CMP

Michaeljohn Clement <inimino@gmail.com>

Version: 20260120

To A. and M.

Abstract
We introduce the universal model $u = (e, t, p, f, \omega)$, a framework that unifies scientific theories, computer programs, machine learning models, and human cognition under a single representation. We show that interpretability and efficiency in learning systems are identical problems, both resolved by recovering domain-natural factorizations. The difficulty of understanding large learned models and the difficulty of training them efficiently are manifestations of the awkward factorization, which gradient descent must overcome through "entropy smuggling" through floats, which we describe in a forthcoming companion paper.

Deep learning's data inefficiency stems from its inability to perform induction: to replace probabilistic patterns with absolute rules when evidence warrants. LLM opacity stems from a mismatch between the architecture-natural factorization (layers, neurons) and the domain-natural factorization (semantic features, concepts). Both problems are solved by correctly factoring the model.

We map our model U to scientific theories such as gravity, to human chess skill, and to the internal structure of machine learning models, and show an algebra over U with convenient practical applications.

Introduction.

We describe a universal model that maps to all domains of knowledge and practice, and of which all models are instances. We introduce the model shape having parameters (e, t, p, f, ω), and show how it maps onto several distinct domains:
First, to an accepted scientific theory, such as gravity, which is already expressed as a mathematical model.
Second, to a human chess player, or to human or animal thought in general.
Third, to logical, physical and philosophical rules of thought.
Fourth, to a computer program such as a fluid dynamics or weather simulation.
Fifth, to a machine learning model, such as a chess playing intelligence or LLM, or to learned programs of a general type, intermediate between classical programs and machine learning models.
Sixth, to chess as a matter of perfect or practical play.
Finally, to ideal human performance in chess, and various generalizations.

The central claim of this paper is that interpretability and efficiency in learning systems are identical problems, both resolved by recovering the correct factorization of the event space.

The model provides a common language for the internal structure of theories, models, and agents, and for exploring how those representations are updated by data or experience, and how factorization determines both efficiency and interpretability. We introduce the product pattern and factorization with examples across these domains, define the standard learning function, and show that the notorious problems of deep learning—its data inefficiency and its opacity—both reduce to incorrect factorization. Interpretability in large learned models is a refactoring problem: finding the domain-natural decomposition hidden within the architecture-natural one.

Universal Model Shape

The UM names three sets and two functions, $u = (e, t, p, f, \omega) \in E \times T \times P \times F \times \Omega$ where a specific model $u = (e, t, p, f, \omega)$ is described as a product of five factors.

Total Information of the UM

The total information of U is the information gained by observing a specific model u when starting from absolute ignorance. Under the uniform prior, this is simply $I(U) = \log |U|$.

Total Event Space $E = \prod E_i$

E is the total event space. It may be factored, or not; as we see later, this is the whole problem. It is some set of states of affairs that we can refer to or name. For example "the switch is open / closed" is a simple binary event space of the kind that governs all deterministic computer circuitry; the set E of possible states of some binary circuit with n switches has 2^n elements, since each of n switches can be either on or off. Or, we can factor it atomically into $E_1, \ldots, E_n$ event spaces, one per switch, such that again the total number of states of the whole system is the same, but the number of event spaces moves from 1 to n and the number of states per event space from 2^n in one event space to two events in each of n event spaces. Total information $I(E) = \sum_i^k I(E_i)$ is invariant under factorizations. Our state of knowledge may be some assignment to some but not all of these factors. E_i in a specific model u is an event space $e_1, \ldots, e_n$ being an analytical epistemic precommitment to divide and describe reality in a particular way. To build an analytic model we factor E by fixing k atomic event spaces $E_1, \ldots, E_n$ (registers in a machine, neurons in a circuit model, measurable channels in an experiment, etc.), such that the total event space is their product

$$E := \prod_{i=1}^{k} E_i.$$

In actual practice, E, T, and P are grown in parallel; this is the only way it can work. However in an analytical approach we must first commit to some epistemic cuts; by establishing E first we say that the first thing you can learn about a thinking being is what categories they have, and then if you have shared names for those categories you can talk to them. From here on we will treat all events as having known, unique, and agreed names, for convenience of description.

An element $e \in E$ is a *total event* which we can regard equally well as either one single event picked (perhaps at random) from all possible states of affairs, isomorphic to n being a selection from $|E|$ total events, or as one event picked from each of the factor event spaces E_i equivalent to choosing a k-tuple or a vector or whatever you like. The choice of the factors E_i is arbitrary so long as they meet certain basic conditions, such as making progress or positive correlation; it determines the breakdown of event e into the joint event $(e_1, \ldots, e_k) \in \prod_i^k E_i$.

To establish an event space is to divide reality into distinguishable parts; to factor an event space is to divide those distinctions into groups. Each event in E may have a name and the event spaces E_i (as sets) may also have names. Shared event spaces and event name spaces (such as the set of names of edible plants in English, such as "strawberry") define the communicable. The space of linguistic thought, or the universe of discourse, is defined by E taken as inseparable from its names. The questions of the correspondence of E to the world is epistemology, and the study of the somewhat arbitrary factor structures of E belongs to ontology.

Total Thought t in space T either $\in \{0,1\}^n$ or $\in (0,1)^n$

The element t of T is the activations of the neurons, the voltages on the electrical leads, the register values in a processor. The elements of T can be modeled as discrete or continuous; we consider both cases. When T is discrete, as it must be in our computer simulations, one element e of E is the current state of the system within the model; t is the corresponding atomic thought that assigns truth (1) or positive support ($\in (0,1), \geq 0$) to e—or to some other event if the model represents an agent in error about the actual state of affairs. Metaphysics describes the relationships between T and reality. The answer for us via the UM is that T is an assignment of belief to a specific state or states from all of those that are possible. Logic governs the relationship between event spaces. Note that the total thought doesn't care if the event spaces are factored or not, because it is just an assignment to whatever the atomic events are. It is an assignment of either $\{0,1\} \subset Z$ or $(0,1) \subset R$ to each e in E corresponding to the rules of an event space. T factors in exactly the way that E does, so we say that T and E are isomorphic, that is, if E is factored then T may be factored in the same way. The factorization of E and T does not impose an order on the factors, but most analysis of models does imply an ordering between factors, which we will explore in what follows. The factors of E determine the sparsity of t and they determine which thoughts can be called contradictory: an assignment of truth values to T is contradictory (or surprising) if it assigns significant positive support to two events that share an event space; thus surprise in our model arises from the event spaces and cannot in general be represented until E is factored.

The fact that we express the total thought in two forms has ramifications on human thinking: specifically, there are symbolic, logical forms of thinking concerned with the discrete domain, and there are practical, probabilistic forms concerned with the world and our actions in it; these correspond to the absolute $0, 1$ and relative $(0, 1)$ representations of atomic thoughts and of patterns.

The rest (P, F, Ω)

After E and T we will introduce the total pattern space P which is a description of observations of changes in

T over time, or the cause of those changes, or both. An atomic pattern is an ordered pair $a = (x \to y) \in E^2$ (or in T^2 via ι). A total pattern is a (typically sparse) set of atomic patterns, i.e. a relation $p \subseteq E^2$. Let P be the set of admissible total patterns (e.g. sparse relations), so $p \in P \subsetneq \mathcal{P}(E^2)$. The reason for the sparsification is the inherent factorization of the patterns in the world which the patterns in the model must reflect. This is obscured when E is unfactored, but revealed when E is factored correctly; then we find that P is defined by the patterns between the factor event spaces. We identify a deterministic update function $f_p : T \to T$ with its graph $\{(t, f_p(t))\} \subseteq T^2$. We introduce the learning function ω as a function on U^2 or relation $f \subset U \times P$, etc, as convenient. As noted previously, even the factoring of U that we have chosen is somewhat arbitrary. In particular, whether we put the details of a model in p or f, the patterns or the update function, is a matter of convenience. However, many results can be proved by assuming a fixed f and fixed ω that are simple and optimal in specific ways. This then requires us to put all the complexity of the model into p and could mean that the size of the pattern space $|P|$ must be larger to accomodate a smaller f, similar to how complexity can be moved between a compression program and the compressed data, or between a compiler and the program.

Total Pattern $p = t \to t' \in P = T^2$
P is the pattern space, and the total pattern p describes how the total thought at one moment in time can influence the total thought in the next moment. If the atomic pattern is $p = e_1 \to e_2$ where e_1 and e_2 are atomic events, then we have the bottom-up description of the total pattern as a set of atomic patterns, e.g., in the brain, this is the connectome or the set of effective synaptic connections from one neuron to another as a sparse subset $p \in E^2$. Note that each atomic event $e \in E$ has a canonical representation in T. Define the embedding $\iota : E \to T$ by

$$\iota(e)(e') := \mathbf{1}[e' = e],$$

(or, if T stores support in $\{0, \ldots, 255\}$, take $\iota(e)(e') := 255 \cdot \mathbf{1}[e' = e]$). Thus we may regard atomic patterns as living in E^2 or, via ι, in T^2, and we abuse notation accordingly. It may be seen that if the event space is taken as a complete collection of possible physical states, for example, then there may be an atomic pattern from each event to each other event, if we are to be able to represent any kind of unknown general pattern between events from one time step to the next. Of course in a physical brain or in a practical computer there must be some reasonable number of connections, so P is a sparse product pattern of pairs of atomic events.

Of course the point of the total pattern is to advance the total thought, so we write

$$t_{n+1} = p \times t_n$$

.

Update Function $f \in P \times T \to T$
We write f_p for the update function f of model u given the instantaneous total pattern p, which we may take as implied and write simply f. f is the update function that takes T at time t_n into T' at time t_{n+1}. It may be that $T = T'$ in which case we can write $f \in T^2$, making f the function from thought to thought. In a simulation we must resolve questions about timing, and the real world also entails similar questions, such as how we treat a connection between two neurons that are close or two that are on opposite sides of the brain. Then the total information, the information gain of learning f, could be $2 \log |T|$. It is a function from T to T and fully characterizes the thinking of an agent through time. From one perspective, if P is already fully general, then we might say that f is unnecessary. However, in P we treat an event e_a as leading to e_b directly, without any consideration given to the passing of time. Clearly this is non-physical and would prevent us from simulating a brain using neuron activations as E, which is reason enough to include f as a parameter, to make the UM capable of representing human thinking in a physical model. If we consider an LLM as a program, then P is the matrix of weights in each layer, and f is the function from T to T_{n+1}. Now if f is defined such that it simply applies the weights in P—the atomic patterns from event to event—according to some fixed formula, then we can take f as a constant function and then if we learn P from data we have the full control over the output of the model; this is how LLMs learn, where P is called Θ.

The Learning Function $\omega \in \Omega = U^2$
ω is the learning function. In a scientific theory, or a computer program, generally, learning is left out

because the behavior is statically determined. However in a machine learning model, or a scientific model that is being fitted to available data, or a learning agent, then learning should be from a memory trace to a delta in P. A memory trace is an element of the set $E \times T \times P^n$, or more naturally for a programmer, it is a sequence of (e, t, p) tuples, each recorded at a later moment in time, showing at any moment the event spaces that are relevant, the total thought, and the total pattern at some instant. (See Appendix: Memory Traces for elaboration on trace types and the connection to databases and biological memory.) Of course, ω is quite powerful if it can entirely rewrite P; the actual case depends on the substrate—in a machine-learning model, ω is backpropagation, where we take the derivative of error, allocating blame for that error to each element of P, and then we multiply the blame by some small negative learning rate and add that change to P.

In the LLM, ω is backpropagation: given observations (the events), it produces pattern updates (the weight changes). The gradient $\partial L/\partial w$ tells us how to update each weight. Information ramifies backward through event spaces—this is how ω integrates information across the factorization.

When gradients are carried by floating point values, there is an effective depth limit. Gradients are products of terms, each ≤ 1. After $d_{\max}$ event spaces, the product falls below representational precision and rounds to zero. This is the same limit as forward processing. Earlier event spaces cannot learn from observations that are too many steps downstream.

In the context of a computer program or any logical formalism, ω represents rewriting and it can make non-linear changes in the outcome. Of course, if we represent both P and f as computer programs in a model of a computer program then it may be convenient to have ω simply write f directly and it can inline whatever values of P (or E) that it wants to use, so that really you only need to write ω. This merely restates the observation already made that our factorization (as all factorizations) is somewhat arbitrary. "Somewhat" because there is some atomic factorization below which nothing can go (in any given domain) and it is all the partitions of this set of atomic factors that defines all possible factorizations that do not lose fidelity.

Product Pattern $E = E_1 \times E_2$

We may regard the total event space E as a product of, say, an input event space I and an output event space O. Then we see that the function $f : I \to O$ represents the behavior of an agent in an environment, to use the terms of classical AI. In this case the total instantaneous state of the agent is represented of course by $e = (i, o) \in E = I \times O$ where i and o are the actual events in the total event space, which is the product of the input and output event spaces, or if we want to admit that the agent has unobservable internal states then we can give the agent $a = (i, o, h)$ where input output and hidden spaces are all high-dimensional, thus we have factored the total event space into three factors. Immediately here we will notice subtleties in practice, e.g. in a physical model we need some neurons to be in both the I and O sets, which is a problem for our model, as now the product IO counts those neurons twice. We prefer for theoretical presentation to consider perfect factorization only.

We can describe the elements of E where e is treated unitarily as a gestalt, as for example the complete first-person experience of a human at any given moment. In this case the total event space is simply a vastly large set of possible states of experience, such as being cold while hearing a car pass and listening to certain music, and so on for all the senses, all at the same time. So we would say that E is simply a large set $e_1, \ldots, e_n$, and nothing more about the structure. Then mathematics is not the right tool for this problem and it becomes metaphysical.

However, as soon as we begin to categorize the total state of the world, to separate being cold from hearing traffic outside, and so on, we factor E into some smaller event spaces, and in particular there must be some first factor that we separate out, such as being cold from all the other immediate sense perceptions and whatever our interiority is subject to.

To be cold presupposes an event space that would eventually contain things like being cold, being comfortable, being uncomfortably hot on one side but not the other, and so on—fundamental to the event space in any real system is that it is always open.

Multiplication

Multiplication appears throughout the UM as the operation that combines factors.

Multiplication in E. For event spaces, multiplication is Cartesian product: $E = E_1 \times E_2$. The event $e \in E$ becomes a joint event (e_1, e_2) where $e_1 \in E_1$ and $e_2 \in E_2$. Total information is additive: $I(E) = I(E_1) + I(E_2) = \log |E_1| + \log |E_2| = \log |E_1 \times E_2|$. Example: if $E_1 = \{\text{heads}, \text{tails}\}$ and $E_2 = \{\text{rain}, \text{sun}\}$, then $E = E_1 \times E_2$ has four joint events like (heads, rain).

Multiplication in T. For thoughts, multiplication composes the beliefs about different factors. If $T = T_1 \times T_2$ where $T_i : E_i \to [0, 255]$, then a total thought $t = (t_1, t_2)$ assigns beliefs to each factor independently. When factors are genuinely independent, $p(e_1, e_2) = p(e_1) \cdot p(e_2)$, which in the log domain is $\log p(e_1, e_2) = \log p(e_1) + \log p(e_2)$. Example: "the first token is A" (255) and "the sentence is declarative" (10) can be held simultaneously; their joint belief is the minimum (in the max/min update described below).

Multiplication in P. For patterns, multiplication combines layers: if $P_1 : E_1 \to E_2$ and $P_2 : E_2 \to E_3$, then $P_1 \cdot P_2 : E_1 \to E_3$ is their composition. In matrix terms, this is literal matrix multiplication. The multi-layer structure of neural networks is exactly this: $P = P_L \cdot P_{L-1} \cdots P_1$. Multiplication in Boolean logic is AND, we see that AND also falls out of our atomic patterns as pattern length, which we discuss in an appendix.

Quotient and factorization. Given E and a factor E_1, we can ask: what is E/E_1? This is the quotient space—the set of equivalence classes where events that differ only in their E_1 component are identified. Example: if $E = A^\ell$ (strings of length ℓ over alphabet A) and we tokenize, the quotient $E/(\text{tokenization})$ is the space of token sequences. The tokenizer maps $E \to E/E_1$; the detokenizer lifts back. The simpler way to understand the quotient E/E_1 is that it is simply the rest of the event spaces in the factorization of the model—but this presupposes the factored perspective.

Concrete Representation

An event space is given as a set of event names.

In the next game of chess I play, I predict there will be one of three outcomes:
"I win." 0.
"I lose." 0.
"The game is a draw." 0.

This event space represents my epistemic precommitment to regard the finished game as in one of three states. I'm excluding the possibility of being in two of these states at once, or in none of them, so that the event space really does divide reality, and not just a part of it. This new epistemic commitment induces $\log 3$ bits of total information which is our information gain when the game concludes in one of the three states, starting out from total ignorance.

In our notation, the event names are always in quotes and are always followed by a number and a dot and the end of the line. This notation with the explicit double quotes and trailing dot is intended to be easy to parse for a machine and easy to spot visually for a human or LLM. A support value of 255 represents the greatest possibility of belief in the organism; functionally, it means an unwillingness to consider contradictory evidence.
"We call this belief format SN." 255.

We may write an atomic pattern as two atomic events followed by the strength of the pattern.
"We see checkmate in one move." "We play that move." 255.

The above pattern may represent a chess player's policy to, if we see checkmate, immediately play it.

Event algebra. We represent E concretely by bijection onto $\mathbb{N}$. If an organism records every total event e it experiences, we may count them: incrementing an internal counter and associating each new event with the count at observation time of previously seen distinct events. Counting imposes both a total order and a labeling simultaneously, making it efficient. In a program, this is a single-column store: writing strings into a file separated by newlines imposes an order and allows reference by position.

Thus every atomic event e is represented by some $n \in \mathbb{N}$ where $n < |E|$.

To factor E into $E_1 \times E_2$ with a renumbering, we may choose $|E_1|$ and $|E_2|$ to be coprime (e.g., by padding to the next suitable size). Then the joint event (e_1, e_2) maps to $e_1 \cdot |E_2| + e_2$, and we recover the factors by division and modulus. This gives us a literal *event algebra*: the joint event e_a AND e_b is literally the integer $e_a \times e_b$ (under appropriate encoding).

With prime factor cardinalities, the Chinese Remainder Theorem guarantees unique recovery regardless of factorization order. Factorizations become equivalent up to permutation—the concrete realization of the abstract claim that factorization order should not matter.

Every time we refactor E, we recalculate the map and remap all indices. Given an event expressed as a number, we can factor it to recover the sub-events.

The forthcoming concrete companion paper develops arithmetic coding as the operational semantics: f's probability outputs determine interval subdivision, compression ratio equals prediction accuracy, and the quotient $Q = N/D$ tracks position exactly, or until finite precision imposes the depth limit, depending on underlying representation.

Scientific Theories and Mathematical Models

We define an accepted scientific theory as some set of event spaces with well-known names, such that we can all communicate about them, along with some total pattern that represents that theory's claims about the relationship between the events.

There is a practical problem in writing down the total pattern as pairs (t, t'), because the number of states of the system is $|E|$ and is large, so we cannot write down the total pattern this way. Writing down p as pairs (t, t') is as impractical as writing down a theory of chess by listing every legal board position and telling what to play in each one. We must factor p, and we will naturally tend to factor it according to how T and E have been factored already.

We will arrive at a construction like this one:
"Object 1 has mass ..." 255.
"Object 1 has the name "Sun"." 255.
...
"The product of the masses of Objects 1 and 2 is ..." 255.
...
"The gravitational force between Object 1 and Object 2 is ..." 255.
"The location of Object 1 at the next time step will be ..." 255.

Then we can easily see how to write some of the corresponding rules, for example, we can capture Newton's first law by having a pattern from the velocity of any object in the system to the same velocity in the next time step. We can write the mathematical operations we need directly into p by having explicit rules, e.g. from "Object 1 has mass X" and "Object 2 has mass Y" to "The product of the masses ...", and similarly from their positions to the distance between them, then to the gravitational force, and so on for each pair of objects.

On the real numbers. Physics traditionally writes state spaces as $\mathbb{R}^n$—positions, momenta, fields are real-valued. But $\mathbb{R}$ has a curious property that a single real value (e.g. chosen at random on $[0, 1]$) has infinite information. Every measurement has finite precision because measurement requires the expenditure of energy. This explains many apparent paradoxes involving the reals.

Standard update function f_p

Let us take E as factored into two disjoint sets I and O of input and output events. Then we take t as initialized in the input event space only, in our standard notation using log support values from 0 to 255. If, as in the presentation of gravity above, we keep all inputs, outputs, and patterns in the logical domain $\{0, 1\}$ and we do not factor I or O further, then p may be trivially written in the relational form as a set of (i, o) pairs. When input or outputs spaces are factored as we have done above patterns naturally factor into the terms that we would expect, such as mass, gravity, the product of the masses, the square of the distance between them, eventually giving us all the formulas of the theory and a set of patterns of reasonable size that we could write down. We can write all of these logical or Boolean rules using only positive patterns of the form "e_a e_b 255." in our notation.

We define the standard update function f_p between two event spaces I and O, with inputs, outputs, and patterns all given in our standard notation, i.e. as log support values in 0..255. We take p as factored into atomic patterns $e_a \to e_b$, and we are here only interested in the subset from I to O. Given only Boolean patterns we can then write the atomic patterns as $p_{ij} = 1$ for the pattern from e_i to e_j, e.g. we can represent p as a Boolean $|I| \times |O|$ matrix. Given only Boolean patterns we can write the atomic patterns as $p_{ij} \in \{0, 255\}$ for the pattern from $e_i \in I$ to $e_j \in O$. Taking conjunction as min and disjunction as max, define

$$(f_p(t))_j := \max_i \min(t_i, p_{ij}).$$

Machine Learning Models

The first factorization of an LLM is from characters to tokens. In the input space, a byte is 8 bits; a character model has $\log |A|$ bits per character. Tokenization compresses this. A vocabulary of $|V|$ tokens costs $\log |V|$ bits per token. For compression, we need the average token to span more bits of input than it costs to represent. With $|V| = 2^{16}$ (16 bits per token), if the average token spans 4 bytes (32 bits), we halve the sequence length. The cost: the model cannot see inside tokens.

Tokenization helps because it increases information per position. If tokens are uniformly distributed (a perfect hash), each carries $\log |V|$ bits—16 bits for $|V| = 2^{16}$. Bytes carry at most 8 bits, and text entropy is lower still due to redundancy. A good tokenizer captures this redundancy: common substrings become single tokens, so $H(\text{token}) \approx \log |V|$. If m tokens represent n bytes with $m < n$ and higher entropy per position, the model sees more information in a shorter sequence—and attention is $O(m^2)$.

Given a trained LLM with layers $L_1, \ldots, L_n$, we represent it as a UM instance. Take E as the union of input tokens, output tokens, and hidden activations: $E = I \cup O \cup H$ where H indexes every neuron in every layer. Take $T : E \to [0, 255]$ as the activation values in a log support representation. Take P as the weights: each weight w_{ij} from neuron i to neuron j becomes a pattern. Take f as the forward pass that applies P to propagate activations.

This gives a UM isomorphic to the original LLM—same structure, same computation, different notation.

There is a problem: patterns in P carry positive support, but neural networks use negative weights. A negative weight from neuron i to neuron j means "if i fires, j should not fire."

We solve this by doubling the number of events. For each neuron j, introduce two events: e_j (fires) and $\bar{e}_j$ (does not fire), forming a binary event space $\{e_j, \bar{e}_j\}$. A positive weight $w_{ij} > 0$ becomes a pattern from e_i to e_j with strength proportional to w_{ij}. A negative weight $w_{ij} < 0$ becomes a pattern from e_i to $\bar{e}_j$ with strength proportional to $|w_{ij}|$. After normalization within each binary event space, support for $\bar{e}_j$ reduces the probability of e_j.

This doubles $|E|$ but makes all patterns positive. The negative pressure of neural networks is mediated by competition within event spaces—exactly as in the UM.

The UM representation above is isomorphic to the original LLM. It is not more interpretable—the events are named by layer and index, not by meaning.

Interpretability is a refactoring problem: find a UM u' that computes the same function but where events have human-meaningful names. Take I and O as given—input and output tokens. The question is how to refactor H, the hidden space.

In the original, H is indexed by layer and neuron: $H = \bigcup_\ell H_\ell$ where $|H_\ell|$ is twice the layer width. In the refactored model, H' would be indexed by semantic features: "subject is plural," "sentiment is negative," "topic is mathematics."

The refactoring map $\phi : H \to H'$ is what interpretability research seeks. The UM provides a useful framework to explore the properties of ϕ and we may prove certain results about it even without giving a construction.

Standard Learning Function ω_0

We introduce here a standard learning function, first in the simplest description, which is the unfactored one.

In fact if E is unfactored, then all we can do is write down every total event, so a sequence of elements of E is all that we can learn; let us consider the minimally-factored case where E is divided into input and output event spaces.

Given some $E = I \times O$ we take some dataset $d \subset E$ as a record of observed events in E. (It is fine to regard d as a set rather than a multiset; if E does not include time or something similar then we may add an index or other synthetic key to it, but in any case we must be able to distinguish between two events if we are to legitimately regard both of them as evidence of the same pattern; thus we are justified in treating the uniqueness of joint events $e \in E$ as given.) The learning function answers the question: given such a dataset, how can we best learn a function from $I \to O$ that learns everything we can from the dataset, purely statistically. That is, the function we seek is also a sufficient statistic that captures everything that we can say about the function based on the dataset that we have. This answer is surprisingly simple: all we have to do is record the joint events, and since we need only their log support in what follows, we can collect the entire statistic as a sparse product of $I \times O$.

Presenting the learning function as an algorithm, letting $p \in I \times O$ be our accumulated pattern, initially zero everywhere; since we know p in advance to be very sparse we will represent it in our algorithm as a list of observed joint occurrences along with log support values, $E \times E \times \{1, \ldots, 254\}$ so that we have:
1. Let `obs` $\leftarrow []$ be the initially-empty list of supported observations.
2. For each observation $(i, o) \in I \times O$, find (i, o, s) in `obs` and log-stochastically increment s; if (i, o, s) does not exist in `obs` then add it (either with $s = 0$ before the log-stochastic increment or after it with $s = 1$).

This algorithm gives us a matrix where the entries are log co-occurrence counts—a log contingency table. Now we can describe our desired function $I \to O$ simply by labeling this matrix in a convenient way. Since we must have assigned I and O to the matrix axes one way or another, and since we are asked for the function $I \to O$, we can find the function from each i onto a distribution over O by simply finding the row (or column) of i and reading off the log support values for O.

Then we have

$$f_p(i) = o_1, \ldots, o_m \in (0,1)^{|O|}, o_j = \exp(p_{ij}) / \sum_j^m \exp(o_j))$$

when i is taken as known (one-hot, $t \in \{0,1\}^{|I|}$). Of course, if the input is also a distribution, in $(0,1)^{|I|}$, say, then we can take softmax on the input side first and then get the expectation over O by the laws of probability theory, or we may also stay in the log domain and do something else that may be more efficient.

To sum up the standard learning function, it is simply this: To learn everything that can be statistically learned about a function from I to O; that is, everything that can be rightly calculated using Bayesian probability theory or frequentist statistics from observed correlations of I and O, we can simply record every observed joint event. Since only the magnitude of observations is statistically meaningful, we only need to record the log counts of occurrences.

The log joint event matrix, the compressed memory of all joint events, is then all that we can say statistically about the influence of any element of I on O.

Since we arbitrarily labeled the matrix dimensions as I and O, the reader has no doubt noticed that the standard learning function over the unfactored $I \times O$ is symmetric and we can read out the function from events in O back onto I just as well.

This explains an observation about causality:

If in the world causality goes from A to B, such as from rain falling to the ground being wet, then in epistemology the information flow may go in either direction; we may infer that it has rained from the ground being wet just as well as that the ground must be wet from the fact that it is raining. Learning takes the directed causal arrow from rain to wet ground into a bidirectional one; here we see how that falls out of the geometric structure of the product pattern being rectangular such that the functions between I and O may be exchanged simply by transposing the matrix of the observations.

The connection between associative memory and correlation matrices was established by Hopfield (1982), who showed that storing patterns via $T_{ij} = \sum_s (2V_i^s - 1)(2V_j^s - 1)$ yields content-addressable memory with capacity approximately $0.15N$ for N neurons.

The Equivalence of Interpretability and Efficiency

We now state the central theoretical claim of this paper. The efficiency of a learning system—its sample complexity, inference cost, and ability to generalize—is determined by the same thing that determines its interpretability: how well the system's internal factorization matches the natural factorization of the domain.

Consider the two notorious problems of deep learning: data inefficiency (requiring billions of examples to learn what humans learn from dozens) and opacity (the inability to explain or edit what the model has learned). These appear to be separate problems requiring separate solutions. We argue they are the same problem.

A model that has found the domain-natural factorization has both properties at once:

- *Efficiency*: Patterns become sparse when expressed in the right factors. Sparse patterns require less data to learn (each factor can be learned independently) and less computation to apply (independent factors can be processed in parallel, and absolute rules require no uncertainty propagation).
- *Interpretability*: When events have human-meaningful names and patterns decompose along domain boundaries, the model becomes transparent. We can read what it knows, edit individual rules, and compose knowledge from different sources.

Conversely, a model with the wrong factorization suffers both problems: its patterns are dense (inefficient), and its internal states have no human-meaningful decomposition (opaque).

The reason LLMs work at all is that they do factor the problem—into layers, attention heads, and residual streams. The reason they are not interpretable is that this architecture-natural factorization does not match the domain-natural one. (It cannot be any other way, because the architecture is fixed in advance, while the natural factorization can only be discovered through the data.) The concepts we want to find (syntax, semantics, world knowledge) are spread across the factorization we have (layers, neurons). The transformation from one to the other is a factor map.

In the sections that follow, we develop this argument in detail: showing how the limiting cases of learning reveal the role of factorization, how induction differs from statistics precisely in its assignment of absolute patterns to correctly-factored evidence, and how this explains the efficiency gap between human and machine learning.

Limiting Cases

Several limiting cases of the standard learning function are of interest.

The most important point here is that we have treated for ease of exposition only the unfactored case where I and O are sets of joint input events and joint output events respectively. As an extreme limiting case, let us recall a model in which we had explicit rules from each possible state of a system to the next state after some small Δt. Let us assume we are given a program that simulates the system perfectly according to the enumerated patterns from one total event to the next, $P \subset E^2$. If we cannot look inside the program but we may run it once with every possible input, we will get the same information back out, giving us a total dataset for this problem. Now we can ask how an ML model might learn this model from this dataset.

We train using the standard product pattern learning function ω_0. We record time-step pairs (t_n, t_{n+1}) into a log contingency table, which will end up being populated with a single 1 in each row and in each column.

We have now automatically and statistically learned a complete function from states of the system to states of the system, so we have something with the shape of a simulation, like the one we started out with, but all the weights of the patterns have changed from 255 to 1.

What is necessary to recover the original model in its full strength would be a kind of sublime induction, to (somehow) intuit that the solution we have is not a probabilistic one but a precise and complete one, and then lift at once all the weights p from 1 back to 255. Induction is the substitution of a logical rule for a probabilistic one derived from observation. It is the only way that logic can be used within a model to make it efficient and the lack of it is why deep learning approaches are inefficient. The inability to induct or abduct explains all the inefficiency in learning (specifically the training set size inefficiency) that is seen in deep learning models as compared to humans and animals. Abduction, or jumping to conclusions, happens not only at the symbolic level but also, necessarily, happens internally at the neural level; in the UM abduction or induction is always setting either an event or a pattern to 255.

We may contrast the sublime induction with the pure statistical training process that would be required to derive the same model, even with all the optimization tricks of deep learning.

The reason why LLMs work at all, of course, is because they do factor the problem, on both the I and O sides, and then in the middle into H internal events, and then they learn patterns from I to H_1 and so on through all the layers to H_k to O.

The reason why they are not interpretable is that the domain factors $E_1 \ldots E_k$ of any convenient factorization that we would like to see of some given model is not the same factorization as that given by the layers, and so instead the factorization we want is spread across the factorization that we have, which is the layers. Interpretability means multiplying all the patterns and layers of the model back together, dividing again by the next factor in the factorization that we prefer, and then carrying the entropy, just like in arithmetic or entropy coding.

When you correctly factor an LLM or any large model you find and localize the event spaces, most of which are about internal thought states of the model (such as "the mass of the planet that we are thinking of is ..."), and by localizing the event spaces in each layer, adding skip connections as necessary or via logical intermediaries (Boolean connectives, one per layer), then you replace the uninterpretable statistical patterns with interpretable ones, which, because they may be understood, may also be replaced wholesale by direct and deterministic code. At the least we can replace floats with integers, but we can also replace entire interior chunks of a learned model by a function, once the edges of that chunk are established (as a function of both factorizations, the one that is natural to the domain and the one that is natural to the model).

This gives us the means to express the problem of induction mathematically. The problem of induction is the difficulty of deriving absolute rules from non-absolute evidence; the difficulty is resolved by factoring the undifferentiated experience of the total recorded event (the total memory) such that the observed log support rises, then we may apply understanding, which means that we may go up a level and operate symbolically on our own awareness of the pattern to explicitly explain it in terms of other patterns, or we may subconsciously perform this same operation, but in any case we may create an efficient pattern from e_a to e_b only by a process that is not purely statistical. This process of jumping to conclusions, the process of setting up absolute patterns from partial evidence, is missing from deep learning because it is purely statistical, which it must by while it remains uninterpretable.

Thus explainability and efficiency in LLMs are the same problem.

Many forms of model collapse and practical difficulties of machine learning can be seen as the model "trying to" perform induction, setting up absolute rules from sufficiently strong evidence; this doesn't work because it breaks back-propagation, which only works in the probabilistic domain, because otherwise the gradients no longer flow, so we must prevent models from collapsing into absolutes because otherwise they will fail to continue learning. The alternative is to explicitly deal with surprise, which in the UM arises directly from the event spaces.

Human Chess Players

To model a human player, we must factor the input space in some way, representing the state of the board, the clocks, and so on. Then we factor the output space, including things like continuing to think (burning time), or making a move, resigning, offering a draw, and so on.

Then we can write down some I containing events like "There is a white rook on a1" as well as "There is less than 2 minutes but more than 1 minute left on your clock" and so on, as we like, and so on for O, and then our model u of some particular player will let us set up a board position and simulate that player by initializing I, running f_p forward until we get an action (move, resignation) and then adding that back to our input and continuing, like running inference on an LLM. We can take the moves for the other side from a model of a different player, if we like.

Now the problem statement is to write down a sequence of patterns, which can be either probabilistic or absolute, such that we can get some sequence of thoughts that ends in us making a move, like:

"There is a white rook on a1." 255.
"We are playing white." 255.
...
"We are thinking about rook moves." 255.
...
"We play rook to a7 checkmate." 255.

It seems it would be easy to solve this problem, if tedious, by simply factoring the human thought process into causative factors manually. In principle, if given infinite time and paper and a place to sit, Magnus Carlsen could write out his total chess pattern p by writing out e as it is factored in his head, and writing out atomic patterns from "we're unhappy with king safety" to "we look again for defensive resources" or whatever. But in practice (and from the history of chess AI) we know that he cannot: if chess grandmasters could program computers to play as well as they do, then they would also be able to write books that would turn all the rest of us into grandmasters. The reason chess knowledge is not introspectable, or communicable, at this level is precisely because it is factored into things like "we're unhappy with king safety" which is not a clearly defined category but clearly a learned product pattern (take, say, board states and game outcomes as the two event spaces) which leads us to feeling a certain way.

But now we know how to easily combine deterministic rulesets, learned models, and learning models all into a single model that correctly behaves according to the only way it can by the very rules of thought. If we want to model Magnus's skill correctly, we must factor it the way that it truly is factored, in his brain, which is first by his visual perception of the board, into some geometric representation, then into some kind of search pattern over fuzzy plans which are like bags of future moves where the move order is only partially factored out, so that the moves drift across move order positions, kind of like a superposition of the tree search that engines do. It's fuzzy of course because it's viewed through learned product patterns on all sides. A lot of these govern the internal mental states, like "we are going back to the top to find a new candidate moves" (because all of these ones are losing) and so on.

We can model Magnus's chess skill as a pattern from states like "thinking about rook moves" to other internal states, and eventually to actions, including those like looking at the clock that merely gain information, and eventually to moves in the game. Some parts of this model will be black boxes, but we know how to model the black boxes and if we put them in the right places, we will get a better fit to Magnus's actual games. A corollary is that we can experimentally support a model of the human brain by doing experiments on a database of human chess games. That is, if we have a hypothesis that a certain kind of pattern matching (such as identifying positions where king safety is a problem) is performed in a certain way, by some set of neurons in the brain that fire in a certain frequency, then we can predict that this will have a certain energetic cost to the organism, that it will take a certain amount of time, and other factors that would affect the goodness of fit of a human player model in silico. The more accurate the model, the more sophisticated the inferences we can make, and the more we learn. Factoring problems across domains in this way is the new wave of human understanding just about to begin, the great unification of human knowledge across every discipline and language. This great unification occurs as a side-effect of understanding our language models and other large models, which have compressed world models that include quite a bit of what we might call the total global pattern of all humanity's combined understanding as reflected in our books and other digital media.

Finally here let us note that in developing his chess skill, Magnus already performed a factorization of the problem. One acquires chess skill by first establishing event spaces, then adding patterns that provide support for those events, all in the service of providing justified support for other, outer events, like "I am a serious student of chess." Acquiring skill is not the factoring process of LLM analysis-in-situ but rather the multiplication of new skills into the mix. This means you can drop in a new product pattern, such as one from board positions to king safety (as estimated probability of checkmate within ten moves, say) and then learn that pattern empirically with ω_0 by replaying memory traces (stored events such as previous games) and this is a model of what a human may be doing as we add concepts to our repertoire and then link them to (internal) actions like looking for defensive ideas.

If two factorizations, such as Stockfish's and Magnus's, both achieve a high accuracy in estimating some unknown function, such as perfect play, then we may prove that both factorizations are equivalent up to, at least, the complement (in information) of the sum of the error terms. This is the principle of explanatory sufficiency: given any two factorizations that both work, there is a translation map between them, which we can find. This is easy to prove mathematically but we can see it even more easily: If Magnus and Stockfish are both good at chess, then they will both make the same moves in a large number of positions. Therefore the explanations they give must overlap over a large area of the space. The principle of explanatory sufficiency says that there is a map from an explanation in terms of one model to the terms of the other. For example, if Magnus used his idea of king safety to correctly play a defensive move, then there is a correlation between Magnus's learned product pattern about king safety onto some subset of the tree search that Stockfish did, leading to checkmates: we could factor both the heuristics like "enemy pieces near my king" and the engine's tree search into the same sets of positions and then see directly how and why they are correlated.

Rules of Thought

It is easy to show that formal logic can be mapped onto the UM. We have already presented briefly how p with a suitably factored E can construct any desired logical circuit. We can present any probabilistic model by taking $T = (0,1)^n$, with $P = T^2$. So we can represent categorical thinking and probabilistic thinking in the UM and combine them naturally. We can prove from the laws of probability that we are learning the most that we can purely from data by our pure learning rule ω_0. What is interesting is when we allow ω to vary over Ω, the entire space of possible learning functions. What is most interesting is when the learning function is allowed to refactor the model. The inability to do this is, of course, why deep learning practice is currently so inefficient.

What we have not mentioned yet is the particular shape we assume for our event spaces and why. We regard the event space and its importance as being a key contribution of the universal model in bringing clarity to existing practice, and to model interpretability in particular. Here we would like to justify the role and form of the event spaces in our model by a philosophical appeal.

We begin by imagining the organism in a state of undifferentiated experience, prior to the formation of any event space. We imagine that some idea arises in the mind (we may factor this as neurons firing in the brain, or as the corresponding subjective experience). Let us say the idea is the awareness of the warmth of the sun falling on the face. Now prior to anything like having a preference for being in the sun or not, the organism must be able to distinguish this state of affairs from all others, so already we have a formative event space including sun falling on the face, and all other states of affairs. So the beginning of our awareness about the world, in any aspect, can always be formulated this way as the creation of some event space. Now if we later associate shade with being under a tree, and we move out from under the tree to feel the sun, then we have used the event space to perform logical reasoning. Now this behavior could be explained purely by a learned product pattern telling us that being not under the tree is positively correlated with sun on the face. In this case we would say the organism has learned the rule but does not understand it. Or we could explain the behavior by a conscious thought process, as we might if justifying our own behavior. We might speculate about a dog who moves into the sun—does it "understand" about the shade, or is it just following a learned pattern? On inspection this question turns into—does it have the event spaces? If it knows that shade and sun are disjoint possibilities, and that being under a tree causes shade, then we could say that it understands. On the other hand, if it moved in response to seeing the sun falling on the grass more brightly, which it associates with warmth, then we could say that it moved into the sun without understanding about the tree, and without involving the under-the-tree event space. When we wonder about any other being's thought process or state of mind or reasoning, usually what we are really wondering about is the event spaces.

This is why we say that event spaces are implicitly always open: in the organism there is always the possibility that something happens that we thought was impossible, and we must always be able to handle this.

Improving Human Play

We can formulate certain objects within our model and then ask if those objects can be found, or how closely they may be approximated.

For example, just as there is some model $u = (e, t, p, f, \omega)$ that plays perfect chess, although we do not know if it is small enough for us to find, there is also some model that represents the perfect teacher of chess to a human, which we can approximate to the best of our current ability.

Just like the perfect anti-human chess player would have a model of human chess so that it could then target human weaknesses, and perfect chess teacher would use that model of the human player to identify the perfect intervention: the right set of puzzles to drill, or the right explanatory remark to resolve the student's biggest current misunderstanding, and so on.

The endgame of chess could be reached when, by using the best of machine learning and traditional programming approaches, which we have now seen how to combine, we understand human chess play at such a level that we can then teach any reasonably-motivated student, in a few weeks, how to play at a level that guarantees a draw against any opponent. This requires only that we have factored perfect or near-perfect chess play into some $E_1, \ldots, E_k$ specific factors and patterns over them, such that a human can learn the patterns with some realistic amount of training.

Conclusion

We have introduced the universal model $u = (e, t, p, f, \omega)$, a mathematical framework that unifies scientific theories, computer programs, machine learning models, and human cognition under a single representation. By treating any thinking or learning system as a combination of event spaces, thoughts, patterns, update functions, and learning functions, we gain a precise language for comparing and translating between these domains.

We have sketched how factorization of the total event space E and total pattern P determines both the efficiency and interpretability of any model. Deep learning's inefficiency stems from its inability to perform induction: to replace probabilistic patterns with absolute rules when evidence warrants. The problem of LLM interpretability reduces to refactoring: finding the domain-natural factorization hidden within the architecture-natural one.

For artificial intelligence, this analysis suggests a path beyond the current paradigm of scaling opaque models with ever more data and compute.

We have shown how human chess skill fits this framework, opening a path toward chess teaching systems that could, by modeling the student as precisely as we model the game, bring any motivated learner to draw-level play. More broadly, the universal model suggests that the great unification of human knowledge—across disciplines, languages, and the boundary between human and machine understanding—is inevitable as we learn to translate between all the factorizations of all the models we have built.

Glossary
We collect here formal definitions of the key terms introduced in this paper.

UM Structure

	Sci	**LLM**	**Human**	**Logic**	**Chess**	**Program**
EventSpace	Γ	$\mathcal{A}^n$	Ψ	$\mathcal{L}$	$\mathcal{G}$	$\mathbf{2}^n$
(factored)	$(E, \Gamma/E)$	(C, R)	(S, O)	(P, C)	$(O, \mathcal{G}/O)$	(I, O, S)
(factored)	(t, Γ_t)	$(L_1, \dots, L_k)$			$(e4, \mathcal{G}/e4)$	
(factored)	$(S, \bar{S})$	$(h_1, \dots, h_n)$			$(C, \mathcal{G}/C)$	
(factored)	(q, p)				$(Z, \mathcal{G}/Z)$	
(factored)	$(M, \Gamma/M)$				$(m, \mathcal{G}/m)$	
Belief	T	T	T	$T \subseteq \mathcal{L}$	$T \subseteq \mathcal{G}$	$T \subseteq \mathbf{2}^n$
(factored)					$(m, T/m)$	(V, F)
(factored)					$(O, T/O)$	(I, O, S)
Learning	ω	ω	ω	ω	ω	ω
(concrete)	Bayes	∇_θ	experience	induction	analysis	ω_0

Arrows

Arrow	Type	**Sci**	**LLM**	**Human**	**Logic**	**Chess**	**Prog**
ι	$D \to T$	measure	train	perceive	assume	observe	input
(induction)		$\to$posterior	$\to$weights	$\to$belief	$\to$premise	$\to$eval	$\to$state
δ	$T \times P \to T$	predict	forward	reason	derive	calculate	execute
(deduction)		law$\to$obs	$h \to h'$	think	$\vdash$	search	step
f	$T \to T$	simulate	infer	update	close	move	run
(update)		$t \mapsto t'$	next tok	revise	T^+	$g \to g'$	tick
ω	$T \times E \to P$	Bayes	SGD	learn	induce	study	ω_0
(learning)		revise H	∇L	memorize	generalize	pattern	count
α	$E \to E_i$	scope	attend	focus	context	candidate	switch
(attention)		choose var	α_{ij}	notice	Γ	see	select
ϕ	$E \cong \prod E_i$	coords	tokenize	categorize	parse	notation	struct
(factor)		(q, p)	(C, R)	(S, O)	(P, C)	$(O, \mathcal{G}/O)$	(I, O, S)

Let p be the model's predicted distribution over outcomes E, and let $e \in E$ be the observed outcome. The surprise of observing e is $-\log p(e)$. When $p(e) \approx 1$, surprise is near zero: the observation confirms what we expected and teaches little. When $p(e) \approx 0$, surprise is unbounded: the observation was "impossible" under the model, and we learn maximally. This asymmetry—that confirmations are cheap but refutations are expensive—is the kernel of truth in falsificationism. But the update rule is Bayesian throughout: we never categorically reject, only reallocate probability. The drama of scientific revolutions is explained by this information-theoretic asymmetry: Kuhn's paradigm shifts occur when accumulated low-probability observations finally outweigh the prior investment in the old theory.

Human analysis is the process of factoring undifferentiated experience into named events. A child learns the word "dog" by establishing an event space that distinguishes dogs from non-dogs. Each new concept is a new factor in E. Language is the technology for sharing factorizations: words are names for events in shared event spaces. The analytical capacity of humans—what we call intelligence—is largely the capacity to factor E in useful ways, and to communicate those factorizations to others.

$T \subseteq \mathcal{L}$ is the set of believed sentences (theorems under the premises). $t : \mathcal{L} \to \{0, 255\}$ — Boolean assignment. $t(\phi) = 255$ iff $\phi \in T$ (believed/proven).

Hidden state $h \in \mathbb{R}^d$ is the belief vector at a layer. For LLMs: $h_t = f(h_{t-1}, x_t)$ where x_t is the current token embedding. Output belief: $p(y|x) = \text{softmax}(Wh + b)$.

During training, belief h is updated to minimize loss. Gradient descent: $h_{t+1} = h_t - \eta \nabla_h L$. Belief converges to a fixed point of the learning dynamics.

Memory trace: sequence of (e, t, p) tuples—events, thoughts, patterns over time. ω: memory trace $\to \Delta P$ (change in pattern space). Human learning updates both P (what patterns exist) and E (what events are distinguished).

ω is backpropagation: $\Delta\theta = -\eta \nabla_\theta L$. Allocate blame for error to each element of θ, multiply by learning rate, update. P (weights) changes; architecture f is fixed.

Theorem (Hopfield). Correlation-based storage $T_{ij} = \sum_s (2V_i^s - 1)(2V_j^s - 1)$ yields content-addressable memory with capacity $\approx 0.15N$ for N neurons. In the limit, perfect recall requires $|P| \geq H(D)$ (information-theoretic bound).

Vibes. Vibes are evidentially-supported thoughts: $T \in [1, 254]$. The only possible evidential support about the world comes from prior observations of relevant events. There is therefore a weight of evidence, measured in bits, for the observations which support every vibe. Vibes cannot reach certainty ($T = 255$) because evidence is always finite. The process of converting vibes to logic—of justifying a felt pattern with explicit reasoning—is the articulation of the implicit memory trace into explicit event spaces and patterns. Cf. binary thinking.

Binary thinking. Binary thinking operates in $\{0, 255\}$: true or false, certain or impossible. The total information in a binary event space is the bit: $\log_2 2 = 1$. Maximum information is gained when you can divide possibilities into two equally-balanced outcomes. Binary thinking includes all categorical thinking, all mathematics, all writing, every computer program. It is the domain of logic and proof, where conclusions follow necessarily from premises. Cf. vibes.

Number systems. When an organism creates an event space $E = \{e_1, \ldots, e_n\}$, it can assign a number to each event. Counting is enumeration of events; numbers are names for positions in an event space. The base of a number system reflects a choice of factorization: binary ($b = 2$) factors repeatedly into pairs; decimal ($b = 10$) factors into groups of ten. There are 10 kinds of people in the world: those who think there are 10 kinds of people in the world, and every other kind of person. If we relax the limitation that each position in a number system follow a geometric series, then we get a generalized number system. If the positions are then mapped onto event factorizations, we can see that E maps onto N exactly as in our concrete representation.

Luck. Given a model's predicted distribution p over events and an observed event e, the *luck* of observing e is $\ell(e) = 1/p(e)$, the inverse probability. High luck means an unlikely event occurred. *Log luck* is $\log \ell(e) = -\log p(e) \geq 0$, with equality when $p(e) = 1$. Log luck is the Kullback information gained by observing e given the model's prior.

Atomic event. An atomic event $e_i \in E_i$ is an element of a single factor event space. It represents one distinguishable state of affairs within that factor—for example, "the switch is on" within the event space $\{\text{on}, \text{off}\}$. Contrast with the total event $e \in E = \prod_i E_i$, which is a joint assignment across all factors. An atomic event is atomic relative to a given factorization; a different factorization would yield different atoms. The information content of an atomic event is $\log |E_i|$ bits.

Atomic pattern. An atomic pattern is an ordered pair $(e_a \to e_b) \in E^2$ representing a directed influence from one event to another. Each atomic pattern may have a strength $s \in [0, 255]$: $s = 0$ means no influence, $s = 255$ means absolute (logical) implication, and intermediate values represent statistical association. The total pattern P is a sparse set of atomic patterns. In neural terms, an atomic pattern is a synapse; in ML terms, a weight. In our notation: "e_a" "e_b" 200. denotes an atomic pattern from e_a to e_b with strength 200.

Factorization. A factorization of the total event space is a decomposition $E = \prod_{i=1}^{k} E_i$ into factor event spaces. Factorization is an epistemic choice, not an ontological claim: it is how we choose to describe reality, not what reality is.

The total information is invariant: $I(E) = \sum_i I(E_i) = \sum_i \log|E_i| = \log|E|$. Refactoring E differently redistributes this information among factors but does not change its total.

Factorization determines sparsity: a pattern $p \in E^2$ that looks dense in one factorization may be sparse in another. The art of modeling is choosing factorizations that make the true patterns sparse.

Factorization determines contradiction: two events $e, e' \in E_i$ in the same factor are mutually exclusive. Assigning positive support to both is surprise; the event space defines what counts as contradictory.

Factorization determines interpretability: a model is interpretable when its factors correspond to human-meaningful categories. The problem of LLM interpretability is finding the domain-natural factorization hidden within the architecture-natural one.

Learning function. The learning function $\omega \in \Omega$ maps memory traces to changes in the pattern space P. A memory trace is a sequence of (e, t, p) tuples recorded over time—the raw record of experience.

The standard learning function ω_0 converts a memory trace into a learned product pattern (LPP) by recording log co-occurrence counts. Given observations $(i, o) \in I \times O$, it accumulates a log contingency table $p_{io} = \log(\text{count}(i, o))$. This is a sufficient statistic: it captures all that can be learned purely statistically from the data.

The log representation gives us Bayesian inference for free: $p(o|i) \propto \exp(p_{io})$, and updates are additive in log-space. This is why the standard learning function is standard: it is the minimal complete statistical learner.

More powerful learning functions can perform induction (setting patterns to 255 from finite evidence) or can refactor E itself. These go beyond statistics into the realm of understanding.

Product pattern. A product pattern arises from factoring E into input and output spaces: $E = I \times O$. The pattern is represented as a matrix p_{io} of strengths from $i \in I$ to $o \in O$. A learned product pattern (LPP) is the result of applying the standard learning function to a memory trace: it is the log contingency table of observed co-occurrences. The product pattern captures correlation: when i and o frequently co-occur, p_{io} is high. A fully-connected layer in an MLP is an example of a learned product pattern.

Total pattern. The total pattern $p \in P \subseteq \mathcal{P}(E^2)$ is a sparse set of atomic patterns. It describes how the total thought at one moment influences the total thought at the next: $t_{n+1} = f_p(t_n)$. In neural terms, the total pattern is the connectome—the set of effective synaptic connections. In ML terms, it is the weights Θ. In a scientific theory, it is the laws—the rules that constrain which states can follow which. The total pattern is typically sparse: most pairs (e_a, e_b) have no direct connection, and computation proceeds through chains of patterns.

Author declaration of the use of AI:
I have used AI to write as much of this paper as I could get the AI to understand.

Some of the appendices contain errors made by the AI that I have not yet had time to correct. If there are errors in the body of the work at this point they are probably my own.

Claude (Opus 4.5) has been very useful in working out mathematical notation and formalizing some proofs. GPT-5.2 has been useful as a critic: anything misunderstood by GPT-5.2 needs clarification or elaboration or restatement. The author's remaining task could be described as stating things simply enough that even GPT-5.2 can understand them.

Future work may replace some of these appendices with prompts, such that the reader can feed the paper itself and a short prompt into a current LLM and get back the desired lemma or explanation or proof.

References

[1] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 2554–2558.

[2] Cox, R. T. (1961). *The Algebra of Probable Inference.* Johns Hopkins Press.

[3] Wittgenstein, L. (1921). *Tractatus Logico-Philosophicus.* Translated by C.K. Ogden (1922). Routledge & Kegan Paul.

Appendix: ANN to UM Compilation

Given a trained neural network with layers $L_1, \ldots, L_n$, we construct a UM instance $u = (e, t, p, f, \omega)$.

Step 1: Event space construction. Let N be the total number of neurons across all layers. For each neuron j, define a binary event space $E_j = \{e_j, \bar{e}_j\}$ (fires, does not fire). The total event space is $E = \prod_{j=1}^{N} E_j$, with $|E| = 2^N$. Input and output token spaces embed into E via the input/output layers.

Step 2: Thought space. Take $T : E \to [0, 255]$. The activation $a_j \in [0, 1]$ of neuron j maps to support values: $t(e_j) = \lfloor 255 \cdot a_j \rfloor$ and $t(\bar{e}_j) = 255 - t(e_j)$. This quantization introduces error bounded by $1/255 \approx 0.4\%$ per neuron.

Step 3: Pattern space. For each weight w_{ij} from neuron i to neuron j:

- If $w_{ij} > 0$: add pattern $(e_i \to e_j)$ with strength $\propto w_{ij}$
- If $w_{ij} < 0$: add pattern $(e_i \to \bar{e}_j)$ with strength $\propto |w_{ij}|$

Normalize strengths to $[0, 255]$ within each layer.

Step 4: Update function. Define f_p as the forward pass: apply patterns layer by layer, using max-min or softmax as appropriate. The UM update $t' = f_p(t)$ corresponds to one forward pass of the network.

Claim (informal). This construction is functionally equivalent: $\Phi(\text{ANN})(x) \approx \text{ANN}(x)$ for all inputs x, with error bounded by quantization. A formal proof would establish the metric d and error bound ϵ such that $|f_{UM}(t) - f_{ANN}(t)| < \epsilon$.

Appendix: Explanation as Factorization and Translation

Event space and reference measure. Let E be the event space. Fix a reference probability measure μ on E (uniform on finite E unless stated otherwise). Let T denote the task/meaning space (e.g. labels, actions, outcomes). Let $t^\star : E \to T$ denote the target behavior.

ε-equality (agreement up to error). For functions $f, g : E \to T$, define

$$f =_\varepsilon g \quad :\Longleftrightarrow \quad \mu\{e \in E :\ f(e) \neq g(e)\} \leq \varepsilon.$$

This relation satisfies the triangle/union bound form:

$$f =_{\varepsilon_1} g \ \wedge \ g =_{\varepsilon_2} h \quad \Longrightarrow \quad f =_{\varepsilon_1 + \varepsilon_2} h.$$

Explanations as factoring map + decoder. An explanation/model M_i consists of

$$\phi_i := (\pi_i, \sigma_i) : E \to A_i \times B_i =: Z_i \qquad \text{(factoring/encoding)}$$

and

$$d_i : Z_i \to T \qquad \text{(decoder/policy/predictor)}.$$

The induced behavior is

$$\hat{t}_i := d_i \circ \phi_i : E \to T,$$

and its error (w.r.t. $t^\star$) is

$$\varepsilon_i := \mu\{e \in E :\ \hat{t}_i(e) \neq t^\star(e)\}.$$

Equivalently,

$$\hat{t}_i =_{\varepsilon_i} t^\star.$$

Meaning-preserving translation (weak claim). A translation from M_1-coordinates to M_2-coordinates is a map

$$\tau_{1\to 2} : Z_1 \to Z_2$$

that preserves decoded meaning:

$$d_2 \circ \tau_{1\to 2} \approx d_1.$$

A convenient sufficient condition for existence is a *section* (a chosen representative) of each decoder:

$$\rho_i : T \to Z_i \quad \text{such that} \quad d_i \circ \rho_i = \mathrm{id}_T \quad (\text{at least on } \mathrm{im}(d_i)).$$

Given such ρ_2, define the translator

$$\boxed{\tau_{1\to 2} := \rho_2 \circ d_1}.$$

Then the commuting law holds (exactly on the domain where the section is defined):

$$\boxed{d_2 \circ \tau_{1\to 2} = d_1}.$$

Consequently, translated predictions from M_1 evaluated by M_2 agree with M_1:

$$d_2 \circ \tau_{1\to 2} \circ \phi_1 = d_1 \circ \phi_1 = \hat{t}_1 =_{\varepsilon_1} t^\star.$$

In particular, if both explanations are accurate,

$$\hat{t}_1 =_{\varepsilon_1} t^\star \quad \text{and} \quad \hat{t}_2 =_{\varepsilon_2} t^\star,$$

then by the triangle property,

$$\boxed{\hat{t}_1 =_{\varepsilon_1+\varepsilon_2} \hat{t}_2}.$$

Equivalently,

$$\mu\{e \in E : \ \hat{t}_1(e) = \hat{t}_2(e)\} \ \geq \ 1 - (\varepsilon_1 + \varepsilon_2).$$

Visibility of factors. Since $Z_i = A_i \times B_i$, any section decomposes as $\rho_i(t) = (\rho_i^A(t), \rho_i^B(t))$ and thus

$$\tau_{1\to 2}(a_1, b_1) = \rho_2(d_1(a_1, b_1)) = \big(\rho_2^A(d_1(a_1, b_1)),\ \rho_2^B(d_1(a_1, b_1))\big),$$

making the translated factors explicit componentwise.

Latent-alignment translation (strong claim; extra assumptions). The weak translation guarantees agreement of *decoded meaning* (T) but does not by itself imply alignment of internal coordinates on E (i.e. it need not be that $\tau_{1\to 2} \circ \phi_1 \approx \phi_2$). A stronger statement requires an identifiability/minimality condition: roughly, that ϕ_i is an (approx.) minimal sufficient representation for $t^\star$.

One convenient formulation: call $\phi : E \to Z$ *ε-sufficient* for $t^\star$ if $\exists d : Z \to T$ with $d \circ \phi =_\varepsilon t^\star$. Call ϕ *(approx.) minimal* if every ε-sufficient $\psi : E \to W$ factors through ϕ (up to small additional error), i.e. $\exists m : W \to Z$ with

$$\phi =_\delta m \circ \psi.$$

Under mutual minimality (with compatible parameters), two such representations are related by (approx.) inverse maps between their latent spaces, yielding (approx.) latent alignment:

$$\tau_{1\to 2} \circ \phi_1 =_\eta \phi_2 \qquad \text{and} \qquad \tau_{2\to 1} \circ \phi_2 =_{\eta'} \phi_1,$$

with $\tau_{2\to 1} \circ \tau_{1\to 2} \approx \mathrm{id}_{Z_1}$ and $\tau_{1\to 2} \circ \tau_{2\to 1} \approx \mathrm{id}_{Z_2}$. (Parameters δ, η, η' depend on the chosen minimality notion and error accounting.)

Summary. Two accurate explanations (small $\varepsilon_1 + \varepsilon_2$) are nearly identical at the level of induced behavior $\hat{t}_i : E \to T$. Moreover, given a decoder section, there exists an explicit translator $\tau_{1\to 2} : Z_1 \to Z_2$ that preserves decoded meaning; stronger claims about aligning latents require additional identifiability/minimality assumptions.

Appendix: Sufficiency of ω_0

We show that the standard learning function ω_0 produces a sufficient statistic for the function $I \to O$.

Setup. Given observations $D = \{(i_1, o_1), \ldots, (i_n, o_n)\} \subset I \times O$, the standard learning function computes the log contingency table:

$$p_{io} = \log(\#\{k : (i_k, o_k) = (i, o)\})$$

for each $(i, o) \in I \times O$, where # denotes count.

Claim. The table (p_{io}) is a sufficient statistic for the conditional distribution $P(O|I)$.

Proof sketch. By the Fisher-Neyman factorization theorem, a statistic $T(D)$ is sufficient for parameter θ iff the likelihood factors as $L(\theta; D) = g(T(D), \theta) \cdot h(D)$ where h does not depend on θ.

For a multinomial model where (i, o) pairs are drawn i.i.d., the likelihood is:

$$L(\theta; D) = \prod_{(i,o)} \theta_{io}^{n_{io}}$$

where n_{io} is the count of (i, o) pairs and $\theta_{io} = P(i, o)$.

This depends on D only through the counts (n_{io}), hence the count table (equivalently, its log) is sufficient.

Implication. No learning function can extract more information about $P(O|I)$ from the data than ω_0 does. Any function that discards the counts loses information; any function that adds to them goes beyond statistics into induction.

Appendix: Negative Weight Construction

Neural networks use negative weights to implement inhibition: if neuron i fires, neuron j should *not* fire. The UM uses only positive pattern strengths. We reconcile these by doubling the event space.

Construction. For each neuron j with activation $a_j \in [0, 1]$, introduce two events:

- e_j: neuron j fires (activation high)
- $\bar{e}_j$: neuron j does not fire (activation low)

These form a binary event space $E_j = \{e_j, \bar{e}_j\}$ with the constraint $t(e_j) + t(\bar{e}_j) = 255$.

Weight translation. For weight w_{ij} from neuron i to neuron j:

- $w_{ij} > 0$: pattern $(e_i \to e_j)$ with strength $s = \alpha|w_{ij}|$
- $w_{ij} < 0$: pattern $(e_i \to \bar{e}_j)$ with strength $s = \alpha|w_{ij}|$

where α is a normalization constant chosen per layer.

Why this works. In the original network, a negative weight $w_{ij} < 0$ means: "when i is active, decrease the pre-activation of j." In the UM, the pattern $(e_i \to \bar{e}_j)$ means: "when e_i has support, give support to $\bar{e}_j$." Since e_j and $\bar{e}_j$ compete within the same event space, supporting $\bar{e}_j$ effectively suppresses e_j.

Cost. This construction doubles $|E|$ from N neurons to $2N$ events. The information content is unchanged: each binary event space contributes 1 bit, same as a binary neuron.

Appendix: Connection to Hopfield Networks

Hopfield (1982) introduced a model of associative memory that stores patterns via correlation matrices. We translate his framework into UM terms.

Hopfield's setup. A network of N neurons, each with state $V_i \in \{0, 1\}$. The total state is a vector $V = (V_1, \ldots, V_N)$. Neurons are connected by symmetric weights $T_{ij} = T_{ji}$. A neuron updates by:

$$V_i \leftarrow \begin{cases} 1 & \text{if } \sum_j T_{ij} V_j > U_i \\ 0 & \text{otherwise} \end{cases}$$

where U_i is a threshold.

Hopfield's storage prescription. To store s memory patterns $V^{(1)}, \ldots, V^{(s)}$, set:

$$T_{ij} = \sum_{k=1}^{s} (2V_i^{(k)} - 1)(2V_j^{(k)} - 1)$$

The term $(2V_i - 1)$ converts $\{0,1\}$ to $\{-1,+1\}$. The product $(2V_i^{(k)} - 1)(2V_j^{(k)} - 1)$ is $+1$ if neurons i and j have the same state in pattern k, and -1 if different. Summing over patterns: T_{ij} counts agreement minus disagreement.

Translation to UM.

Hopfield	**UM**
Neuron i	Event $e_i \in E$ (or event space $E_i = \{e_i, \bar{e}_i\}$)
State V_i	Thought $t(e_i) \in \{0, 255\}$
Weight T_{ij}	Pattern strength p_{ij}
Memory pattern $V^{(k)}$	Observed joint event (memory trace element)
Storage prescription	Standard learning function ω_0

The correspondence in detail. Hopfield's storage prescription computes:

$$T_{ij} = \sum_{k} (2V_i^{(k)} - 1)(2V_j^{(k)} - 1) = \#(\text{agree}) - \#(\text{disagree})$$

Our standard learning function records co-occurrences:

$$p_{ij} = \log(\#\{k : e_i \text{ and } e_j \text{ both active in observation } k\})$$

The difference: Hopfield uses signed counts (agreement minus disagreement); we use log counts of co-activation.

With the doubled event space $E_i = \{e_i, \bar{e}_i\}$, these coincide:

- Pattern from e_i to e_j when both active: corresponds to $+1$ in Hopfield's sum
- Pattern from e_i to $\bar{e}_j$ when i active, j inactive: corresponds to -1 in Hopfield's sum

The UM representation with doubled E and positive-only patterns is equivalent to Hopfield's signed weights.

Content-addressable memory. Hopfield showed: presenting a partial or noisy pattern causes the network to settle into the nearest stored pattern. In UM terms: initializing t with partial information about a stored event, then iterating f_p, recovers the full event. This is why the standard learning function produces a sufficient statistic: the co-occurrence matrix contains everything needed for recall.

Capacity. Hopfield networks reliably store approximately $0.15N$ patterns for N neurons. This is the information-theoretic limit of correlation storage. Beyond this, patterns interfere and recall fails. The UM inherits this limit: ω_0 on unfactored E has bounded capacity. Factoring E into smaller event spaces increases effective capacity by reducing interference.

Appendix: Category-Theoretic Presentation

The Universal Model admits a natural category-theoretic formulation that clarifies the relationships between its components.

The category **UM**. Objects are triples (E, T, P) where:

- E is an event space (finite set, possibly factored as $\prod_i E_i$)
- $T = [0, 255]^E$ is the belief space over E

- $P \subseteq E \times E \times [0, 255]$ is a pattern space (weighted relations)

Morphisms. A morphism $\phi : (E_1, T_1, P_1) \to (E_2, T_2, P_2)$ consists of:

- A function $\phi_E : E_1 \to E_2$ (event embedding)
- An induced map $\phi_T : T_1 \to T_2$ given by $\phi_T(t)(e_2) = \max\{t(e_1) : \phi_E(e_1) = e_2\}$
- A pattern map $\phi_P : P_1 \to P_2$ preserving structure: if $(e, e', s) \in P_1$, then $(\phi_E(e), \phi_E(e'), s) \in P_2$

These must commute with update: $\phi_T(f_p(t)) = f_{\phi_P(p)}(\phi_T(t))$.

Key arrows within a single UM.

- **Update** $f_p : T \to T$ — apply pattern p to advance belief
- **Learning** $\omega : T \times E \to P$ — from belief and observation to revised patterns
- **Projection** $\pi_i : E \to E_i$ — attention to factor i of a factored space
- **Injection** $\iota_i : E_i \to E$ — embedding a factor back into the whole

Adjunctions. The projection-injection pair (π_i, ι_i) forms an adjunction: $\pi_i \dashv \iota_i$. This captures the intuition that focusing on a factor (projection) and embedding into context (injection) are dual operations.

More speculatively: if we view induction $I : D \to T$ (from data to belief) and deduction $D : T \times P \to T$ (from belief and pattern to new belief), there may be an adjoint relationship capturing the asymmetry between evidence-gathering and inference.

The ANN-to-UM functor. The compilation map $\Phi : \mathbf{ANN} \to \mathbf{UM}$ from §**??** should be a functor:

- Φ sends each neural network to a UM instance
- Φ sends network morphisms (weight-preserving maps) to UM morphisms
- Φ preserves composition: $\Phi(g \circ f) = \Phi(g) \circ \Phi(f)$

This functoriality ensures that the translation is systematic, not ad hoc.

Appendix: Formalizing Human Chess Skill

A human chess player can be modeled as a UM instance $u = (e, t, p, f, \omega)$ where:

Event Spaces. Factor E into:

- E_{board}: board states ($\sim 10^{44}$ legal positions, but the player works with compressed representations)
- E_{clock}: time control states (binned intervals: > 1 hour, 30–60 min, etc.)
- E_{eval}: position evaluation events (“winning,” “equal,” “losing,” with gradations)
- E_{plan}: strategic plan events (“attack kingside,” “improve knight,” etc.)
- E_{tactic}: tactical pattern events (“pin on e-file,” “back rank weakness,” etc.)
- E_{action}: output events (moves, clock actions, resignation, draw offer)

Thoughts. The total thought t assigns support to each event. A chess position activates multiple events simultaneously: the board state, an evaluation, recognized patterns, candidate plans. Internal deliberation is the update $f(t) \to t'$ cycling through candidate moves until an action threshold is reached.

Patterns. The player’s skill resides in P:

- Perceptual patterns: board $\to$ tactical features (learned, statistical)

- Evaluative patterns: features $\to$ evaluation (learned, may approach 255 for known endgames)
- Planning patterns: evaluation + features $\to$ candidate plans
- Selection patterns: plans + time $\to$ action

A grandmaster's patterns differ from an amateur's primarily in granularity and strength.

The Explanatory Sufficiency Principle. If two players (or a player and an engine) both achieve high accuracy, their factorizations must overlap substantially. The overlap is at least $1 - \epsilon_1 - \epsilon_2$ where ϵ_i is player i's error rate. This means we can, in principle, translate "king safety intuition" to "tree search depth on king-attack lines"—they correlate because they both track the same underlying truth (checkmate probability).

Experimental Predictions. This model predicts:

- Blunders increase when time pressure disrupts $E_{\text{clock}} \to E_{\text{action}}$ patterns
- Fatigue degrades statistical patterns before absolute ones (tactics before endgame knowledge)
- A player's "style" is the relative strength of evaluative vs. tactical patterns

These predictions are testable against large game databases.

Appendix: Algebra on Universal Models

We define algebraic operations on UMs, starting with the simplest modifications. Throughout, we use lowercase u, u_1, u_2 for model instances and uppercase E, T, P for their component spaces.

The zero model u_0. The zero model has $E = \emptyset$, $T = \{0\}$, $P = \emptyset$, trivial f and ω. It represents "no knowledge, no capacity to learn." For any u: $u + u_0 = u$ (disjoint union with empty is identity) and $u \times u_0 = u_0$ (product with empty event space collapses).

Minimal additions. We can modify a model $u = (e, t, p, f, \omega)$ by adding one element:

Add one event: $E' = E \cup \{e_{\text{new}}\}$. This expands what the model can represent. Cost: $\log |E'| - \log |E| \approx 1/|E|$ bits (negligible for large E).

Add one atomic pattern: $P' = P \cup \{(e_a \to e_b, s)\}$ for some $e_a, e_b \in E$ and strength s. This adds one causal connection. Cost: $\log |E|^2 + \log 256 = 2 \log |E| + 8$ bits to specify.

Add one bit to f: Extend the description of f by one bit, doubling the space of possible update functions. This allows f to make one additional distinction.

Add one bit to ω: Similarly for the learning function.

Sum of models: $u_1 + u_2$. The sum represents "either u_1 or u_2 but not both."

$$\begin{aligned} E_{1+2} &= E_1 \sqcup E_2 \quad \text{(disjoint union)} \\ T_{1+2} &= T_1 \sqcup T_2 \\ P_{1+2} &= P_1 \sqcup P_2 \quad \text{(no cross-model patterns)} \end{aligned}$$

This is coproduct in the category of UMs. Example: a model that handles either English or French but treats them as separate.

Product of models: $u_1 \times u_2$. The product represents "both u_1 and u_2 simultaneously."

$$\begin{aligned} E_{1\times 2} &= E_1 \times E_2 \\ T_{1\times 2} &= T_1 \times T_2 \\ P_{1\times 2} &= (P_1 \times E_2) \cup (E_1 \times P_2) \cup P_{\text{cross}} \end{aligned}$$

Here $P_1 \times E_2$ means patterns from u_1 acting on the first component while the second is unchanged, and $P_{\text{cross}} \subseteq (E_1 \times E_2)^2$ contains any cross-model patterns. Without cross-patterns ($P_{\text{cross}} = \emptyset$), the models run independently. With cross-patterns, they interact.

Example: Chess $\times$ Clock. Let u_{chess} model board positions with patterns for legal moves and evaluations. Let u_{clock} model time remaining with patterns for urgency levels. The product $u_{\text{chess}} \times u_{\text{clock}}$ has events like (position, time-left). Cross-patterns encode time pressure: "(complex position, low time) $\to$ (prefer simple continuation)" with high strength.

Quotient: $u/\sim$. Given an equivalence relation $\sim$ on E, the quotient collapses equivalent events:

$$E_{/\sim} = E/\sim = \{[e] : e \in E\}$$

How patterns lift. Given $p : e_a \to e_b$ with strength s, define $p_{/\sim} : [e_a] \to [e_b]$ with strength

$$s_{/\sim}([e_a], [e_b]) = \max_{e'_a \in [e_a], e'_b \in [e_b]} s(e'_a, e'_b).$$

We take the maximum strength over all pairs in the equivalence classes. This is well-defined but may lose information: distinct patterns may merge.

Example. Tokenization is a quotient—many character sequences map to one token. The pattern "s-t-r $\to$ likely followed by vowel" becomes "str $\to$ likely followed by vowel" at the token level.

Embedding: $u_1 \hookrightarrow u_2$. An embedding is an injective morphism: u_1 sits inside u_2 as a submodel. Formally, ϕ_E, ϕ_T, ϕ_P are all injective and structure-preserving. Example: a model of tic-tac-toe embeds into a model of all 3×3 grid games.

Composition: $u_1 \circ u_2$. If u_1's output space is compatible with u_2's input space (i.e., there exists $\phi : O_1 \to I_2$), we can compose:

$$(u_1 \circ u_2)(e) = u_2(\phi(u_1(e)))$$

This is function composition on the induced maps $E \to T$. Example: perception model composed with decision model.

The identity model $u_{id}^{(E)}$. For each event space E, there is an identity model: $E = T$, $P = \{(e \to e, 255) : e \in E\}$, $f = \text{id}$. For any u with matching types: $u \circ u_{\text{id}}^{(O_u)} = u$ and $u_{\text{id}}^{(I_u)} \circ u = u$. Note: identity models are indexed by event space; there is no single universal identity.

Morphisms. A morphism $\phi : u_1 \to u_2$ consists of maps (ϕ_E, ϕ_T, ϕ_P) preserving structure:

- $\phi_E : E_1 \to E_2$ (events map to events)
- $\phi_T : T_1 \to T_2$ such that $\phi_T(\iota_1(e)) = \iota_2(\phi_E(e))$ (embedding commutes)
- $\phi_P : P_1 \to P_2$ such that $\phi_P(e_a \to e_b) = \phi_E(e_a) \to \phi_E(e_b)$
- $f_2 \circ \phi_T = \phi_T \circ f_1$ (update commutes)

An isomorphism is a bijective morphism with bijective inverse. Isomorphism implies behavioral equivalence (same input-output function), but not conversely: two models may compute the same function while having different internal structure.

Information under operations. Let $I(u) = \log|E| + \log|T| + \log|P| + \ldots$ denote total information.

- $I(u_1 + u_2) = I(u_1) + I(u_2) + 1$ bit (to select which model)
- $I(u_1 \times u_2) = I(u_1) + I(u_2)$ (total information is additive under product)
- $I(u/\sim) \le I(u)$ (quotient loses information; equality iff $\sim$ is trivial)

Factorization as inverse of product. The main text emphasizes factoring E into $E_1 \times \cdots \times E_k$. In algebraic terms, this means finding $u_1, \ldots, u_k$ such that $u \cong u_1 \times \cdots \times u_k$ (with minimal cross-patterns). A model is *fully factorizable* if it decomposes with $P_{\text{cross}} = \emptyset$; such models consist of independent submodels. Interpretability seeks factorizations where the E_i have human-meaningful names.

Algebraic properties. Sum and product satisfy the expected identities (up to isomorphism):

- Associativity: $(u_1 + u_2) + u_3 \cong u_1 + (u_2 + u_3)$, and similarly for $\times$
- Commutativity: $u_1 + u_2 \cong u_2 + u_1$, and similarly for $\times$
- Distributivity: $u \times (v + w) \cong (u \times v) + (u \times w)$
- Identity: $u + u_0 \cong u$; $u \times u_1 \cong u$ where u_1 is the trivial one-event model
- Zero: $u \times u_0 \cong u_0$

Thus UMs with $+$ and $\times$ form a commutative semiring (up to isomorphism). Composition is associative but not commutative.

Categorical structure. UMs with morphisms form a category **UM**. Sum is coproduct; product is product; quotient is coequalizer. The zero model u_0 is initial (unique morphism $u_0 \to u$ for any u). There is no terminal object: a model receiving morphisms from all others would need to "forget" all distinctions, but even u_0 has nowhere to send its (vacuous) structure.

Appendix: Character-to-Token Factor Map

Tokenization is a quotient map: many character sequences become one token. We trace the factorization from bits to tokens and compare how RNNs and Transformers implement it.

The tokenization chain.

$$2^\ell \xrightarrow{\phi_1} A^n \xrightarrow{\phi_2} V^m$$

- 2^ℓ: raw bits (the physical medium)
- A^n: characters/bytes ($A = \{0, \ldots, 255\}$, so $|A| = 256 = 2^8$)
- V^m: tokens (V is vocabulary, typically $|V| = 2^{16}$ for modern LLMs)

Each arrow is a quotient: ϕ_1 groups 8 bits into one byte; ϕ_2 groups variable-length byte sequences into one token.

Worked example. The string " the" (space-t-h-e) in GPT-2:

- Bytes: [32, 116, 104, 101] (ASCII codes)
- Bits: 32 bits total (4×8)
- Token: 262 (a single vocabulary index)

The tokenizer maps 32 bits of input to $\log_2 |V| \approx 16$ bits of token index. Compression ratio: $32/16 = 2\times$. The model sees one position instead of four.

RNN factorization. An RNN processes bits (or bytes) sequentially:

$$P_{\text{RNN}} = P_{\text{step}}^n$$

where $P_{\text{step}} : H \times A \to H$ is the recurrent update (hidden state plus input to new hidden state). For character-level processing, $n = 4$ steps for " the". The hidden state h accumulates information via shift-and-add:

$$h_{t+1} = \sigma(W_h h_t + W_x x_t + b)$$

Time factorization: 4 sequential steps, each using the same P_{step}.

Transformer factorization. A Transformer processes tokens in parallel:

$$P_{\text{Trans}} = P_{L_k} \circ \cdots \circ P_{L_1}$$

where each P_{L_i} is one layer (attention + feedforward). For " the", the entire token is processed at once: 1 position, k layers. Depth factorization: 1 token processed through k layers simultaneously via attention.

Same function, different factorization. Both architectures can compute the same input-output function (given enough capacity). The difference is how they factor the computation:

	RNN	**Transformer**
Input unit	byte/char	token
Time steps	n (sequence length)	1 (per token)
Depth	1 (recurrent)	k (layers)
Factorization	P^n_{step}	$P_{L_k} \circ \cdots \circ P_{L_1}$
Parallelism	Sequential	Parallel over positions

The translation map. Given an RNN with hidden state h and a Transformer with residual stream r, the translation $\phi : H \to R$ maps:

$$\phi(h_{\text{after } n \text{ chars}}) \approx r_{\text{after token}}$$

if both models have learned the same function. Finding ϕ explicitly is an interpretability problem. The principle of explanatory sufficiency guarantees ϕ exists when both models are accurate.

Appendix: Boolean Circuits and Layer Depth

The standard update function determines what logical operations a single layer can compute.

The standard update is OR-like. Recall the standard update:

$$(f_p(t))_j = \max_i \min(t_i, p_{ij})$$

If any input t_i with a strong pattern p_{ij} to output j is active, then j activates. This is disjunction: j fires if i_1 OR i_2 OR . . . fires (for any i with $p_{ij} > 0$).

One layer computes OR only. With one layer and positive patterns, we can compute:

- OR: multiple patterns to the same output
- Identity: single pattern $i \to j$
- Fan-out: single input to multiple outputs

We cannot compute AND in one layer with positive patterns alone.

AND requires two layers. To compute $j = i_1 \wedge i_2$, we need:

- Layer 1: Check i_1, produce intermediate h only if i_1 is active
- Layer 2: Check both h and i_2, but this still gives OR

The trick: use the doubled event space. Let $\bar{i}_1$ mean "i_1 is not active."

- Layer 1: Pattern $i_1 \to h_1$ and $i_2 \to h_2$
- Layer 2: Pattern $\bar{h}_1 \to \bar{j}$ and $\bar{h}_2 \to \bar{j}$

Now j is active only if neither $\bar{h}_1$ nor $\bar{h}_2$ activated $\bar{j}$—i.e., only if both h_1 and h_2 are active. This is AND via De Morgan: $i_1 \wedge i_2 = \neg(\neg i_1 \vee \neg i_2)$.

n-way AND requires depth n. To compute $j = i_1 \wedge i_2 \wedge \cdots \wedge i_n$, we can chain:

- Layer 1: Check i_1, produce h_1 if present
- Layer 2: Check $h_1 \wedge i_2$, produce h_2
- . . .
- Layer n: Check $h_{n-1} \wedge i_n$, produce j

Each layer adds one conjunct. Depth n is necessary and sufficient for n-way AND.

Character recognition example. To recognize a specific 8-bit character (e.g., ASCII 'A' = 01000001):

- Input: 8 bits $b_0, \ldots, b_7$ and their complements $\bar{b}_0, \ldots, \bar{b}_7$
- Required: $\bar{b}_0 \wedge b_1 \wedge \bar{b}_2 \wedge \bar{b}_3 \wedge \bar{b}_4 \wedge \bar{b}_5 \wedge \bar{b}_6 \wedge b_7$
- Depth: 8 layers (one per bit check)

This explains why transformers need multiple layers even for simple pattern matching.

Circuit depth equals layer count.

Circuit	**UM**
OR gate	one layer, multiple input patterns
AND gate	two layers via De Morgan
NOT gate	swap $e \leftrightarrow \bar{e}$ (free with doubled E)
n-way AND	n layers
circuit depth d	d layers

Any Boolean circuit of depth d can be implemented in a UM with d layers.

Appendix: Mate in One

We construct a minimal UM for a chess player recognizing and executing checkmate in one.

Position. White to play. White has King on g1, Queen on h5, Rook on f1. Black has King on g8, pawns on f7, g7, h7. The winning move is Qxh7#.

Event space factorization. We factor E minimally for this problem:

$$\begin{aligned}
E_{\text{board}} &= \{\text{current position}\} \quad \text{(singleton for this example)}\\
E_{\text{piece}} &= \{Q, R, K, \text{none}\} \quad \text{(which piece we're considering)}\\
E_{\text{square}} &= \{a1, \ldots, h8\} \quad \text{(64 squares)}\\
E_{\text{legal}} &= \{0, 1\} \quad \text{(is the considered move legal?)}\\
E_{\text{check}} &= \{0, 1\} \quad \text{(does the move give check?)}\\
E_{\text{escape}} &= \{0, 1\} \quad \text{(can the king escape?)}\\
E_{\text{action}} &= \{\text{move}, \text{think}\} \quad \text{(output)}
\end{aligned}$$

Key patterns. The player's skill includes these atomic patterns (all strength 255):

Move generation:

$$(\text{piece} = Q, \text{square} = h7) \to (\text{legal} = 1)$$

The queen can legally move to h7 (capture is legal).

Check detection:

$$(\text{piece} = Q, \text{square} = h7, \text{legal} = 1) \to (\text{check} = 1)$$

Queen on h7 attacks the king on g8.

Escape analysis:

$$(\text{check} = 1, \text{king on } g8, \text{Q on } h7, \text{R on } f1) \to (\text{escape} = 0)$$

King cannot move to f8 (rook), h8 (queen), f7/g7/h7 (blocked or queen).

Mate recognition:

$$(\text{check} = 1, \text{escape} = 0) \to (\text{pattern: mate})$$

Policy:

$$(\text{pattern: mate}, \text{piece} = Q, \text{square} = h7) \to (\text{action} = \text{move})$$

If we see checkmate, play it immediately.

Thought evolution. Initial t_0: high support for board position, low support elsewhere. After $f_p(t_0) = t_1$: support propagates through move generation, check detection. After $f_p(t_1) = t_2$: escape analysis completes, mate pattern activates. After $f_p(t_2) = t_3$: action = move reaches threshold.

The entire sequence—see position, recognize mate, decide to play it—is four applications of f_p.

Why this is hard to introspect. A grandmaster executes this in milliseconds via compiled patterns. The intermediate events (legal, check, escape) may not surface in consciousness. The player reports "I saw Qxh7 mate" without accessing the derivation. The patterns exist; they're just too fast and automatic to observe.

Appendix: Two-Body Gravity

We construct a UM instance for the Sun-Earth system under Newtonian gravity, showing one timestep of evolution.

Event space factorization. Factor E into:

$$\begin{aligned} E_{\text{pos}} &= \{(x_1, y_1, x_2, y_2) : x_i, y_i \in \mathbb{Z}, |x_i|, |y_i| \leq L\} \\ E_{\text{vel}} &= \{(v_{x1}, v_{y1}, v_{x2}, v_{y2}) : v_i \in \mathbb{Z}, |v_i| \leq V\} \\ E_{\text{mass}} &= \{(m_1, m_2) : m_i \in \mathbb{Z}^+, m_i \leq M\} \end{aligned}$$

where subscript 1 denotes the Sun and 2 denotes the Earth. Discretizing to integers with bounds L, V, M keeps $|E|$ finite. The total event space is $E = E_{\text{pos}} \times E_{\text{vel}} \times E_{\text{mass}}$.

Initial thought. Take $t \in T$ as the one-hot encoding of the current state:

$$t(e) = \begin{cases} 255 & \text{if } e = (x_1^0, y_1^0, x_2^0, y_2^0, v_{x1}^0, \ldots, m_2^0) \\ 0 & \text{otherwise} \end{cases}$$

That is, we know the exact state with certainty.

Patterns for Newton's laws. The physics decomposes into atomic patterns. Let $r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ be the distance.

Force calculation: For each configuration of positions and masses, we have a pattern:

$$(x_1, y_1, x_2, y_2, m_1, m_2) \to (F_x, F_y)$$

where $F_x = Gm_1m_2(x_2 - x_1)/r^3$ and $F_y = Gm_1m_2(y_2 - y_1)/r^3$. These are deterministic (strength 255).

Acceleration:

$$(F_x, F_y, m_i) \to (a_{xi}, a_{yi}) \quad \text{where } a = F/m$$

Velocity update:

$$(v_{xi}, v_{yi}, a_{xi}, a_{yi}) \to (v'_{xi}, v'_{yi}) \quad \text{where } v' = v + a \cdot \Delta t$$

Position update:

$$(x_i, y_i, v'_{xi}, v'_{yi}) \to (x'_i, y'_i) \quad \text{where } x' = x + v' \cdot \Delta t$$

Update function. The standard update f_p chains these patterns: positions and masses $\to$ forces $\to$ accelerations $\to$ new velocities $\to$ new positions. Each layer applies max-min but since all patterns have strength 255 and inputs are one-hot, this reduces to function composition. One application of f_p advances the system by Δt.

Factorization matters. Without factorization, we would need $|E|^2$ patterns (one per state transition). With factorization into position, velocity, mass, force, and acceleration, we need $O(|E_{\text{pos}}| + |E_{\text{vel}}| + |E_{\text{mass}}|)$ patterns—exponentially fewer. This is why physics is possible: the laws factor.

Appendix: Circuit Depth and Cognition

We explore the connection between pattern complexity (as measured by depth) and cognitive phenomena.

Pattern depth. A pattern $p = (e_1 \to e_2)$ has depth 1. A compound pattern $p_1 \circ p_2 = ((e_1 \to e_2) \to e_3)$ has depth 2. In neural terms, depth corresponds to the number of sequential layers required to compute the pattern.

AND requires depth. A single neuron computes a threshold function: "fire if weighted sum exceeds threshold." This is OR-like: "if ANY sufficient subset of inputs is active, fire." To compute AND ("fire only if ALL inputs are active"), we need to chain neurons—each checking one condition before passing to the next. Thus AND has inherent depth.

System 1 and System 2. Kahneman's distinction corresponds to circuit depth:

- **System 1 (fast):** Shallow circuits, chunked patterns, depth 1–2. Immediate recognition, intuition.
- **System 2 (slow):** Deeper circuits, serial verification, depth 10+. Deliberate reasoning, checking.

The "slow" feeling of System 2 reflects the sequential nature of deep patterns—each step must complete before the next begins.

The depth limit. Finite-precision representations can only carry finite information across processing steps. Each step accumulates entropy; when accumulated entropy exceeds representational precision, information is lost.

The maximum pattern depth is:

$$d_{\max} = \frac{\text{precision (bits)}}{\text{entropy per step}}$$

For neural representations with ∼20–30 bits effective precision and ∼3–4 bits per reasoning step, this gives $d_{\max} \approx 7$—the classic working memory limit.

This is not coincidence. The "magic number 7 ± 2" IS the depth limit imposed by finite neural precision. Different tasks have different entropy per step, explaining the variance.

Chunking as depth compression. An expert's "chunk" is a compiled pattern: what once required depth n (learning to recognize a configuration step by step) becomes depth 1 (immediate recognition). A chess grandmaster sees "weak pawn structure" in one glance; a beginner must check multiple conditions sequentially.

Expertise acquisition is pattern compilation: converting deep AND-chains into shallow lookups. This explains why expertise is often unconscious—the derivation that built the chunk is gone, leaving only the compiled result.

Implications. The depth limit is fundamental, not a bug. To reason deeper:

- Chunk intermediate results (reduce depth at cost of learning time)
- Use external memory (paper, computers) to extend precision
- Match factorization to domain (reduce entropy per step)

The concrete companion paper develops the mathematics fully.

Appendix: Trust and the Source of 255

A support value of 255 represents certainty—unwillingness to update. Where does certainty come from?

Three sources of 255.

1. Definition. "Bachelor" means "unmarried man"—by construction, with certainty 255. Definitional truths are set when we establish event spaces. They cost nothing to verify because they're true by fiat.

2. Proof. "$2 + 2 = 4$"—derived from axioms via valid inference. Each step preserves 255; the conclusion inherits certainty from the premises. Proof transfers 255 from axioms to theorems.

3. Abduction. "The sun will rise tomorrow"—a decision, not an inference. No amount of evidence logically entails future events. We *choose* to treat it as certain because acting otherwise is impractical.

Abduction is choosing 255. Evidence is never sufficient for certainty about contingent facts. Yet we must act, and acting requires commitment. Abduction is the decision to set a pattern to 255 based on:

- Strong (but not conclusive) evidence
- Tolerable cost of error

- High cost of perpetual uncertainty

This is rational when the expected value of commitment exceeds the expected value of continued deliberation.

Trust as delegated abduction. Most of our 255s come from other people: teachers, books, institutions. We don't personally verify that Paris is the capital of France; we trust sources. Trust is outsourcing the abduction decision: "I set this to 255 because source s set it to 255, and I trust s."

The trust equation has the same structure as the standard update:

$$t(P) = \max_s \min(\text{trust}(s), \text{support}_s(P))$$

My belief in P is the maximum over sources s of the minimum of my trust in s and s's support for P.

Root 255s. Some certainties are foundational—set by architecture, not learning:

- "I exist" (Descartes' cogito)
- "Modus ponens is valid" (logic itself)
- "The past happened" (memory is evidence)
- "My sensory experience correlates with reality" (minimal realism)

These cannot be derived; they are presupposed by any derivation. Denying them is coherent but unlivable.

LLMs and trust. Current LLMs have no trust mechanism:

- All training data is weighted equally (by frequency, not reliability)
- They learn correlations, never setting patterns to 255
- They cannot abduce because there is no "self" to commit
- Contradictory information averages rather than resolves

An LLM that could track provenance and assign trust would be categorically more capable. This requires extending ω to update trust weights, not just pattern strengths.

Appendix: Smooth Event Spaces and Tractability

We show that factoring event spaces is integer factorization, and that tractability requires smooth (highly composite) cardinalities.

Event spaces map to natural numbers. Every finite event space E has cardinality $|E| = n$ for some $n \in \mathbb{N}$. Factoring E means finding event spaces E_1, E_2 such that $E \cong E_1 \times E_2$, which requires $|E_1| \cdot |E_2| = n$. This is exactly the problem of integer factorization.

Smooth numbers and tractability. An integer is *smooth* if it has only small prime factors. For example, $60 = 2^2 \times 3 \times 5$ is 5-smooth (no prime factor exceeds 5), while 59 is prime and cannot be factored at all.

When $|E|$ is smooth, we can decompose E into small factors:

$$E = E_1 \times E_2 \times \cdots \times E_k, \quad \text{each } |E_i| \text{ small}$$

This makes computation tractable: instead of $|E|$ operations, we need $\sum_i |E_i|$ operations for many algorithms.

When $|E|$ is prime, E is atomic—it cannot be factored and must be treated monolithically.

Reality is smooth. Physical event spaces tend to have highly composite cardinalities:

- Binary distinctions: $|E_i| = 2$
- Bytes: $|E_i| = 256 = 2^8$

- Human working memory: ~ 7 independent factor spaces, each holding one item

Products of these are smooth: $2^a \times 3^b \times 5^c \times 7^d$ with small exponents. The total event space $E = \prod_i E_i$ has cardinality that is a product of small primes—highly factorable.

Connection to probabilistic independence. Factoring an event space as $E = E_1 \times E_2 \times \cdots \times E_k$ is a structural claim: outcomes are tuples $(e_1, \ldots, e_k)$ drawn from the component spaces. Independence is a separate claim about the probability measure: variables E_i and E_j are independent when $P(e_i, e_j) = P(e_i) \cdot P(e_j)$. A factored space can carry correlated distributions.

Double-counting evidence occurs when two observations share a hidden common cause. Suppose observations A and B both correlate with a hidden factor H. Naïvely treating A and B as independent evidence for some hypothesis counts H's contribution twice. The inclusion-exclusion principle corrects this: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$, subtracting the overlap.

This is why *understanding* the factorization matters for correct reasoning. When you know which factors are shared between evidence sources, you can avoid overcounting. Without the factor map, you cannot tell whether A and B provide independent support or merely reflect the same underlying cause.

Conjecture. Natural problems have natural factorizations with small factors. NP-hardness arises when problems are encoded in representations that destroy the original factorization. Integer factorization is hard precisely because integers carry no hint of their factors—the factorization has been erased from the representation.

Appendix: Chomsky, Merge, and Universal Grammar

We interpret Chomsky's linguistic theory in terms of the UM, showing that Universal Grammar is innate factorization of the event space.

Universal Grammar as innate event spaces. Children acquire language from limited data ("poverty of the stimulus"), yet converge on similar grammars. Chomsky's explanation: the space of possible grammars is constrained by innate structure. In UM terms: children come with pre-factored event spaces.

The innate factorization includes:

- Phoneme inventory: $\sim$40 atoms per language (small $|E_{\text{phoneme}}|$)
- Syntactic categories: N, V, Adj, etc. (small $|E_{\text{syntax}}|$)
- Structural positions: subject, object, specifier (small $|E_{\text{position}}|$)

Learning a language means learning which events in these pre-given spaces are active, not creating the spaces from scratch.

Poverty of the stimulus resolved. A child hears finite examples but learns to generate infinitely many sentences. Without innate structure, the hypothesis space is 2^n for n possible grammars—intractable. With innate factorization, the search space is $40 \times 40 \times 20 \times \ldots$, a product of small numbers. The child learns parameters within fixed factors, not the factors themselves.

Merge as the product operation. Chomsky's Merge operation combines two syntactic objects into one: $\text{Merge}(A, B) = \{A, B\}$. This is exactly the product: $E_A \times E_B$, creating a binary factor. All of syntax arises from iterated binary products.

The simplicity of Merge (binary, symmetric, recursive) explains why human syntax is context-free: bounded depth, small factors, no crossing dependencies.

The Chomsky hierarchy as factorization depth.

- **Regular:** Flat products, no recursion. Finite-state.
- **Context-free:** Recursive products, $S \to NP \times VP$. Pushdown automata.
- **Context-sensitive:** Products with constraints across factors.
- **Recursively enumerable:** Arbitrary.

Human language is (mostly) context-free—a sweet spot of expressiveness and tractability.

Why LLMs learn language. Language is smooth: it factors into products of small event spaces. LLMs discover this factorization empirically through massive data. Humans come pre-factored and need little data. Same destination (linguistic competence), different paths (statistical vs innate).

Appendix: P = NP as Factorization (Speculative)

We offer a speculative reframing of P vs NP in terms of factorization, without claiming to resolve it.

The standard question. P vs NP asks: is finding a solution as easy as verifying one? For NP-complete problems, verification is polynomial but known algorithms for finding are exponential.

Reframing in UM terms. Consider the pattern:

- **Training:** Hard (months, huge compute)
- **Inference:** Easy (milliseconds, single forward pass)
- **NP-complete:** Search is hard, verification is easy

In each case, *finding* the right structure is hard; *using* it is easy. This suggests: finding the correct factorization is the hard part.

Speculation 1: NP-hardness as lost factorization. Natural problems may have natural factorizations that make them tractable. When we encode a problem (e.g., into SAT), we may destroy the original factorization. SAT looks hard because we're searching over 2^n assignments—but the original problem might have had small factors that made it easy. NP-hardness might be an artifact of representation, not inherent difficulty.

Speculation 2: P = NP iff universal factorization-finder. If there existed an efficient algorithm to recover the "natural" factorization of any problem, then P = NP: we'd find the factorization, then solve efficiently. Conversely, if P $\neq$ NP, no such universal factorization-finder exists.

Why integer factorization is hard. Integers carry no hint of their factors—the factorization has been completely erased. This is the degenerate case: no structure left to exploit. Contrast with real-world problems that retain traces of their origin.

Caveat. This is speculation, not a proof or even a conjecture. It's a way of thinking about computational complexity through the UM lens. The standard complexity-theoretic treatment remains authoritative.

Notation. Let $|E|$ denote the total count of atomic events in some event space. Let $\mathcal{N}(k)$ denote the set of integers arising as products of k factors drawn from a distribution μ over small primes (modeling natural processes). Let $\mathcal{U}[1, N]$ denote the uniform distribution over $\{1, \ldots, N\}$.

The anti-realism tax. Define the density:

$$\rho_k(N) = \frac{|\mathcal{N}(k) \cap [1, N]|}{N}$$

Then $\rho_k(N) \to 0$ as $N \to \infty$ for any fixed k. Uniformly sampled large integers are almost surely *non-understood*: they do not arise from any bounded-depth factorization process.

Integers as event-space projections. Say n is *understood relative to* E if $n \mid |E|^m$ for some m, i.e., n's prime factors appear in the factorization of $|E|$. If $n \sim \mathcal{U}[1, N]$ with $N \gg |E|$, then with high probability n contains prime factors absent from $|E|$—it has no interpretation as a substructure of E. This is the formal sense in which uniform sampling over large integers is "anti-realistic."

Version 2: Integration with the UM.

Problems as models. A computational problem instance can be viewed as a UM: $u = (E, T, P, f, \omega)$ where E is the space of candidate solutions, T is the belief state (which candidates are plausible), and P encodes constraints. Solving the problem means finding $e^* \in E$ such that $t_{e^*} = 255$.

Factorization morphisms. Let $\phi : E \to \prod_i E_i$ be a factorization morphism. We say ϕ is *tractable* if the induced patterns $\phi(P)$ are sparse—i.e., if the constraints, when viewed through this factorization, decompose into local constraints on small subsets of factors.

The encoding problem. Encoding a problem into SAT defines a morphism $\psi : E_{\text{natural}} \to \{0,1\}^n$. If ψ is not structure-preserving—if the natural factors E_i do not map to contiguous or sparse subsets of variables—then ψ destroys the tractable factorization. The quotient $E_{\text{natural}}/\ker(\psi)$ may collapse distinctions that made the problem easy.

Restatement of Speculation 2.

> P = NP iff there exists a universal factorization-finder: an algorithm $\mathcal{A}$ such that for any problem u encoded as u', $\mathcal{A}(u')$ recovers a tractable factorization ϕ in polynomial time.

In UM terms: P = NP iff interpretability is always efficient. Since LLM interpretability is "finding the domain-natural factorization hidden within the architecture-natural one," and this is known to be hard, we have circumstantial evidence that P $\neq$ NP.

Depth and factorization. Recall that n-way AND requires circuit depth n. A factorization ϕ that aligns with problem structure reduces the effective depth: if constraints factor as $C = C_1 \times C_2$, we can check C_1 and C_2 in parallel (depth = max, not sum). Destroying factorization forces serial verification: depth grows with problem size.

The learning function and search. The standard learning function ω extracts patterns from data. Search is the inverse: given a pattern (the constraints), find data (a solution) that matches. If ω can learn sparse patterns efficiently, ω^{-1} should find solutions efficiently—but only if the factorization is known. Without the factorization, ω^{-1} must search over both solutions *and* factorizations: this is the source of exponential blowup.

Integers revisited. Under the bijection $E \to \mathbb{N}$, a factorization $E = E_1 \times E_2$ with $|E_1|, |E_2|$ coprime becomes literal integer factorization. The anti-realism tax (from Version 1) is the special case where E is not a real event space but an arbitrary range $[1, N]$—a "problem" with no natural structure. Hardness of integer factorization reflects the absence of any factorization morphism to exploit.

Toward P = NP.
The framework suggests a path toward P = NP, or at least a restricted form of it.

Natural problems carry structure. Real-world problems originate from physical processes with natural factorizations. When we encode into SAT, we obscure that structure but do not destroy it—the factorization is still present, hidden in the encoding. If P = NP, there exists an algorithm that recovers it.

Hard instances are anti-realistic. The anti-realism tax (Version 1) implies that hard instances—random SAT at the threshold, factoring random semiprimes—are hard precisely because they are constructed to lack exploitable structure. No natural process generates them. They may be measure-zero in the space of instances that actually arise.

Learning as factorization-finding. The paper argues that interpretability and efficiency are the same problem. LLMs demonstrably learn efficient representations of natural problems. This suggests the universal factorization-finder exists: it is ω, the learning function. P = NP might be witnessed by: "train a sufficiently capable model."

The statement is the hint. A problem instance that can be *stated* has structure in its statement. The encoding into language, into SAT, into any formal system, necessarily reflects the origin of the problem. The provocative claim: every well-posed problem carries its solution's shadow. The "hard" problems are those posed adversarially, designed to hide their structure—but such problems may not correspond to anything real.

Reframing the question. P vs NP becomes: do structure-free problems exist? If every problem that can be stated necessarily has recoverable structure, then P = NP. If there exist genuinely structure-free problems (and SAT is one), then P $\neq$ NP. The separation, if it exists, measures the gap between physics and pure mathematics.

Appendix: The Extraction Research Program

We outline a research program for extracting explicit patterns from trained models—making the implicit knowledge in weights Θ available as inspectable, editable pattern lists.

Phase 0: Ground truth. Establish patterns we *know* are in the model:

- Factual: "capital of France" $\to$ "Paris"
- Linguistic: "un-" + adjective $\to$ negation
- Arithmetic: "2 + 2 =" $\to$ "4"

These serve as test cases for extraction methods.

Phase 1: Local extraction. One pattern, one model, full detail. For a known pattern p, identify:

- Which neurons activate when p is triggered?
- Which attention heads are involved?
- What is the minimal circuit that computes p?

Success criterion: ablating the circuit removes the pattern; the rest of the model is unaffected.

Phase 2: Extraction methods. Compare tools: linear probes, sparse autoencoders, activation patching, attention analysis, logit lens. Goal: a unified pipeline from model + prompt $\to$ activations $\to$ features $\to$ pattern descriptions.

Phase 3: Validation. How do we know the extracted pattern is real?

- **Behavioral:** Does the pattern predict other model outputs?
- **Ablation:** Does removing the circuit remove the behavior?
- **Transplant:** Can we copy the circuit to another model?
- **Synthesis:** Can we compile a pattern description back to weights that exhibit it? (The hard test.)

Phase 4: Scaling. Automated extraction, coverage metrics, contradiction detection. Key question: what fraction of a model's knowledge can we extract? Can we detect when two extracted patterns contradict each other? Can we trace patterns back to training data provenance?

Phase 5: Unification. Cross-domain patterns, translation between models, merging knowledge bases. The end goal: complete extraction of all human knowledge encoded in trained models into explicit, editable, verifiable form.

Appendix: Economics of Pattern Injection

We speculate on economic implications if pattern-to-weight compilation becomes tractable.

Current paradigm. Knowledge enters models via training:

$$\text{Knowledge} \to \text{Training data} \to \text{Gradient descent} \to \text{Weights}$$

Cost: data curation (\$10M+), compute (\$100M+), time (months). Value accrues to compute providers.

Pattern injection paradigm. If we could compile directly:

$$\text{Knowledge} \to \text{Pattern list} \to \text{Compilation} \to \text{Weights}$$

And if compilation is cheap ($O(n)$ in pattern count), the economics change dramatically.

What changes.

- Training from scratch → fork and edit existing models
- Months of compute → hours of curation
- Scaling laws → diminishing returns on raw compute
- Closed source moats → open pattern libraries

Who gains and loses. Gains: domain experts (source of valuable patterns), curators (quality control), tool builders (extraction/compilation), open source community.

Loses: GPU manufacturers (training demand drops), hyperscalers (less training to sell), closed-source labs (moats erode).

Caveats. This is speculation. We have neither reliable extraction nor compilation. The timeline is unknown. The actual economics may differ substantially. But the possibility is worth considering for strategic planning.

Appendix: Refactoring and Theory Building

The software engineering term "refactoring" captures exactly what the UM formalizes.

Naur's insight. Peter Naur, in "Programming as Theory Building" (1985), argued:

- The program is not the code
- The program is the theory in the programmer's head
- Code is an artifact; understanding is the real product
- When the original programmers leave, the theory is lost even if the code remains

In UM terms: the factorization (how E, T, P are structured) is the theory. The code is one representation of that factorization. Different code can express the same factorization; the same code can obscure its factorization.

Refactoring as factorization change. Fowler's definition: "Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior."

In UM terms:

$$\text{Refactor} : (E, T, P, f) \mapsto (E', T', P', f') \quad \text{such that} \quad f_P = f'_{P'}$$

The factorization changes; the computed function is preserved.

The correspondence.

Software engineering	**UM**
Codebase	Factorization of E, P, f
Refactoring	Change factorization, preserve function
Technical debt	Poor factorization accumulating
Design pattern	Known-good factorization
Code smell	Factorization that obscures structure
API	Event space shared between modules
Module	Factor of the total pattern P

Fowler's catalog. Martin Fowler's refactoring catalog is a catalog of factorization transformations:

- **Extract Method**: factor a pattern out of a larger pattern
- **Inline Method**: multiply a factor back in
- **Rename**: change event names without changing structure
- **Move Method**: reassign a pattern to a different factor of P

- **Extract Class**: create a new factor of E

Each preserves the function while changing the factorization.

Why refactoring is hard. Finding the "right" factorization requires understanding the domain. A factorization natural to the problem differs from one natural to the implementation. Technical debt accumulates when expedient factorizations diverge from domain-natural ones. Refactoring is expensive because it requires recovering the lost theory.

The insight. Software engineers discovered the UM piecemeal, through practice. "Refactoring," "separation of concerns," "single responsibility principle"—these are all statements about factorization. The UM provides the mathematical framework that unifies them.

Appendix: AI Slop Detection

We sketch a diagnostic for distinguishing original contribution from low-value recombination ("slop") using *pattern novelty*, and we state the failure mode that occurs if novelty is turned into a training objective.

Setup: patterns and closure. Fix a representation in which an artifact x (text, code, proof, design) can be mapped to a multiset of patterns:

$$\Pi(x) \subseteq \mathcal{P}.$$

A "pattern" may be an n-gram, AST subtree, rewrite schema, proof step template, citation graph motif, API-usage idiom, or any other extractable structure chosen for the domain.

Let Π_{train} be the union of extracted patterns from the training corpus, and let $\mathrm{cl}(\cdot)$ be a *derivational closure operator* capturing what counts as "not genuinely new" given the training set:

$$\mathrm{cl}(S) \supseteq S, \quad S \subseteq T \Rightarrow \mathrm{cl}(S) \subseteq \mathrm{cl}(T), \quad \mathrm{cl}(\mathrm{cl}(S)) = \mathrm{cl}(S).$$

Intuitively, $\mathrm{cl}(\Pi_{\text{train}})$ includes patterns obtainable by trivial transformations (renaming, formatting), straightforward composition, or other derivations deemed non-original for the task.

Novelty score. For an output y, define its novelty set and novelty mass by

$$N(y) \;=\; \Pi(y) \setminus \mathrm{cl}(\Pi_{\text{train}}), \qquad \nu(y) \;=\; w(N(y)),$$

where $w(\cdot)$ is a weighting that can discount fragile artifacts (e.g., one-off strings) and emphasize stable structure (e.g., reusable lemmas, nontrivial algorithmic ideas). High $\nu(y)$ is evidence that y contains patterns not attributable to recombination under the chosen closure.

Under this lens:

- **Pure recombination (slop):** $N(y) = \varnothing$ (or $\nu(y)$ negligible).
- **Original contribution:** $N(y) \neq \varnothing$ with substantial mass under w.

The Goodhart failure mode. If $\nu(y)$ is used as a training reward ("maximize novelty"), the generator is incentivized to create *adversarial* patterns that evade $\mathrm{cl}(\Pi_{\text{train}})$ while carrying little semantic value. The model learns to manufacture artifacts that *look* new under the extractor but are not meaningful improvements: syntactic perturbations, degenerate constructions, or incoherent inventions that maximize the metric.

This is an instance of Goodhart's law: when a measure becomes a target, it ceases to be a good measure.

Proper use. Pattern novelty is best treated as an *evaluation* tool, not an optimization objective:

- forensic analysis of suspect content,
- post-hoc assessment of claimed research contributions,
- human-in-the-loop verification of purported discoveries.

The evaluator should remain separate from the generator; novelty scores should not be fed back as training targets. This is how you train a model to lie.

Appendix: Neuron Reuse Across Event Spaces

We speculate on when different event spaces can share the same neural implementation.

The question. Brains are small relative to the number of possible event spaces. This suggests massive reuse: the same neurons compute functions over many different event spaces. When is such sharing possible?

Condition for sharing. Let $f : E_1 \to T$ and $g : E_2 \to T$ be functions over different event spaces. If there exists a relabeling $\sigma : E_1 \to E_2$ such that $g = f \circ \sigma^{-1}$, then the same circuit can compute both f and g—just rewire the inputs.

Example: "distance from expectation" is structurally the same whether computing surprise, prediction error, or deviation from baseline. One circuit, multiple uses.

Connection to superposition. Recent work (Anthropic and others) shows that neural networks encode many more features than they have neurons, using *superposition*: features share dimensions and are disambiguated by context.

In UM terms: many event spaces share the same neural substrate. The same neuron participates in multiple E_i, distinguished by which other neurons are active (context).

Efficiency through abstraction. If many event spaces share structure, the brain can implement them all with shared circuits plus a small "addressing" overhead. Abstraction is exactly this: recognizing structural similarity across domains and implementing it once.

Open questions. What is the equivalence relation on event spaces that permits sharing? How do we formalize "same function under relabeling"? This remains speculative and needs formal development.

Appendix: Memory Traces

The learning function ω operates on memory traces to produce patterns. Here we elaborate on what memory traces are and how they connect to learning.

Memory as time-indexed event sequences. Factor time into E explicitly: $E = E_{\text{content}} \times E_{\text{time}}$. A memory trace is a sequence $(e_1, t_1), (e_2, t_2), \ldots$ recording what event was observed at what time. This is exactly what databases do: a log table is a memory trace. Each row records a joint event; the table accumulates the organism's experience.

Types of memory traces.

- **Raw trace:** sequence of (e, t) tuples, stores everything observed.
- **Compressed trace:** the log contingency table from the standard learning function—stores only co-occurrence counts, discarding order and duplicates beyond count.
- **Absolute trace:** when patterns reach 255, we've crossed from statistical to symbolic storage. The organism now "knows" rather than "believes."

From trace to pattern. The standard learning function ω_0 takes a memory trace and produces a product pattern:

$$\omega_0 : (E \times T)^* \to P$$

The trace is the input (raw experience); the learned product pattern (LPP) is the output. Thus: learning = compression of memory into patterns.

Database analogy.

Database	**UM**
Table schema	Event space E
Row (record)	Joint event $e \in E$
INSERT operation	Append to memory trace
SELECT with GROUP BY	Extract co-occurrence counts
Query result	Thought t
Stored procedure	Pattern p

A relational table is a matrix where columns are factor event spaces and rows are joint events that "occurred" at some moment.

Biological connection (speculative). This framework suggests a mapping to neuroscience:

- Hippocampus stores episodic memory (raw traces)
- Consolidation moves traces to cortex (compressed patterns)
- Sleep may be when ω runs on the day's traces
- Replay during sleep = re-presenting traces to the learning function

These connections require careful experimental validation.

Factored vs. unfactored learning. The unfactored learning function ω_0 operates on $E \times E$, recording joint events without structure. The factored learning function operates on $Q \times Q$ where $Q = E/k$ for some quotient. Factoring E before learning reduces the table size from $|E|^2$ to $(|E|/k)^2$—exponential savings. The cost: factoring requires knowing the right equivalence relation in advance.

Appendix: Connection to the Universal Weight Subspace Hypothesis

Kaushik et al. demonstrate empirically that trained models from a fixed architecture exhibit weight variation concentrated in a shared low-dimensional spectral subspace—the Universal Weight Subspace Hypothesis (UWSH). We connect this finding to CMP theory.

Our main claim: *CMP provides a theoretical explanation for why UWSH eigenspaces exist.*

Embedding the Pattern Space. CMP defines P set-theoretically as relations $p \subseteq E \times E$. To connect with UWSH's spectral analysis, we define an explicit embedding. Let the vectorization map be:

$$v : P \to \mathbb{R}^{|E|^2}, \quad v(p)_{(e_1,e_2)} = p(e_1, e_2)$$

with inner product $\langle p, q\rangle := \langle v(p), v(q)\rangle$. All linear-algebraic language (subspace, basis, rank, PCA) applies to $v(P) \subseteq \mathbb{R}^d$, not to P directly.

The Covariance Operator. For a distribution τ over tasks, each task induces a learned pattern p_τ. Define the covariance operator:

$$C := \mathbb{E}_\tau\left[(v(p_\tau) - \mu)(v(p_\tau) - \mu)^\top\right]$$

where $\mu = \mathbb{E}_\tau[v(p_\tau)]$. UWSH's empirical finding: $\mathrm{rank}(C)$ is small relative to $|E|^2$, with the top-k eigenspace capturing most variance.

The Factor-Respecting Subspace. Let $E = E_1 \times E_2 \times \cdots \times E_k$ be a factorization. A pattern p *respects the factorization* when it decomposes as:

$$v(p) \in \mathrm{span}\big\{v(p_1) \otimes I_2 \otimes \cdots \otimes I_k,\ I_1 \otimes v(p_2) \otimes \cdots \otimes I_k,\ \ldots,\ \text{low-order interactions}\big\}$$

Using a low-order interaction framing: order-0 patterns are constant; order-1 patterns depend on single factors; order-m patterns depend on m-tuples of factors where $m \ll k$.

The factor-respecting subspace F has dimension:

$$\dim(F) = O(k \cdot \max_i |E_i|^2) \ \ll\ O(|E|^2) = O\Big(\prod_i |E_i|\Big)^2$$

The dimensional reduction is exponential when k is large.

Theorem (Factorization Forces Concentration). If the distribution τ concentrates on factor-respecting patterns, then the covariance operator C has low effective rank, and the top-k eigenspace of C lies within the factor-respecting subspace F.

Argument. (i) Factor-respecting patterns live in subspace F of dimension $d_F = O(k \cdot \max_i |E_i|^2)$. (ii) The covariance of any distribution supported on a d-dimensional subspace has rank $\leq d$. (iii) Therefore $\text{rank}(C) \leq d_F$, and $\text{TopEig}_k(C) \subseteq F$ for any $k \leq d_F$.

This connects CMP's structural claim (factor-respecting patterns are preferred) to UWSH's empirical observation (low-rank covariance across trained models).

The Model Architecture Gap. CMP shows that ω is statistically optimal only when operating under an optimal factorization of E. But current LLM architectures do *not* explicitly factor E—they work with an undifferentiated embedding space.

The UWSH finding reveals this gap: the low-dimensionality of learned patterns is caused by the inherent factorability of the semantic space E, "showing through" despite the architecture's failure to exploit it explicitly.

In other words:

- E has natural factorization $E_1 \times \cdots \times E_k$ (semantic structure)
- LLMs work because this factorization exists
- LLMs are suboptimal because they don't represent it explicitly
- UWSH detects the shadow of this hidden structure in weight space

This points to a concrete architectural improvement: models that explicitly represent the factorization of E should achieve better compression and generalization than current monolithic architectures.

Scope of the Claim. We claim: (a) the UWSH eigenspace is *contained in* the CMP factor-respecting subspace; (b) CMP provides a theoretical *explanation* for why UWSH eigenspaces exist.

We do *not* claim: that UWSH eigenspace equals the factor-respecting subspace (it may be a proper subset); that SGD/Adam are information-theoretically optimal; or that we can identify the specific factors E_i from UWSH data alone.

Limitations. This is an argument sketch; full proofs require formalizing the measure τ over tasks. The gap between CMP's normative theory and empirical optimizer behavior remains. Identifying the semantic factors E_i from empirical UWSH data is future work.