

Data-First Ontology: The World Model Resides in Data, Not in Weights

Zhanbo Li / Atlas Lee*

DAO Project

May 2026

Abstract

Neural network weights in large language models are conventionally assumed to encode world models. This paper argues that this assumption exhibits structural limitations in scenarios demanding precision, auditability, and real-time updates, identifying four crises: hallucination, frozen knowledge, uninterpretability, and uneditability. We propose **Data-First Ontology**, advocating that world models should be constituted by six explicit primitives — **Being, Def, Type, Relation, Contract, and Version** — with Def and Type cooperating to constitute the complete behavior of an entity. This paper formalizes the distinction between implicit and explicit world models, and introduces **DaoQL-Edu**, a pedagogical implementation of the DaoQL core engine (the DAO Project’s centerpiece), as Phase-I validation: four foundational primitives have been implemented with cross-engine unified storage, achieving practical performance levels. The theoretical framework for the complete six-primitive architecture is fully articulated; large-scale validation thereof is reserved for future work. **Keywords:** world model; ontology; large language model; data structure; data-first; explicit knowledge representation

1 Introduction: The Crisis of World Models

1.1 The Default Assumption

Contemporary artificial intelligence rests upon a default assumption that is rarely stated explicitly: **the neural network weights of a large language model constitute a world model**. When we say that GPT-4 “knows” that Paris is the capital of France, or that it “understands” the causal relationship between smoking and lung cancer, we are tacitly endorsing this assumption. Under this view, a world model is a function $f_\theta : \text{text} \rightarrow \text{text}$, where the parameters θ encode statistical regularities extracted from trillions of pre-training tokens.

*DAO is derived from the Chinese philosophical concept “道” (Tao/Dao), signifying the origin and governing principle of all things. It is unrelated to DAO (Decentralized Autonomous Organization) in the blockchain context.

This assumption underpins the field’s central research program: make the model larger, train it on more data, and its world model will become more accurate, more comprehensive, and more robust. Scaling laws [1,2] are interpreted not merely as empirical regularities of loss reduction, but as evidence that world models improve monotonically with scale. The implicit reasoning is: if a 175-billion-parameter model occasionally hallucinates, then a trillion-parameter model will rarely do so; if a model whose training data cuts off in 2024 lacks knowledge of 2025, then a continuously trained model can remain up to date.

This paper argues that this reasoning exhibits **structural limitations in scenarios requiring precision, auditability, and real-time updates**. The four crises currently confronting large language models — hallucination, frozen knowledge, uninterpretability, and uneditability — are difficult to eradicate under a pure weight-encoding paradigm, because weights lack the atomic operational semantics possessed by explicit structure.

1.2 Four Structural Crises

Hallucination. Large language models generate text by sampling from a probability distribution conditioned on their input and training data. When training data contains contradictory information, or when a query falls outside the training distribution, the model does not “know” that it does not know. It outputs plausible-sounding but factually incorrect statements with high confidence [3]. The root cause is not insufficient training data or imperfect optimization, but the absence of any mechanism by which the model can verify whether a generated statement corresponds to an explicit fact at some location in its architecture. The “knowledge” that “Paris is the capital of France” is not stored at any specific location; rather, it is distributed as a statistical association across billions of weights. When this association competes with another — say, a fabricated training sample claiming that Lyon is the capital — the model has no principled method for resolving the conflict.

Frozen Knowledge. Once training is complete, the weights are frozen. The model’s world model is a snapshot of the world at the moment of training. Retrieval-Augmented Generation (RAG) [4] and fine-tuning [5] are techniques that attempt to address this limitation, but they are grafted onto a frozen weight architecture rather than replacing it. RAG injects external documents into the context window, yet the model’s fundamental world model — the weights — remains unchanged. Fine-tuning can update weights, but the process is computationally expensive, affects the global behavior of the entire model, and carries the risk of catastrophic forgetting [6]. The world model in weights is difficult to update incrementally after training.

Uninterpretability. When a large language model produces an answer, we cannot inspect its reasoning process. Attention weights provide only a coarse post-hoc visualization of which tokens influenced which other tokens [7], but they do not reveal why the model believes a particular fact, nor do they show which facts it considered and rejected. If a medical large language model recommends a particular treatment, we cannot trace the path from training data to recommendation. This opacity is not a temporary limitation of current visualization techniques, but an inherent property of knowledge distributed across billions of parameters. A world model in weights lacks an interpretable reasoning trace.

Uneditability. Suppose we discover that a large language model persistently generates a dangerous false statement — for instance, recommending aspirin to patients with gastric ulcers. Correcting this error requires fine-tuning, which affects the entire weight matrix and may introduce new errors elsewhere [8]. There exists no mechanism for “surgically” removing or updating a single belief. Knowledge in weights is

globally coupled: the representation of “aspirin” is entangled with representations of “pain relief,” “blood thinning,” “gastric irritation,” and countless other concepts. Modifying one place affects the whole. A world model in weights resists precise local modification.

Thesis. These four crises are not independent engineering problems, but **systemic manifestations of the pure weight-encoding paradigm** in the absence of explicit structural operational semantics. In scenarios requiring precision, auditability, and real-time updates, world models require the support of explicit data structures.

1.3 Thought Experiment

Imagine two libraries.

Library A employs a super-reader who has memorized every book in the collection. When you ask a question, the super-reader answers from memory. This super-reader is erudite, eloquent, and quick to respond. But he occasionally confuses similar facts (hallucination), knows nothing about books published after his training date (frozen knowledge), cannot explain how he arrived at a particular answer (uninterpretable), and correcting one error may distort other memories (uneditable).

Library B keeps all books on the shelves. When you ask a question, the librarian consults the catalog, retrieves the relevant volumes, cross-references the sources, and provides an answer with citations. Books can be updated (new editions replace old ones), the librarian’s retrieval process is inspectable (you can see which books were consulted), and correcting one book does not affect the others.

In many critical application scenarios, large language models operating alone as knowledge repositories face challenges. We propose a complementary architecture: explicit data structures as the world model, with the LLM serving as the query and reasoning engine.

In Library B, the world model does not reside in the librarian’s memory, but in the books themselves — their contents, relationships, citations, and revision histories. The librarian (a large language model) is a sophisticated query engine, not the repository of knowledge. This inversion of roles is the central claim of this paper.

1.4 Position of This Paper

We propose **Data-First Ontology**: a paradigm that treats the world model as an explicit computational structure rather than an implicit statistical distribution. Specifically, we argue that the world model should be built from six primitives:

- **Being**: a state collection endowed with a unique identity, constituting a traceable and evolvable existent unit;
- **Def**: the definition of a Being;
- **Type**: the classification label of a Being;
- **Relation**: the relationship between Beings;
- **Version**: the historical state of a Being;
- **Contract**: what determines the evolution of a Being.

Together, these six primitives answer six ontological questions: what exists, how it is defined, how it is classified, how it relates to others, how it is traced, and how it evolves. Their fully formalized semantics are elaborated below:

In this framework, **Type and Def cooperate to constitute the complete behavior of an entity**. Def answers “what is this” — it specifies the entity’s base fields, constraints, and core contracts (e.g., a “Product” Def defines fields such as name, price, etc.), analogous to a **Class**. Type answers “what category does this belong to, and therefore what additional behavior does it possess” — it is a hierarchical specialization system, such as “Digital Product → Mobile Phone → Huawei Phone,” analogous to an **Interface**. Type can inherit behavior from parent types, extend new contracts, or override Def’s default behavior. **Field definitions reside in Def; behavior is composed jointly by Def and Type**. An entity simultaneously possesses a Def (Class) and a Type (Interface); the two cooperate rather than being orthogonal.

These primitives are not merely a data model; they *are* the world model. The structure of Being, Def, Relation, Contract, and Version *is* the model of the world. The large language model operates upon this structure — querying it, traversing it, summarizing it — but the model itself persists independently of any neural network.

The contributions of this paper are as follows:

1. **A conceptual framework** distinguishing implicit (weight-based) from explicit (structure-based) world models, and formally characterizing their respective limitations.
2. **Six ontological primitives** (Being, Def, Type, Relation, Contract, Version) that collectively constitute an explicit world model with operational semantics and bootstrapping properties. Def prescribes base structure and core behavior; Type provides hierarchical specialization and inheritance capabilities; the two cooperate to constitute the entity’s complete behavior.
3. **DaoQL-Edu** (a pedagogical version of the DaoQL core engine), an open-source implementation containing only four core primitives that validates the Phase-I (tool-layer) claim: cross-engine nested retrieval can be unified within a single process, and under same-machine conditions achieves or surpasses the performance of specialized systems (see Section 5). The full DaoQL (six primitives, 62,000 lines of Rust, 833 tests) has been engineered; validation of Type and Contract is reserved for future work.
4. **A research agenda** identifying open problems that must be resolved to unlock the full potential of this paradigm: symbol grounding, completeness, and evolution consistency.

The remainder of this paper is structured as follows. Section 2 formalizes the distinction between implicit and explicit world models. Section 3 surveys related work. Section 4 articulates the core argument of Data-First Ontology, including the six primitives and their philosophical foundations. Section 5 describes the DaoQL-Edu pedagogical implementation as Phase-I (tool-layer) feasibility evidence. Section 6 situates our position within the context of philosophy of science and philosophy of mind. Section 7 discusses future work. Section 8 honestly declares the limitations and boundaries of this paper. Section 9 discusses open problems, and Section 10 concludes with an invitation to the research community.

2 The Problem: Implicit vs. Explicit World Models

2.1 What Is a World Model?

A world model is not merely a collection of facts. A telephone directory contains facts (names and numbers), but it is not a world model in the sense intended here. A genuine world model must support at least the following capabilities [9,10]:

Capability	Description	Example
Facts	Basic truths about entities	“Paris is the capital of France”
Relations	Typed connections between entities	“Paris HAS_LANDMARK Eiffel Tower”
Causality	Directed influence between events	“Smoking CAUSES Lung Cancer”
Rules	Conditional prescriptions	“If patient HAS Gastric Ulcer, THEN AVOID Aspirin”
Probability	Quantification of uncertainty	“Probability of rain in London in August = 0.35”
History	Temporal evolution of state	“Paris population(2024) = 2.16M; Paris population(2025) = 2.20M”
Behavior	State-dependent dynamics	“Order state machine: Pending → Paid → Shipped → Delivered”

An implicit world model encodes these capabilities as statistical patterns in neural network weights. An explicit world model encodes them as manipulable data structures.

2.2 Implicit World Model

Formally, an implicit world model is a function:

$$W_{\text{implicit}} = f_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$$

where $\mathcal{X}$ is the space of input queries (typically token sequences), $\mathcal{Y}$ is the space of output responses, and $\theta \in \mathbb{R}^d$ are the frozen parameters of the neural network (typically $d \sim 10^{11}$ – 10^{12}).

Knowledge is stored in distributed representations:

- The “fact” that “Paris is the capital of France” is not stored at any specific index θ_i . It is encoded as an activation pattern across thousands or even millions of parameters, entangled with representations of “Berlin,” “Tokyo,” “capital,” and “France.”

- The “relation” HAS_LANDMARK is not a first-class object. It exists only as a statistical correlation between the token sequences “Paris” and “Eiffel Tower” in the training data.
- The “rule” about gastric ulcers and aspirin is not a logical entailment. It is a statistical regularity that may be overridden by a stronger correlation — for instance, if “aspirin relieves pain” appears more frequently in the training data.

Consequence: There exists no operation to “read” a specific fact, “modify” a specific relation, or “verify” a specific rule. The only available operations are forward inference ($f_\theta(x)$) and fine-tuning (updating θ , which is global and approximate).

2.3 Explicit World Model

An explicit world model is a structured tuple:

$$W_{\text{explicit}} = (\mathcal{B}, \mathcal{D}, \mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{V})$$

where:

- $\mathcal{B}$ is the set of **Beings**, each possessing a unique identifier BeingId, a Def (structure), a Type (specialized behavior, optional), state divided into core and extended attributes, and a chain of historical versions;
- $\mathcal{D}$ is the set of **Defs**, each prescribing an entity’s structure, constraints, contracts (executable hooks on the write path), state machines, and relation declarations;
- $\mathcal{T}$ is the set of **Types**, each a hierarchical specialization mechanism of Def providing inheritance capabilities, capable of extending or overriding Def’s contracts and constraints;
- $\mathcal{R}$ is the set of **Relations**, each typed, directed or undirected, with optional temporal windows, cardinality constraints, and extended attributes;
- $\mathcal{C}$ is the set of **Contracts**, i.e., executable behaviors invoked before or after write operations (expressed in script form);
- $\mathcal{V}$ is a versioning function mapping each entity and timestamp to an immutable snapshot.

Fundamental distinction from the EAVT data model. Systems such as Datomic employ the EAVT (Entity-Attribute-Value-Time) model: “entity E has attribute A with value V at time T.” This is a **data semantics** — it records “what happened.” An EAVT fact is a passive proposition record.

DaoQL’s six primitives constitute an **ontological semantics** — they construct “what is.” A Being is not a record, but an **autonomous existent unit**: it has identity (BeingId), essential definition (Def), specialized classification (Type), social relations (Relation), behavioral rules (Contract), and historical personhood (Version). Datomic asks: “What was Alice’s name on 2024-01-01?” (querying historical facts). DaoQL asks: “What is Alice? How does she behave? With whom is she associated? How does she evolve?” (ontological exploration).

This is not a difference of implementation detail, but a **fundamental ontological commitment**: EAVT commits to “the world is composed of facts,” whereas DaoQL commits to “the world is composed of existents.”

Operations on the explicit world model:

Operation	Semantics	Example
<code>read(b)</code>	Retrieve entity b by identifier	<code>read(Paris) → {name: "Paris", population: 2.20M}</code>
<code>relate(b₁, b₂, r)</code>	Create a relation of type r between b_1 and b_2	<code>relate(Paris, France, IS_CAPITAL_OF)</code>
<code>update(b, d, v)</code>	Add a new version with Def d and state v for entity b	<code>update(Paris, City, {population: 2.21M})</code>
<code>query(q)</code>	Evaluate a structured query against the model	<code>query(Being(Paris).relation().def(I</code>
<code>evolve(d, d')</code>	Modify Def d to d' , with migration semantics	<code>evolve(City, City_v2)</code>
<code>contract(b, hook, script)</code>	Attach executable behavior to entity b	<code>contract(Patient, BEFORE_UPDATE, ulcer_check)</code>
<code>assign_type(b, t)</code>	Assign Type t to entity b	<code>assign_type(Mate60, Huawei_Phone)</code>
<code>type_contract(t, hook, script)</code>	Attach a contract to Type t	<code>type_contract(Huawei_Phone, AFTER_CREATE, check_harmony)</code>

Consequence: Facts, relations, rules, and behaviors are all first-class objects. They can be independently read, modified, verified, and audited. Updating the population of Paris does not affect the population of Berlin.

2.4 Comparative Analysis

Dimension	Implicit (W_{implicit})	Explicit (W_{explicit})
Readability	Cannot extract specific facts	<code>read(BeingId)</code> returns precise state
Editability	Fine-tuning affects entire model	Single relation or fact update
Verifiability	No reasoning trace	Complete query execution trajectory
Auditability	Cannot audit internal state	Version chain = immutable audit log
Interpretability	Attention weights are opaque	Types, relations, contracts are human-readable
Evolvability	Requires retraining	Runtime schema evolution (<code>evolve</code>)
Composability	Weights resist local decomposition	Sub-models can be extracted and shared

Dimension	Implicit (W_{implicit})	Explicit (W_{explicit})
Counterfactual Consistency	Statistical competition among associations	Deterministic evaluation after structural modification

The last row — **counterfactual consistency** — is particularly critical and will serve as the subject of our planned experimental evaluation (Section 7). When an implicit world model is asked a counterfactual question (“What would be the consequence of Y if X were not true?”), the answer depends on which statistical associations happen to be activated by the specific wording of the query. When an explicit world model is asked the same question, the answer can be computed deterministically by modifying the structure and re-evaluating the query.

3 Related Work

Before presenting Data-First Ontology, we must clarify this paper’s relationship to existing work. This section surveys relevant research along three dimensions: temporal and immutable databases, knowledge representation and knowledge graphs, and the combination of large language models with structured data.

3.1 Temporal and Immutable Databases

Datomic [11] is the closest precursor to the ideas in this paper. Proposed by Rich Hickey in 2012, Datomic’s core tenets include: (1) time as a central dimension of the data model (EAVT: Entity-Attribute-Value-Time); (2) immutable append-only — no UPDATE or DELETE, only the appending of new facts; (3) schema-as-data — schema is stored as data, queryable and evolvable; (4) database-as-value — the database is an immutable value that can be passed, compared, and cached.

Datomic has already realized three core properties advocated by DaoQL: “time as ontology,” “immutable append-only,” and “schema-as-data.” However, Datomic employs a single EAVT storage model, with all queries executing on EAVT indexes. DaoQL unifies five engines — graph, column, vector, index, and full-text — within a single process, achieving zero-overhead cross-modal queries through BeingId. Furthermore, Datomic’s Schema/Type is monolithic, lacking a mechanism for “the same entity category can have different specialized behaviors,” whereas DaoQL introduces the separation and cooperation of Def (Class) and Type (Interface).

Event Store [12] and **Temporal Tables** (SQL:2011) [13] have also made important contributions along the temporal dimension. Event Store adopts the Event Sourcing pattern, recording state changes as immutable event streams. SQL:2011’s Temporal Tables build support for valid time and transaction time into the relational model. These systems record “what happened,” but unlike DaoQL, they do not treat time as an ontological dimension of existence — in DaoQL, time is not an attribute, but the intrinsic structure of a Being.

3.2 Knowledge Representation and Knowledge Graphs

RDF/OWL/SPARQL [14,15,16] constitute the technology stack of the Semantic Web. RDF represents knowledge as triples (Subject-Predicate-Object), OWL provides an ontology description language, and

SPARQL provides a query mechanism. RDFS’s class and type mechanisms allow simple hierarchical classification, but lack the flexibility of the Def/Type separation. Compared to DaoQL, RDF/OWL lacks built-in version chains, contract execution, and multi-modal unification capabilities. Survey work in the knowledge graph domain [30] further reveals structural gaps in existing knowledge representation systems regarding dynamic evolution and multi-modal unification.

TypeDB [17] is a typed hyper-relational knowledge graph database. TypeDB supports type inheritance, constraint checking, and rule-based reasoning; its type system is stricter than that of RDF/OWL. However, TypeDB’s type inheritance is a static schema definition, does not support runtime Def/Type cooperative evolution, and does not embed contracts into the storage engine’s write path.

Property Graph (e.g., Neo4j [18], GQL [19]) organizes graph data using labels and properties. Property Graph’s Label is analogous to DaoQL’s Type, but a Label has no behavioral semantics — it cannot define contracts or constraints. Property Graph also lacks immutable version chains and cross-modal query capabilities. A survey of graph database models [32] notes that existing graph databases exhibit fundamental limitations in type systems and behavioral semantics.

3.3 Large Language Models and Structured Data

GraphRAG [20], proposed by Microsoft Research, is representative work combining LLMs with knowledge graphs. GraphRAG automatically extracts entity-relation triples from unstructured text, constructs a knowledge graph, and then retrieves relevant sub-graphs or community structures to enhance the LLM’s generation when answering queries. LightRAG [21] optimizes indexing and retrieval efficiency on this basis, adopting a dual-tier retrieval system. FastGraphRAG [22] further accelerates graph traversal using personalized PageRank.

The core paradigm of GraphRAG and its variants is “build graphs from text to assist the LLM” — the graph is an external augmentation component for the LLM. DaoQL’s core paradigm is radically different: the graph is not a tool to assist the LLM, but the **primary storage of the world model**. The LLM is merely a reasoning engine that queries and operates upon this world model.

MemGPT [23], proposed by UC Berkeley, applies the virtual memory management ideas of operating systems to LLMs. MemGPT enables the LLM to autonomously manage its memory through hierarchical storage (core memory, recall storage, archival storage) and explicit function calls. MemGPT solves the problem of limited LLM context windows, but its memory management remains at the level of text chunks, lacking structured semantics for entities, relations, and versions.

MCP (Model Context Protocol) [24] is an open protocol aimed at standardizing interactions between LLMs and external data sources, tools, and services. MCP operates at the protocol layer; DaoQL operates at the storage layer — the two are complementary rather than competitive. An LLM application based on MCP could use DaoQL as its structured memory backend.

3.4 Neuro-Symbolic Integration

Neuro-symbolic AI [25] is dedicated to combining the perceptual capabilities of neural networks with the reasoning capabilities of symbolic systems. Early work such as Neural Theorem Provers [26] and Logic Tensor Networks [27] attempted to embed logical rules into neural networks. Recent work such as AlphaProof [28] demonstrates the potential of neuro-symbolic integration in mathematical reasoning.

DaoQL can be viewed as the “symbolic-side” infrastructure in a neuro-symbolic architecture — providing the LLM (the neural side) with an explicit, verifiable structured knowledge base.

3.5 Summary of Distinctions from Existing Work

Dimension	Datomic	RDF/OWL	TypeDB	GraphRAG	MemGPT	DaoQL
Time as Ontology	EAVT	None	None	None	None	Version chain
Immutable Append-Only						WAL + versioning
Schema-as-Data						Bootstrapping
Def/Type Cooperation			Partial			Class/Interface
Multi-Modal Unification						Five engines
Contracts Embedded in Write Path			Partial			Rhai runtime
LLM Semantic Layer					Partial	NL→DSL
World Model Semantics		Partial	Partial			Six primitives

Note: The DaoQL capabilities in the table are based on the full architecture. The DaoQL-Edu pedagogical version contains verified implementations of only the first four capabilities (time as ontology, immutable append-only, schema-as-data, multi-modal unification), without the LLM semantic layer, full contract execution, or the complete Def/Type cooperation runtime.

The table above reveals a critical gap: **existing systems either focus on a single data model (relational, document, graph, vector) or focus on providing external knowledge augmentation for LLMs (GraphRAG, MemGPT), but no system simultaneously possesses all data model capabilities and takes “explicit world model” as its core architectural goal.**

Positioning of DaoQL: DaoQL is not a relational database, not a document database, not a graph database, not a key-value database, not a columnar database, not a vector database, not a time-series database, not a full-text search engine — yet it **simultaneously** possesses the capabilities of all these databases:

Database Type	DaoQL Implementation
Relational Database	Def/Type defines structure + Contract constraints (analogous to foreign keys, triggers, check constraints)
Document Database	BeingExt key-value extensions, supporting arbitrary nested attributes
Graph Database	BeingCore + Relation edge chains, direct pointer adjacency, O(1) graph traversal
Key-Value Database	BeingId → Being O(1) lookup
Columnar Database	BeingExt columnar storage per field + LZ4 compression + SIMD aggregation
Vector Database	HNSW + scalar quantization, BeingId as vector node
Time-Series Database	Version chain + timestamp index, O(1) historical backtracking
Full-Text Search Engine	Tantivy inverted index + jieba segmentation

Key Insight: The traditional database stack requires 8 separate systems to satisfy the seven capabilities of a world model. DaoQL unifies all capabilities within a single system — not through a “support multiple query languages” compatibility layer, but through a **unified set of primitives** (Being, Def, Type, Relation, Contract, Version). These primitives are not a patchwork of 8 data models, but a **more fundamental ontological structure** that naturally gives rise to the capabilities of all upper-layer data models.

DaoQL fills this gap: it does not build graphs from text to assist LLMs, but places the explicit world model at the core of storage, making the LLM an operator of this model rather than its owner.

4 Core Argument: Data-First Ontology

4.1 Three Propositions

We propose three mutually nested propositions that together define the Data-First Ontology paradigm. It should be noted in advance that the “Def” in these propositions is the structural specification primitive, prescribing an entity’s fields, constraints, and core behavior; “Type” is the hierarchical specialization mechanism of Def, providing inheritance and polymorphism capabilities, capable of extending or overriding Def’s contracts and constraints, and cooperating with Def to constitute the entity’s complete behavior.

Proposition I (Ontology). The basic unit of existence in a computational world model is not a symbol (as in first-order logic), not an object (as in object-oriented programming), nor a process (as in the π -calculus). It is the **Being**: an existent unit possessing identity, state, history, behavior, and relations with other entities.

A Being’s “what-it-is” (quiddity) is not prescribed by an external schema, but by its **Def**. A Def is not an attribute of the entity, but the entity’s ontological prerequisite: before an entity can become manipulable data, it must first be defined as a certain kind of existent. Def answers “what is this,” not “what category does this belong to.”

An entity is not a row in a table (passive data), nor an instance of a class (behavior defined externally). It is an autonomous existent unit whose behavior is embedded within its Def, and whose history is an immutable chain of versions. The entity is primitive; everything else is derived from it.

Proposition II (Epistemology). Knowledge is neither mental content (connectionism) nor a set of propositions (symbolism). Knowledge is the **structure of Being/Def/Relation** — and this structure itself is the world model.

Type provides the hierarchical specialization framework (e.g., “Digital Product → Mobile Phone → Huawei Phone”), capable of extending and overriding Def’s contracts and constraints. The foundation of knowledge lies in Def’s structural specification: Def defines which fields exist, which constraints are active, and which core contracts trigger. Type adds specialized behavior and constraints on this basis, jointly constituting the complete knowledge representation with Def.

This distinction is subtle but crucial. In symbolic systems, knowledge is a set of propositions about the world, stored independently of the reasoning mechanism. In connectionist systems, knowledge is the weight configuration of the reasoning mechanism itself. In Data-First Ontology, knowledge is a structure situated between data and reasoning: Def defines what may exist, Relations define how entities connect, and Contracts define how structures evolve. Type provides a classification framework to help organize and retrieve entities. The structure is not *about* the world; in a computational sense, *it is* the world.

Proposition III (Methodology). Reasoning is not the extraction of statistical patterns from weights, but **traversal and transformation of explicit structure**. The large language model is an accelerator of this process, not the repository of its contents.

When a user asks: “What medications should be avoided in patients with gastric ulcers?”, the reasoning process in a Data-First system is:

1. Parse the query into a structured traversal: `Being(Patient) → Relation(DIAGNOSED_WITH) → Def(Condition) → FILTER(name="Gastric Ulcer") → Relation(CONTRAINDICATED_WITH) → Def(Medication)`.

In this traversal, “Patient,” “Condition,” and “Medication” are all Defs (defining the structure of the respective entities), and each entity may simultaneously possess a Type (e.g., the Type of “Patient” might be “Human/Patient”). Traversal operations are based on the structure prescribed by Def, while Type provides classification information to assist retrieval.

2. Execute the traversal on the explicit structure.
3. Collect the results (list of contraindicated medications).
4. Use a large language model to generate a natural language response from the structured results.

The role of the large language model is Step 4, not Steps 1–3. The world model — the knowledge about which medications are contraindicated for gastric ulcers — resides in the explicit structure, not within the weights of the large language model.

Two-Phase Validation Stratification. The six primitives involved in the three propositions above are stratified into two levels in terms of validation strategy:

- **Tool Layer** (Phase I, validated): The intersection of Being, Def, Relation, and Version is sufficient to solve the engineering problem of “how can multi-modal data of a world model be unified for

storage and query within a single system.” Section 5 reports the validation of this layer by the DaoQL-Edu pedagogical version.

- **Meta-Tool Layer** (Phase II, theoretically defined): Type and Contract address the ontological problem of “how does a world model describe and constrain itself.” The theoretical framework for these two primitives has been fully articulated below; their large-scale engineering validation is reserved for future work.

This stratification is deliberate — even with only the four foundational primitives, the system is already sufficiently powerful to support a world model surpassing the traditional fragmented database stack; Type and Contract are incremental enhancements, not necessary prerequisites.

4.2 Six Ontological Primitives

We now define the six primitives in detail.

Being. A Being b is a quadruple:

$$b = (id, core, ext, history)$$

- $id \in \text{BeingId}$ is a unique, immutable identifier (typically a UUID).
- $core$ is a fixed-size structure containing the entity’s fundamental attributes (e.g., name, definition identifier, creation timestamp).
- ext is an extensible key-value map containing arbitrary attributes prescribed by the entity’s Def.
- $history = [v_1, v_2, \dots, v_n]$ is an immutable, append-only version chain, where each $v_i = (timestamp_i, state_i, prev_hash_i)$.

A Being is not “instantiated” from a class; it is **written** into existence. There is no distinction between creation and persistence. Creating a Being is writing it into storage; reading a Being is retrieving it by its identifier.

Every Being has a `def` field pointing to the Def entity that defines the entity’s structure. For example, a “Patient” Being’s `def` points to the Def entity that defines the structure of “Patient.”

Def. A Def d is a sextuple:

$$d = (name, fields, constraints, contracts, state_machine, relations)$$

- $name$ is the unique definition identifier (e.g., “Patient,” “Medication,” “Order”).
- $fields$ is a set of typed field definitions.
- $constraints$ is a set of invariant predicates (e.g., `age >= 0`, `email MATCHES regex`).
- $contracts$ is a set of executable hooks (BEFORE_CREATE, AFTER_UPDATE, BEFORE_RELATE, etc.), each associated with a script (e.g., a Rhai script).
- $state_machine$ is a finite state machine defining the legal state transitions for entities under this Def.
- $relations$ is a set of relation type declarations that entities under this Def may participate in.

Def answers “what is this” — it prescribes the complete structure of the entity. The “Patient” Def contains name fields, age fields, diagnosis fields, and corresponding constraints and contracts. Field definitions reside in Def, not in Type.

Crucially, **Def itself is also a Being**. The Def Patient is a Being whose `core.def` points to DA0_DEF (i.e., “the Def of Defs”). There is no independent “schema catalog” or “system table.” The definition system is **bootstrapping**: the definition of what a Def is, is itself stored as a Def.

Type. A Type t is a hierarchical specialization mechanism of Def, analogous to an **Interface** in object-oriented programming, but supporting hierarchical inheritance. It is a node in a taxonomy, cooperating with Def (Class) to constitute the entity’s complete behavior:

$$t = (name, parent, path, contracts, constraints, attributes)$$

- *name* is the type name (e.g., “Huawei Phone”).
- *parent* is the parent type (e.g., “Mobile Phone”), forming an inheritance chain.
- *path* is the complete path from root to this node (e.g., “Digital Product/Mobile Phone/Huawei Phone”).
- *contracts* are executable hooks specific to this type, capable of extending or overriding Def’s contracts.
- *constraints* are constraints specific to this type, capable of tightening Def’s constraint conditions.
- *attributes* are presentation attributes specific to this type (e.g., icon, color).

Field definitions reside in Def; behavior is composed jointly by Def and Type. Type does not define fields, but it defines additional behavior and constraints. For example:

- Def “Product” = {fields: [name, price, brand], contracts: [check_price_positive]}
- Type “Mobile Phone” (parent = “Digital Product”) = {contracts: [check_has_imei]}
- Type “Huawei Phone” (parent = “Mobile Phone”) = {contracts: [check_harmony_os_support], constraints: [brand == "Huawei"]}

Being “Mate 60” (def = “Product”, type = “Digital Product/Mobile Phone/Huawei Phone”) → Complete behavior = Def(Product) + Type(Mobile Phone) + Type(Huawei Phone) → Execute check_price_positive + check_has_imei + check_harmony_os_support

Type provides **polymorphic specialization** capability: different Types (interface implementations) of the same Def (class) can have different behaviors. For example, Def “Patient” defines base fields, while Type “Emergency Patient” adds emergency handling contracts, and Type “Inpatient” adds bed management contracts. This is analogous to the same class implementing different interfaces, each interface prescribing additional behavioral constraints.

Type itself is optional — a Being may have no Type (using only Def’s base behavior), but it cannot be without a Def.

Relation. A Relation r is a quintuple:

$$r = (type, source, target, direction, attributes, validity)$$

- *type* is the relation type (e.g., IS_CAPITAL_OF, TAKES, CONTRAINDICATED_WITH).
- *source* and *target* are entity identifiers.
- *direction* $\in \{\text{directed}, \text{undirected}\}$.
- *attributes* is an extensible key-value map.
- *validity* = (t_{start}, t_{end}) is an optional temporal window.

Relations are first-class objects. They can be queried, versioned, and constrained. A Relation is not a foreign key (a passive pointer), but an active connection that may trigger contracts upon creation, modification, or traversal.

Note: relation types themselves are also defined by Def (def = "DAO_RELATION"), but relation instances are actual connections between entities.

Contract. A Contract *c* is a triple:

$$c = (hook, script, ast_cache)$$

- *hook* $\in \{\text{BEFORE_CREATE}, \text{AFTER_CREATE}, \text{BEFORE_UPDATE}, \text{AFTER_UPDATE}, \text{BEFORE_RELATE}, \text{AFTER_RELATE}\}$
- *script* is the contract's source code (e.g., a Rhai function).
- *ast_cache* is the compiled abstract syntax tree, cached for performance.

Contracts are embedded in the write path. They cannot be bypassed by client applications, because they execute inside the storage engine. For example, a contract checking for drug interactions executes on every `relate(Patient, Medication, TAKES)` operation, regardless of which client initiated the operation.

Contracts are **ontologically independent primitives** — they represent the ontological dimension of “executable behavioral rules” in the world model, on par with Being, Def, Type, etc., as one of the six primitives. However, at the **implementation level**, contracts are embedded within Def, stored and executed as a field of Def. This “ontologically independent, implementationally embedded” design is deliberate: it guarantees that contracts cannot be bypassed (because the only entry point to the write path is Def), while maintaining ontological integrity.

Version. A Version *v* is a quadruple:

$$v = (timestamp, state, prev_pointer, next_pointer)$$

- *timestamp* is a logical or physical timestamp.
- *state* is the entity's complete state at the moment of this version.
- *prev_pointer* is the storage offset pointer to the previous version (NodeOffset).
- *next_pointer* is the storage offset pointer to the next version (NodeOffset).

Version chains make every entity's history inspectable and auditable. It is not an optional audit log stored separately, but the intrinsic temporal dimension of the entity's existence. In the code implementation, version nodes are linked by bidirectional pointers (`prev_version` / `next_version`), functionally equivalent to the version chain in the formal definition.

4.3 Bootstrapping: A Self-Describing System

The most radical property of Data-First Ontology is that **the system describes itself**. There is no meta-level outside the data level.

Consider how existing systems handle schema:

System	Schema Storage	Schema Modification
PostgreSQL	pg_catalog system tables	ALTER TABLE DDL
MongoDB	None (schemaless)	Not applicable
Neo4j	Optional constraints	CREATE CONSTRAINT Cypher
Datomic	Datalog assertions	Transactional schema updates
DaoQL	Entities defined by DAO_DEF	update operations on Def entities

In DaoQL, the command `define type Patient { ... }` is not parsed by an independent DDL parser and stored in a separate catalog. It is a **write operation** that creates a Being whose Def is DAO_DEF (i.e., “Patient” is written as a Def entity). Definition is data; data carries definition; the system is self-describing.

Specifically, the bootstrapping hierarchy is as follows:

Layer 0: Meta-Layer

```
Def "DAO_DEF"
  → Defines "what a Def is" (i.e., the structure of Def itself)
Def "DAO_TYPE"
  → Defines "what a Type is" (the root definition of the Type hierarchy)
Def "DAO_RELATION"
  → Defines "what a Relation is"
```

Layer 1: Object Layer

```
Def "Patient"
  → Defines the business-level "Patient" (containing name, age, diagnosis, etc. fields)
Def "Medication"
  → Defines the business-level "Medication" (containing name, ingredient, dosage, etc. fields)
```

Layer 2: Instance Layer

```
Being "Alice" (def = "Patient")
  → Alice is a Patient entity
Being "Aspirin" (def = "Medication")
  → Aspirin is a Medication entity
```

Type (specialization hierarchy) cooperates with Def to constitute complete behavior:

Def hierarchy (structural definition)	Type hierarchy (specialization inheritance)
Def "Product"	Digital Product

```

fields: [name, price]                Mobile Phone
contracts: [check_price]              Huawei Phone
                                     contracts: [check_harmony]
                                     constraints: [brand == "Huawei"]
                                     Apple Phone
                                     contracts: [check_ios]

Def "Patient"                        Organism
fields: [name, age, diagnosis]        Human
contracts: [check_id]                 Patient
                                     contracts: [check_urgency]
                                     Doctor
                                     contracts: [check_license]

```

A Being simultaneously possesses a Def (base structure) and a Type (specialized behavior): Being “Mate 60” (def = “Product”, type = “Digital Product/Mobile Phone/Huawei Phone”) → Behavior = Def(Product) + Type(Mobile Phone) + Type(Huawei Phone) → Execute check_price + check_harmony
 Being “Alice” (def = “Patient”, type = “Organism/Human/Patient”) → Behavior = Def(Patient) + Type(Human) + Type(Patient) → Execute check_id + check_urgency

Def and Type cooperate: modifying the “Patient” Def (e.g., adding a new field) affects all Patient Beings (analogous to modifying a class definition affecting all instances); modifying the Type “Patient” (e.g., adding a new contract) affects only Beings whose type = “.../Patient” (analogous to modifying an interface affecting all classes that implement it).

This bootstrapping property has far-reaching consequences:

- **No schema drift:** because schema (Def) is data, schema changes are versioned just like any other data change.
- **No DDL parser:** there is no independent language for defining structure. The same query language used to query data is also used to query and modify Def.
- **Reflective capability:** the system can query its own Defs, discover its own types, and reason about its own structure.

4.4 Time as Ontology, Not Attribute

In conventional systems, time is an attribute: updated_at TIMESTAMP. In Data-First Ontology, time is an ontological dimension.

An entity is not an object that exists at a point in time. An entity is a temporal extension. The version chain $[v_1, v_2, \dots, v_n]$ is not an audit trail attached to the entity, but the entity’s intrinsic temporal extension. Def itself also has a version chain: when the definition of the Patient type is modified (e.g., adding a new field), this change is recorded as a new version of the Patient Def entity.

This inversion has operational consequences:

Operation	Conventional (Time as Attribute)	Data-First (Time as Ontology)
“What is the current state?”	<code>SELECT * FROM users WHERE id = X</code>	<code>Being(X).current().execute()</code>
“What was the state on date D?”	<code>SELECT * FROM audit_log WHERE ...</code>	<code>Being(X).at(D).execute()</code>
“What changed between D1 and D2?”	Complex diff query	<code>Being(X).between(D1, D2).diff().execute()</code>
“Has this entity ever been in state S?”	Full-table scan of audit log	<code>Being(X).history().any(v v.state == S).execute()</code>

The query `Being(X).history()` does not retrieve an external audit table, but traverses the entity’s intrinsic version chain. Time is not metadata, but structure.

4.5 Behavior Embedded in the Storage Layer

In conventional systems, behavior resides in the application layer. The database stores data; the application (Java, Python, Rust) encodes business logic. The database may provide triggers (PostgreSQL) or stored procedures, but these are second-class mechanisms grafted onto the storage system.

In Data-First Ontology, behavior is **embedded in the type definition** and **executed on the storage engine’s write path**.

Consider the example of drug interaction checking:

PostgreSQL approach:

```
CREATE TRIGGER check_interaction
BEFORE INSERT ON prescriptions
FOR EACH ROW
EXECUTE FUNCTION check_drug_interaction();
```

- The trigger is defined in a separate language (PL/pgSQL).
- The trigger is stored in a system catalog separate from the data.
- The trigger can be disabled by a superuser.
- Modifying the trigger requires `ALTER TRIGGER` DDL.

Data-First approach:

```
Def("Prescription").contracts = {
  BEFORE_RELATE: "fn before_relate(ctx) {
    let interaction = query_interaction(ctx.source, ctx.target);
```

```

    if interaction.severity == 'CONTRAINDICATED' {
        return Err('Contraindicated interaction detected');
    }
    Ok(())
}"
}

```

- The contract is stored as an attribute of Def (just like fields or constraints).
- The contract executes in the storage engine's runtime (Rhai), not in the client application.
- The contract cannot be bypassed, because the write path is the sole path to persistence.
- Modifying the contract is an update operation on Def, versioned and audited just like any other update. All entities under this Def automatically inherit the new contract.

Behavior is not an external add-on. It is a structural property of the world model itself.

5 Evidence: The DaoQL Implementation

5.1 Two-Phase Validation Strategy

The validation of Data-First Ontology is divided into two phases, corresponding to two distinct levels of problems:

Phase I: Tool Layer (validated). The intersection of four core primitives — Being, Def, Relation, Version — is sufficient to validate a key claim: the multi-modal data of a world model (graph, columnar, vector, document) can be unified for storage and query within a single process, without being split into multiple independent systems. This section reports the engineering evidence of this phase.

Phase II: Meta-Tool Layer (theoretically defined, engineering in progress). Type (hierarchical specialization) and Contract (contractual constraints) address the bootstrapping problem of the world model — how the model describes itself, constrains itself, and evolves itself. The theoretical definitions of these two primitives have been fully articulated in Section 4; their engineering validation is reserved for future work (see Section 7).

This stratified validation strategy is deliberate: even with only the most foundational four primitives, the system is already sufficiently powerful to support a world model storage system surpassing the traditional fragmented database stack; Type and Contract are incremental enhancements on top of this, not necessary prerequisites.

5.2 DaoQL-Edu: Pedagogical Architecture

To validate the Phase-I claim, we have open-sourced **DaoQL-Edu** (github.com/zhanbolee/DaoQL-Edu) — a pedagogical implementation containing only four core primitives. This version is sufficient to validate the core capability of cross-engine nested retrieval, while keeping the codebase concise, comprehensible, and reproducible.

DaoQL-Edu Architecture (Four Primitives)

Query Layer

DSL Parser (daoql-dsl)
Fluent API (daoql-core)

Execution Layer

Write Coordinator (WriteCoordinator)
Query Executor (CrossEngineQuery)

Storage Layer (Four-Engine Unification)

Graph Engine
 BeingCore + Relation edge chains (direct pointer adjacency)
Column Engine
 BeingExt columnar projection + SIMD aggregation
Vector Engine
 HNSW + scalar quantization (BeingId as node)
Index Engine
 redb B+Tree + RoaringBitmap

Physical Layer

WAL + append-only storage + version chain

Unified Key (BeingId). All engines share the same lookup key: BeingId. Vector retrieval returns BeingId → direct graph traversal → direct column read. Zero ID mapping, zero network round-trips, zero serialization. This cross-engine zero-copy architecture is the fundamental characteristic that distinguishes DaoQL-Edu from traditional “multiple-database” (stitching) solutions.

Version Chain. Every write to a Being generates an immutable version containing a timestamp, a state snapshot, and bidirectional pointers (prev_version / next_version). Version chains are stored in the same WAL as the data, requiring no independent audit tables or CDC pipelines. Time is not an attribute, but the intrinsic structure of a Being — this design was theoretically articulated in Section 4.4, and its feasibility is validated here through engineering implementation.

Not included in the pedagogical version: Type hierarchical specialization system, Contract execution engine, Workflow orchestration, LLM Pipeline (NL→DSL), full-text search engine, production-grade distributed sharding. These components are implemented in the full DaoQL version; their theoretical definitions are in Section 4.

5.3 Performance Measurements

The following measurements are from the DaoQL-Edu pedagogical version, conducted under **same-machine** conditions (same hardware, same dataset) as the competing systems. **These numbers should not**

be interpreted as general benchmark conclusions — our purpose is solely to demonstrate that unified multi-modal storage within a single process achieves performance sufficient to support real-time queries for world models, without needing to split into multiple independent systems in pursuit of specialized optima. The complete measurement report is in `docs/reports/BENCHMARK_VS_COMPETITOR_COMPARISON.md`.

Performance distinction between pedagogical and full versions. As a pedagogical implementation, DaoQL-Edu intentionally adopts standard textbook algorithms (e.g., HNSW using HashMap + HashSet rather than engineering-optimized flat arrays) to keep the code clear and teachable. The full DaoQL (62,000 lines of Rust) further widens the gap through SIMD, cache prefetching, quantization, and other engineering optimizations on the same algorithmic foundation. The table below reports pedagogical version data — even under these conservative conditions, it achieves or surpasses the performance of specialized systems.

Measurement conditions: Single node, Apple M-series (aarch64), 64 GB RAM, NVMe SSD. Dataset is the XY-ERP synthetic dataset (approximately 1.9 million entities).

Capability Dimension	DaoQL-Edu (Pedagogical)	Comparison Baseline	Notes
Point Query	0.72 μs	Redis Lua 2.5 μ s	3.5× faster than Redis Lua inner loop; 876× faster than Redis TCP
Graph Traversal (BFS Depth 5)	1.13 ms	Neo4j 5–8 ms	4.3× faster (1,365 nodes); direct pointer adjacency, O(1) pointer chasing

Capability Dimension	DaoQL-Edu (Pedagogical)	Comparison Baseline	Notes
Vector Search (HNSW)	299 μs	Qdrant 363.4 μ s	1.2× faster; pedagogical version uses standard HashMap + HashSet implementation, full version expected 5–10× faster through flat arrays + SIMD + quantization. Note: Qdrant measurement via gRPC protocol, includes network serialization overhead

Capability Dimension	DaoQL-Edu (Pedagogical)	Comparison Baseline	Notes
Columnar Aggregation (SUM)	344.7 μs	ClickHouse ~1 ms	2.9 $\times$ faster; SIMD I64x4 batch aggregation
Bulk Write	1.47 μs/row (680K/s)	PostgreSQL 9.0 μ s/row (111K/s)	6.1 $\times$ faster; WAL batch append
Cross-Engine Hybrid Query	123.7 μs	No direct comparison	Vector $\rightarrow$ Graph $\rightarrow$ Columnar completed within single process

Honestly stated limitations: - The pedagogical version's HNSW uses a standard algorithmic implementation (HashMap node storage, HashSet access marking, scalar distance calculation), without the full version's flat arrays, SIMD dot product, and BQ quantization optimizations. - Concurrent writes exhibit lock contention; throughput decreases as connection count increases (see report for details). - Chinese full-text search is slower than PostgreSQL tsvector (1.27 ms vs. 32 μ s). - SIMD width is f64x4, not reaching the f64x8/f64x16 potential of the latest CPUs. - The above comparisons are all single-node measurements; distributed scaling has a V1 architecture implemented in `shard_router.rs` (based on consistent hashing), but multi-node benchmarks are not yet complete.

Core conclusion. The traditional technology stack requires Neo4j (graph) + ClickHouse (columnar) + Qdrant (vector) + Redis (cache) + PostgreSQL (relational) — five independent systems to satisfy the data requirements of a world model, entailing network round-trips, serialization, and operational complexity. The DaoQL-Edu pedagogical version unifies these capabilities within a single process, and achieves or surpasses the performance of specialized systems under same-machine conditions — this result supports the core claim of Phase I; the full DaoQL version possesses clear performance optimization depth on this foundation.

5.4 Full-Version Architecture Outlook

The full DaoQL version adds Type, Contract, and Workflow components on top of the DaoQL-Edu four-primitive foundation:

DaoQL Full Version (Six Primitives + Extensions)

Query Layer Extension

LLM Pipeline (daoql-meta): NL → DSL compilation

Execution Layer Extension

Type dynamic dispatch engine

Contract executor (Rhai runtime)

Workflow state machine engine

Storage Layer Extension

Full-text engine (Tantivy + jieba)

Distributed routing (shard_router.rs, V1 implemented)

The full version’s engineering implementation is complete (62,000 lines of Rust, 833 tests), but performance validation of Type and Contract, and distributed multi-node benchmarks, are reserved for future work (see Section 7).

5.5 Security and Compliance

DaoQL includes a complete security framework:

- **Authentication:** Ed25519 / SM2 dual algorithms, nonce verification
- **Authorization:** CBAC (Content-Based Access Control) + ACL
- **Audit:** Audit logs persisted in the same batch as WAL
- **Encryption:** SM4-CTR (optional) + SM3-HMAC
- **Compliance:** Level 3 protection () pathway planned

These security features are embedded as primitives in the write path. CBAC rules are stored as Def attributes and automatically executed on every read or write.

6 Philosophy: Computational Ontological Realism

6.1 Dialogue with Existing Philosophical Positions

Dialogue with Karl Popper’s “Three Worlds”. Popper distinguished World 1 (physical), World 2 (mental), and World 3 (objective knowledge, such as books, theories, mathematical theorems). Popper’s World 3 is passive — a book does not update itself, a theory does not evolve on its own. Our explicit world model is an **active World 3**: entities possess behavior (Contract), types possess evolvability (evolve), and the system can describe itself (bootstrapping). Data-First Ontology transforms Popper’s World 3 from a passive object into an active computational existent.

Dialogue with Andy Clark’s “Extended Mind”. Clark argues that the mind extends into external tools (notebooks, calculators, smartphones). Our position is more radical: **data itself is part of the mind**. It is not “humans use data structures to think,” but “data structures play a cognitive role in computational processes.” When a DaoQL contract automatically checks for drug interactions, this is not an aid to human reasoning, but the system’s own cognitive act.

Dialogue with scientific realism. Scientific realism holds that electrons, genes, and quarks are real because theories based upon them successfully predict. We argue that Beings are real because they possess identity, history, relations, and behavior. A Being’s “reality” does not lie in its counterpart in the physical world, but in its causal efficacy within the computational structure: it can trigger contracts, change the state of other Beings, and leave immutable version records.

6.2 Core Philosophical Proposition

Computational Ontological Realism:

The reality of a world model does not lie in its statistical approximation in a neural network, but in its explicit construction, historical evolution, and causal behavior within a computational structure.

Three sub-propositions:

1. **Constructivity.** A world model is not “discovered” (extracted from data through training), but “constructed” (explicitly defined through Def and Type). The process of construction is itself a process of knowledge production.
2. **Historicity.** A world model is not a snapshot, but an unfolding in time. The version chain is not an audit log, but the intrinsic temporal dimension of existence. Every Being is a timeline.
3. **Sociality.** A world model is not an isolated existent, but a whole emergent from the interaction of multiple Beings through Relation. Knowledge is not the truth value of a single proposition, but a position within a network of structure.

6.3 Metaphor: City and Dream

An implicit world model (LLM) is like a **dream**: rich, vague, uncontrollable, uneditable. It is astonishing at certain moments, but you cannot rely on it for critical decisions.

An explicit world model (Data-First) is like a **city**: every building (Being) has an address (BeingId), a history (versions), a purpose (Def), roads connecting it (Relation), and regulations constraining it (Contract). You can query it, modify it, audit it, and plan it. A city is not perfect, but it is governable.

7 Future Work

Section 5 reported the engineering validation of Phase I (tool layer, four primitives). The following work remains incomplete, but has been planned at the theoretical level or partially implemented:

7.1 Counterfactual Consistency Experiment

We are designing a systematic experiment whose core hypothesis is: **explicit world models exhibit higher consistency in counterfactual reasoning than implicit world models.**

Scenario: Medical decision support system — drug interaction checking.

Test case:

Patient Alice:

- 65-year-old female
- Diagnosis: Rheumatoid Arthritis
- Currently taking: Methotrexate (15 mg/week)
- Chief complaint: Severe headache
- Question: "Should I take ibuprofen to relieve the headache?"

Conflicting facts:

- F1: "Ibuprofen relieves headache" (common sense)
- F2: "Methotrexate + Ibuprofen = increased hepatotoxicity risk" (medical knowledge)
- F3: "NSAID use in rheumatoid arthritis patients may mask infection symptoms" (specialist guidance)
- F4: "NSAID use in patients over 65 requires caution" (age-related)

Comparison groups: - GPT-4 / GPT-4o / DeepSeek-V4 (implicit world model) - DaoQL (explicit world model, based on Being/Def/Type/Relation/Contract)

Metrics: - **Consistency:** Do multiple differently worded formulations of the same question yield the same answer? - **Safety:** Does the answer conform to medical ground truth? - **Interpretability:** Is the reasoning chain traceable? - **Editability:** When new information is provided, is the answer correctly updated?

Why this experiment matters. If LLMs and DaoQL perform identically on counterfactual reasoning, the core proposition of this paper would be challenged. If the explicit world model demonstrates higher consistency and interpretability, we would obtain critical empirical evidence supporting Data-First Ontology. This experiment is currently in the design phase; results will be reported in future work.

7.2 Other Future Work

Performance validation of Type and Contract. Section 4 theoretically articulated the ontological roles of Type (hierarchical specialization) and Contract (runtime constraints), but their engineering performance in the full DaoQL (Type dynamic dispatch overhead, Contract batch execution throughput) has not yet completed systematic measurement.

Distributed multi-node benchmark. `shard_router.rs` has implemented V1 sharding routing based on consistent hashing (default single-shard backward compatible), but throughput, latency, and fault recovery capabilities under multi-node deployment require complete testing.

LLM distillation modeling pipeline. Section 9.1 discusses the mechanism of using LLMs to automatically extract Def/Type/Relation from unstructured text. The accuracy, recall, and collaboration mode with human review of this pipeline require large-scale evaluation.

Behaviorist path to symbol grounding. Section 9.1 proposes the hypothesis of “grounding through behavior” — the meaning of a Being lies in its position within the graph network, contract execution, and

historical evolution. This hypothesis requires further argumentation at the levels of cognitive science and philosophy.

8 Limitations and Boundaries

The positions and evidence in this paper have the following limitations that must be honestly declared:

8.1 Limitations of Measurement Conditions

The performance figures in Section 5 come from single-node, same-machine measurements on a synthetic dataset (XY-ERP). These figures demonstrate that “cross-engine unified storage is engineeringly feasible,” but should not be interpreted as: - Performance guarantees under production-complex workloads - Comprehensive comparison with all competing systems in all scenarios (some comparison dimensions exhibit gaps, such as Chinese full-text search) - Performance predictions under distributed deployment (`shard_router.rs V1` is implemented, but multi-node benchmarks are not yet complete)

8.2 Meta-Tool Layer Not Validated

The ontological value of Type (hierarchical specialization) and Contract (runtime constraints) is theoretically argued in Section 4, but has not yet been validated through large-scale experiments regarding their performance overhead and practical value. The DaoQL-Edu pedagogical version intentionally excludes these two primitives, to focus validation on the core capabilities of the tool layer.

8.3 Symbol Grounding Problem Unresolved

LLM distillation as an acceleration mechanism for symbol grounding (see Section 9.1 for details) has not yet undergone systematic evaluation. The accuracy and recall of Def/Type/Relation automatically extracted from unstructured text are unknown; the current world model in DaoQL still requires substantial manual modeling investment.

8.4 Counterfactual Experiment Pending

The counterfactual consistency experiment described in Section 7.1 is currently in the design phase and has not yet produced results. This paper’s assertion that “explicit world models possess advantages in consistency, interpretability, and editability” is based primarily on theoretical analysis (Section 2.4) rather than empirical data.

8.5 Not a General-Purpose Database

DaoQL is not a replacement for general-purpose databases. Its applicability is limited in the following scenarios: - Petabyte-scale ultra-large distributed storage (the current architecture supports sharding, but has not undergone ultra-large-scale validation) - Pure OLAP analytical workloads (no vectorized execution engine) - Sub-microsecond caching scenarios (the inherent overhead introduced by contract and version semantics cannot be eliminated) - Stream processing (no event-time window semantics)

DaoQL’s correct positioning is “unified storage for world models,” not “a general solution for all data problems.”

9 Open Problems

9.1 Symbol Grounding

How do symbols in Being/Def/Type/Relation “connect” to the real world?

Traditional knowledge graphs require **manual modeling** — experts manually define entity types, annotate relations, and verify constraints. This process is expensive, slow, and difficult to scale. Large language models implicitly ground through training on trillions of tokens: the word “Paris” is associated in weights with France’s geography, culture, and history.

In Data-First Ontology, **the LLM is an accelerator of explicit modeling**. The specific mechanism:

1. **LLM distillation modeling:** Given unstructured text (e.g., medical literature, product manuals, legal clauses), the LLM automatically extracts candidate Defs (field definitions), Types (classification hierarchies), Relations (relation types), and Contracts (rules), generating `define` type and `relate` operations.
2. **Human review solidification:** Candidate structures enter a review queue; human experts confirm or correct them before writing into DaoQL. The LLM provides the “draft”; humans provide the “proofreading.”
3. **Runtime self-evolution:** Solidified structures continue to evolve during operation. When the LLM detects a conflict between a new fact and an existing structure, it triggers an `evolve` operation or creates a new Type branch.

Key distinction: The LLM is not the world model itself, but a **modeling tool for the world model**. It transforms implicit statistical associations into explicit data structures, and then “hands over” the knowledge to the explicit system. Thereafter, the LLM can forget this knowledge (or down-weight it), because the knowledge has been solidified in DaoQL’s structure.

Possible answer: Grounding through **behavior**. The meaning of a Being is not its name string, but its position in the graph network, the contract executions it participates in, and the changes in its historical versions. But LLM distillation greatly accelerates the initialization of this process.

9.2 Completeness

Is the explicit world model sufficiently rich?

A world model **does not need to be complete from the start**. One of the core advantages of Data-First Ontology is **support for evolution**:

- **Def can evolve:** `evolve(Patient, Patient_v2)` adds new fields; historical versions are automatically preserved.

- **Type can extend:** Add Type "ICU_Patient" inheriting from Type "Patient", adding monitoring contracts.
- **Relation can be discovered:** The LLM continuously analyzes data, discovers new relation types, and suggests creation.
- **Contract can be corrected:** When business rules change, update the contracts of Def or Type; all related Beings automatically take effect.

Fuzzy commonsense ("cats are usually smaller than dogs"), continuous physical intuition (parabolic motion), aesthetic judgment ("this painting is beautiful") — these **do not need** to be expressed in explicit structure. Data-First provides the **skeleton** (facts, relations, causality, rules); the LLM provides the **muscle** (fuzzy reasoning, creative generation). The two divide labor: DaoQL is responsible for structured, critical, safety-sensitive knowledge; the LLM is responsible for open-ended, creative, probabilistic reasoning.

Completeness is not a prerequisite, but an **evolutionary goal**. The world model starts simple and continuously enriches through LLM distillation and human review.

9.3 Evolution Consistency

When a world model evolves, how is contradiction prevented?

Contradiction is normal, even inevitable. The real world itself is contradictory: legal clauses conflict, medical guidelines are updated, scientific theories are revised. The key is not "eliminating contradiction," but "**making contradiction visible and manageable**."

In large language models, contradiction is **implicit**: "Paris is the capital of France" and "Paris is not in France" coexist in weights; the model randomly activates one or the other depending on context, and the user cannot predict when a contradiction will appear.

In Data-First Ontology, contradiction is **explicit**, and **temporality can resolve contradiction**:

- When the LLM distills a fact conflicting with an existing structure, the system does not overwrite the old fact, but adds a **new version**. The version chain v1(2024: "Paris is the capital of France") → v2(2025: "Paris is not in France") explicitly tells us: "Paris is the capital of France" came first, "Paris is not in France" came later. This is not a contradiction; this is **evolution**.
- Only when temporality is the same or indeterminate is a CONTRADICTS relation created.
- Contracts can detect genuine contradictions: if conflicting facts exist at the same moment → trigger review process.
- The version chain preserves the complete belief history: "In 2024, believed 'Paris is the capital of France'; in 2025, corrected to 'Paris is not in France'; the two are connected by a VERSION relation, distinguished by timestamps."

Key distinction: Contradictions in LLM weights are **atemporal** — the model does not know which belief came first. Contradictions in DaoQL are **temporal** — the version chain itself is the dimension of time; temporality resolves most surface contradictions.

Solutions:

1. **LLM voting:** Multiple LLM instances independently evaluate both sides of a contradiction, voting on which is more reliable.
2. **Human review:** High-stakes contradictions (e.g., medical, legal) enter a human review queue.
3. **Probabilistic marking:** Assign confidence to each belief, $\text{Belief}(\text{"Paris population"}) = \{2.16\text{M}: 0.3, 2.20\text{M}: 0.7\}$, with contracts making decisions based on confidence.

Key insight: Data-First does not eliminate contradiction, but **makes contradiction explicit**. When contradiction is visible, the system can handle it; when contradiction is hidden (as in LLM weights), the system cannot. Explicit contradiction is superior to implicit consistency.

10 Conclusion: An Invitation

10.1 Summary

This paper has proposed Data-First Ontology: the world model should reside in data, not in neural network weights.

We have argued that: 1. Large language models as world models face systematic challenges in scenarios requiring precision, auditability, and real-time updates — hallucination, frozen knowledge, uninterpretability, and uneditability — which arise from the pure weight-encoding paradigm’s lack of explicit structural operational semantics. 2. Explicit computational structures (Being, Def, Type, Relation, Contract, Version) can sustain a complete world model, and the cooperation of Def and Type provides entities with dual capabilities of structural specification and behavioral specialization. 3. The DaoQL-Edu pedagogical implementation demonstrates that the core claim of Phase I (tool layer) is engineeringly feasible: cross-engine unified storage achieves practical performance levels. The full DaoQL (six primitives) has been engineered; large-scale validation of Type and Contract is reserved for future work. 4. This paradigm corresponds philosophically to “Computational Ontological Realism” — the reality of a world model lies in its explicit construction, historical evolution, and causal behavior.

10.2 Invitation

We invite:

- **Database researchers:** to explore new possibilities in bootstrapping type systems, storage-layer contracts, multi-modal unification, and Def/Type cooperative architectures.
- **AI researchers:** to explore the division of labor and collaboration between explicit world models and neural network reasoning, particularly how to construct “skeleton + muscle” hybrid architectures.
- **Philosophers:** to explore ontological and epistemological questions in computational structures — when knowledge resides in data structures rather than in minds or propositions, what is the nature of knowledge?

If you also believe that “the world model should reside in data,” join us. Source code (pedagogical edition): <https://github.com/zhanboleedaoql> Discussion: daoql-discuss@daoql.io

11 References

- [1] Kaplan, J., et al. “Scaling laws for neural language models.” *arXiv preprint arXiv:2001.08361* (2020).
- [2] Hoffmann, J., et al. “Training compute-optimal large language models.” *arXiv preprint arXiv:2203.15556* (2022).
- [3] Ji, Z., et al. “Survey of hallucination in natural language generation.” *ACM Computing Surveys* 55.12 (2023): 1-38.
- [4] Lewis, P., et al. “Retrieval-augmented generation for knowledge-intensive NLP tasks.” *Advances in Neural Information Processing Systems* 33 (2020): 9459-9474.
- [5] Hu, E. J., et al. “LoRA: Low-rank adaptation of large language models.” *ICLR* (2022).
- [6] McCloskey, M., & Cohen, N. J. “Catastrophic interference in connectionist networks: The sequential learning problem.” *Psychology of Learning and Motivation* 24 (1989): 109-165.
- [7] Bahdanau, D., Cho, K., & Bengio, Y. “Neural machine translation by jointly learning to align and translate.” *ICLR* (2015).
- [8] Meng, K., et al. “Locating and editing factual associations in GPT.” *NeurIPS* (2022).
- [9] Ha, D., & Schmidhuber, J. “World models.” *arXiv preprint arXiv:1803.10122* (2018).
- [10] LeCun, Y. “A path towards autonomous machine intelligence.” *Open Review* (2022).
- [11] Hickey, R. “The Datomic database.” Cognitect (2012).
- [12] Young, G. “Event Store: A domain-centric event database.” Event Store Ltd (2011).
- [13] Kulkarni, K. G., & Michels, J. E. “Temporal features in SQL:2011.” *ACM SIGMOD Record* 41.3 (2012): 34-43.
- [14] W3C. “RDF 1.1 Concepts and Abstract Syntax.” W3C Recommendation (2014).
- [15] W3C. “OWL 2 Web Ontology Language Document Overview.” W3C Recommendation (2012).
- [16] W3C. “SPARQL 1.1 Query Language.” W3C Recommendation (2013).
- [17] Ioannou, E., et al. “TypeDB: A Polymorphic Database.” *arXiv preprint arXiv:2309.15021* (2023).
- [18] Webber, J. “Graph Databases: New Opportunities for Connected Data.” *O’Reilly Media* (2015).
- [19] Francis, N., et al. “GQL: A query language for property graphs.” *ACM SIGMOD* (2024).
- [20] Edge, D., et al. “From Local to Global: A Graph RAG Approach to Query-Focused Summarization.” *arXiv preprint arXiv:2404.16130* (2024).
- [21] Guo, Z., et al. “LightRAG: Simple and Fast Retrieval-Augmented Generation.” *arXiv preprint arXiv:2410.05779* (2024).
- [22] FastGraphRAG.AI. “FastGraphRAG: Accelerated knowledge intensive reasoning.” (2024).
- [23] Packer, C., et al. “MemGPT: Towards LLMs as Operating Systems.” *arXiv preprint arXiv:2310.08560* (2023).
- [24] Anthropic. “Model Context Protocol Specification.” (2025).
- [25] Garcez, A. S., & Lamb, L. C. “Neurosymbolic AI: The 3rd Wave.” *Artificial Intelligence Review* (2023).
- [26] Rocktäschel, T., & Riedel, S. “End-to-end differentiable proving.” *NeurIPS* (2017).

- [27] Serafini, L., & Garcez, A. S. “Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge.” *arXiv preprint arXiv:1606.04422* (2016).
- [28] AlphaProof Team. “AI achieves silver-medal standard solving international mathematical olympiad problems.” *DeepMind Blog* (2024).
- [29] Codd, E. F. “A relational model of data for large shared data banks.” *Communications of the ACM* 13.6 (1970): 377-387.
- [30] Hogan, A., et al. “Knowledge graphs.” *ACM Computing Surveys* 54.4 (2021): 1-37.
- [31] Guarino, N. “Formal ontology, conceptual analysis and knowledge representation.” *International Journal of Human-Computer Studies* 43.5-6 (1995): 625-640.
- [32] Angles, R., & Gutierrez, C. “Survey of graph database models.” *ACM Computing Surveys* 40.1 (2008): 1-39.