

# JANUS BACKWARD

## Sleep State: Memory Management

*Knowledge Consolidation, Schema Formation, and  
Long-Term Learning*

**Classification: Technical Implementation Guide**

**Version: 1.0 (Implementation-Ready)**

**Author:** Jordan Smith  
*[github.com/nuniesmith](https://github.com/nuniesmith)*

**Date:** December 26, 2025

**JANUS Backward Overview:**

- **Purpose:** Offline memory consolidation and schema learning
- **Cold Path:** Batch processing during market closure
- **Components:** Three-timescale memory, prioritized replay, UMAP visualization
- **Goal:** Transform raw experiences into abstract knowledge structures

## Abstract

JANUS Backward represents the "sleep state" of the JANUS trading system, responsible for offline memory consolidation, schema formation, and long-term learning during market closure. This document provides a comprehensive mathematical and implementation specification for the Backward service, which implements:

- **Three-Timescale Memory Architecture** spanning short-term (hippocampus), medium-term (SWR replay), and long-term (neocortex) storage
- **Prioritized Experience Replay** using TD-error, logical violation scores, and reward magnitude
- **Sharp Wave Ripple Simulation** for time-compressed memory consolidation
- **Recall-Gated Learning** that filters updates based on familiarity and logical validity
- **UMAP Visualization** for real-time cognitive monitoring and anomaly detection

The Backward service operates on a cold path with no strict latency requirements, enabling sophisticated batch processing and offline optimization that would be infeasible during live trading.

# Contents

<b>1</b>	<b>Memory Hierarchy: Three-Timescale Architecture</b>	<b>3</b>
1.1	Short-Term Memory (Hippocampus)	3
1.1.1	Episodic Buffer	3
1.1.2	Pattern Separation	3
1.1.3	Sparse Encoding	4
1.2	Medium-Term Consolidation (SWR Simulator)	4
1.2.1	Replay Prioritization	4
1.2.2	Sampling Probability	5
1.2.3	Importance Sampling Correction	5
1.2.4	Time Compression	5
1.2.5	SWR Replay Algorithm	6
1.3	Long-Term Memory (Neocortex)	6
1.3.1	Schema Representation	6
1.3.2	Schema Assignment	6
1.3.3	Recall-Gated Consolidation	7
1.3.4	Consolidation Update Rule	7
<b>2</b>	<b>UMAP Visualization: Cognitive Dashboard</b>	<b>7</b>
2.1	AlignedUMAP for Schema Formation	7
2.1.1	Objective Function	8
2.1.2	Alignment Weights	8
2.1.3	Schema Cluster Detection	8
2.2	Parametric UMAP for Real-Time Monitoring	8
2.2.1	Neural Network Projection	9
2.2.2	Anomaly Detection	9
<b>3</b>	<b>Integration with Vector Database (Qdrant)</b>	<b>9</b>
3.1	Schema Storage	9
3.2	Similarity Search	10
3.3	Periodic Schema Pruning	10
<b>4</b>	<b>Sleep Cycle: Complete Algorithm</b>	<b>10</b>
<b>5</b>	<b>Implementation Checklist</b>	<b>12</b>
5.1	Core Components	12
5.2	Integration & Storage	13
5.3	Monitoring & Debugging	13

5.4	Performance Optimization . . . . .	14
<b>6</b>	<b>Rust Implementation Considerations</b>	<b>14</b>
6.1	Cold Path Optimization . . . . .	14
6.2	Data Structures . . . . .	15
6.2.1	Episodic Buffer . . . . .	15
6.2.2	Schema Representation . . . . .	16
6.3	Error Handling . . . . .	17

# 1 Memory Hierarchy: Three-Timescale Architecture

The memory system is organized into three distinct timescales, each with specialized functions and computational properties.

## 1.1 Short-Term Memory (Hippocampus)

The hippocampal subsystem provides rapid encoding of recent experiences with pattern separation to prevent interference.

### 1.1.1 Episodic Buffer

The hippocampus maintains an episodic buffer of recent transitions:

$$\mathcal{B}_{\text{STM}} = \{(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})\}_{t=1}^{T_{\text{episode}}} \quad (1)$$

where each tuple represents a state-action-reward-nextstate transition.

#### Implementation Details:

- Maximum capacity:  $|\mathcal{B}_{\text{STM}}| \leq 10,000$  transitions
- FIFO replacement policy when capacity exceeded
- Indexed by timestamp for temporal queries

### 1.1.2 Pattern Separation

To prevent catastrophic interference between similar market states, the hippocampus implements pattern separation:

$$\mathbf{h}_{\text{separated}} = \text{ReLU}(\mathbf{W}_{\text{sep}}\mathbf{s} + \mathbf{b}_{\text{sep}}) \quad (2)$$

where  $\mathbf{W}_{\text{sep}} \in \mathbb{R}^{d_h \times d_s}$  is initialized to promote orthogonality.

#### Orthogonality Initialization:

$$\mathbf{W}_{\text{sep}} \sim \mathcal{N}(0, \sigma^2), \quad \text{where } \sigma = \sqrt{\frac{2}{d_s + d_h}} \quad (3)$$

During training, add orthogonality regularization:

$$\mathcal{L}_{\text{ortho}} = \lambda_{\text{ortho}} \cdot \|\mathbf{W}_{\text{sep}}^\top \mathbf{W}_{\text{sep}} - \mathbf{I}\|_F^2 \quad (4)$$

### 1.1.3 Sparse Encoding

The hippocampus uses sparse representations to maximize information capacity:

$$\mathbf{c}_{\text{sparse}} = \text{TopK}(\mathbf{h}_{\text{separated}}, k) \quad (5)$$

where TopK selects the  $k$  largest activations and zeros others.

**Sparsity Level:**

$$k = \lceil \rho \cdot d_h \rceil, \quad \rho \in [0.05, 0.15] \quad (6)$$

Typically  $\rho = 0.1$  (10% activation).

## 1.2 Medium-Term Consolidation (SWR Simulator)

The Sharp Wave Ripple (SWR) simulator implements prioritized replay with time compression, mimicking biological memory consolidation during sleep.

### 1.2.1 Replay Prioritization

Each transition is assigned a priority score combining three components:

$$p_i = |\delta_i| + \lambda_{\text{logic}} \cdot v_i + \lambda_{\text{reward}} \cdot |r_i| \quad (7)$$

where:

- $\delta_i$  = TD-error:  $r_i + \gamma Q(\mathbf{s}_{i+1}, \mathbf{a}_{i+1}) - Q(\mathbf{s}_i, \mathbf{a}_i)$
- $v_i$  = logical violation score from LTN (higher = more constraint violations)
- $r_i$  = reward magnitude (prioritize high-reward experiences)
- $\lambda_{\text{logic}} = 2.0$  (weight for constraint violations)
- $\lambda_{\text{reward}} = 0.5$  (weight for reward magnitude)

**Rationale:**

- High TD-error  $\rightarrow$  surprising transitions that require learning
- High violation score  $\rightarrow$  dangerous patterns to avoid
- High reward  $\rightarrow$  successful strategies to reinforce

### 1.2.2 Sampling Probability

Transitions are sampled stochastically with probability proportional to priority:

$$P(i) = \frac{p_i^\alpha}{\sum_{j=1}^{|\mathcal{B}_{\text{STM}}|} p_j^\alpha} \quad (8)$$

where  $\alpha \in [0, 1]$  controls prioritization strength:

- $\alpha = 0 \rightarrow$  uniform sampling
- $\alpha = 1 \rightarrow$  greedy prioritization
- $\alpha = 0.6 \rightarrow$  recommended default (balanced)

### 1.2.3 Importance Sampling Correction

To correct for sampling bias, apply importance-sampling weights:

$$w_i = \left( \frac{1}{|\mathcal{B}_{\text{STM}}|} \cdot \frac{1}{P(i)} \right)^\beta \quad (9)$$

where  $\beta \in [0, 1]$  is annealed from 0.4 to 1.0 during training.

Normalized weights:

$$\bar{w}_i = \frac{w_i}{\max_j w_j} \quad (10)$$

Gradients are scaled by importance weights:

$$\nabla_{\theta} \mathcal{L}(\tau_i) \leftarrow \bar{w}_i \cdot \nabla_{\theta} \mathcal{L}(\tau_i) \quad (11)$$

### 1.2.4 Time Compression

During replay, transitions are replayed at  $C \times$  speed to accelerate consolidation:

$$\Delta t_{\text{replay}} = \frac{\Delta t_{\text{original}}}{C} \quad (12)$$

where  $C \in [10, 20]$  is the compression factor (typically  $C = 15$ ).

**Biological Motivation:** Real hippocampal replay occurs at 10-20 $\times$  speed during sleep.

### 1.2.5 SWR Replay Algorithm

---

**Algorithm 1** Sharp Wave Ripple Replay
 

---

**Require:** Buffer  $\mathcal{B}_{\text{STM}}$ , compression factor  $C$ , batch size  $B$ , prioritization exponent  $\alpha$

**Ensure:** Replay batch  $\mathcal{B}_{\text{replay}}$ , importance weights  $\mathbf{w}$

- 1: Compute TD-errors  $\delta_i$  for all transitions
  - 2: Compute logical violations  $v_i$  via LTN evaluation
  - 3: Compute priorities  $p_i = |\delta_i| + \lambda_{\text{logic}} \cdot v_i + \lambda_{\text{reward}} \cdot |r_i|$
  - 4: Compute sampling probabilities  $P(i) = p_i^\alpha / \sum_j p_j^\alpha$
  - 5: Sample  $B$  transition indices with probabilities  $P(i)$
  - 6: Compute importance weights  $w_i = (1/(|\mathcal{B}_{\text{STM}}| \cdot P(i)))^\beta$
  - 7: Normalize weights  $\bar{w}_i = w_i / \max_j w_j$
  - 8: **for** each sampled transition  $\tau_i = (\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})$  **do**
  - 9:   Compress time:  $\Delta t \leftarrow \Delta t / C$
  - 10:   Add  $(\tau_i, \bar{w}_i)$  to  $\mathcal{B}_{\text{replay}}$
  - 11: **end for**
  - 12: **return**  $\mathcal{B}_{\text{replay}}, \mathbf{w}$
- 

## 1.3 Long-Term Memory (Neocortex)

The neocortical subsystem maintains abstract schemas—statistical summaries of recurring market patterns.

### 1.3.1 Schema Representation

Each schema  $k$  is represented as a Gaussian distribution:

$$\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (13)$$

where:

$$\boldsymbol{\mu}_k = \frac{1}{|\mathcal{S}_k|} \sum_{\mathbf{s} \in \mathcal{S}_k} \mathbf{s} \quad (14)$$

$$\boldsymbol{\Sigma}_k = \frac{1}{|\mathcal{S}_k|} \sum_{\mathbf{s} \in \mathcal{S}_k} (\mathbf{s} - \boldsymbol{\mu}_k)(\mathbf{s} - \boldsymbol{\mu}_k)^\top \quad (15)$$

$\mathcal{S}_k$  is the set of all states assigned to schema  $k$ .

### 1.3.2 Schema Assignment

New states are assigned to schemas via maximum likelihood:

$$k^* = \arg \max_k \mathcal{N}(\mathbf{s}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (16)$$

If  $\max_k \mathcal{N}(\mathbf{s}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) < \tau_{\text{schema}}$ , create a new schema.

### 1.3.3 Recall-Gated Consolidation

Updates to long-term memory are gated by two factors: recall strength (familiarity) and logical validity.

**Recall Strength:**

$$g(r_{\text{STM}}(\tau)) = \text{sigmoid}(\mathbf{W}_r \mathbf{r}_{\text{STM}} + b_r) \quad (17)$$

where  $\mathbf{r}_{\text{STM}}$  is the hippocampal representation of transition  $\tau$ .

**Logical Validity:**

$$g_{\text{sym}}(\tau) = \text{SatAgg}(\mathcal{K}_{\text{episode}}) \quad (18)$$

where  $\mathcal{K}_{\text{episode}}$  is the knowledge base evaluated on the episode containing  $\tau$ .

**Gated Update Rule:**

$$\Delta \mathbf{W}_{\text{LTM}} = \eta_{\text{sleep}} \cdot g(r_{\text{STM}}(\tau)) \cdot g_{\text{sym}}(\tau) \cdot \nabla_{\mathbf{W}} \mathcal{L}_{\text{policy}}(\tau) \quad (19)$$

Only update if both gates exceed thresholds:

$$\text{Update if: } g(r_{\text{STM}}) > \tau_{\text{recall}} \text{ AND } g_{\text{sym}} > \tau_{\text{logic}} \quad (20)$$

Typical thresholds:  $\tau_{\text{recall}} = 0.3$ ,  $\tau_{\text{logic}} = 0.7$ .

### 1.3.4 Consolidation Update Rule

For schema  $k$ , update mean and covariance:

$$\boldsymbol{\mu}_k^{(t+1)} = (1 - \eta_{\text{schema}}) \boldsymbol{\mu}_k^{(t)} + \eta_{\text{schema}} \cdot \mathbf{s}_{\text{new}} \quad (21)$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = (1 - \eta_{\text{schema}}) \boldsymbol{\Sigma}_k^{(t)} + \eta_{\text{schema}} \cdot (\mathbf{s}_{\text{new}} - \boldsymbol{\mu}_k^{(t)})(\mathbf{s}_{\text{new}} - \boldsymbol{\mu}_k^{(t)})^\top \quad (22)$$

where  $\eta_{\text{schema}}$  is the schema learning rate (typically 0.01).

## 2 UMAP Visualization: Cognitive Dashboard

UMAP (Uniform Manifold Approximation and Projection) provides a real-time 3D visualization of the system's internal knowledge structure.

### 2.1 AlignedUMAP for Schema Formation

AlignedUMAP ensures consistency across multiple sleep cycles, enabling tracking of schema evolution over time.

### 2.1.1 Objective Function

AlignedUMAP minimizes:

$$\mathcal{L}_{\text{AlignedUMAP}} = \sum_{t=1}^{T_{\text{cycles}}} \left[ \mathcal{L}_{\text{UMAP}}(\mathbf{X}_t) + \lambda_{\text{align}} \sum_{i,j} w_{ij} ||\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(t-1)}||^2 \right] \quad (23)$$

where:

- $\mathbf{X}_t \in \mathbb{R}^{N_t \times d}$  = high-dimensional embeddings at sleep cycle  $t$
- $\mathbf{y}_i^{(t)} \in \mathbb{R}^3$  = 3D projection of point  $i$  at cycle  $t$
- $w_{ij}$  = alignment weights (higher for points in same schema)
- $\lambda_{\text{align}} = 0.1$  = alignment strength

**Standard UMAP Loss:**

$$\mathcal{L}_{\text{UMAP}}(\mathbf{X}) = \sum_{i,j} \left[ v_{ij} \log \frac{v_{ij}}{w_{ij}} + (1 - v_{ij}) \log \frac{1 - v_{ij}}{1 - w_{ij}} \right] \quad (24)$$

where  $v_{ij}$  is high-dimensional similarity and  $w_{ij}$  is low-dimensional similarity.

### 2.1.2 Alignment Weights

$$w_{ij} = \begin{cases} 1.0 & \text{if schema}(i) = \text{schema}(j) \\ 0.1 & \text{otherwise} \end{cases} \quad (25)$$

### 2.1.3 Schema Cluster Detection

Schemas are identified as dense clusters in UMAP space using DBSCAN:

$$\text{Schema}_k = \{\mathbf{y}_i : ||\mathbf{y}_i - \boldsymbol{\mu}_k|| < \tau_{\text{cluster}}\} \quad (26)$$

where  $\tau_{\text{cluster}}$  is the cluster radius (typically 0.5 in normalized UMAP space).

## 2.2 Parametric UMAP for Real-Time Monitoring

Parametric UMAP learns a neural network mapping for fast projection of new points during live trading.

### 2.2.1 Neural Network Projection

A feedforward network  $f_{\text{UMAP}} : \mathbb{R}^d \rightarrow \mathbb{R}^3$  is trained to approximate UMAP projection:

$$\mathbf{y} = f_{\text{UMAP}}(\mathbf{e}; \theta_{\text{UMAP}}) \quad (27)$$

**Architecture:**

$$\mathbf{h}_1 = \text{ReLU}(\mathbf{W}_1 \mathbf{e} + \mathbf{b}_1), \quad \mathbf{h}_1 \in \mathbb{R}^{256} \quad (28)$$

$$\mathbf{h}_2 = \text{ReLU}(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2), \quad \mathbf{h}_2 \in \mathbb{R}^{128} \quad (29)$$

$$\mathbf{y} = \mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3, \quad \mathbf{y} \in \mathbb{R}^3 \quad (30)$$

**Training Objective:**

$$\mathcal{L}_{\text{ParametricUMAP}} = \sum_i ||\mathbf{y}_i - f_{\text{UMAP}}(\mathbf{e}_i)||^2 + \mathcal{L}_{\text{UMAP}} \quad (31)$$

### 2.2.2 Anomaly Detection

During live trading, a point is flagged as anomalous if it falls outside all known schemas:

$$\text{Anomaly}(\mathbf{y}) = \mathbb{I} \left[ \min_k ||\mathbf{y} - \boldsymbol{\mu}_k|| > \tau_{\text{anomaly}} \right] \quad (32)$$

where  $\tau_{\text{anomaly}} = 2.0$  (units in UMAP space).

**Response to Anomalies:**

- Log anomaly with full context
- Increase risk threshold temporarily
- Alert human operator if anomaly persists
- Add to high-priority replay buffer

## 3 Integration with Vector Database (Qdrant)

Long-term memory schemas are persisted in Qdrant for efficient similarity search and retrieval.

### 3.1 Schema Storage

Each schema is stored as a point in Qdrant:

- **Vector:**  $\boldsymbol{\mu}_k \in \mathbb{R}^d$  (schema centroid)

- **Payload:**

- schema\_id: Unique identifier
- covariance: Flattened  $\Sigma_k$
- num\_points:  $|\mathcal{S}_k|$
- avg\_reward: Mean reward for transitions in schema
- created\_at: Timestamp
- last\_updated: Timestamp

### 3.2 Similarity Search

Given a new state  $s_{\text{new}}$ , retrieve top- $k$  similar schemas:

$$\text{TopK}(s_{\text{new}}) = \arg \max_{k, |K|=k} \{\text{cosine}(s_{\text{new}}, \mu_i)\}_{i=1}^{N_{\text{schemas}}} \quad (33)$$

**Use Cases:**

- Retrieve historical context during decision-making
- Find similar market conditions for transfer learning
- Identify schema membership for new states

### 3.3 Periodic Schema Pruning

Remove low-quality schemas to prevent memory bloat:

$$\text{Prune if: } |\mathcal{S}_k| < \tau_{\text{min\_points}} \text{ OR } \text{age}(k) > \tau_{\text{max\_age}} \quad (34)$$

where  $\tau_{\text{min\_points}} = 10$  and  $\tau_{\text{max\_age}} = 90$  days.

## 4 Sleep Cycle: Complete Algorithm

The sleep cycle runs nightly (or after market close) to consolidate the day's experiences.

**Algorithm 2** JANUS Backward Sleep Cycle**Require:** Short-term buffer  $\mathcal{B}_{\text{STM}}$ , long-term schemas  $\{\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}_k$ **Ensure:** Updated schemas, trained policy

```

1: Phase 1: Prioritized Replay (SWR Simulation)
2: for  $n_{\text{replays}}$  iterations (e.g., 1000) do
3:   Sample batch  $\mathcal{B}_{\text{replay}}$  using SWR algorithm
4:   Compute losses:  $\mathcal{L}_{\text{policy}}, \mathcal{L}_{\text{logic}}$ 
5:   Update policy:  $\theta \leftarrow \theta - \eta \cdot \bar{w}_i \cdot \nabla_{\theta} \mathcal{L}_{\text{total}}$ 
6:   Update priorities:  $p_i \leftarrow |\delta_i| + \lambda_{\text{logic}} v_i + \lambda_{\text{reward}} |r_i|$ 
7: end for
8:
9: Phase 2: Schema Consolidation
10: for each transition  $\tau_i \in \mathcal{B}_{\text{STM}}$  do
11:   Compute recall gate:  $g_{\text{recall}} = \text{sigmoid}(\mathbf{W}_r \mathbf{r}_{\text{STM}} + b_r)$ 
12:   Compute logic gate:  $g_{\text{logic}} = \text{SatAgg}(\mathcal{K})$ 
13:   if  $g_{\text{recall}} > \tau_{\text{recall}}$  AND  $g_{\text{logic}} > \tau_{\text{logic}}$  then
14:     Find matching schema:  $k^* = \arg \max_k \mathcal{N}(s_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ 
15:     if  $\mathcal{N}(s_i; \boldsymbol{\mu}_{k^*}, \boldsymbol{\Sigma}_{k^*}) < \tau_{\text{schema}}$  then
16:       Create new schema:  $\boldsymbol{\mu}_{\text{new}} \leftarrow s_i, \boldsymbol{\Sigma}_{\text{new}} \leftarrow \epsilon \mathbf{I}$ 
17:     else
18:       Update schema  $k^*$  using consolidation rule
19:     end if
20:   end if
21: end for
22:
23: Phase 3: UMAP Update
24: Extract all schema centroids:  $\{\boldsymbol{\mu}_k\}_k$ 
25: Fit AlignedUMAP with previous cycle alignment
26: Update parametric UMAP network
27: Detect new clusters via DBSCAN
28:
29: Phase 4: Vector Database Sync
30: Upsert updated schemas to Qdrant
31: Prune low-quality schemas
32: Create snapshot for recovery
33:
34: Phase 5: Metrics & Logging
35: Compute and log:
    • Number of schemas:  $N_{\text{schemas}}$ 
    • Mean schema size:  $\mathbb{E}[|\mathcal{S}_k|]$ 
    • Constraint satisfaction rate:  $\mathbb{E}[\text{SatAgg}(\mathcal{K})]$ 
    • Average TD-error improvement

```

## 5 Implementation Checklist

sec:checklist

This section provides a sequential checklist for implementing JANUS Backward.

### 5.1 Core Components

#### 1. Short-Term Memory (Hippocampus)

- ☐ Implement episodic buffer with FIFO eviction
- ☐ Implement pattern separation layer
- ☐ Add orthogonality regularization
- ☐ Implement TopK sparse encoding
- ☐ Add timestamp indexing for temporal queries
- ☐ Test buffer operations (insert, retrieve, evict)

#### 2. Sharp Wave Ripple (SWR) Simulator

- ☐ Implement TD-error computation
- ☐ Implement logical violation scoring via LTN
- ☐ Implement composite priority function
- ☐ Implement prioritized sampling with importance weights
- ☐ Add time compression simulation
- ☐ Test replay batch generation
- ☐ Validate importance weight correction

#### 3. Long-Term Memory (Neocortex)

- ☐ Implement schema representation (Gaussian)
- ☐ Implement schema assignment via maximum likelihood
- ☐ Implement recall gate computation
- ☐ Implement logical validity gate
- ☐ Implement gated consolidation update
- ☐ Add schema creation logic
- ☐ Test schema updates with edge cases

#### 4. UMAP Visualization

- ☐ Implement AlignedUMAP objective
- ☐ Add alignment weight computation
- ☐ Implement parametric UMAP network
- ☐ Implement DBSCAN cluster detection
- ☐ Add anomaly detection logic
- ☐ Test visualization updates across cycles
- ☐ Validate cluster stability

## 5.2 Integration & Storage

### 1. Vector Database Integration (Qdrant)

- ☐ Set up Qdrant connection
- ☐ Define schema collection structure
- ☐ Implement schema upsert operations
- ☐ Implement similarity search queries
- ☐ Add periodic pruning logic
- ☐ Implement backup/restore functionality
- ☐ Test concurrent access patterns

### 2. Sleep Cycle Orchestration

- ☐ Implement 5-phase sleep cycle algorithm
- ☐ Add progress tracking and logging
- ☐ Implement graceful shutdown on errors
- ☐ Add checkpoint/resume capability
- ☐ Test full sleep cycle end-to-end
- ☐ Validate schema evolution over cycles

## 5.3 Monitoring & Debugging

### 1. Metrics Collection

- ☐ Track number of schemas over time
- ☐ Monitor mean schema size
- ☐ Track constraint satisfaction rates

- ☐ Monitor TD-error distribution
- ☐ Track UMAP cluster count
- ☐ Log replay batch statistics

## 2. Visualization & Debugging

- ☐ Export UMAP projections for visualization
- ☐ Add schema evolution timeline
- ☐ Visualize priority distributions
- ☐ Plot constraint satisfaction heatmaps
- ☐ Add interactive schema browser

## 5.4 Performance Optimization

### 1. Batch Processing

- ☐ Parallelize TD-error computation
- ☐ Vectorize schema likelihood calculations
- ☐ Batch Qdrant upsert operations
- ☐ Use GPU for UMAP fitting (if available)
- ☐ Profile and optimize bottlenecks
- ☐ Target: <10 minutes for 10k transitions

## 6 Rust Implementation Considerations

sec:rust

### 6.1 Cold Path Optimization

Unlike Forward, Backward has no strict latency requirements, allowing focus on throughput and correctness.

- **Batch parallelism:** Use `rayon` for parallel replay processing
- **Memory efficiency:** Use `ndarray` for linear algebra operations
- **Async I/O:** Use `tokio` for non-blocking Qdrant operations
- **Checkpointing:** Serialize intermediate state with `serde`

## 6.2 Data Structures

### 6.2.1 Episodic Buffer

```

1 use std::collections::VecDeque;
2
3 #[derive(Clone, Debug)]
4 pub struct Transition {
5     pub state: Array1<f32>,
6     pub action: Action,
7     pub reward: f32,
8     pub next_state: Array1<f32>,
9     pub timestamp: u64,
10 }
11
12 pub struct EpisodicBuffer {
13     buffer: VecDeque<Transition>,
14     capacity: usize,
15 }
16
17 impl EpisodicBuffer {
18     pub fn new(capacity: usize) -> Self {
19         Self {
20             buffer: VecDeque::with_capacity(capacity),
21             capacity,
22         }
23     }
24
25     pub fn push(&mut self, transition: Transition) {
26         if self.buffer.len() >= self.capacity {
27             self.buffer.pop_front();
28         }
29         self.buffer.push_back(transition);
30     }
31
32     pub fn sample_prioritized(
33         &self,
34         priorities: &[f32],
35         batch_size: usize,
36         alpha: f32,
37     ) -> (Vec<Transition>, Vec<f32>) {
38         // Prioritized sampling implementation

```

```

39         todo!()
40     }
41 }

```

## 6.2.2 Schema Representation

```

1  use ndarray::{Array1, Array2};
2
3  #[derive(Clone, Debug, serde::Serialize, serde::Deserialize)]
4  pub struct Schema {
5      pub id: uuid::Uuid,
6      pub mean: Array1<f32>,
7      pub covariance: Array2<f32>,
8      pub num_points: usize,
9      pub avg_reward: f32,
10     pub created_at: chrono::DateTime<chrono::Utc>,
11     pub last_updated: chrono::DateTime<chrono::Utc>,
12 }
13
14 impl Schema {
15     pub fn likelihood(&self, state: &Array1<f32>) -> f32 {
16         // Compute Gaussian likelihood
17         let diff = state - &self.mean;
18         let inv_cov = self.covariance.inv().unwrap();
19         let exponent = -0.5 * diff.dot(&inv_cov.dot(&diff));
20         exponent.exp()
21     }
22
23     pub fn update(
24         &mut self,
25         new_state: &Array1<f32>,
26         learning_rate: f32,
27     ) {
28         let diff = new_state - &self.mean;
29         self.mean = &self.mean + learning_rate * &diff;
30         // Update covariance (outer product)
31         let outer = diff.clone().insert_axis(Axis(1))
32             .dot(&diff.clone().insert_axis(Axis(0)));
33         self.covariance = (1.0 - learning_rate) * &self.covariance
34             + learning_rate * outer;
35         self.num_points += 1;

```

```
36         self.last_updated = chrono::Utc::now();
37     }
38 }
```

## 6.3 Error Handling

```
1  #[derive(Debug, thiserror::Error)]
2  pub enum BackwardError {
3      #[error("Insufficient data for replay: {0} transitions")]
4      InsufficientData(usize),
5
6      #[error("Schema update failed: {0}")]
7      SchemaUpdateError(String),
8
9      #[error("UMAP fitting failed: {0}")]
10     UmapError(String),
11
12     #[error("Qdrant operation failed: {0}")]
13     VectorDbError(#[from] qdrant_client::QdrantError),
14
15     #[error("Linear algebra error: {0}")]
16     LinalgError(String),
17 }
18
19 pub type BackwardResult<T> = Result<T, BackwardError>;
```

## References

- [1] Jordan Smith, "Project JANUS: Implementation Guide v1.0," 2025.
- [2] Schaul et al., "Prioritized Experience Replay," ICLR 2016.
- [3] "A Unified Dynamic Model for Learning, Replay, and Ripples," 2015/2025.
- [4] Foster, Wilson, "Reverse Replay of Behavioural Sequences in Hippocampal Place Cells," Nature 2006.
- [5] Tse et al., "Schema-Dependent Gene Activation and Memory Encoding in Neocortex," Science 2011.
- [6] McInnes et al., "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," arXiv:1802.03426, 2018.
- [7] Aynaud et al., "AlignedUMAP: Temporal Alignment for Multi-Dataset Visualization," bioRxiv, 2020.
- [8] Qdrant Team, "Qdrant Vector Database Documentation," 2024.