learning cost model; the quantitative threshold depends on its precise formalization.

10.3 Cross-Domain Examples

The Simplicity Tax applies to the following domains:

Domain	**"Simple" Choice**	**"Complex" Choice**	**Tax per Site**
Type Systems	Dynamic typing	Static typing	Runtime type errors
Python	Manual patterns	Metaclasses/descriptors	Boilerplate code
Data Validation	Ad-hoc checks	Schema/ORM	Validation logic
Configuration	Hardcoded values	Config management	Change propagation
APIs	Stringly-typed	Rich type models	Parse/validate code

In each case, the "simple" choice has lower learning cost (H_{central}) but higher per-site cost ($H_{\text{distributed}}$). For n use sites, the simple choice costs $n \times$ tax.

Example: Python Metaclasses. Python's community resists metaclasses as "too complex." But consider a problem requiring automatic subclass registration, attribute validation, and interface enforcement (three axes of variation).

Approach	**Native Axes**	**Tax/Class**	**Total for 50 classes**
Metaclass	$\{$registration, validation, interface$\}$	0	0
Manual decorators	$\{$registration$\}$	2	100
Fully manual	$\emptyset$	3	150

The "simplest" approach (fully manual) creates the most work. The community's resistance to metaclasses is the Simplicity Preference Fallacy in action.

Example: Static vs. Dynamic Typing. Dynamic typing has lower learning cost. But type errors are a per-site tax: each call site that admits a wrong type in some execution is an error site. For n call sites:

- Static typing: type checker verifies once, 0 runtime type errors
- Dynamic typing: n error sites, each requiring defensive code or debugging when a wrong type arrives

The "simplicity" of dynamic typing distributes type-checking to every call site.

10.4 Unification with Papers 1–3

The Simplicity Tax Theorem unifies results across the pentalogy:

Paper 1 (Typing Disciplines). Fixed-axis type systems are incomplete for domains requiring additional axes. The Simplicity Tax quantifies the cost: $|\text{requiredAxes}(D) \setminus \text{fixedAxes}|$ per use site. Parameterized type systems are complete (zero tax).

Paper 2 (SSOT). Non-SSOT architectures distribute specification across n locations. Each location is a potential error site. SSOT centralizes specification: $H_{\text{distributed}} = 0$.

Paper 3 (Leverage). High-leverage solutions have high H_{central} and low $H_{\text{distributed}}$. Leverage = impact/effort = n/H_{central} when $H_{\text{distributed}} = 0$. Low-leverage solutions pay per-site.

Paper 4 (This Paper). Identifying which axes matter is coNP-complete. If you guess wrong and use an incomplete tool, you pay the Simplicity Tax. The tax is the cost of the coNP-hard problem you did not solve.

Theorem 10.13 (Unified Dominance)**.** *Across Papers 1–4, solutions with higher $H_{central}$ and zero $H_{distributed}$ strictly dominate solutions with lower $H_{central}$ and positive $H_{distributed}$, for $n > n^*$.*

These are not four separate claims. They are four views of a single phenomenon: the conservation and distribution of intrinsic problem complexity.

10.5 Formal Competence

Definition 10.14 (Formal Competence)**.** An engineer is *formally competent* with respect to complexity distribution if they correctly account for both H_{central} and $H_{\text{distributed}}$ when evaluating tools.

The Competence Test:

1. Identify intrinsic problem complexity: $|R(P)|$
2. Identify tool's native axes: $|A(T)|$
3. Compute the gap: $|R(P) \setminus A(T)|$
4. Compute total cost: $H_{\text{central}}(T) + n \times |R(P) \setminus A(T)|$
5. Compare tools by total cost, not H_{central} alone

Failing step 4 (evaluating tools by learning cost alone) is a formal mistake.

Remark 10.15 (The Zen of Python, Correctly Read). Python's Zen states: "Simple is better than complex. Complex is better than complicated." This is misread as endorsing simplicity unconditionally. The correct reading:

- **Simple**: Low intrinsic complexity (both H_{central} and $H_{\text{distributed}}$ low)
- **Complex**: High intrinsic complexity, *structured* (high H_{central}, low $H_{\text{distributed}}$)
- **Complicated**: High intrinsic complexity, *tangled* (low H_{central}, high $H_{\text{distributed}}$)

The Zen says: when the problem has intrinsic complexity, *complex* (centralized) beats *complicated* (distributed). The community conflates complex with complicated.

10.6 Lean 4 Formalization

All theorems in this section are machine-checked in `DecisionQuotient/HardnessDistribution.lean`:

Theorem	Lean Name
Simplicity Tax Conservation	`simplicityTax_conservation`
Complete Tools Pay No Tax	`complete_tool_no_tax`
Incomplete Tools Pay Positive Tax	`incomplete_tool_positive_tax`
Tax Grows Linearly	`simplicityTax_grows`
Complete Dominates Incomplete	`complete_dominates_incomplete`
The Fallacy Theorem	`simplicity_preference_fallacy`
Amortization Threshold	`amortization_threshold`
Dominance Transitivity	`dominates_trans`
Tax Antitone w.r.t. Expressiveness	`simplicityTax_antitone`

The formalization uses `Finset` $\mathbb{N}$ for axes, making the simplicity tax a computable natural number. The `Tool` type forms a lattice under the expressiveness ordering, with tax antitone (more expressive $\Rightarrow$ lower tax).

All proofs compile with zero `sorry` placeholders.

11 Related Work

11.1 Formalized Complexity Theory

Machine verification of complexity-theoretic results remains sparse compared to other areas of mathematics. We survey existing work and position our contribution.

Coq formalizations. Forster et al. [10] developed a Coq library for computability theory, including undecidability proofs. Their work focuses on computability rather than complexity classes. Kunze et al. [17] formalized the Cook-Levin theorem in Coq, proving SAT is NP-complete. Our work extends this methodology to coNP-completeness and approximation hardness.

Isabelle/HOL. Nipkow and colleagues formalized substantial algorithm verification in Isabelle [22], but complexity-theoretic reductions are less developed. Recent work on algorithm complexity [13] provides time bounds for specific algorithms rather than hardness reductions.

Lean and Mathlib. Mathlib's computability library [35] provides primitive recursive functions and basic computability results. Our work extends this to polynomial-time reductions and complexity classes. To our knowledge, this is the first Lean 4 formalization of coNP-completeness proofs and approximation hardness.

The verification gap. Published complexity proofs occasionally contain errors [20]. Machine verification eliminates this uncertainty. Our contribution demonstrates that complexity reductions are amenable to formalization with reasonable effort (7071 lines for the full reduction suite).

11.2 Computational Decision Theory

The complexity of decision-making has been studied extensively. Papadimitriou [23] established foundational results on the complexity of game-theoretic solution concepts. Our work extends this to the meta-question of identifying relevant information. For a modern treatment of complexity classes, see Arora and Barak [3].

Closest prior work and novelty. Closest to our contribution is the literature on feature selection and model selection hardness, which proves NP-hardness of selecting informative feature subsets and inapproximability for minimum feature sets [4, 2]. Those results analyze predictive relevance or compression objectives. We study decision relevance and show coNP-completeness for sufficiency checking, a different quantifier structure (universal verification) with distinct proof techniques and a full hardness/tractability landscape under explicit encoding assumptions, mechanized in Lean 4. The formalization aspect is also novel: prior work establishes hardness on paper, while we provide machine-checked reductions with explicit polynomial bounds.

11.3 Feature Selection Complexity

In machine learning, feature selection asks which input features are relevant for prediction. Blum and Langley [4] survey the field, noting hardness in general settings. Amaldi and Kann [2] proved that finding minimum feature sets for linear classifiers is NP-hard, and established inapproximability bounds: no polynomial-time algorithm approximates the minimum feature set within factor $2^{\log^{1-\epsilon} n}$ unless NP $\subseteq$ DTIME($n^{\text{polylog } n}$).

Our results extend this line: the decision-theoretic analog (SUFFICIENCY-CHECK) is coNP-complete, and MINIMUM-SUFFICIENT-SET inherits this hardness. The key insight is that sufficiency checking is "dual" to feature selection; rather than asking which features predict a label, we ask which coordinates determine optimal action. The coNP (rather than NP) classification reflects this duality: insufficiency has short certificates (counterexample state pairs), while sufficiency requires universal verification.

11.4 Sufficient Statistics

Fisher [9] introduced sufficient statistics: a statistic $T(X)$ is *sufficient* for parameter θ if the conditional distribution of X given $T(X)$ does not depend on θ. Lehmann and Scheffé [18] characterized minimal sufficient statistics and their uniqueness properties.

Our coordinate sufficiency is the decision-theoretic analog: a coordinate set I is sufficient if knowing s_I determines optimal action, regardless of the remaining coordinates. The parallel is precise:

- **Statistics:** T is sufficient $\iff$ $P(X|T(X), \theta) = P(X|T(X))$
- **Decisions:** I is sufficient $\iff$ $\text{Opt}(s) = \text{Opt}(s')$ whenever $s_I = s'_I$

Fisher's factorization theorem provides a characterization; our Theorem 4.7 shows that *finding* minimal sufficient statistics (in the decision-theoretic sense) is computationally hard.

11.5 Causal Inference and Adjustment Sets

Pearl [24] and Spirtes et al. [31] developed frameworks for identifying causal effects from observational data. A central question is: which variables must be adjusted for to identify a causal effect? The *adjustment criterion* and *back-door criterion* characterize sufficient adjustment sets.

Our sufficiency problem is analogous: which coordinates must be observed to determine optimal action? We conjecture that optimal adjustment set selection is intractable; recent work on the complexity of causal discovery supports this [5].

The connection runs deeper: Shpitser and Pearl [29] showed that identifying causal effects is NP-hard in general graphs. Our coNP-completeness result for SUFFICIENCY-CHECK is the decision-theoretic counterpart.

11.6 Minimum Description Length and Kolmogorov Complexity

The Minimum Description Length (MDL) principle [26, 11] formalizes model selection as compression: the best model minimizes description length of data plus model. Kolmogorov complexity [19] provides the theoretical foundation (the shortest program that generates the data).

Our decision quotient connects to this perspective: a coordinate set I is sufficient if it compresses the decision problem without loss—knowing s_I is as good as knowing s for decision purposes. The minimal sufficient set is the MDL-optimal compression of the decision problem.

The complexity results explain why MDL-based model selection uses heuristics: finding the true minimum description length is uncomputable (Kolmogorov complexity) or intractable (MDL approximations). Our results show the decision-theoretic analog is coNP-complete—intractable but decidable.

11.7 Value of Information

The value of information (VOI) framework [14] quantifies the maximum rational payment for information. Our work addresses a different question: not the *value* of information, but the *complexity* of identifying which information has value.

In explicit-state representations, VOI is polynomial to compute in the input size, while identifying which information *to value* (our problem) is coNP-complete. This separation explains why VOI is practical while optimal sensor placement remains heuristic.

11.8 Sensitivity Analysis

Sensitivity analysis asks how outputs change with inputs. Local sensitivity (derivatives) is polynomial; global sensitivity (Sobol indices [30]) requires sampling. Identifying which inputs *matter* for decision-making is our sufficiency problem, which we show is coNP-complete.

This explains why practitioners use sampling-based sensitivity analysis rather than exact methods: exact identification of decision-relevant inputs is intractable. The dichotomy theorem (Theorem 5.1) characterizes when sensitivity analysis becomes tractable (logarithmic relevant inputs) versus intractable (linear relevant inputs).

11.9 Model Selection

Statistical model selection (AIC [1], BIC [28], cross-validation [33]) provides practical heuristics for choosing among models. Our results provide theoretical justification: optimal model selection is intractable, so heuristics are necessary.

The Simplicity Tax Theorem (Section 10) adds a warning: model selection heuristics that favor "simpler" models incur hidden costs when the true model is complex. The simplicity preference fallacy (choosing low-parameter models without accounting for per-site costs) is the decision-theoretic formalization of overfitting-by-underfitting.

12 Proof Engineering Insights

This section discusses lessons learned from formalizing complexity-theoretic reductions in Lean 4, intended to guide future formalization efforts.

12.1 Patterns That Worked

Bundled reductions. Packaging the reduction function, correctness proof, and polynomial bound into a single structure (`KarpReduction`) was essential. Early attempts using separate lemmas led to proof state explosion when composing reductions.

Definitional equality for simple cases. We defined concepts so that simple cases reduce definitionally whenever the definitions admit it:

```
-- Sufficiency of all coordinates is definitionally true
example (D : DecisionProblemWithCoords n) :
    Sufficient D Finset.univ := fun _ _ _ => rfl
```

Separation of polynomial bounds. We prove polynomial-time bounds separately from correctness, then combine. This mirrors the structure of pen-and-paper proofs and makes debugging easier.

Explicit size functions. Rather than relying on implicit encodings, we define explicit `size` functions for each type. This avoids universe issues and makes polynomial bounds concrete:

```
def size_formula : Formula → ℕ
  | .var _ => 1
  | .not φ => 1 + size_formula φ
  | .and φ ψ => 1 + size_formula φ + size_formula ψ
  | .or φ ψ => 1 + size_formula φ + size_formula ψ
```

12.2 Patterns That Failed

Unbundled type classes. Early attempts used type classes for complexity properties:

```
class InNP (P : DecisionProblem α) where
  witness_type : Type*
  verify : α → witness_type → Prop
  ...
```

This failed because instance search couldn't handle the necessary universe polymorphism. Bundled structures with explicit witnesses worked better.

Definitional unfolding for reductions. Attempting to make reduction correctness hold by `rfl` led to unwieldy definitions. It's better to accept that `correct` requires a short proof.

Direct SAT encoding. Our first reduction encoded SAT variables as coordinates directly. This required dependent types indexed by the number of variables, causing universe issues. The solution: encode via finite types with explicit bounds.

12.3 Challenges Specific to Complexity Theory

Polynomial composition. Proving that polynomial-time reductions compose to polynomial-time requires polynomial arithmetic. Mathlib's `Polynomial` provides this, but connecting abstract polynomials to concrete time bounds requires care.

The oracle model. For Σ_2^{P}-completeness, we need oracle Turing machines. We model these abstractly:

```
structure OracleTM (Oracle : Type*) where
  query : Oracle → Bool
  compute : (Oracle → Bool) → Input → Output
```

Full Turing machine formalization is future work; our proofs work at the reduction level.

ETH and SETH. Conditional lower bounds require assuming the Exponential Time Hypothesis. We encode this as an axiom in a separate file, clearly marked:

```
-- This is an ASSUMPTION, not a theorem
axiom ETH : ¬∃ (f : Formula → Bool),
  (∀ φ, f φ = true ↔ Satisfiable φ) ∧
  ∃ c < 2, ∀ n, time_on_size f n ≤ c ^ n
```

Parameterized complexity. The W-hierarchy requires careful definition. We model W[t] via weighted satisfiability:

```
def InW (t : ℕ) (P : DecisionProblem α) : Prop :=
  ∃ (reduce : α → WeightedFormula t),
    ∀ x, P x ↔ (reduce x).satisfiable
```

12.4 Automation Opportunities

Automation targets for repetitive proof patterns:

1. **Reduction templates:** Given a mapping and correctness statement, generate the `KarpReduction` structure.
2. **Polynomial bound synthesis:** Given a recursive function, synthesize its polynomial bound from the recurrence.
3. **Witness extraction:** For NP membership, automatically extract witness types from existential statements.
4. **Counterexample search:** For coNP-hardness, search for counterexamples to proposed reductions.

We implemented (1) as a macro; (2)–(4) remain manual. A `complexity` tactic analogous to `continuity` or `measurability` is the direct extension for automating routine reduction steps.

12.5 Recommendations for Future Work

Start with membership, then hardness. Proving "$P \in$ coNP" is usually easier than "P is coNP-hard." The membership proof clarifies the witness structure needed for the hardness reduction.

Formalize the target problem first. Before reducing from TAUTOLOGY to SUFFICIENCY-CHECK, we formalized SUFFICIENCY-CHECK completely. This caught encoding issues early.

Use `#print axioms` continuously. We ran axiom checks after each major lemma. This caught unintended classical dependencies in constructive components.

Separate "math" from "encoding." Keep the mathematical content ("sufficiency is the same as tautology under this encoding") separate from encoding details ("how to represent formulas as coordinates"). This separation aids both clarity and reuse.

12.6 Formalization Size

- Total Lean lines: 7071
- Theorem/lemma statements: 310
- Proof files: 42
- `sorry` placeholders: 0

These totals are generated from the current proof artifact at build time.

13 Conclusion

We have presented a Lean 4 framework for formalizing polynomial-time reductions and demonstrated it through a comprehensive complexity analysis of decision-relevant information.

Formalization Contributions

The primary contributions are methodological:

1. **Reusable reduction infrastructure.** The bundled `KarpReduction` type, polynomial bound tracking, and composition lemmas provide reusable infrastructure for future complexity formalizations.
2. **Demonstrated methodology.** Five complete reduction proofs (TAUTOLOGY, SET-COVER, ETH, W[2], and the Simplicity Tax) show that complexity-theoretic arguments are tractable to formalize with $\sim$600 lines per reduction.
3. **Integration patterns.** We show how to connect custom complexity definitions with Mathlib's computability and polynomial libraries.
4. **Artifact quality.** Zero `sorry` placeholders, documented axiom dependencies, and reproducible builds via `lake build`.

Verified Complexity Results

Through the case study, we machine-verified:

- Checking whether a coordinate set is sufficient is coNP-complete
- Finding the minimum sufficient set is coNP-complete (the Σ_2^{P} structure collapses)
- Anchor sufficiency (fixed-coordinate subcube) is Σ_2^{P}-complete
- A complexity dichotomy separates easy (logarithmic) from hard (linear) cases
- Tractable subcases exist for explicit-state encoding, separable utility, and tree-structured utility with explicit local factors

These results establish a fundamental principle of rational decision-making under uncertainty within the formal model:

Determining what you need to know is harder than knowing everything.

This is not a metaphor or heuristic observation. It is a mathematical theorem with universal scope *within the formal model*. Any agent facing structured uncertainty (whether a climate scientist, financial analyst, software engineer, or artificial intelligence) faces the same computational constraint when their decision problem fits the coordinate-structured encoding. The ubiquity of over-modeling across domains is not coincidence, laziness, or poor discipline. It is a rational response to intractability in the general case.

The principle has immediate normative force: criticizing engineers for including "irrelevant" parameters or demanding minimal models conflicts with computational reality in the general case. The dichotomy theorem (Theorem 5.1) characterizes when tractability holds; outside those boundaries, over-modeling is not a failure mode—it is the rational strategy under the model.

All proofs are machine-checked in Lean 4, ensuring correctness of the core mathematical claims including the reduction mappings and equivalence theorems. Complexity classifications follow from standard complexity-theoretic results (TAUTOLOGY is coNP-complete, $\exists\forall$-SAT is Σ_2^{P}-complete) under the encoding model described in Section 3.4.

Why These Results Are Tight (Within the Formal Model)

The theorems proven here are *tight within the formal model and standard assumptions*: stronger claims would require changing the model, the encoding, or the complexity assumptions.

1. **Exact complexity characterization (not just lower bounds).** We prove SUFFICIENCY-CHECK is coNP-complete (Theorem 4.6). This is *both* a lower bound (coNP-hard) and an upper bound (in coNP). The complexity class is exact. Additional lower or upper bounds would be redundant.
2. **Universal over the formal model ($\forall$), not probabilistic prevalence ($\mu = 1$).** Theorems quantify over *all* decision problems satisfying the stated structural constraints and encodings, not measure-1 subsets. Measure-theoretic claims like "hard instances are prevalent" would *weaken* the results from "always hard (unless P = coNP)" to "almost always hard."

3. **Constructive reductions, not existence proofs.** Theorem 4.6 provides an explicit polynomial-time reduction from TAUTOLOGY to SUFFICIENCY-CHECK. This is stronger than proving hardness via non-constructive arguments (e.g., diagonalization). The reduction is machine-checked and executable.

4. **Dichotomy separates logarithmic and linear regimes (Theorem 5.1).** The complexity separates into polynomial behavior when the minimal sufficient set has size $O(\log |S|)$ and exponential behavior under ETH when size is $\Omega(n)$. Intermediate regimes are not ruled out by ETH and fall outside the lower-bound statement.

5. **Explicit tractability conditions (Section 6).** The tractability conditions are stated with explicit encoding assumptions (Section 3.4).

What would NOT strengthen the results:

- **Additional complexity classes:** SUFFICIENCY-CHECK is coNP-complete. Proving it is also NP-hard, PSPACE-hard, or #P-hard would add no information (these follow from coNP-completeness under standard reductions).

- **Average-case hardness:** We prove worst-case hardness. Average-case results would be *weaker* (average $\leq$ worst) and would require distributional assumptions not present in the problem definition.

- **#P-hardness of counting:** When the decision problem is asking "does there exist?" (existential) or "are all?" (universal), the corresponding counting problem is trivially at least as hard. Proving #P-hardness separately would be redundant unless we change the problem to count something else.

- **Approximation hardness beyond the proved bound:** We prove $(1 - \varepsilon) \ln n$ inapproximability via L-reduction. Stronger inapproximability statements would require additional assumptions or different reductions.

These results close the complexity landscape for coordinate sufficiency within the formal model. Within classical complexity theory, the characterization is tight for the encodings considered.

The Simplicity Tax: A Major Practical Consequence

A widespread belief holds that "simpler is better" (that preferring simple tools and minimal models is a mark of sophistication). Within the formal model, that belief is false whenever the tool is incomplete for the problem.

The *Simplicity Tax Theorem* (Section 10) establishes: when a problem requires k axes of variation and a tool natively supports only $j < k$ of them, the remaining $k - j$ axes must be handled externally at *every use site*. For n use sites, the "simpler" tool creates $(k - j) \times n$ units of external work. A tool matched to the problem's complexity creates zero external work.

Corollary (Simplicity Tax). For any incomplete tool and any $n > 0$ use sites, total external work is strictly larger than for a complete tool, and the gap grows linearly in n (Theorem 10.8).

Interpretation (within the finite coordinate model; any finite decision problem admits such an encoding).

> **True sophistication is matching tool complexity to problem complexity.**

Preferring "simple" tools for complex problems is not wisdom; it is a failure to account for distributed costs. The simplicity tax is paid invisibly, at every use site, by every user, forever. The sophisticated engineer asks not "which tool is simpler?" but "which tool matches my problem's intrinsic complexity?"

This result is machine-checked in Lean 4 (`HardnessDistribution.lean`). The formalization proves conservation (you cannot eliminate the tax, only redistribute it), dominance (complete tools always beat incomplete tools), and the amortization threshold (beyond which the "complex" tool is strictly cheaper).

Scope and Implications

This paper proves a universal constraint on optimization under uncertainty *within the formal model.* The constraint is:

- **Mathematical**, not empirical: it follows from the structure of computation
- **Universal within the model**, not domain-specific: it applies to any decision problem with coordinate structure as defined in Section 3
- **Robust under standard assumptions**, not provisional: no algorithmic breakthrough circumvents coNP-completeness (unless P = coNP)

The result explains phenomena across disciplines *within the scope of the model*: why feature selection uses heuristics, why configuration files grow, why sensitivity analysis is approximate, why model selection is art rather than science. These are not separate problems with separate explanations. They are manifestations of a single computational constraint, now formally characterized.

The Simplicity Tax Theorem yields the corollary stated above: for any incomplete tool and any $n > 0$ use sites, total external work is strictly larger than for a complete tool (Theorem 10.8). Therefore, avoiding over-modeling by choosing an incomplete "simpler" tool increases total work.

Open questions remain (fixed-parameter tractability, quantum complexity, average-case behavior under natural distributions), but the central question—*is identifying relevance fundamentally hard?*—is answered: yes.

Future Work

Several directions extend this work:

1. **Mathlib integration.** Contribute the reduction framework to Mathlib's computability library, providing standard definitions for Karp reductions, NP/coNP membership, and polynomial bounds.
2. **Additional reductions.** The methodology extends to other classical reductions (3-SAT to CLIQUE, HAMPATH to TSP, etc.). A library of machine-verified reductions would be valuable for both education and research.
3. **Automation.** The patterns in Section 12 define targets for tactic development. Define a `complexity` tactic analogous to `continuity` to automate routine reduction steps.

4. **Turing machine formalization.** Our current work operates at the reduction level, assuming polynomial-time bounds. Full Turing machine formalization would enable end-to-end verification from machine model to complexity class.

5. **Parameterized complexity library.** W-hierarchy and FPT definitions are not yet in Mathlib. Our W[2]-hardness proof provides a starting point.

Methodology Disclosure

This paper was developed through human-AI collaboration. The author provided problem formulations, hardness conjectures, and proof strategies. Large language models (Claude) assisted with LaTeX drafting, Lean tactic exploration, and proof search.

The Lean 4 compiler served as the ultimate arbiter: proofs either compile or they don't. The validity of machine-checked theorems is independent of generation method. We disclose this methodology in the interest of academic transparency.

A Lean 4 Proof Listings

The complete Lean 4 formalization is available at:

https://doi.org/10.5281/zenodo.18140966

The formalization is not a transcription; it exposes a reusable reduction library with compositional polynomial bounds, enabling later mechanized hardness proofs to reuse these components directly.

A.1 On the Nature of Definitional Proofs

The Lean proofs are straightforward applications of definitions and standard complexity-theoretic constructions. Formalization produces insight through precision.

Definitional vs. derivational proofs. The core theorems establish definitional properties and reduction constructions. For example, the polynomial reduction composition theorem (Theorem A.1) proves that composing two polynomial-time reductions yields a polynomial-time reduction. The proof follows from the definition of polynomial time: composing two polynomials yields a polynomial.

Precedent in complexity theory. This pattern occurs throughout classical complexity theory:

- **Cook-Levin Theorem (1971):** SAT is NP-complete. The proof constructs a reduction from an arbitrary NP problem to SAT. The construction itself is straightforward (encode Turing machine computation as boolean formula), but the *insight* is recognizing that SAT captures all of NP.

- **Ladner's Theorem (1975):** If $\mathrm{P} \neq \mathrm{NP}$, then NP-intermediate problems exist. The proof is a diagonal construction; conceptually simple once the right framework is identified.

- **Toda's Theorem (1991):** The polynomial hierarchy is contained in $\mathrm{P}^{\#\mathrm{P}}$. The proof uses counting arguments that are elegant but not technically complex. The profundity is in the *connection* between counting and the hierarchy.

Why simplicity indicates strength. A definitional theorem derived from precise formalization is *stronger* than an informal argument. When we prove that sufficiency checking is coNP-complete (Theorem 4.6), we are not saying "we tried many algorithms and they all failed." We are saying something general: *any* algorithm solving sufficiency checking solves TAUTOLOGY, and vice versa. The proof is a reduction construction that follows from the problem definitions.

Where the insight lies. The semantic contribution of our formalization is:

1. **Precision forcing.** Formalizing "coordinate sufficiency" in Lean requires stating exactly what it means for a coordinate subset to contain all decision-relevant information. This precision eliminates ambiguity about edge cases (what if projections differ only on irrelevant coordinates?).

2. **Reduction correctness.** The TAUTOLOGY reduction (Section 4) is machine-checked to preserve the decision structure. Informal reductions are error-prone; Lean verification guarantees the mapping is correct.

3. **Complexity dichotomy.** Theorem 5.1 separates logarithmic and linear regimes: polynomial behavior when the minimal sufficient set has size $O(\log |S|)$, and exponential lower bounds under ETH when it has size $\Omega(n)$. Intermediate regimes are not ruled out by the lower-bound statement.

What machine-checking guarantees. The Lean compiler verifies that every proof step is valid, every definition is consistent, and no axioms are added beyond Lean's foundations (extended with Mathlib for basic combinatorics and complexity definitions). 0 `sorry` placeholders means 0 unproven claims. The 7071 lines establish a verified chain from basic definitions (decision problems, coordinate spaces, polynomial reductions) to the final theorems (hardness results, dichotomy, tractable cases). Reviewers need not trust our informal explanations; they run `lake build` and verify the proofs themselves.

Comparison to informal complexity arguments. Prior work on model selection complexity (Chickering et al. [5], Teyssier & Koller [34]) presents compelling informal arguments but lacks machine-checked proofs. Our contribution is not new *wisdom*—the insight that model selection is hard is old. Our contribution is *formalization*: making "coordinate sufficiency" precise enough to mechanize, constructing verified reductions, and proving the complexity results hold for the formalized definitions.

This follows the tradition of verified complexity theory: just as Nipkow & Klein [21] formalized automata theory and Cook [7] formalized NP-completeness in proof assistants, we formalize decision-theoretic complexity. The proofs are simple because the formalization makes the structure clear. Simple proofs from precise definitions are the goal, not a limitation.

A.2 Module Structure

The formalization consists of 42 files organized as follows:

- `Basic.lean` – Core definitions (DecisionProblem, CoordinateSet, Projection)
- `AlgorithmComplexity.lean` – Complexity definitions (polynomial time, reductions)
- `PolynomialReduction.lean` – Polynomial reduction composition (Theorem A.1)
- `Reduction.lean` – TAUTOLOGY reduction for sufficiency checking

- `Hardness/` – Counting complexity and approximation barriers
- `Tractability/` – Bounded actions, separable utilities, tree structure, FPT
- `Economics/` – Value of information and elicitation connections
- `Dichotomy.lean` and `ComplexityMain.lean` – Summary results
- `HardnessDistribution.lean` – Simplicity Tax Theorem (Section 10)

A.3 Key Theorems

Theorem A.1 (Polynomial Composition, Lean)**.** *Polynomial-time reductions compose to polynomial-time reductions.*

```
theorem PolyReduction.comp_exists
    (f : PolyReduction A B) (g : PolyReduction B C) :
    exists h : PolyReduction A C,
      forall a, h.reduce a = g.reduce (f.reduce a)
```

Theorem A.2 (Simplicity Tax Conservation, Lean)**.** *The simplicity tax plus covered axes equals required axes (partition).*

```
theorem simplicityTax_conservation :
    simplicityTax P T + (P.requiredAxes inter T.nativeAxes).card
      = P.requiredAxes.card
```

Theorem A.3 (Simplicity Preference Fallacy, Lean)**.** *Incomplete tools always cost more than complete tools for $n > 0$ use sites.*

```
theorem simplicity_preference_fallacy (T_simple T_complex : Tool)
    (h_simple_incomplete : isIncomplete P T_simple)
    (h_complex_complete : isComplete P T_complex)
    (n : Nat) (hn : n > 0) :
    totalExternalWork P T_complex n < totalExternalWork P T_simple n
```

A.4 Verification Status

- Total lines: 7071
- Theorems/lemmas: 310
- Files: 42
- Status: All proofs compile with 0 `sorry`

B Discussion

We address anticipated questions about the scope and implications of the main results.

B.1 On Tractability Despite Hardness

Question: "coNP-complete problems have good heuristics or approximations in practice. Does the hardness result preclude practical solutions?"

Response: This observation actually *strengthens* our thesis. The point is not that practitioners cannot find useful approximations; they clearly do (feature selection heuristics in ML, sensitivity analysis in economics, configuration defaults in software). The point is that *optimal* dimension selection is provably hard.

The prevalence of heuristics across domains is itself evidence of the computational barrier. If optimal selection were tractable, we would see optimal algorithms, not heuristics. The widespread adoption of "include more than necessary" strategies is a rational response to coNP-completeness.

B.2 On Coordinate Structure Assumptions

Question: "Real decision problems are messier than the clean product-space model. Is the coordinate structure assumption too restrictive?"

Response: The assumption is weak. Any finite state space admits a binary-coordinate encoding; our hardness results apply to this encoding. More structured representations make the problem *easier*, not harder; so hardness for structured problems implies hardness for unstructured ones.

The coordinate structure abstracts common patterns: independent sensors, orthogonal configuration parameters, factored state spaces. These are ubiquitous in practice precisely because they enable tractable reasoning in special cases (Theorem 6.1).

B.3 On the SAT Reduction

Question: "The reduction from SAT/TAUT is an artifact of complexity theory. Do real decision problems encode Boolean formulas?"

Response: All coNP-completeness proofs use reductions. The reduction demonstrates that TAUT instances admit an encoding as sufficiency-checking problems while preserving computational structure. This is standard methodology [6, 16].

The claim is not that practitioners encounter SAT problems in disguise, but that sufficiency checking is *at least as hard as* TAUT. If sufficiency checking is tractable, then TAUT is solvable in polynomial time, contradicting the widely-believed $\P \neq \mathsf{NP}$ conjecture.

B.4 On Tractable Subcase Scope

Question: "The tractable subcases (bounded actions, separable utility, tree structure) seem restrictive. Do they cover real problems?"

Response: These subcases characterize *when* dimension selection becomes feasible:

- **Bounded actions:** Many real decisions have few options (buy/sell/hold, accept/reject, left-/right/straight)
- **Separable utility:** Additive decomposition is common in economics and operations research
- **Tree structure:** Hierarchical dependencies appear in configuration, organizational decisions, and causal models

The dichotomy theorem (Theorem 5.1) precisely identifies the boundary in the stated encoding regimes. The contribution is not that all problems are hard, but that hardness is the *default* unless special structure exists.

B.5 On the Value of Formalization

Question: "Feature selection is known to be hard. Does this paper just add mathematical notation to folklore?"

Response: The contribution is unification. Prior work established hardness for specific domains (feature selection in ML [12], factor identification in economics, variable selection in statistics). We prove a general result that applies to decision problems with coordinate structure within the formal model.

This generality explains why the same "over-modeling" pattern recurs across unrelated domains. It's not that each domain independently discovered the same heuristic; it's that each domain independently hit the same computational barrier.

B.6 On the Lean Formalization Scope

Question: "The Lean formalization models an idealized version of the problem. Real coNP-completeness proofs involve Turing machines."

Response: The Lean formalization captures the mathematical structure of the reduction, not the Turing machine details. We prove:

1. The sufficiency-checking problem is in coNP (verifiable counterexample)
2. TAUT reduces to sufficiency checking (polynomial-time construction)
3. The reduction preserves yes/no answers (correctness)

These are the mathematical claims that establish coNP-completeness. The Turing machine encoding is implicit in Lean's computational semantics. The formalization provides machine-checked verification that the reduction is correct.

B.7 On the Dichotomy Gap

Question: "The dichotomy between $O(\log n)$ and $\Omega(n)$ minimal sufficient sets leaves a gap. What about $O(\sqrt{n})$?"

Response: The dichotomy is tight in the stated regimes under ETH. The gap corresponds to problems reducible to a polynomial number of SAT instances; these sit in the polynomial hierarchy between P and coNP.

In practice, the dichotomy captures the relevant cases: either the problem has logarithmic dimension (tractable) or linear dimension (intractable). Intermediate cases exist theoretically; we do not rule them out.

B.8 On Practical Guidance

Question: "Proving hardness doesn't help engineers solve their problems. Does this paper offer constructive guidance?"

Response: Understanding limits is constructive. The paper provides:

1. **Tractable subcases** (Theorem 6.1): Check if your problem has bounded actions, separable utility, or tree structure
2. **Justification for heuristics**: Over-modeling is not laziness—it's computationally rational

3. **Focus for optimization**: Don't waste effort on optimal dimension selection; invest in good defaults and local search

Knowing that optimal selection is coNP-complete frees practitioners to use heuristics without guilt. This is actionable guidance.

B.9 On Learning Costs

Question: "The Simplicity Tax analysis ignores learning costs. Simple tools have lower barrier to entry, which matters for team adoption."

Response: This conflates H_{central} (learning cost) with total cost. Yes, simple tools have lower learning cost. But for n use sites, the total cost is:

$$H_{\text{total}} = H_{\text{central}} + n \times H_{\text{distributed}}$$

The learning cost is paid once; the per-site cost is paid n times. For $n > H_{\text{central}}/H_{\text{distributed}}$, the "complex" tool with higher learning cost has lower total cost.

B.10 On Organizational Constraints

Question: "In practice, teams use what they know. Advocating for 'complex' tools ignores organizational reality."

Response: The Simplicity Tax is paid regardless of whether your team recognizes it. If your team writes boilerplate at 50 locations because they don't know metaclasses, they pay the tax; in time, bugs, and maintenance.

Organizational reality is a constraint on *implementation*, not on *what is optimal*. The Simplicity Tax Theorem identifies the optimal; the practitioner's task is to approach it within organizational constraints.

B.11 On Deferred Refactoring

Question: "Start simple, refactor when needed. Technical debt is manageable."

Response: Refactoring from distributed to centralized is $O(n)$ work; you pay the accumulated Simplicity Tax all at once. If you have n sites each paying tax k, refactoring costs at least nk effort.

Moreover, distributed implementations create dependencies. Each workaround becomes a local assumption that must be preserved during refactoring. Under dependency models where local assumptions interact, refactoring cost is *superlinear* in n.

B.12 On Weighted Importance

Question: "Real problems have axes of varying importance. A tool that covers the important axes is good enough."

Response: The theorem is conservative: it counts axes uniformly. Weighted versions strengthen the result.

If axis a has importance w_a, define weighted tax:

$$\text{WeightedTax}(T, P) = \sum_{a \in R(P) \setminus A(T)} w_a$$

The incomplete tool pays $\sum w_a \times n$ while the complete tool pays 0. The qualitative result is unchanged.

The "cover important axes" heuristic only works if you *correctly identify* which axes are important. By Theorem 4.6, this identification is coNP-complete; returning us to the original hardness result.

References

[1] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] Edoardo Amaldi and Viggo Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209(1-2):237–260, 1998.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[4] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[5] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-hard. In *Journal of Machine Learning Research*, volume 5, pages 1287–1330, 2004.

[6] 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.

[7] Stephen A. Cook. The P versus NP problem, 2018. Millennium Prize Problems, Clay Mathematics Institute.

[8] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[9] Ronald A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A*, 222:309–368, 1922.

[10] Yannick Forster, Edith Heiter, and Gert Smolka. Verified programming of Turing machines in Coq. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 114–127. ACM, 2019.

[11] Peter D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[12] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[13] Maximilian P. L. Haslbeck and Tobias Nipkow. Verified analysis of list update algorithms. In *Journal of Automated Reasoning*, volume 65, pages 117–156. Springer, 2021.

[14] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[15] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[16] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[17] Fabian Kunze, Gert Smolka, and Yannick Forster. Formal small-step verification of a call-by-value lambda calculus machine. In *Asian Symposium on Programming Languages and Systems*, pages 264–283. Springer, 2019.

[18] Erich L. Lehmann and Henry Scheffé. Completeness, similar regions, and unbiased estimation. *Sankhyā: The Indian Journal of Statistics*, 10(4):305–340, 1950.

[19] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 3rd edition, 2008.

[20] Richard J. Lipton. The P=NP question and Gödel's lost letter. *Communications of the ACM*, 52(5):31–33, 2009.

[21] Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014.

[22] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[23] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[24] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

[25] Howard Raiffa and Robert Schlaifer. *Applied Statistical Decision Theory*. Harvard University Press, 1961.

[26] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[27] Leonard J. Savage. *The Foundations of Statistics*. John Wiley & Sons, 1954.

[28] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[29] Ilya Shpitser and Judea Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. *Proceedings of the National Conference on Artificial Intelligence*, 21(2):1219–1226, 2006.

[30] Ilya M. Sobol. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1-3):271–280, 2001.

[31] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT Press, 2nd edition, 2000.

[32] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[33] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

[34] Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. *arXiv preprint arXiv:1207.1429*, 2012.

[35] The mathlib Community. The Lean mathematical library. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 367–381, 2020.