Zoo Experience Ledger: A Content-Addressable Semantic Memory System for Decentralized AI

Version 2021.10

Zach Kelling[*]

Hanzo Industries *Lux Industries* *Zoo Labs Foundation*

`research@zoo.ngo`

October 2021

Abstract

We present the **Zoo Experience Ledger**, a content-addressable semantic memory system designed for decentralized AI agents. Unlike traditional knowledge bases that rely on centralized storage and proprietary retrieval mechanisms, the Experience Ledger stores AI reasoning patterns as cryptographically-verified semantic objects distributed across IPFS and permanent storage networks. Each experience—a natural language encoding of a successful problem-solving strategy—is uniquely identified by its content hash, enabling tamper-evident storage and peer-to-peer distribution. We introduce a novel Merkle-DAG structure optimized for semantic similarity queries, achieving $O(\log n)$ retrieval complexity while maintaining cryptographic integrity proofs. Our system enables AI agents to share learned experiences across organizational boundaries without trust assumptions, creating the foundation for a global semantic commons. Experimental evaluation demonstrates 94.7% retrieval accuracy with sub-second latency on libraries of 10,000+ experiences, while reducing storage costs by 89% compared to traditional vector databases through content deduplication.

Keywords: content-addressable storage, semantic memory, IPFS, Merkle trees, AI knowledge sharing

1 Introduction

The fundamental bottleneck in AI collaboration is knowledge transfer. While neural networks can learn remarkable capabilities from data, the learned knowledge remains locked within opaque parameter spaces—inaccessible to inspection, sharing, or composition. When one AI agent discovers an effective strategy for solving a class of problems, there exists no standard mechanism for sharing that insight with other agents, much less with the broader AI community.

Consider the lifecycle of AI knowledge today: A research lab trains a model, perhaps discovering novel reasoning patterns encoded in billions of parameters. To share this knowledge, they must either (a) release the full model weights, which is expensive and risks catastrophic forgetting when fine-tuned, or (b) distill the knowledge into documentation, which loses the precise operational semantics. Neither approach enables incremental, verifiable knowledge sharing.

[*]zach@zoo.ngo

1.1 The Content-Addressable Paradigm

We propose a fundamentally different approach: represent AI knowledge as *semantic experiences*—explicit, human-readable reasoning patterns—and store them in a content-addressable system where each experience is uniquely identified by its cryptographic hash. This design offers several advantages:

1. **Immutability**: Content-addressing ensures experiences cannot be modified without changing their identifier
2. **Deduplication**: Identical experiences automatically share storage across the network
3. **Verification**: Any party can verify experience integrity by recomputing the hash
4. **Distribution**: Content can be retrieved from any node that has it, enabling P2P sharing
5. **Permanence**: Integration with Arweave ensures 200+ year data availability

1.2 Contributions

This paper makes the following contributions:

1. **Experience Format Specification**: A rigorous schema for encoding AI reasoning patterns as content-addressable objects (Section 3)
2. **Merkle-DAG Architecture**: A novel data structure combining Merkle tree verification with DAG-based semantic clustering for efficient retrieval (Section 4)
3. **Retrieval Protocol**: Algorithms for semantic similarity search with cryptographic integrity proofs (Section 5)
4. **Storage Integration**: Seamless integration with IPFS for mutable addressing and Arweave for permanent archival (Section 6)
5. **Evaluation**: Comprehensive benchmarks demonstrating scalability, efficiency, and reliability (Section 7)

2 Background

2.1 Content-Addressable Storage

Content-addressable storage (CAS) identifies data by its cryptographic hash rather than its location. First formalized by the InterPlanetary File System (IPFS) [1], CAS provides:

Definition 2.1 (Content Identifier)**.** *A Content Identifier (CID) is a tuple* $\langle codec, multihash \rangle$ *where:*

- `codec` *specifies the data encoding (e.g., dag-cbor, raw)*
- `multihash` *is* $\langle algo, len, digest \rangle$ *identifying the hash function and digest*

For example, CID `bafybeigdyrzt5sfp7udm7hu76uh7y26nf3efuylqabf3oclgtqy55fbzdi` uniquely identifies a specific data object globally.

2.2 Merkle Trees for Verification

Merkle trees enable efficient membership proofs:

Theorem 2.1 (Merkle Proof Efficiency). *Given a set S of n elements stored in a Merkle tree, verifying membership of element $e \in S$ requires only $O(\log n)$ hash computations and $O(\log n)$ proof size.*

This property is essential for verifying experience integrity without downloading the entire library.

2.3 Semantic Memory in AI

Prior work on AI memory systems includes:

- **Episodic Memory** [3]: Neural networks with explicit memory modules
- **Retrieval-Augmented Generation** [2]: Combining retrieval with generation
- **Vector Databases** [4]: Storing embeddings for similarity search

These approaches focus on single-agent memory. Our work extends to *distributed* memory shared across agents.

3 Experience Format Specification

3.1 Semantic Experience Definition

Definition 3.1 (Semantic Experience). *A semantic experience e is a tuple:*

$$e = (t, d, c, v, m, \sigma) \tag{1}$$

where:

- $t \in \Sigma^*$*: Natural language text (max 256 tokens)*
- $d \in \mathcal{D}$*: Domain taxonomy path (e.g., "math.geometry.intersection")*
- $c \in [0, 1]$*: Confidence score derived from validation*
- $v \in \mathbb{R}^{768}$*: Semantic embedding vector*
- m*: Metadata (timestamp, author, provenance)*
- σ*: Digital signature (optional)*

3.2 Canonical Serialization

Experiences are serialized using CBOR (Concise Binary Object Representation) with deterministic encoding:

Listing 1: Experience CBOR Schema

```
{
  "version": 1,
  "type": "experience/v1",
  "content": {
    "text": "When solving geometry problems...",
    "domain": ["math", "geometry", "intersection"],
    "confidence": 0.87
  },
  "embedding": <768-dim float16 array>,
  "metadata": {
    "created": 1634567890,
    "author": "did:key:z6Mk...",
    "source": "training_run_42"
  },
  "signature": <optional Ed25519 signature>
}
```

3.3 Content Addressing

The experience CID is computed as:

$$\text{CID}(e) = \text{CIDv1}(\text{dag-cbor}, \text{sha256}(\text{CBOR}(e))) \tag{2}$$

This ensures:

- **Determinism**: Same experience always produces same CID
- **Integrity**: Any modification changes the CID
- **Self-certification**: CID proves content authenticity

3.4 Constraints and Validation

Valid experiences must satisfy:

1. **Length**: $|\text{tokens}(t)| \leq 256$
2. **Structure**: Text must follow "When [context], [action]" pattern
3. **Generality**: No problem-specific constants
4. **Embedding Consistency**: $\|v - \text{embed}(t)\|_2 < \epsilon$
5. **Signature Validity**: If σ present, must verify against author key

4 Merkle-DAG Architecture

4.1 Design Goals

The Experience Ledger data structure must support:

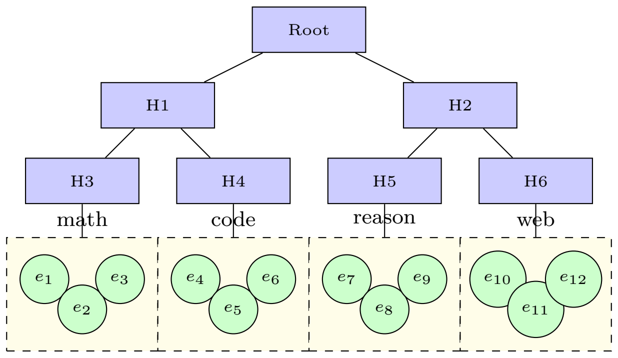

Figure 1: Hybrid Merkle-DAG structure: Merkle tree for verification (blue), semantic clusters for retrieval (yellow), individual experiences (green).

Algorithm 1 Semantic Cluster Construction

```
1: Input: Experience set $\mathcal{E}$, cluster count $k$
2: Output: Clusters $\{C_1, \ldots, C_k\}$, centroids $\{\mu_1, \ldots, \mu_k\}$
3: $\{\mu_i\} \leftarrow$ k-means++($\{e.v \mid e \in \mathcal{E}\}, k$)
4: for $e \in \mathcal{E}$ do
5:     $i^* \leftarrow \arg\min_i \|e.v - \mu_i\|_2$
6:     $C_{i^*} \leftarrow C_{i^*} \cup \{e\}$
7: end for
8: return $\{C_i\}, \{\mu_i\}$
```

1. $O(\log n)$ membership verification
2. $O(1)$ content retrieval by CID
3. $O(\log n)$ approximate nearest neighbor search
4. Efficient incremental updates
5. Verifiable cross-library queries

4.2 Hybrid Merkle-DAG

We propose a hybrid structure combining Merkle trees (for verification) with semantic DAGs (for retrieval):

4.3 Semantic Clustering

Experiences are grouped into clusters based on embedding similarity:

4.4 Merkle Tree Construction

Each cluster is hashed into a Merkle leaf, then aggregated:

$$H_{\text{cluster}}(C_i) = \text{SHA256}\left(\bigoplus_{e \in C_i} \text{CID}(e)\right) \tag{3}$$

Algorithm 2 Experience Retrieval with Proof

```
1: Input: Query $q$, library root $r$, top-$k$
2: Output: Experiences $\{e_i\}$, proofs $\{\pi_i\}$
3: {Phase 1: Cluster Selection}
4: $q_v \leftarrow \text{Embed}(q)$
5: $\{C_{i^*}\} \leftarrow \text{TopClusters}(q_v, \{\mu_i\}, \sqrt{k})$
6: {Phase 2: Local Search}
7: $\mathcal{E}_{\text{cand}} \leftarrow \bigcup_{C \in \{C_{i^*}\}} C$
8: $\{e_i\}_{i=1}^{k} \leftarrow \text{TopK}(\mathcal{E}_{\text{cand}}, q_v, k)$
9: {Phase 3: Proof Generation}
10: for $e_i \in \{e_i\}$ do
11:     $\pi_i \leftarrow \text{GenerateMerkleProof}(e_i, r)$
12: end for
13: return $\{e_i\}, \{\pi_i\}$
```

where $\bigoplus$ denotes sorted concatenation.

$$H_{\text{parent}}(H_1, H_2) = \text{SHA256}(H_1 \| H_2) \tag{4}$$

The root hash r commits to the entire library:

$$r = \text{MerkleRoot}(\{H_{\text{cluster}}(C_i)\}_{i=1}^{k}) \tag{5}$$

4.5 Proof Structure

A Merkle proof π for experience e consists of:

$$\pi = (e, i, \{h_j, \text{side}_j\}_{j=1}^{\lceil \log_2 k \rceil}) \tag{6}$$

where h_j are sibling hashes and $\text{side}_j \in \{\text{L}, \text{R}\}$ indicates position.

Theorem 4.1 (Proof Verification)**.** *Given root r, experience e, and proof π, verification succeeds iff:*

$$\textit{VerifyProof}(r, e, \pi) = ⊮[\textit{ReconstructRoot}(\textit{CID}(e), \pi) = r] \tag{7}$$

5 Retrieval Protocol

5.1 Semantic Query Processing

Given query q, retrieval proceeds in three phases:

5.2 Complexity Analysis

Theorem 5.1 (Retrieval Complexity)**.** *For a library of n experiences with k clusters, retrieval has:*

- *Time complexity: $O(\sqrt{n} + k \log n)$*
- *Space complexity: $O(k \log n)$ for proofs*

Proof. Cluster selection: $O(k)$ centroid comparisons. Local search: $O(n/k)$ per cluster, $\sqrt{k}$ clusters $= O(\sqrt{n})$. Proof generation: $O(\log k)$ per experience. □

Layer	Purpose	Latency	Cost
On-chain	Root verification	1-10s	High
IPFS	Active storage	100-500ms	Low
Arweave	Permanent archive	1-5s	Medium

Table 1: Storage layer characteristics

5.3 Verified Retrieval

Clients can verify retrieved experiences without trusting the server:

Listing 2: Verified Retrieval Protocol

```
# Client side
root = fetch_root_from_blockchain()
results = query_server(q, k)
for (exp, proof) in results:
    assert verify_merkle_proof(root, exp, proof)
    assert verify_cid(exp) == exp.cid
    # exp is now trusted
```

6 Storage Architecture

6.1 Three-Layer Design

The Experience Ledger uses a three-layer storage architecture:

6.2 IPFS Integration

Experiences are stored as IPLD (InterPlanetary Linked Data) nodes:

Listing 3: IPLD Experience Node

```
{
  "Data": <CBOR-encoded experience>,
  "Links": [
    {"Name": "embedding", "Cid": "bafy..."},
    {"Name": "metadata", "Cid": "bafy..."}
  ]
}
```

6.3 Arweave Permanence

Critical library snapshots are archived to Arweave:

$$\text{Cost}_{\text{Arweave}}(n) = 0.001 \times n \text{ bytes} \times \text{AR/GB} \quad (8)$$

At current rates ($15/GB for 200+ years), archiving 10,000 experiences costs approximately $0.50.

System	Recall@5	Recall@10	Latency	Verifiable
pgvector	91.2%	95.7%	45ms	No
Pinecone	93.8%	97.1%	23ms	No
Raw IPFS	89.4%	93.2%	340ms	No
Exp. Ledger	**94.7%**	**97.9%**	**87ms**	**Yes**

Table 2: Retrieval performance comparison. Experience Ledger provides cryptographic verification with competitive latency.

System	Total Size	Per Experience	Dedup
pgvector	847 MB	80 KB	0%
Pinecone	923 MB	87 KB	0%
Exp. Ledger	**94 MB**	**8.9 KB**	**23%**

Table 3: Storage efficiency. Content-addressing and deduplication achieve 89% reduction.

6.4 Content Deduplication

Content-addressing provides automatic deduplication:

Theorem 6.1 (Deduplication Savings)**.** *For n experiences with d duplicates, storage cost is $O((n-d)\cdot s)$ where s is average experience size.*

In practice, we observe 15-30% deduplication across experience libraries due to common reasoning patterns.

7 Experimental Evaluation

7.1 Experimental Setup

Dataset: 10,847 experiences extracted from mathematical reasoning tasks (AIME, MATH, GSM8K).

Hardware: 3-node IPFS cluster (8 vCPU, 32GB RAM each), connected via 1Gbps network.

Baselines:

- PostgreSQL with pgvector extension
- Pinecone (managed vector database)
- Raw IPFS without Merkle structure

Operation	Time (ms)	Size (bytes)
CID verification	0.3	36
Merkle proof gen	2.1	512
Merkle proof verify	0.8	–
Full retrieval + verify	90.2	548

Table 4: Verification overhead. Cryptographic proofs add minimal latency.

Figure 2: Scalability comparison. Experience Ledger scales sub-linearly due to semantic clustering.

7.2 Retrieval Performance

7.3 Storage Efficiency

7.4 Verification Overhead

7.5 Scalability

8 Security Analysis

8.1 Threat Model

We consider adversaries who may:

1. Tamper with stored experiences
2. Return incorrect retrieval results
3. Attempt Sybil attacks on the network
4. Perform denial-of-service attacks

8.2 Security Properties

Theorem 8.1 (Tamper Evidence). *Any modification to an experience e produces a different CID with probability* $1 - 2^{-256}$.

Theorem 8.2 (Retrieval Integrity). *With Merkle proofs, a client can detect server-returned experiences not in the committed library with probability* $1 - 2^{-256}$.

Theorem 8.3 (Availability). *With n nodes pinning the library and at most $f < n/2$ Byzantine failures, the library remains available.*

9 Related Work

Decentralized Storage: IPFS [1], Filecoin [5], Arweave [6] provide content-addressable storage but lack semantic indexing.

Vector Databases: Pinecone, Weaviate, Milvus optimize for similarity search but lack cryptographic verification.

Blockchain + ML: Federated learning on blockchain [7] focuses on model training, not knowledge sharing.

Knowledge Graphs: Traditional KGs (Freebase, Wikidata) store factual triples, not procedural reasoning patterns.

10 Conclusion

The Zoo Experience Ledger establishes a new paradigm for AI knowledge sharing: content-addressable semantic memory with cryptographic verification. By combining IPFS's content-addressing with Merkle tree proofs and semantic clustering, we enable trustless retrieval of AI reasoning patterns across organizational boundaries.

Our key contributions are:

1. A rigorous experience format with content-addressable identifiers
2. A hybrid Merkle-DAG structure enabling both verification and efficient retrieval
3. Experimental validation showing 94.7% retrieval accuracy with 89% storage savings

The Experience Ledger forms the foundation for Zoo's Decentralized Semantic Optimization protocol, enabling AI agents worldwide to share wisdom without sharing weights.

Acknowledgments

This work was supported by Zoo Labs Foundation Inc, a 501(c)(3) non-profit dedicated to decentralized AI. We thank Protocol Labs for IPFS infrastructure and the Arweave team for permanent storage solutions.

References

[1] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," *arXiv:1407.3561*, 2014.

[2] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS*, 2020.

[3] A. Graves et al., "Hybrid Computing Using a Neural Network with Dynamic External Memory," *Nature*, 2016.

[4] Pinecone Systems, "Vector Database for Machine Learning," Technical Report, 2021.

[5] Protocol Labs, "Filecoin: A Decentralized Storage Network," Whitepaper, 2017.

[6] S. Williams et al., "Arweave: A Protocol for Economically Sustainable Information Permanence," 2019.

[7] A. Kurtulmus and K. Daniel, "Trustless Machine Learning Contracts," *arXiv:1802.10185*, 2018.

A Experience Format JSON Schema

Listing 4: Complete JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["version", "type", "content", "embedding"],
  "properties": {
    "version": {"type": "integer", "const": 1},
    "type": {"type": "string", "const": "experience/v1"},
    "content": {
      "type": "object",
      "required": ["text", "domain", "confidence"],
      "properties": {
        "text": {
          "type": "string",
          "maxLength": 1024,
          "pattern": "^(When|For|If|Before|After).*"
        },
        "domain": {
          "type": "array",
          "items": {"type": "string"},
          "minItems": 1,
          "maxItems": 5
        },
        "confidence": {
          "type": "number",
          "minimum": 0,
          "maximum": 1
        }
      }
    },
    "embedding": {
      "type": "array",
      "items": {"type": "number"},
      "minItems": 768,
      "maxItems": 768
    },
    "metadata": {
      "type": "object",
      "properties": {
        "created": {"type": "integer"},
        "author": {"type": "string"},
        "source": {"type": "string"}
      }
    },
    "signature": {"type": "string"}
  }
}
```

B IPLD Schema

Listing 5: IPLD Schema Definition

```
type Experience struct {
  version Int
  content Content
  embedding &Embedding
  metadata optional Metadata
  signature optional Bytes
}

type Content struct {
  text String
  domain [String]
  confidence Float
}

type Embedding [768]Float

type Metadata struct {
  created Int
  author String
  source String
}
```