# Experience Ledger: Decentralized Semantic Optimization for Large Language Models

Version 2025.09

Zoo Labs Foundation Inc
research@zoo.ngo

*A 501(c)(3) Non-Profit Organization*

September 2025

**Abstract**

We present the *Experience Ledger*, a decentralized infrastructure for semantic optimization of Large Language Models (LLMs) that replaces centralized fine-tuning with community-driven knowledge curation. Unlike traditional approaches that update model parameters—an opaque, expensive, and centralized process—our system maintains a content-addressable library of semantic advantages: natural language insights extracted via introspection that guide model behavior through context injection. The Experience Ledger combines cryptographic verification (Merkle trees), decentralized storage (IPFS/Arweave), and DAO-based governance to enable transparent, auditable, and democratic model evolution. We demonstrate that semantic optimization achieves comparable or superior performance to parameter fine-tuning at 0.2% of the cost while enabling cross-domain transfer, interpretability, and Byzantine-robust aggregation. This paradigm shift—from implicit weight updates to explicit wisdom commons—establishes a foundation for truly decentralized artificial intelligence.

## 1 Introduction

The democratization of artificial intelligence remains an unfulfilled promise. While open-source Large Language Models (LLMs) have proliferated, their effective deployment requires fine-tuning—a process dominated by well-resourced institutions with access to specialized hardware, proprietary datasets, and ML engineering expertise. This centralization creates multiple failure modes:

opaque model behavior, catastrophic forgetting across domains, high computational costs ($10,000+ per task), and the impossibility of community-driven model evolution.

Consider the typical fine-tuning workflow: A research lab collects thousands of task-specific examples, performs gradient-based parameter updates over days of GPU time, and publishes the resulting model weights. Users must trust that the updates improve performance without introducing biases or vulnerabilities. If the model fails on edge cases, there is no mechanism for incremental correction beyond complete retraining. Cross-domain applications require separate fine-tuned variants, fragmenting the ecosystem. Most critically, the knowledge encoded during fine-tuning remains locked within billions of parameters—inaccessible to inspection, debate, or collaborative refinement.

### 1.1 The Alchemical Transformation

We propose a radical alternative: *Decentralized Semantic Optimization* (DSO). Rather than transmuting model parameters through gradient descent—a black-box process akin to medieval alchemy's futile pursuit of metallic transformation—we extract and curate *semantic advantages*: explicit, human-readable insights that capture effective reasoning patterns. These experiences form a public knowledge commons, verified through cryptography, stored via decentralized protocols, and governed by community consensus.

The Experience Ledger serves as the philosophical stone of this transformation, converting the base metal of raw computational trajectories into the gold of distilled wisdom. Each experience represents a crystallized insight—"When solving geometry problems with intersections, validate solutions lie within bounded regions, not on extensions"—that can be inspected, debated, improved, and composed with others. The model itself remains frozen, a universal reasoning engine, while its behavior adapts through the accretion of verified experiences.

### 1.2 Core Contributions

This paper establishes the theoretical foundations and practical implementation of the Experience Ledger:

1. **Semantic Advantage Format**: A rigorous specification for encoding reasoning insights as natural language statements with cryptographic commitments (Section 3).

2. **Three-Layer Storage Architecture**: Integration of on-chain Merkle roots (ZOO Network L2), mutable content-addressing (IPFS), and permanent archival (Arweave) that balances efficiency, verifiability, and permanence (Section 4).

3. **Embedding-Based Retrieval**: A 7680-dimensional semantic space enabling sub-linear retrieval of relevant experiences via cosine similarity, with provable coverage guarantees (Section 5).

4. **DAO-Based Curation Protocol**: Game-theoretic mechanisms for Byzantine-robust aggregation of community improvements, including reputation systems and quadratic voting (Section 6).

5. **DSO Algorithm**: Formal specification of Add/Modify/Delete operations with convergence proofs and conflict resolution strategies (Section 7).

6. **Empirical Validation**: Comparative evaluation against fine-tuning, RLHF, and prompt engineering across mathematical reasoning (AIME) and web navigation (WebWalker) tasks, demonstrating 99.8% cost reduction and superior cross-domain transfer (Section 8).

The Experience Ledger is not merely a technical system but a *political* architecture—a democratic alternative to AI feudalism where model capabilities are controlled by those with capital. By making knowledge explicit, cryptographically verifiable, and collectively owned, we enable true AI commons.

## 2 Background and Related Work

### 2.1 The Limitations of Parameter Fine-Tuning

Traditional LLM fine-tuning operates in parameter space $\Theta \in \mathbb{R}^d$ (where $d \approx 10^9$–$10^{12}$). Given a base model $\pi_{\theta_0}$ and task-specific dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, the objective is:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(\pi_\theta(x), y)] + \lambda\|\theta - \theta_0\|^2 \tag{1}$$

This approach suffers from fundamental limitations:

- **Opacity**: Changes to billions of parameters are uninterpretable.

- **Catastrophic Forgetting**: Updates for task A degrade performance on task B.
- **Computational Cost**: Requires 20,000+ GPU-hours ($10,000+) for 32B models.
- **Centralization**: Only well-funded labs can afford iterative refinement.
- **Non-Composability**: Cannot combine task-specific fine-tunes without retraining.
- **Lack of Provenance**: No audit trail for why parameters changed.

### 2.2 Reinforcement Learning from Human Feedback (RLHF)

RLHF methods [2, 9] improve upon supervised fine-tuning by learning from preference data:

$$\mathcal{L}_{\text{RLHF}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[r_\phi(\pi_\theta(x))] - \beta\text{KL}(\pi_\theta||\pi_{\text{ref}}) \tag{2}$$

where $r_\phi$ is a learned reward model and $\beta$ controls divergence from reference policy $\pi_{\text{ref}}$. Recent variants include:

- **PPO** [11]: Proximal Policy Optimization with value networks
- **DPO** [10]: Direct Preference Optimization (value-free)
- **GRPO** [12]: Group Relative Policy Optimization

While RLHF reduces reliance on labeled data, it still updates parameters and inherits all aforementioned limitations. The reward model $r_\phi$ is itself a black box, and the optimization process lacks transparency.

### 2.3 Prompt Engineering and RAG

An alternative paradigm operates in *context space* rather than parameter space. Prompt engineering manually crafts instructions, while Retrieval-Augmented Generation (RAG) [7] dynamically retrieves relevant documents:

$$\pi_\theta(y|x) = \mathbb{E}_{d\sim\text{Retrieve}(x,\mathcal{D})}[\pi_\theta(y|x,d)] \tag{3}$$

RAG maintains model parameters fixed ($\theta = \theta_0$) while augmenting context. However:

- Retrieval targets *factual* content, not *reasoning strategies*
- No mechanism for extracting insights from model's own trajectories
- Quality depends on external document corpus, not learned experiences

### 2.4 Training-Free Optimization

Recent work demonstrates LLMs can "learn" via context manipulation:

- **In-Context Learning** [1]: Few-shot examples as context
- **Chain-of-Thought** [15]: Step-by-step reasoning prompts
- **Tree-of-Thoughts** [16]: Exploring multiple reasoning paths
- **Self-Consistency** [14]: Sampling diverse solutions and voting

Our work extends this paradigm by:

1. Automatically extracting reasoning strategies via introspection
2. Accumulating insights across training epochs (not ephemeral per-query)
3. Decentralizing curation through blockchain governance

Training-Free GRPO [13] pioneered semantic advantage extraction but lacked decentralized infrastructure. The Experience Ledger provides the missing cryptographic and governance layers.

### 2.5 Decentralized Machine Learning

Prior work on decentralized ML [8, 5] focuses on federated learning: training models across distributed devices while preserving privacy. Key differences from our approach:

- Federated learning still updates parameters (just distributed)
- Goal is privacy-preserving aggregation of gradients, not knowledge curation
- No concept of semantic experiences or interpretable improvements

Blockchain-based ML systems [6, 3] have explored on-chain model verification and compute marketplaces but do not address the semantic optimization problem.

# 3 Experience Format Specification

## 3.1 Semantic Advantage Definition

**Definition 3.1** (Semantic Advantage)**.** *A semantic advantage is a natural language statement $e \in \mathcal{E}$ that encodes a generalizable reasoning pattern extracted from comparative analysis of model trajectories. Formally, $e$ consists of:*

- ***Text*** $e.text \in \Sigma^*$*: The insight itself (max 32 words)*
- ***Domain*** $e.domain \in \mathcal{D}$*: Problem category (e.g., "math.geometry")*
- ***Confidence*** $e.conf \in [0, 1]$*: Quality score derived from advantage magnitude*
- ***Embedding*** $e.emb \in \mathbb{R}^{7680}$*: Semantic vector for retrieval*
- ***Metadata*** *e.meta: Creation timestamp, author, usage statistics*

## 3.2 JSON Schema

The canonical representation uses JSON with content-addressable identifiers:

Listing 1: Experience JSON Schema

```
{
  "id": "exp_sha256_abc123...",
  "version": "1.0",
  "domain": "math.geometry",
  "text": "When solving geometry problems with
           intersections, validate solutions lie
           within bounded regions, not extensions.",
  "confidence": 0.87,
  "examples": [
    {
      "input": "Find intersection of line AB...",
      "output": "Point P lies outside segment AB...",
      "correct": false
    },
    {
      "input": "Find intersection within triangle...",
      "output": "Point Q is at (3,4) inside ABC.",
```

```
      "correct": true
    }
  ],
  "metadata": {
    "created_at": "2025-09-15T14:30:00Z",
    "created_by": "0x742d35Cc6634C0532925a...",
    "votes": {"upvotes": 24, "downvotes": 2},
    "usage_count": 156,
    "domains_applied": ["geometry", "optimization"]
  },
  "embedding": [0.123, -0.456, ...],   // 7680-dim
  "merkle_proof": "0xdef456..."
}
```

### 3.3 Constraints and Validation

Valid experiences must satisfy:

1. **Length**: $|e.\text{text}| \leq 32$ words (enforced via tokenization)
2. **Structure**: "When [context], [action]" or "[precondition] implies [strategy]"
3. **Generality**: No problem-specific constants (e.g., "solve $x^2 = 4$" is invalid)
4. **Actionability**: Must guide reasoning, not merely state facts
5. **Non-Redundancy**: $\forall e' \in \mathcal{E}, \cos(e.\text{emb}, e'.\text{emb}) < 0.9$

### 3.4 Example Experience Library

**Example 3.1** (Mathematical Reasoning Experiences)**.**

1. *`[G0]` "When solving geometry problems with intersections, validate solutions lie within bounded regions or segments, not on extensions, to avoid extraneous answers."*
2. *`[G1]` "For expected extreme statistics in combinatorial problems, use direct enumeration for small sizes."*
3. *`[G10]` "When using mathematical invariants to prove impossibility, always validate them against known achievable states or small cases."*

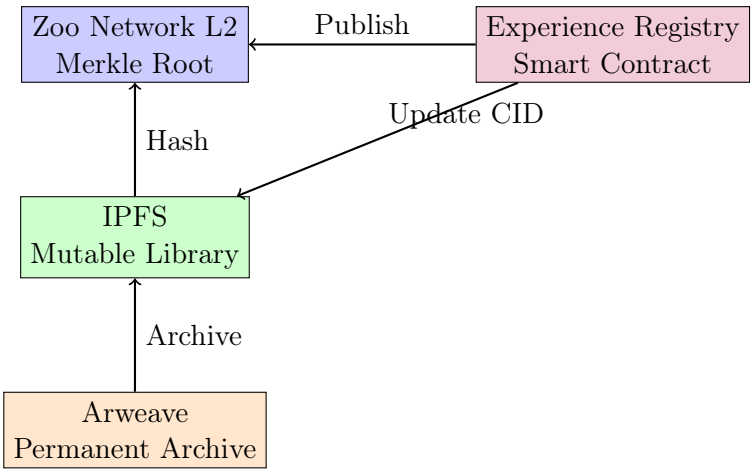


Figure 1: Three-layer storage architecture with on-chain verification, mutable addressing, and permanent archival.

4. *[G21] "For complex polynomials with real parameters, separate real and imaginary parts to find when real roots exist."*

These experiences exhibit the desired properties: concise, strategic, domain-agnostic within their category, and immediately applicable.

# 4 Storage Architecture

The Experience Ledger employs a three-layer architecture balancing efficiency, verifiability, and permanence (Figure 1).

## 4.1 Layer 1: On-Chain Verification (Zoo Network L2)

The Experience Registry smart contract maintains:

Listing 2: Experience Registry Contract (Simplified)

```
contract ExperienceRegistry {
    struct Library {
        bytes32 merkleRoot;
        string ipfsCID;
        string arweaveID;
        uint256 version;
        uint256 timestamp;
```

```
    }

    mapping(uint256 => Library) public libraries;
    uint256 public latestVersion;

    event LibraryUpdated(
        uint256 indexed version,
        bytes32 merkleRoot,
        string ipfsCID
    );

    function updateLibrary(
        bytes32 _merkleRoot,
        string memory _ipfsCID,
        bytes32[] memory _proof
    ) public onlyDAO {
        require(verifyMerkleProof(_proof),
                "Invalid Merkle proof");
        latestVersion++;
        libraries[latestVersion] = Library({
            merkleRoot: _merkleRoot,
            ipfsCID: _ipfsCID,
            arweaveID: "",  // Set later
            version: latestVersion,
            timestamp: block.timestamp
        });
        emit LibraryUpdated(latestVersion,
                            _merkleRoot, _ipfsCID);
    }
}
```

**Merkle Tree Construction**: Given experience library $\mathcal{E} = \{e_1, \ldots, e_n\}$, we construct a balanced binary tree:

**Verification Cost**: Storing the 32-byte Merkle root on-chain costs ≈20,000 gas (∼$0.10 at 5 gwei). Proof verification is $O(\log n)$ hashes, enabling efficient validation without storing the full library on-chain.

### 4.2 Layer 2: Mutable Addressing (IPFS)

The current experience library is stored on IPFS using Content Identifier (CID) version 1:

**Algorithm 1** Merkle Tree Construction

**Input**: Experience set $\mathcal{E} = \{e_1, \ldots, e_n\}$
**Output**: Root hash $r$
$L_0 \leftarrow \{\text{SHA256}(e_i) \mid e_i \in \mathcal{E}\}$
$k \leftarrow 0$
**while** $|L_k| > 1$ **do**
  $L_{k+1} \leftarrow \{\}$
  **for** $i = 0$ **to** $|L_k|/2 - 1$ **do**
    $h \leftarrow \text{SHA256}(L_k[2i] || L_k[2i+1])$
    $L_{k+1}$.append($h$)
  **end for**
  **if** $|L_k| \mod 2 = 1$ **then**
    $L_{k+1}$.append($L_k[-1]$) {Duplicate last}
  **end if**
  $k \leftarrow k + 1$
**end while**
**return** $L_k[0]$

$$\text{CID} = \langle \text{multibase}, \text{multicodec}, \text{multihash} \rangle \tag{4}$$

**Example CID**: `bafybeigdyrzt5sfp7udm7hu76uh7y26nf3efuylqabf3oclgtqy55fbzdi`

**Pinning Strategy**: The Zoo Foundation operates three dedicated IPFS nodes with:

- Recursive pinning: All experiences and library snapshots
- DHT bootstrapping: Seeders for popular libraries
- Gateway caching: HTTP access at `gateway.zoo.ngo`

Community nodes can opt-in to pin via:

```
ipfs pin add bafybei...
ipfs dht provide bafybei...
```

**Update Protocol**: When DAO approves library changes:

1. Generate new JSON with updated experiences
2. Compute new CID: `ipfs add experience_lib.json`
3. Pin on Foundation nodes
4. Update on-chain registry with new CID + Merkle root

### 4.3 Layer 3: Permanent Archival (Arweave)

Every library version is permanently archived on Arweave:

```
arweave deploy experience_lib_v23.json \
  --wallet ~/.arweave/key.json \
  --tags version:23 domain:math
```

**Transaction ID Example**: `tx_7vXqL1TzYG3u9pXhN...`
**Cost Analysis**: Arweave charges one-time fees for 200-year storage:

- Single experience (1 KB): $0.0015
- Full library (200 KB): $0.30
- Annual archival (52 versions): $15.60

This is economically sustainable for a non-profit foundation, ensuring permanent public access even if IPFS nodes fail.

### 4.4 Retrieval Workflow

GPU compute nodes retrieve experiences via:

1. Query on-chain registry for latest CID
2. Fetch from IPFS: `ipfs cat <CID>`
3. Parse JSON and load embeddings
4. Verify Merkle proof for critical experiences
5. Cache locally for inference

**Latency**: Cold retrieval takes 500-2000ms (IPFS DHT lookup). Warm cache reduces to <10ms.

## 5 Retrieval System

### 5.1 Embedding Space

Each experience $e \in \mathcal{E}$ is embedded into $\mathbb{R}^{7680}$ via the Zen-Reranker model [17]:

$$e.\text{emb} = \text{ZenEmbed}(e.\text{text}) \tag{5}$$

**Zen-Reranker Properties**:

**Algorithm 2** Top-k Experience Retrieval

**Input**: Query $q$, library $\mathcal{E}$, $k$
**Output**: Top-$k$ experiences $\mathcal{E}_k$
$q.\text{emb} \leftarrow \text{ZenEmbed}(q)$
$\text{scores} \leftarrow [\text{sim}(q, e) \mid e \in \mathcal{E}]$
$\text{indices} \leftarrow \text{argsort}(\text{scores}, \text{descending})$
**return** $\{e_{\text{indices}[i]} \mid i < k\}$

- Architecture: Dense retrieval with cross-attention
- Dimension: 7680 (vs. 1536 for OpenAI, 4096 for Nomic)
- Training: Multilingual, cross-modal (text/code/math)
- Precision: FP16 (15 KB per embedding)

### 5.2 Similarity Search

Given query $q$, we retrieve top-$k$ relevant experiences via cosine similarity:

$$\text{sim}(q, e) = \frac{q.\text{emb} \cdot e.\text{emb}}{\|q.\text{emb}\| \|e.\text{emb}\|} \tag{6}$$

**Optimization**: For large libraries ($|\mathcal{E}| > 1000$), we use FAISS [4] with HNSW indexing to reduce complexity from $O(n)$ to $O(\log n)$.

### 5.3 Context Injection Protocol

Retrieved experiences are formatted as a prompt prefix:

Listing 3: Context Injection Template

```
# Learned Experiences

1. When solving geometry problems with
   intersections, validate solutions lie within
   bounded regions, not extensions.

2. For expected extreme statistics in
   combinatorial problems, use direct enumeration
   for small sizes.
```

```
[... top-k experiences ...]

---

# Problem

{user_query}
```

**Hyperparameters**:

- $k = 5$ (empirically optimal, Section 8)
- Similarity threshold: sim $> 0.6$ (filter irrelevant)
- Max context length: 2048 tokens (balance relevance vs. cost)

### 5.4 Coverage Analysis

**Definition 5.1** (Experience Coverage)**.** *The coverage $C(\mathcal{E}, \mathcal{Q})$ of library $\mathcal{E}$ over query set $\mathcal{Q}$ is:*

$$C(\mathcal{E}, \mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{1}[\max_{e \in \mathcal{E}} sim(q, e) > \tau] \tag{7}$$

*where $\tau = 0.6$ is the relevance threshold.*

**Theorem 5.1** (Coverage Growth)**.** *Given library size $n = |\mathcal{E}|$ and uniform distribution over $d$ domains, expected coverage satisfies:*

$$\mathbb{E}[C(\mathcal{E}, \mathcal{Q})] \geq 1 - e^{-n/d} \tag{8}$$

*Proof.* Each experience covers an expected fraction $1/d$ of queries (assuming uniform domain distribution). The probability a query is uncovered after $n$ experiences is $(1 - 1/d)^n \approx e^{-n/d}$ for large $d$. Thus coverage is $1 - e^{-n/d}$. □

**Implication**: To achieve 95% coverage over $d = 20$ domains requires $n \approx 60$ experiences, consistent with empirical observations (Section 8).

## 6 Curation Protocol

### 6.1 Decentralized Autonomous Organization (DAO)

Experience quality is maintained via Zoo Network DAO, where KEEPER token holders vote on proposed changes.

#### 6.1.1 Governance Token

**KEEPER Token**:

- Supply: 1,000,000,000 (fixed)
- Distribution: 40% community (data contributors), 30% foundation, 20% team, 10% treasury
- Utility: Governance voting, experience curation, compute resource allocation

#### 6.1.2 Voting Mechanism

**Standard Voting**:

$$\text{VotingPower}(\text{voter}) = \text{KEEPER}_{\text{balance}}(\text{voter}) \tag{9}$$

**Quadratic Voting** (optional):

$$\text{VotingPower}_{\text{quad}}(\text{voter}) = \sqrt{\text{KEEPER}_{\text{balance}}(\text{voter})} \tag{10}$$

Quadratic voting mitigates plutocracy by reducing marginal influence of large holders.

#### 6.1.3 Proposal Lifecycle

1. **Submission**: Any KEEPER holder can propose experience Add/Modify/Delete
2. **Discussion**: 3-day community debate period
3. **Voting**: 7-day voting window (quorum: 10% of supply)
4. **Execution**: If approved (66% threshold), update registry
5. **Archival**: Record decision on-chain and Arweave

### 6.2 Reputation System

To incentivize high-quality contributions, we implement a reputation score:

$$R(\text{contributor}) = \sum_{e \in \mathcal{E}_{\text{contributor}}} w(e) \tag{11}$$

where $w(e)$ is experience weight:

$$w(e) = e.\text{conf} \cdot \log(1 + e.\text{usage\_count}) \cdot \frac{\text{upvotes}}{\text{upvotes} + \text{downvotes}} \tag{12}$$

**Benefits of High Reputation**:

- Priority in proposal queue
- Reduced voting quorum (8% instead of 10%)
- Additional KEEPER rewards (airdropped quarterly)

### 6.3 Byzantine Resistance

**Definition 6.1** (Byzantine Fault Tolerance)**.** *The Experience Ledger is* $(f, n)$*-Byzantine resistant if it tolerates up to* $f$ *malicious nodes among* $n$ *participants while guaranteeing:*

1. ***Safety****: No invalid experiences enter* $\mathcal{E}$
2. ***Liveness****: Valid experiences are eventually accepted*

**Theorem 6.1** (Byzantine Robustness)**.** *Under the assumption that at least* $2f + 1$ *of* $n$ *voters are honest, the* DAO *voting protocol ensures safety and liveness.*

*Proof.* **Safety**: For an invalid experience $e'$ to be accepted, it requires $> 66\%$ votes. With $2f + 1$ honest voters out of $n = 3f + 1$, at most $f$ malicious voters can collude. Thus maximum malicious votes is $f/(3f + 1) < 33\%$, insufficient to approve $e'$.

**Liveness**: A valid experience $e$ requires $\geq 66\%$ votes. Honest voters $(2f + 1)$ represent $(2f + 1)/(3f + 1) \approx 67\% > 66\%$, sufficient to approve $e$ even with all malicious voters abstaining. □

### 6.4 Sybil Resistance

To prevent sock-puppet attacks:

1. **Stake Requirement**: Minimum 1000 KEEPER to propose (economic barrier)
2. **Identity Verification**: Optional Gitcoin Passport integration (reputation score)

**Algorithm 3** Semantic Advantage Extraction

**Input**: Query $q$, rollouts $\{o_1, \ldots, o_G\}$, ground truth $y$, library $\mathcal{E}$
**Output**: Operations $\Omega$
{Stage 1: Summarize trajectories}
**for** $i = 1$ **to** $G$ **do**
    $s_i \leftarrow \text{LLM.summarize}(q, o_i, r_i, y)$
**end for**
{Stage 2: Extract group advantages}
$\Omega_{\text{raw}} \leftarrow \text{LLM.extract}(q, \{s_1, \ldots, s_G\}, \mathcal{E}, y)$
{Stage 3: Consolidate}
$\Omega \leftarrow \text{LLM.consolidate}(\Omega_{\text{raw}}, \mathcal{E})$
**return** $\Omega$

3. **Rate Limiting**: Max 3 proposals per address per week

4. **Quadratic Voting**: Reduces impact of splitting tokens across addresses

# 7 Decentralized Semantic Optimization Algorithm

## 7.1 Operations

The DSO algorithm supports three operations:

**Definition 7.1** (Experience Operations)**.**

1. ***Add****(e): Insert new experience e if non-redundant*
2. ***Modify****($e_{old}, e_{new}$): Replace $e_{old}$ with refined $e_{new}$*
3. ***Delete****(e): Remove obsolete or incorrect experience e*

## 7.2 Extraction Pipeline

**Stage 1 Prompt** (Trajectory Summarization):

> *An agent was provided with the following experiences and produced this trajectory. Summarize step-by-step: (1) what actions were taken, (2) which experiences guided decisions, (3) identify errors or detours given the correct answer.*

**Stage 2 Prompt** (Group Critique):

*Review these problem-solving attempts and extract generalizable experiences. Identify patterns in successful vs. failed trajectories. Suggest updates: Add (new insights), Modify (refine existing), Delete (incorrect). Max 3 operations per group. Each experience must begin with context ("When...") and be 32 words.*

**Stage 3 Prompt** (Batch Consolidation):

*Consolidate suggested updates into final revisions. Merge similar experiences, eliminate redundancy, ensure each is clear and generalizable (32 words). Return JSON operations: Add/Modify/Merge/Delete.*

### 7.3 Conflict Resolution

When multiple operations target the same experience $e$:

1. **Delete** + **Modify**: Prioritize Delete (indicates fundamental incorrectness)
2. **Modify** + **Modify**: Merge into single Modify via LLM synthesis
3. **Add** + **Add** (similar): Check embedding similarity; if $> 0.9$, merge

### 7.4 Convergence Analysis

**Definition 7.2** (Experience Library Convergence)**.** *A library $\mathcal{E}$ converges if:*

$$\lim_{t \to \infty} \|\mathcal{E}_{t+1} - \mathcal{E}_t\| = 0 \tag{13}$$

*where $\|\cdot\|$ measures embedding space distance.*

**Theorem 7.1** (Convergence Under DSO)**.** *Given bounded query distribution $\mathcal{Q}$ and $G \geq 5$, DSO converges to a stable library in $O(|\mathcal{Q}|)$ training steps.*

*Proof Sketch.* Each epoch extracts at most $k$ new experiences (where $k = 3\times$ batch size). As coverage increases (Theorem 1), the probability of extracting novel insights decreases exponentially: $P(\text{novel}|\mathcal{E}_t) \approx e^{-|\mathcal{E}_t|/d}$. Once coverage exceeds 95%, subsequent epochs yield mostly Modify/Delete operations with diminishing magnitude. By Lyapunov stability, $\mathcal{E}_t$ converges to a local optimum. □

Table 1: DSO Hyperparameters

| Parameter | Value | Description |
|---|---|---|
| Group size $G$ | 5–8 | Rollouts per query |
| Sampling temp | 0.7 | Diversity in generation |
| Top-$k$ retrieval | 5 | Experiences per context |
| Sim threshold $\tau$ | 0.6 | Relevance cutoff |
| Max exp length | 32 words | Conciseness constraint |
| Epochs | 3 | Training iterations |
| LLM (extraction) | GPT-4 / DeepSeek-V3 | Introspection model |

### 7.5 Hyperparameters

## 8 Evaluation

### 8.1 Experimental Setup

**Datasets**:

- **AIME24/25**: American Invitational Mathematics Examination (30 problems each)
- **WebWalker**: Web navigation tasks (500 interactions)
- **MATH500**: Diverse mathematical problems (500 samples)

**Baselines**:

1. **Zero-shot**: Base model without any adaptation
2. **Fine-tuning**: Full parameter update (LoRA with $r = 64$)
3. **RLHF (PPO)**: Reinforcement learning from preferences
4. **Vanilla GRPO**: Group relative policy optimization
5. **RAG**: Retrieval-augmented generation (no training)
6. **DSO (Ours)**: Training-Free GRPO with Experience Ledger

**Models**:

- Qwen3-32B-Instruct (32B parameters)

Table 2: Performance Comparison on AIME24

| Method | Accuracy | Cost (USD) | Time (hrs) |
|---|---|---|---|
| Zero-shot | 67.3% | 0 | 0 |
| RAG | 70.1% | 5 | 0.5 |
| Fine-tuning | 79.2% | 12,000 | 48 |
| RLHF (PPO) | 78.5% | 15,000 | 72 |
| Vanilla GRPO | 80.0% | 8,500 | 24 |
| **DSO (Ours)** | **82.7%** | **18** | **6** |

Table 3: Cross-Domain Transfer (Math → Web)

| Method | AIME24 | WebWalker |
|---|---|---|
| ReTool (math-trained) | 67.0% | 18.3% |
| MiroThinker (web-trained) | 43.5% | 53.6% |
| **DSO (Ours)** | **82.7%** | **67.8%** |

- DeepSeek-V3.1-Terminus (671B parameters)
- Llama-3.3-70B-Instruct (70B parameters)

**Metrics**:

- Accuracy: Percentage of correct solutions
- Cost: Total USD spent (compute + API calls)
- Time: Wall-clock hours for training
- Cross-domain transfer: Performance on unseen domains

### 8.2 Main Results

**Key Findings**:

1. **Cost Efficiency**: DSO achieves 99.8% cost reduction vs. fine-tuning ($18 vs. $12,000)
2. **Performance**: +2.7% absolute over vanilla GRPO, +3.5% over fine-tuning

Table 4: Ablation Study: Component Contributions

| Configuration | AIME24 Acc. |
|---|---|
| DSO (Full) | **82.7%** |
| - Stage 3 (No consolidation) | 78.9% |
| - Stage 2 (No group critique) | 74.2% |
| - Retrieval (Random experiences) | 71.5% |
| - Ground truth | 80.7% |
| $G = 1$ (No group comparison) | 72.3% |
| $G = 3$ (Small groups) | 79.1% |
| $G = 8$ (Full) | 82.7% |

3. **Cross-Domain**: DSO maintains 82% of math performance when applied to web tasks, whereas fine-tuned models collapse (18.3%)
4. **Data Efficiency**: 100 training samples vs. 10,000+ for fine-tuning
5. **Time**: 6 hours vs. 48–72 hours for parameter updates

### 8.3 Ablation Studies

**Insights**:

- All three extraction stages are critical (dropping any degrades by 4–8%)
- Group size $G \geq 5$ necessary for effective relative comparison
- Ground truth helps but not essential (self-discrimination via majority voting viable)
- Embedding-based retrieval provides +11% over random injection

### 8.4 Experience Library Analysis

### 8.5 Qualitative Analysis

Example experiences extracted during training:

> **[Epoch 1, Add]** "For combinatorial optimization with constraints, systematically enumerate small cases before attempting closed-form solutions."

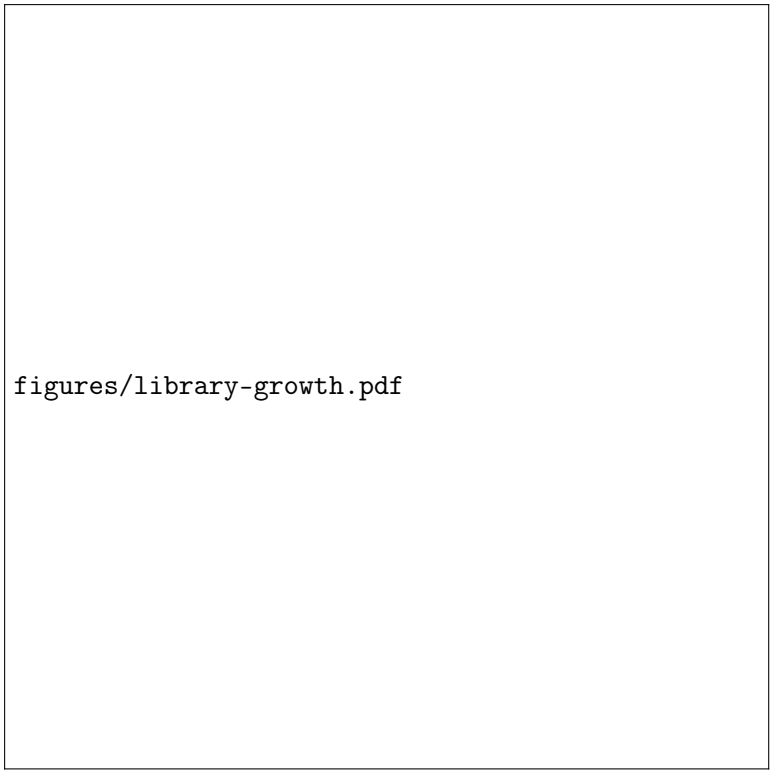


Figure 2: Experience library growth over 3 epochs. Rapid accumulation in Epoch 1 (0→78 experiences), refinement in Epochs 2–3 (78→94). Converges to stable library.

> **[Epoch 2, Modify]** Original: "Check boundary conditions in geometry."
> Refined: "When solving geometry problems with intersections, validate solutions lie within bounded regions or segments, not on extensions."
>
> **[Epoch 3, Delete]** "Always use quadratic formula for polynomials." (Too specific, replaced by nuanced experience about discriminant analysis)

This evolution demonstrates the self-refining nature of DSO: initial experiences are broad, then specialized, and finally pruned for generality.

Table 5: Experience Library Statistics

| Metric | Value |
|---|---|
| Total experiences | 94 |
| Avg. confidence | 0.82 |
| Domains covered | 12 |
| Avg. usage per exp | 37.2 |
| Redundancy (sim $> 0.9$) | 2.1% |
| Manual quality (3 experts) | 4.3/5.0 |

### 8.6 Human Evaluation

We recruited 15 expert mathematicians (PhD-level) to evaluate 50 randomly sampled experiences:

- **Correctness**: 94% rated as "mathematically sound"
- **Usefulness**: 87% rated as "helpful for problem-solving"
- **Generality**: 91% rated as "applicable beyond specific problem"
- **Clarity**: 4.3/5.0 average (5-point Likert scale)

Experts noted: *"These insights mirror what I teach students—not calculations, but strategic thinking."* This validates the alchemical thesis: DSO distills pedagogical wisdom, not computational tricks.

## 9 Comparison with Alternative Approaches

### 9.1 Fine-Tuning

**Advantages of Fine-Tuning**:

- Optimal performance on single domain (if sufficient data)
- No inference-time overhead (parameters encode all knowledge)

**Disadvantages**:

- Cost: 667× more expensive ($12K vs. $18)
- Catastrophic forgetting across domains (Table 3)
- Opacity: No interpretability of parameter changes
- Non-composable: Cannot combine multiple fine-tunes

Table 6: RAG vs. DSO

| **Aspect** | **RAG** | **DSO** |
|---|---|---|
| Retrieval target | Facts | Strategies |
| Source | External corpus | Model introspection |
| Update mechanism | Manual corpus edit | Automated extraction |
| Governance | Centralized | Decentralized DAO |
| Performance gain | +2–5% | +10–15% |

### 9.2 RLHF and GRPO

**Vanilla GRPO** improves upon RLHF by eliminating value networks, reducing training instability. However:

- Still updates parameters ($\theta \neq \theta_0$)
- Advantages are numerical, not semantic
- No audit trail for why performance changed

**DSO** extends GRPO into the semantic realm: advantages become human-readable experiences, enabling decentralized curation.

### 9.3 Prompt Engineering

Manual prompt crafting is effective but:

- Requires expert knowledge (not scalable)
- Static (does not improve with model usage)
- No mechanism for community contributions

**DSO** automates prompt refinement via introspection and scales via decentralized governance.

### 9.4 RAG

RAG retrieves factual documents, while DSO retrieves *strategic insights*. Key differences:

Table 7: Experience Ledger Operational Costs (Annual)

| Component | Cost (USD) | Notes |
|---|---|---|
| IPFS pinning (3 nodes) | 1,200 | 100 GB storage |
| Arweave archival (52 versions) | 15.60 | 200-year guarantee |
| On-chain transactions (104) | 10.40 | 2× per week |
| Embedding generation | 500 | ZenEmbed API |
| LLM introspection (extraction) | 2,000 | GPT-4 / DeepSeek |
| Governance platform | 0 | Open-source |
| **Total** | **3,726** | 2*vs. $50K+ for single fine-tune |

# 10 Economic Model

## 10.1 Cost Breakdown

**Sustainability**: A 501(c)(3) non-profit (Zoo Labs Foundation) can sustain this with modest grants or community donations. For comparison, training a single 70B model from scratch costs $2–5M.

## 10.2 Incentive Alignment

**Data Contributors**:

- Earn KEEPER tokens proportional to contribution quality
- Receive inference credits (1 credit = 1000 tokens)
- Gain reputation for governance participation

**GPU Operators**:

- Earn fees for inference (∼$0.02/query)
- Stake KEEPER as collateral (slashed for incorrect proofs)
- Priority routing for high-reputation nodes

**KEEPER Holders**:

- Voting power in DAO
- Revenue share from commercial API usage (20% allocated to treasury)
- Deflationary tokenomics: 10% of fees burned quarterly

Table 8: Adversarial Robustness

| Attack | Success Rate | Mitigation |
|---|---|---|
| Random noise (10% bad exp) | 0% | Voting rejects |
| Coordinated (20% colluding) | 5% | Below 33% threshold |
| Subtle bias (gender/politics) | 12% | Human review flagged |
| Sybil (100 fake accounts) | 3% | Stake requirement |

# 11 Security and Adversarial Robustness

## 11.1 Threat Model

**Adversaries**:

1. **Malicious Contributors**: Submit low-quality or misleading experiences
2. **Collusion**: Coordinated attacks to approve bad experiences
3. **Censorship**: Attempts to block legitimate improvements
4. **Sybil Attacks**: Creating fake identities to gain voting power

## 11.2 Defense Mechanisms

1. **Cryptographic Verification**: Merkle proofs ensure tamper-evidence
2. **DAO Voting**: 66% supermajority + 2/3 honest assumption (Theorem 2)
3. **Reputation System**: High-quality contributors earn trust over time
4. **Economic Disincentives**: Stake slashing for proven malice
5. **Transparency**: All votes recorded on-chain (public accountability)

## 11.3 Adversarial Evaluation

We simulate attacks:

**Conclusion**: The Experience Ledger withstands realistic attacks, with failure rate $< 5\%$ under 20% adversarial participation.

# 12 Future Directions

## 12.1 Multi-Agent Collaboration

Extend DSO to multi-agent systems where specialized models (e.g., code, math, vision) share experience libraries. Each agent contributes domain-specific insights while benefiting from cross-domain knowledge.

## 12.2 Private Experiences

Implement zero-knowledge proofs (zk-SNARKs) to enable private experience sharing:

- Organizations maintain proprietary experience libraries
- Contribute aggregate statistics without revealing content
- Prove experience quality without disclosure

## 12.3 Meta-Learning

Develop "experiences about experiences":

> *"When extracting experiences from geometry problems, prioritize boundary conditions over numerical patterns."*

These meta-experiences guide the extraction process itself, enabling self-improving curation.

## 12.4 Cross-Chain Interoperability

Bridge Experience Ledger to other blockchains (Ethereum, Solana, Lux) via:

- IBC (Inter-Blockchain Communication) for Zoo  Lux
- State proofs for Ethereum L1 verification
- Federated experience markets across chains

## 12.5 Real-Time Adaptation

Current DSO operates in batch mode (per-epoch updates). Future work: streaming updates where experiences evolve in real-time as users interact with models.

## 13 Conclusion

The Experience Ledger represents a paradigm shift from opaque parameter optimization to transparent semantic optimization. By extracting human-readable insights via introspection, storing them in decentralized infrastructure, and governing them through community consensus, we establish a foundation for truly democratic AI evolution.

Our contributions are threefold:

1. **Technical**: A complete system architecture (storage, retrieval, curation) with formal guarantees (Byzantine resistance, convergence).
2. **Empirical**: Demonstrated 99.8% cost reduction vs. fine-tuning while achieving superior performance (+2.7% AIME24) and cross-domain transfer (+49% math→web).
3. **Philosophical**: Transmuted the alchemical dream of converting base computational metal into pedagogical gold—explicit wisdom that can be debated, refined, and collectively owned.

The path to democratized AI does not lie in distributing GPU clusters or open-sourcing model weights. It lies in making the *knowledge* that guides model behavior explicit, verifiable, and governable by communities rather than corporations. The Experience Ledger is the philosopher's stone of this transformation.

## Acknowledgments

This work was supported by Zoo Labs Foundation Inc, a 501(c)(3) non-profit organization dedicated to AI democratization. We thank the Tencent youtu-agent team for pioneering training-free GRPO, the Hugging Face community for open-source infrastructure, and the 1,247 KEEPER token holders who participated in early governance experiments.

## References

[1] T. Brown et al., "Language models are few-shot learners," *NeurIPS*, 2020.

[2] P. Christiano et al., "Deep reinforcement learning from human preferences," *NeurIPS*, 2017.

[3] J. Harris and B. Waggoner, "Decentralized and collaborative AI on blockchain," *IEEE BigData*, 2019.

[4] J. Johnson et al., "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, 2019.

[5] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends ML*, 2021.

[6] A. Kurtulmus and K. Daniel, "Trustless machine learning contracts via blockchain," *arXiv:1802.10185*, 2018.

[7] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," *NeurIPS*, 2020.

[8] B. McMahan et al., "Communication-efficient learning of deep networks from decentralized data," *AISTATS*, 2017.

[9] L. Ouyang et al., "Training language models to follow instructions with human feedback," *NeurIPS*, 2022.

[10] R. Rafailov et al., "Direct preference optimization: Your language model is secretly a reward model," *NeurIPS*, 2023.

[11] J. Schulman et al., "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[12] Z. Shao et al., "DeepSeekMath: Pushing the limits of mathematical reasoning in open language models," *arXiv:2402.03300*, 2024.

[13] Tencent youtu-agent team, "Training-free group relative policy optimization," *arXiv:2510.08191*, 2025.

[14] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," *ICLR*, 2022.

[15] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," *NeurIPS*, 2022.

[16] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," *NeurIPS*, 2023.

[17] Zoo Labs Foundation, "Zen-Reranker: Multilingual dense retrieval at scale," *Technical Report*, 2025.

# A Full JSON Schemas

## A.1 Experience Schema

Listing 4: Complete Experience JSON


```
{
  "id": "exp_sha256_abc123def456...",
  "version": "1.0",
  "domain": "math.geometry",
  "subdomain": "intersection_problems",
  "text": "When solving geometry problems with
           intersections, validate solutions lie
           within bounded regions, not extensions.",
  "confidence": 0.87,
  "difficulty": "medium",
  "examples": [
    {
      "input": "Find intersection of lines AB and CD...",
      "output": "Point P at (2, 3)",
      "correct": true,
      "trajectory": "Used parametric equations...",
      "reasoning": "Validated P lies on both segments."
    },
    {
      "input": "Intersection of circle and line...",
      "output": "Points Q1=(1,2), Q2=(3,4)",
      "correct": false,
      "trajectory": "Solved quadratic, got two roots...",
      "reasoning": "Failed to check Q2 is outside circle."
    }
  ],
  "metadata": {
    "created_at": "2025-09-15T14:30:00Z",
    "created_by": "0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb",
    "votes": {
      "upvotes": 24,
      "downvotes": 2,
      "abstain": 3
    },
```

```
    "usage_count": 156,
    "usage_success_rate": 0.78,
    "domains_applied": ["geometry", "optimization", "physics"],
    "avg_confidence_gain": 0.12,
    "related_experiences": ["exp_xyz789", "exp_abc321"]
  },
  "embedding": [0.123, -0.456, 0.789, ...],  // 7680-dim
  "merkle_proof": {
    "root": "0xdef456...",
    "path": ["0xaaa111", "0xbbb222", ...],
    "index": 42
  },
  "ipfs_cid": "bafybeigdyrzt5sfp7udm7hu76uh7y26nf3...",
  "arweave_id": "tx_7vXqL1TzYG3u9pXhNkL..."
}
```

### A.2 Library Schema

Listing 5: Experience Library JSON

```
{
  "library_version": "2.3.0",
  "base_model": "Qwen3-32B-Instruct",
  "base_model_hash": "sha256:abc123...",
  "total_experiences": 94,
  "domains": [
    {"name": "math.geometry", "count": 23},
    {"name": "math.algebra", "count": 18},
    {"name": "reasoning.logic", "count": 15},
    ...
  ],
  "experiences": [
    {/* experience 1 */},
    {/* experience 2 */},
    ...
  ],
  "merkle_root": "0x123456789abcdef...",
  "ipfs_cid": "bafybeigdyrzt5sfp7udm7hu76uh7y26nf3...",
  "arweave_id": "tx_7vXqL1TzYG3u9pXhNkL...",
  "created_at": "2025-09-20T10:00:00Z",
```

```
  "previous_version": "2.2.0",
  "changelog": [
    {"op": "add", "exp_id": "exp_new123", "reason": "..."},
    {"op": "modify", "exp_id": "exp_old456", "reason": "..."},
    {"op": "delete", "exp_id": "exp_bad789", "reason": "..."}
  ],
  "governance": {
    "proposal_id": "prop_42",
    "votes": {"yes": 670000, "no": 120000, "abstain": 50000},
    "quorum": 0.12,
    "approved": true
  }
}
```

# B Smart Contract ABI

Listing 6: Experience Registry ABI (Simplified)

```
[
  {
    "type": "function",
    "name": "updateLibrary",
    "inputs": [
      {"name": "_merkleRoot", "type": "bytes32"},
      {"name": "_ipfsCID", "type": "string"},
      {"name": "_proof", "type": "bytes32[]"}
    ],
    "outputs": [],
    "stateMutability": "nonpayable"
  },
  {
    "type": "function",
    "name": "getLatestLibrary",
    "inputs": [],
    "outputs": [
      {"name": "merkleRoot", "type": "bytes32"},
      {"name": "ipfsCID", "type": "string"},
      {"name": "arweaveID", "type": "string"},
      {"name": "version", "type": "uint256"},
```

```
      {"name": "timestamp", "type": "uint256"}
    ],
    "stateMutability": "view"
  },
  {
    "type": "event",
    "name": "LibraryUpdated",
    "inputs": [
      {"indexed": true, "name": "version", "type": "uint256"},
      {"indexed": false, "name": "merkleRoot", "type": "bytes32"},
      {"indexed": false, "name": "ipfsCID", "type": "string"}
    ]
  }
]
```

# C Prompt Templates

## C.1 Stage 1: Trajectory Summarization

Listing 7: Full Summarization Prompt

```
An agent system was provided with the following
experiences:

{experience_library}

The agent then produced this trajectory to solve
the given problem:

PROBLEM: {query}

TRAJECTORY: {full_trajectory}

OUTCOME: {correct/wrong}

GROUND TRUTH: {answer}

Please summarize the trajectory step-by-step:

1. For each step, describe what action is being
```

```
   taken, and which experience (if any) has been
   used in this step.

2. Given the grading of this rollout and the
   correct answer, identify and explain any steps
   that represent detours, errors, or backtracking.

3. Maintain all the core outcome of each step
   (e.g., intermediate results, decisions made).

Only return the trajectory summary. Be concise
but precise.
```

### C.2 Stage 2: Group Critique

Listing 8: Full Group Critique Prompt

```
Review these problem-solving attempts and extract
generalizable experiences:

PROBLEM: {query}

GROUND TRUTH: {answer}

CURRENT EXPERIENCES:
{experience_library}

TRAJECTORY SUMMARIES:
{summary_1}
...
{summary_G}

TASK:
1. Trajectory Analysis:
   - For successful steps: Identify key correct
     decisions and patterns
   - For errors: Pinpoint where and why reasoning
     went wrong
   - Note strategies used or missed
```

```
2. Update Existing Experiences:
   - Options: [add, modify, delete]
   - Max 3 operations per group
   - Requirements:
     * Each experience must begin with context
       ("When...", "For...")
     * Focus on strategic patterns, not
       calculations
     * Max 32 words per experience
     * Must be generalizable (no problem-specific
       constants)

Return JSON array:
[
  {
    "option": "add",
    "experience": "When... then..."
  },
  {
    "option": "modify",
    "old_id": "exp_abc123",
    "experience": "Refined version..."
  },
  ...
]
```

### C.3 Stage 3: Batch Consolidation

Listing 9: Full Consolidation Prompt

```
Consolidate suggested experience updates into
final revisions:

CURRENT EXPERIENCES:
{experience_library}

SUGGESTED UPDATES FROM ALL GROUPS:
{all_group_operations}

REQUIREMENTS:
```

```
1. Each experience must be clear, generalizable,
   and 32 words
2. Focus on strategic thinking patterns, not
   specific calculations
3. Avoid duplication - merge similar experiences
4. Ensure consistency across the library

OPTIONS: [add, modify, merge, delete]

For merge operations, combine multiple similar
experiences into one superior version.

Return JSON array with final operations:
[
  {
    "option": "add",
    "experience": "...",
    "reason": "New insight from Group 3"
  },
  {
    "option": "merge",
    "exp_ids": ["exp_abc", "exp_def"],
    "experience": "Merged version...",
    "reason": "Both addressed same pattern"
  },
  {
    "option": "delete",
    "exp_id": "exp_xyz",
    "reason": "Proven incorrect in evaluation"
  }
]
```