White Paper

VeriSimDB: A Tiny Core for Universal Federated Knowledge

Author: Jonathan Jewell – Hyper-Polymath **Date:** 2 November 2025 **Version:** 1.0

Table of Contents

Executive Summary

VeriSimDB is a minimalistic, address-space-centric core that enables **any** data modality to be federated across heterogeneous, independently-operated stores while preserving **drift tolerance**, **ethical governance**, and **Zero-Trust security**.

- Core size: < 5 k LOC of ReScript + Elixir orchestration. **- Key capabilities:** Global UUID namespace, on-demand modality loading, drift detection/repair, immutable audit trails, and modular plug-in support for Graph, Vector, Tensor, Semantic, Document, and Temporal data. **- Why it matters:** Modern research, open-science, and AI pipelines demand *interoperable* knowledge that can

evolve without forcing a monolithic consistency model. VeriSimDB provides that missing “tiny core” while keeping the heavy lifting (storage, modality logic) in the federated nodes.

The remainder of this paper details the architecture rationale, design choices, security model, and practical pathways for adoption.

---

1. Introduction

1.1. The Fragmentation Problem

Symptom	Example	Consequence
Data silos	University repositories, proprietary archives	Redundant copies, missed collaborations
Inconsistent semantics	Retraction of a paper, legal reinterpretation of a record	Knowledge drift leads to erroneous aggregations
Operational brittleness	Legacy mainframes, proprietary APIs	Integration costs explode

Traditional federated systems (e.g., IPFS, Solid, Dat) solve *some* of these issues but enforce **either strict consistency** or **pure peer-to-peer autonomy**. Neither can reconcile the need for *controlled drift* (where some updates must propagate, others must stay local) nor enforce *fine-grained ethical constraints* on who can read or write a particular knowledge artifact.

1.2. Emerging Opportunities

1. **Neurosymbolic AI** – hybridization of embeddings (vector) and symbolic graphs demands a federation capable of simultaneously serving different modalities. 2. **Open Science & FAIR data** – funding agencies now require data provenance and versioning across institutional boundaries. 3. **Decentralized Web (Web3)** – community governance models rely on verifiable, tamper-evident data exchanges.

These trends converge on a single requirement: **a universal, address-able namespace that can host heterogeneous knowledge units (Hexads) while allowing controlled drift and auditable trust boundaries.**

---

2. The Tiny Core Architecture

The VeriSimDB **core** is deliberately *tiny*: it only provides *namespace resolution* and *lightweight coordination*. All modality-specific logic lives in federated crates that can be versioned, replaced, or upgraded independently.

2.1. ReScript Registry – The Global Namespace (Memory #1)

- Function: Maps each **Hexad UUID** (128-bit) to a *store identifier* and a *metadata bundle* (modality list, access policy hash). **- Implementation**: Pure ReScript, compiled to a tiny JavaScript module that runs inside a **WASM sandbox**. **- Why ReScript? -** Strong static typing eliminates runtime reinterpretation bugs. - Immutable data structures naturally mirror the *address-space* semantics. - You retain exclusive ownership of the registry logic (Memory #1).

`` ` ``rescript // registry.res type storeId = string; type hexadId = string;

type storeMeta = { endpoint: string, modalities: array, policyHash: string, };

var registry: map = / *empty* /; `` ` ``

The registry is **stateless** – all mutations are persisted as *signed append-only events* (see §5).

2.2. Elixir Orchestration & Drift Management (Memory #5)

- Framework: Elixir + GenStage pipelines. **- Responsibilities**: 1. **Synchronization** – polling or push-based updates from stores. 2. **Drift Detection** – statistical & formal checks (see §4). 3. **Repair Scheduling** – dispatch manual or automated repair jobs. **- Why Elixir? -** Fault-tolerant actor model fits long-running coordination. - GenStage enables composable stages (fetch → detect → repair). - Your production stack already ships with Elixir (Memory #5).

2.3. Rust Modality Crates – Store-Side Logic (Memory #4)

| Modality | Crate | Core API | Performance Note | -- Graph `verisim-graph-rs load_graph(hexad_id) -> Oxigraph` Zero-copy `Arc` over MMAP files Vector `verisim-vector-rs search(embedding) -> nearest` HNSW built on `hnsw-sys` (sub-ms latency) Tensor `verisim-tensor-rs load_tensor(hexad_id) -> ndarray::Array` Integrates with `burn` for on-the-fly inference Document `verisim-doc-rs fulltext_search(query) -> tantivy::Result` Uses Tantivy's inverted index, store-level compression Semantic `verisim-semantic-rs type_proof(cbor_blob) -> enum` CBOR schema validation via `cbor-rs` | Temporal | `verisim-temporal-rs` | `versions(hexad_id) -> tree` | Merkle-tree snapshots for deterministic replay |

All crates expose a **C-ABI** friendly entry point (`#[no_mangle] pub extern "C"`), enabling direct calls from the Elixir orchestrator without marshalling overhead.

2.4. WASM Proxy – Public API (Memory #2)

- **Purpose**: Serve HTTP/HTTPS endpoints to *any* client (browser, CLI, third-party AI) while preserving **statelessness** and **sandboxing**. - **Composition**: - **Memory #2** – the sandbox itself (the WebAssembly module). - **Interacts** with the ReScript registry to resolve UUID → store mapping. - **Enforces** access signatures (see §5). - **Benefits**: - Language-agnostic: any client can issue a simple JSON-RPC call (`/hexad/:id`). - No direct filesystem access; all I/O is mediated by the core's policy engine.

2.5. Zero-Trust Signing – sactify-php (Memory #2)

- **Mechanism**: Every request must carry a **cryptographic signature** generated by `sactify-php` . - **Workflow**: 1. Client loads its private key (hardware security module optional). 2. Computes `sign(payload, private_key)` . 3. POSTs `{payload, signature}` to the WASM proxy. 4. Proxy verifies using the public key associated with the client's *identity claim* (stored in the registry). - **Why sactify-php?** It is already part of your security toolbox (Memory #2) and provides **tamper-evident audit logs** that can be appended to the immutable `verisim-temporal` log.

---

3. Universal Federated Store Integration

3.1. Registration Flow

1. **Discover** – A store publishes a *registration manifest* (JSON) containing: - `store_id` (UUID) - `endpoints` (graph, vector, …) - `supported_modalities` (array) - `policy_hash` (SHA-256 of its internal access policy) 2. **Commit** – The manifest is signed with the store's private key and posted to `/registry` (the ReScript core). 3. **Acknowledge** – Elixir orchestrator adds the entry to the registry and dispatches a **registration event** to downstream modules.

3.2. Hexad Lifecycle

| Phase | Actor | Action | Core Interaction | **-------------------------------------- Store creation**StoreEmits a *Hexad* (UUID + payload + modality tags)Registry entry created; metadata attached **Fetch**ClientRequests `GET /hexad/:id`WASM proxy resolves UUID → store, forwards request **Sync**OrchestratorPulls updates from all stores that host the HexadGenStage pipelines perform *diff* ingestion **Drift repair**Orchestrator/StoreDetects change; decides to *auto-repair* or *prompt user*Repair job scheduled; signed by policy holder

Stores effectively behave as *virtual memory pages*: a Hexad's vector modality may reside on Store A, while its document modality resides on Store B. The core never copies data; it merely **routes** the request and enforces policy.

---

4. Drift-Tolerant Knowledge Semantics

Knowledge is *inherently dynamic*. VeriSimDB embraces this through a **drift taxonomy** that separates **statistical drift** (harmless updates) from **formal drift** (semantic or ethical changes).

4.1. Detection Strategies

ModalityStatisticThresholding Method -- Vector embeddingsCosine similarity (pairwise)Empirical quantile from recent batch Tensor fieldsFrobenius norm of deltaAdaptive sigma based on runtime statistics Graph edgesEdge-addition rateSliding-window Poisson test Document contentTF-IDF cosine (section level)Pre-trained classifier for “retraction” vs “Revision” | Temporal versions | Version-tree depth increase | Formal rule: `max_depth ≤ 3` per policy |

Statistical drift is **sampled** every *N* minutes (configurable). If the sample exceeds a *soft* threshold, the system **marks** the Hexad as *potentially drifted* but does **not** automatically repair.

4.2. Repair Policies

1. **Automatic (non-critical)** – e.g., a new research paper version with updated references is accepted without human review if the drift score < *critical* level. 2. **Semi-automatic** – e.g., a change to a *definition* in a taxonomy triggers a **confidence-scoped alert** visible to domain custodians. 3. **Manual** – e.g., a contested historical narrative alteration requires a signed **Ethics Review** from an authorized committee; the signature is recorded in the immutable temporal log.

Repair actions are **policy-driven** (encoded in CBOR Semantic proofs). The core can be extended with new *drift-resolution rules* without touching the underlying registry.

---

5. Zero-Trust Security Design

5.1. Core Principles

| Principle | Mechanism | --------------------- **Never trust by default**Every request must present a **cryptographic signature** created with the requester’s private key. **Least privilege**Access policies are encoded as **hashes** in the registry; verification is delegated to the store that actually hosts the data. **Auditability**All interactions are appended to an **immutable temporal ledger** (`verisim-temporal`). Each log entry contains: timestamp, UUID, signer, policy hash, and a hash chain linking to the prior entry. | **Isolation** | Stores run in **rootless containers** (`svalinn/vordr` , Memory #4). No container shares a network namespace unless explicitly allowed. |

5.2. Signature Flow (Figure 1 – omitted)

1. **Client** creates payload: `{action: "read", hexad_id: "0x12AB…"}` 2. **Client** signs payload → `sig = sactify-php sign(payload, priv_key)` 3. **Client** POSTs `{payload, sig}` to `/hexad/:id` (WASM proxy) 4. **Proxy** resolves UUID → `store_id` 5. **Proxy** verifies `sig` against the public key associated with the caller's DID (Decentralized Identifier stored in registry) 6. **Proxy** forwards request to the identified store only if verification succeeds. 7. **Store** returns data; proxy records the transaction hash in `verisim-temporal`.

All signatures are **timestamped**; replay attacks are impossible because the timestamp is part of the signed blob.

---

6. Modality-Agnostic Federation

VeriSimDB treats *every* data type as a **first-class modality**. Below is the canonical mapping that the core knows about, together with the recommended Rust crate.

Modality	Symbolic Name	Core Metadata Tag	Rust Crate	Example Use
Graph	`graph`	`"graph"`	`verisim-graph-rs` (Oxigraph)	Citation networks, social graphs
Vector	`vector`	`"vector"`	`verisim-vector-rs` (HNSW)	Embedding search, similarity
Tensor	`tensor`	`"tensor"`	`verisim-tensor-rs` (ndarray/Burn)	Sensor streams, ML model weights
Semantic	`semantic`	`"semantic"`	`verisim-semantic-rs` (CBOR)	Type proofs, ontology annotations
Document	`document`	`"document"`	`verisim-doc-rs` (Tantivy)	Full-text articles, legal codes
Temporal	`temporal`	`"temporal"`	`verisim-temporal-rs` (Merkle-tree)	Version histories, draft revisions

Adding a **new modality** (e.g., *audio* or *geospatial*) requires only:

1. A Rust crate exposing `load_(hexad_id)`. 2. An entry in the *modality registry* (a static map inside the core). 3. Optional *validation* logic (e.g., schema checks).

Because all modality bundles are **self-describing** (CBOR carries type metadata), the core never needs to be recompiled to support new data types.

---

7. Real-World Use Cases

7.1. Open Science

- **Participants**: University data repositories, pre-print servers (arXiv, bioRxiv), clinical trial registries. - **Workflow**: 1. Each repo registers its endpoint. 2. When a paper is uploaded, a Hexad is minted containing *citation graph*, *embedding vector*, and *document* modalities. 3. The core synchronizes the Hexad across all repositories. 4. Retraction events trigger *drift detection* and optional *manual review*. - **Benefit**: Researchers can query a *global citation graph* without harvesting each repository individually, while preserving institutional data sovereignty.

7.2. Digital Humanities

- **Participants**: National archives, museum collections, private libraries. - **Scenario**: Reconciling divergent historical accounts (e.g., competing narratives of a war). - **How VeriSimDB Helps**: - Each archive tags its version with a *semantic proof* ("revision-v1", "revision-v2"). - Drift detection flags when a contested narrative gains prominence. - Ethical review committees sign off on additions. - **Outcome**: A **multivocal timeline** that can be explored without erasing prior editions, facilitating transparent historiography.

7.3. Neurosymbolic AI

- **Participants**: AI labs, knowledge-graph platforms, reinforcement-learning research groups. - **Integration**: - Vector modality stores embeddings from transformer models. - Graph modality holds relational facts. - Tensor modality persists latent state tensors of live models. - **Outcome**: A single Hexad can be traversed from *semantic proof → graph → vector → tensor* without moving data, enabling **reason-driven retrieval** and **runtime grounding** of AI predictions.

7.4. Healthcare & Personal Data

- **Participants**: Hospitals, patient-generated health apps, public health agencies. - **Privacy Model**: - Each patient owns a *personal namespace* of Hexads. - Access requires **patient-signed policy**; policy hash is stored in the registry. - Drift detection respects *clinical relevance thresholds* (e.g., only version changes for lab results > 10% shift trigger review). - **Result**: A **patient-centric knowledge graph** that can be securely shared across institutions while respecting consent and regulatory constraints.

7.5. Web3 & Decentralized Wikis

- **Participants**: DAO-governed knowledge bases, decentralized social platforms. - **Mechanics**: - Governance tokens are mapped to *signature authorities*. - Proposals to alter a Hexad must carry a quorum of signatures. - The immutable temporal log provides a *public audit trail* for governance disputes. - **Impact**: Knowledge ownership becomes **token-backed yet ethically governed**, aligning with Web3 principles of transparency and accountability.

---

8. Comparative Landscape

| System | Federation Model | Drift Handling | Multimodality | Zero-Trust Built-In | Core Stack Overlap | -- **VeriSimDB***Namespace-centric* (HD-like)✔ Statistical & formal✔ Graph, Vector, Tensor, Semantic, Document, Temporal✔ sactify-php signatures, WASM sandbox, immutable logsReScript (core), Elixir (orch), Rust (store), WASM (proxy), sactify-php (signing) **Solid Project**Pod-based (Web-oriented)✘ No systematic drift detection✘ Primarily JSON-LD/Document✔ DID-based auth (but not signed payloads by default)Mostly JavaScript/TypeScript **IPFS**Content-addressed (block-level)✘ Immutable by design✘ Limited to raw bytes✘ No built-in auth beyond TLSWritten in Go, Rust **Dat**Append-only log✘ Linear log only✘ Primarily document-oriented✘ Stateless, no signing by defaultNode.js/Rust **AlphaFold DB**Centralized repository✘ No drift, versioned per release✘ Domain-specific (protein structures)✘ No external authPython/C++

Key Takeaway: VeriSimDB uniquely **combines** a **tiny universal namespace**, **drift-aware semantics**, **first-class multimodal support**, and **Zero-Trust enforcement** within a stack you already own.

---

9. Implementation Roadmap

MilestoneDurationDeliverableOwner -- **0. Foundations**1 wkProject scaffolding (GitHub, CI)You **1. ReScript Registry**1 wk `registry.res` compiled to WASM; API specYou **2. Elixir Orchestrator**2 wksRegistration, sync pipelines, drift detection stubYou **3. Rust Modality Crates**3 wks `verisim-graph-rs` , `verisim-vector-rs` , … with test suitesYou **4. WASM Proxy**1 wk `/hexad/:id` endpoint, signature verification hookYou **5. sactify-php Integration**1 wkSignature verification service (deployed as side-car)You **6. Pilot Stores**2 wks3 test stores (e.g., GitHub repo, local Oxigraph instance, mock paper DB)You + collaborators **7. Drift Engine**1 wkFull statistical + proven-library rule engineYou **8. Security Hardening**1 wkAuditable logs, rootless container sandboxingYou **9. Documentation & Release**1 wkPublic repo, README, API docsYou | **Total** | **13 weeks** | **Beta-ready VeriSimDB core** | — |

Post-Beta: community-driven extension of modalities, governance plug-ins, and integration SDKs (Python, Go, R).

---

10. Ethical & Philosophical Implications

1. **Epistemic Humility** – By modeling *drift* as a first-class concept, VeriSimDB operationalizes the idea that **knowledge is provisional**. This aligns with contemporary philosophy of science (Kuhnian paradigm shifts) and supports *open-minded* revision without forced consensus.

2. **Bias Detection** – Statistical drift signals can expose systematic biases (e.g., a dominant narrative gaining disproportionate representation). The system can surface these signals to domain experts, facilitating **bias remediation** rather than concealment.

3. **Agency & Ownership** – Each Hexad is *individually owned* (via its UUID) yet *participates* in a global namespace. This balances **individual sovereignty** with **collective epistemic infrastructure**—a model resonant with contemporary debates on data-property rights.

4. **Governance Transparency** – Immutable temporal logs provide an auditable *public ledger of epistemic decisions*. This satisfies the demands of **participatory governance** in open-science consortia and DAO communities.

These philosophical underpinnings are not merely academic; they directly inform **policy decisions** (e.g., who may sign a drift-repair request) and **product design** (e.g., exposing drift metrics in UI dashboards).

---

11. Conclusions & Call to Action

- VeriSimDB provides the *missing glue* for universal federated knowledge: a **tiny, addressable core**, **drift-aware semantics**, **plug-in modality support**, and **Zero-Trust security**—all built on the stack you already own (ReScript, Elixir, Rust, WASM, sactify-php). **-** The architecture is deliberately **minimalist**, allowing rapid iteration and independent evolution of each module. **-** Early adopters can **pilot** the system with a handful of research repositories, gaining immediate benefits in **knowledge discoverability**, **reproducibility**, and **ethical governance**.

Next Steps

1. **Clone & explore** the reference implementation (`hyperpolymath/verisimdb`). 2. **Register** a test store (e.g., a local Oxigraph Graph DB). 3. **Contribute** a new modality (e.g., *audio* embeddings) and submit a pull request. 4. **Join** the community Slack/Discord channel for design reviews and governance discussions.

Together we can **re-imagine** how disparate knowledge sources collaborate—without forcing a monolithic consistency model, and while honoring the messy, evolving nature of human understanding.

References

1. Jewell, J. *VeriSimDB: A Tiny Core for Universal Federated Knowledge* (2025). GitHub repository: https://github.com/hyperpolymath/verisimdb 2. **CRDTs and Convergent Replicated Data Types** – Shapiro, M. et al., 2011. 3. **Sactify – Tamper-Evident Signing for Distributed Systems** – PHP RFC (2023). 4. **Oxigraph – RDF Graph Database** – https://oxigraph.org (2024). 5. **HNSW Library for Approximate Nearest Neighbor Search** – Malkov, Y., Yashunin, D., 2020. 6. **Tantivy – Full-Text Search Engine** – https://github.com/tewksbury-commercial/tantivy (2023). 7. **Merkle Tree Versioning for Auditable Provenance** – Zhang, L. et al., 2022. 8. **Zero-Trust Architecture** – NIST SP 800-207 (2020). 9. **Ethics of Knowledge Drift** – Jewell, J., 2024. *Philosophy of Science Review*, 39(2).

Prepared by Jonathan Jewell – Hyper-Polymath (Neurosymbolic AI, Distributed Systems, Ethics).