Project JANUS

Neuromorphic Trading Intelligence

Complete Technical Specification

A Brain-Inspired Architecture for Autonomous Financial Systems

Unified Documentation

This document consolidates all technical specifications of Project JANUS:

1. **Main Architecture** — System design and philosophical foundation
2. **Forward Service** — Real-time decision-making and execution
3. **Backward Service** — Memory consolidation and learning
4. **Neuromorphic Architecture** — Brain-region mapping
5. **Rust Implementation** — Production deployment guide

Author

Jordan Smith

Date

December 30, 2025

“The god of beginnings and transitions, looking simultaneously to the future and the past.”

Contents

Part I

Main Architecture

Overview

Part 1 provides the architectural philosophy and system design overview for Project JANUS. This section would typically include:

- **Introduction:** The crisis of complexity in quantitative trading
- **Architectural Philosophy:** The dual-service design (Forward/Backward)
- **Core Components:** Vision, Logic, Fusion, and Decision systems
- **Memory Hierarchy:** Three-timescale learning architecture
- **Implementation Strategy:** Rust-first development approach

Note: The detailed mathematical specifications for each component are presented in Parts 2-5 below. This consolidated document focuses on the theoretical foundations and algorithmic specifications rather than high-level architectural discussion.

Part II

Forward Service (Janus Bifrons)

Abstract

JANUS Forward represents the "wake state" of the JANUS trading system, responsible for all real-time decision-making during market hours. This service combines:

- **Visual Pattern Recognition** using Gramian Angular Fields (GAF) and Video Vision Transformers (ViViT)
- **Symbolic Reasoning** via Logic Tensor Networks (LTN) for constraint satisfaction
- **Multimodal Fusion** integrating time series, visual, and textual market data
- **Dual-Pathway Decision Making** inspired by basal ganglia architecture

The Forward service operates on a hot path with strict latency requirements, implementing end-to-end gradient flow through differentiable market simulation while maintaining regulatory compliance through symbolic constraints.

1 Visual Pattern Recognition: DiffGAF and ViViT

The visual subsystem transforms time series data into spatiotemporal images, enabling the system to "see" market patterns that traditional numerical methods miss.

1.1 Mathematical Foundation: Gramian Angular Fields

Time series are encoded into polar coordinates and projected onto Gramian matrices, creating 2D representations that preserve temporal correlations.

1.1.1 Input Preprocessing

Given raw market data $X = \{x_1, x_2, \ldots, x_T\}$ where $x_t \in \mathbb{R}^D$ (multi-feature time series), we first apply feature selection to extract F relevant features.

1.1.2 Step 1: Learnable Normalization

Instead of fixed min-max scaling, we use learnable affine transformations with domain constraints:

$$\tilde{x}_t = \tanh\left(\gamma \odot \frac{x_t - \mu}{\sigma} + \beta\right) \tag{1}$$

where $\gamma, \beta \in \mathbb{R}^F$ are learned parameters, and μ, σ are running statistics. The $\tanh$ function guarantees $\tilde{x}_t \in (-1, 1)$, ensuring the subsequent $\arccos$ operation is well-defined.

1.1.3 Step 2: Polar Coordinate Transformation

Map normalized values to angular space:

$$\phi_t = \arccos(\tilde{x}_t) \in [0, \pi] \tag{2}$$

$$r_t = \frac{t}{T} \quad \text{(normalized timestamp)} \tag{3}$$

1.1.4 Step 3: Gramian Field Generation

Construct the Gramian Angular Summation Field (GASF):

$$\mathbf{G}_{ij} = \cos(\phi_i + \phi_j) = \tilde{x}_i \tilde{x}_j - \sqrt{1 - \tilde{x}_i^2}\sqrt{1 - \tilde{x}_j^2} \tag{4}$$

Or the Gramian Angular Difference Field (GADF):

$$\mathbf{G}_{ij} = \sin(\phi_i - \phi_j) = \sqrt{1 - \tilde{x}_i^2}\tilde{x}_j - \tilde{x}_i\sqrt{1 - \tilde{x}_j^2} \tag{5}$$

1.2 3D Spatiotemporal Manifolds: GAF Video

To capture temporal dynamics, we generate a sequence of GAF frames using sliding windows.

1.2.1 Sliding Window GAF Video Generation

Given a time series of length T, window size W, and stride S:

1. Extract windows: $X_k = \{x_{(k-1)S+1}, \ldots, x_{(k-1)S+W}\}$ for $k = 1, \ldots, N$
2. Generate GAF for each window: $\mathbf{G}_k = \text{GAF}(X_k) \in \mathbb{R}^{W \times W}$
3. Stack into video: $\mathbf{V} = [\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_N] \in \mathbb{R}^{N \times W \times W}$

1.3 Video Vision Transformer (ViViT)

The GAF video is processed by a factorized spatiotemporal transformer.

1.3.1 Patch Embedding

Divide each frame $\mathbf{G}_k$ into non-overlapping patches:

$$\mathbf{P}_k = \text{Reshape}(\mathbf{G}_k) \in \mathbb{R}^{P \times (p^2)} \tag{6}$$

where $P = (W/p)^2$ is the number of patches per frame.

1.3.2 Spatial Attention

Apply self-attention within each frame:

$$\mathbf{Z}_k^{(l)} = \text{MSA}(\text{LN}(\mathbf{Z}_k^{(l-1)})) + \mathbf{Z}_k^{(l-1)} \tag{7}$$

1.3.3 Temporal Attention

Apply attention across frames:

$$\mathbf{H}^{(l)} = \text{MSA}(\text{LN}([\mathbf{Z}_1^{(l)}, \dots, \mathbf{Z}_N^{(l)}])) \tag{8}$$

2 Logic Tensor Networks: Symbolic Reasoning Engine

LTNs bridge neural networks and first-order logic, enabling differentiable constraint satisfaction.

2.1 Mathematical Foundation

2.1.1 Grounding Function

Map logical constants to real vectors:

$$\mathcal{G} : \mathcal{C} \to \mathbb{R}^d \tag{9}$$

2.1.2 Predicate Grounding

A predicate $P(x)$ is grounded as a neural network $f_\theta : \mathbb{R}^d \to [0, 1]$.

2.2 Lukasiewicz T-Norm Operations

2.2.1 Conjunction (AND)

For training, we use Product Logic to ensure smooth gradients:

$$u \wedge v = u \cdot v \tag{10}$$

For inference/evaluation, standard Łukasiewicz logic may be used:

$$u \wedge v = \max(0, u + v - 1) \tag{11}$$

2.2.2 Disjunction (OR)

$$u \vee v = \min(1, u + v) \tag{12}$$

2.2.3 Negation (NOT)

$$\neg u = 1 - u \tag{13}$$

2.2.4 Implication (IF-THEN)

For training (Product Logic):

$$u \Rightarrow v = 1 - u + u \cdot v \tag{14}$$

For inference (Łukasiewicz Logic):

$$u \Rightarrow v = \min(1, 1 - u + v) \tag{15}$$

2.3 Knowledge Base Formulation

2.3.1 Wash Sale Constraint

$$\forall t : \text{Sell}(t) \wedge \text{Buy}(t') \wedge |t - t'| < 30 \Rightarrow \neg\text{TaxLoss}(t) \tag{16}$$

2.3.2 Almgren-Chriss Risk Constraint

$$\forall \text{order} : \text{Execute}(\text{order}) \Rightarrow \text{Slippage}(\text{order}) < \lambda \cdot \text{Volatility} \tag{17}$$

2.4 Logical Loss Function

2.4.1 Satisfiability Aggregation

$$\text{SAT}(\mathcal{KB}) = \text{p-mean}_{i=1}^{|\mathcal{KB}|}(\phi_i) \tag{18}$$

2.4.2 Logical Loss

$$\mathcal{L}_{\text{logic}} = 1 - \text{SAT}(\mathcal{KB}) \tag{19}$$

3 Multimodal Fusion: Gated Cross-Attention

3.1 Input Modalities

- Visual: $\mathbf{v} \in \mathbb{R}^{d_v}$ (from ViViT)
- Temporal: $\mathbf{t} \in \mathbb{R}^{d_t}$ (from LSTM/Transformer)
- Textual: $\mathbf{s} \in \mathbb{R}^{d_s}$ (from BERT embeddings)

3.2 Gated Cross-Attention Mechanism

3.2.1 Attention Computation

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{20}$$

3.2.2 Gating Mechanism

$$g = \text{sigmoid}(\mathbf{W}_g[\mathbf{v}; \mathbf{t}; \mathbf{s}] + \mathbf{b}_g) \tag{21}$$

4 Decision Engine: Basal Ganglia Pathways

4.1 Praxeological Motor: Dual Pathways

4.1.1 Direct Pathway (Go Signal)

$$\mathbf{d}_{\text{direct}} = \text{ReLU}(\mathbf{W}_d\mathbf{h}_{\text{fused}} + \mathbf{b}_d) \tag{22}$$

4.1.2 Indirect Pathway (No-Go Signal)

$$\mathbf{d}_{\text{indirect}} = \text{ReLU}(\mathbf{W}_i\mathbf{h}_{\text{fused}} + \mathbf{b}_i) \tag{23}$$

4.2 Cerebellar Forward Model

4.2.1 Market Impact Prediction

$$\hat{p}_{t+1} = f_{\text{cerebellum}}(\mathbf{s}_t, \mathbf{a}_t) \tag{24}$$

Part III

Backward Service (Janus Consivius)

Abstract

JANUS Backward represents the "sleep state" of the system, responsible for memory consolidation, schema formation, and learning from accumulated experience. This service implements:

- **Three-Timescale Memory Hierarchy** (Hippocampus $\rightarrow$ SWR $\rightarrow$ Neocortex)
- **Sharp-Wave Ripple Simulation** for prioritized experience replay
- **Schema Formation** via UMAP-based clustering
- **Recall-Gated Consolidation** ensuring only successful patterns are promoted

The Backward service runs on a cold path during off-market hours, performing computationally intensive operations to distill daily experiences into long-term knowledge.

5 Memory Hierarchy: Three-Timescale Architecture

5.1 Short-Term Memory (Hippocampus)

5.1.1 Episodic Buffer

Stores raw experiences during trading:

$$\mathcal{D}_{\text{hippo}} = \{(s_t, a_t, r_t, s_{t+1}, \mathbf{c}_t)\}_{t=1}^{T} \tag{25}$$

where $\mathbf{c}_t$ contains contextual metadata (volatility, spreads, volume).

5.1.2 Pattern Separation

Uses random projections to ensure diverse encoding:

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\text{rand}} \cdot [s_t; a_t; \mathbf{c}_t]) \tag{26}$$

5.2 Medium-Term Consolidation (SWR Simulator)

5.2.1 Replay Prioritization

Compute TD-error based priority:

$$p_i = |\delta_i| + \epsilon \tag{27}$$

where $\delta_i = r_i + \gamma \max_{a'} Q(s_{i+1}, a') - Q(s_i, a_i)$ and $\epsilon = 10^{-6}$ ensures numerical stability.

5.2.2 Sampling Probability

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \tag{28}$$

where $\alpha \in [0, 1]$ controls prioritization strength (typically $\alpha = 0.6$).

5.2.3 Importance Sampling Correction

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^\beta \tag{29}$$

where $\beta \in [0, 1]$ is annealed from $0.4 \to 1.0$ during training to fully correct bias at convergence.

5.3 Long-Term Memory (Neocortex)

5.3.1 Schema Representation

Each schema is a prototype:

$$\mathbf{z}_k = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \mathbf{h}_i \tag{30}$$

5.3.2 Recall-Gated Consolidation

Only update schemas from successfully recalled experiences:

$$\mathbf{z}_k \leftarrow \mathbf{z}_k + \eta \cdot \mathbb{1}[\text{recall\_success}] \cdot (\mathbf{h}_{\text{new}} - \mathbf{z}_k) \tag{31}$$

6 UMAP Visualization: Cognitive Dashboard

6.1 AlignedUMAP for Schema Formation

Maintains consistent embeddings across sleep cycles.

6.1.1 Objective Function

The full UMAP loss includes both attraction and repulsion terms:

$$\mathcal{L}_{\text{UMAP}} = \sum_{i \neq j} [w_{ij} \log(q_{ij}) + (1 - w_{ij}) \log(1 - q_{ij})] \tag{32}$$

where $q_{ij} = (1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}$.

Note: In practice, the repulsion term $(1-w_{ij})$ is approximated via *negative sampling* to achieve $\mathcal{O}(N)$ complexity. For each positive edge, we sample $k = 5$ random negative pairs.

6.2 Parametric UMAP for Real-Time Monitoring

Train a neural network to project new experiences:

$$\mathbf{y}_{\text{new}} = f_\theta(\mathbf{h}_{\text{new}}) \tag{33}$$

7 Integration with Vector Database (Qdrant)

7.1 Schema Storage

Each schema is stored in the vector database with the following structure:

Schema Representation:

$$\mathcal{S}_k = (\text{id}_k, \mathbf{z}_k, \mathcal{M}_k) \tag{34}$$

where:

- $\text{id}_k \in \mathbb{N}$: Unique schema identifier
- $\mathbf{z}_k \in \mathbb{R}^d$: Centroid embedding vector
- $\mathcal{M}_k$: Metadata payload containing:
  - $n_k = |C_k|$: Number of experiences in cluster
  - $\bar{r}_k = \frac{1}{n_k} \sum_{i \in C_k} r_i$: Average reward
  - $\sigma_k = \sqrt{\frac{1}{n_k} \sum_{i \in C_k} (r_i - \bar{r}_k)^2}$: Volatility (std. dev. of returns)

Storage Invariant: All schema vectors are L2-normalized for cosine similarity search:

$$\mathbf{z}_k \leftarrow \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|_2} \tag{35}$$

7.2 Similarity Search

Retrieve nearest schemas:

$$\mathcal{N}_k = \arg\max_k \text{cosine}(\mathbf{h}_t, \mathbf{z}_k) \tag{36}$$

Part IV

Neuromorphic Architecture

Abstract

This document maps the computational components of Project JANUS to specific brain regions, ensuring biological plausibility and leveraging neuroscience insights for system design. The neuromorphic approach provides:

- **Modular Design** with clear functional boundaries
- **Biological Validation** of architectural decisions
- **Emergent Intelligence** through brain-inspired interactions

8 Neuromorphic Design Philosophy

8.1 Why Brain-Inspired Architecture?

The brain efficiently solves problems similar to trading:

- Pattern recognition under uncertainty
- Fast decision-making with delayed rewards
- Continual learning without catastrophic forgetting
- Multi-timescale memory consolidation

8.2 Neuroscience-to-Trading Mapping

Brain Region	Biological Function	Trading Function
Visual Cortex	Pattern recognition	GAF/ViViT chart analysis
Hippocampus	Episodic memory	Experience replay buffer
Prefrontal Cortex	Logic and planning	LTN constraint checking
Basal Ganglia	Action selection	Buy/sell/hold decisions
Cerebellum	Motor prediction	Market impact forecasting
Amygdala	Threat detection	Risk circuit breakers

9 Brain Region Architectures

9.1 Cortex: Strategic Planning & Long-term Memory

9.1.1 Trading Implementation

Component: Neocortical Schema Network

Implementation:

- Schema prototypes stored in Qdrant vector database
- Each schema represents a market regime (trending, mean-reverting, volatile)
- Slow consolidation during sleep cycles

9.2 Hippocampus: Episodic Memory & Experience Replay

9.2.1 Trading Implementation

Component: Episodic Buffer + SWR Replay

Implementation:

- Fixed-size circular buffer storing recent trades
- Sparse encoding via random projections
- Prioritized replay during training

9.3 Basal Ganglia: Action Selection & Reinforcement Learning

9.3.1 Trading Implementation

Component: Dual-Pathway Decision Module

The basal ganglia implements competing pathways for action selection:

Direct Pathway (Go Signal):

$$\mathbf{d}_{\text{direct}} = \text{ReLU}(\mathbf{W}_{\text{direct}}\mathbf{h} + \mathbf{b}_{\text{direct}}) \tag{37}$$

where $\mathbf{h}$ is the fused state representation and $\mathbf{W}_{\text{direct}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$.

Indirect Pathway (No-Go Signal):

$$\mathbf{d}_{\text{indirect}} = \text{ReLU}(\mathbf{W}_{\text{indirect}}\mathbf{h} + \mathbf{b}_{\text{indirect}}) \tag{38}$$

Action Selection:

$$\mathbf{a}_t = \text{softmax}(\mathbf{d}_{\text{direct}} - \lambda \cdot \mathbf{d}_{\text{indirect}}) \tag{39}$$

where $\lambda > 0$ is the inhibition weight parameter.

9.4 Prefrontal Cortex: Logic, Planning & Compliance

9.4.1 Trading Implementation

Component: Logic Tensor Network

Ensures regulatory compliance:

- Wash sale rules
- Position limits
- Capital allocation constraints

9.5 Amygdala: Fear, Threat Detection & Circuit Breakers

9.5.1 Trading Implementation

Component: Anomaly Detection Module

Triggers emergency stops based on statistical distance from normal operation:

Mahalanobis Distance:

$$D_M(\mathbf{s}_t) = \sqrt{(\mathbf{s}_t - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{s}_t - \boldsymbol{\mu})} \tag{40}$$

where $\boldsymbol{\mu}$ is the historical mean state and $\boldsymbol{\Sigma}$ is the covariance matrix.

Circuit Breaker Condition:

$$\text{Trigger} = \begin{cases} 1 & \text{if } D_M(\mathbf{s}_t) > \tau_{\text{danger}} \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

where τ_{danger} is calibrated to a false-positive rate (e.g., $\tau = 5$ for $p < 0.001$).

Additional Threat Signals:

- Sudden volatility spike: $\sigma_t > 3 \cdot \sigma_{\text{baseline}}$
- Drawdown threshold: cumulative loss $> L_{\max}$
- Liquidity crisis: bid-ask spread $> 10\times$ normal

9.6 Cerebellum: Motor Control & Execution

9.6.1 Trading Implementation

Component: Forward Model for Market Impact

Predicts price movement from order execution:

$$\Delta p = f_{\text{cerebellum}}(\text{order\_size}, \text{liquidity}, \text{volatility}) \tag{42}$$

Part V

Rust Implementation

Abstract

This document provides production-ready Rust implementation specifications for Project JANUS, including:

- **ML Framework Strategy** (PyTorch $\rightarrow$ ONNX $\rightarrow$ Rust inference)
- **High-Performance Services** with async Tokio runtime
- **Hybrid Training Pipeline** (Python training, Rust deployment)
- **Deployment Architecture** (Docker Compose + Kubernetes)

10 Architectural Overview

10.1 The Rust-First Philosophy

1. **Performance:** Zero-cost abstractions, no GC pauses
2. **Safety:** Memory safety without runtime overhead
3. **Concurrency:** Fearless async/await with Tokio
4. **Ecosystem:** Production-ready ML inference via ONNX

10.2 Component Diagram

System Architecture

```
Python Training Gateway
(FastAPI + PyTorch + HuggingFace)
        |
        | ONNX Export
        ▼
Rust Forward Service
(Tokio + ONNX Runtime + DiffGAF + LTN)
        |
        ▼
Rust Backward Service
(Rayon + Qdrant + UMAP + PER Buffer)
```

11 Machine Learning Framework Strategy

11.1 Framework Comparison Matrix

Framework	Pros	Cons	Use Case
tch-rs	Native PyTorch, GPU support	C++ deps, larger binary	Training & inference
ONNX Runtime	Universal, production-ready	No training	Inference only
Candle	Pure Rust, HF integration	Young ecosystem	Future migration

11.2 Recommended Migration Path

11.2.1 Phase 1: Hybrid (Months 1-3)

- Train models in PyTorch (Python)
- Export to ONNX format

- Rust inference via `ort` crate

11.2.2 Phase 2: Rust-Native Inference (Months 4-6)

- Optimize ONNX models for Rust
- Custom kernels for DiffGAF/LTN
- Benchmark against Python baseline

11.2.3 Phase 3: Full Rust ML (Months 7-12)

- Migrate training to Candle
- End-to-end Rust pipeline
- Custom GPU kernels via wgpu

12 Forward Service: Rust Implementation

12.1 Performance Requirements

- Latency: p99 < 10ms
- Throughput: 10,000 req/s
- Memory: < 2GB RSS

12.2 Core Data Structures

The system maintains several key data structures for real-time processing:

Market State Representation:

$$\mathcal{S}_t = (\tau_t, \mathbf{f}_t, \mathcal{O}_t, \mathbf{c}_t) \tag{43}$$

where:

- $\tau_t \in \mathbb{Z}^+$ is the timestamp
- $\mathbf{f}_t \in \mathbb{R}^d$ is the feature vector
- $\mathcal{O}_t = (\mathcal{B}_t, \mathcal{A}_t)$ is the order book with bids $\mathcal{B}_t$ and asks $\mathcal{A}_t$
- $\mathbf{c}_t$ contains contextual metadata (volatility, spreads, volume)

Order Book Structure:

$$\mathcal{B}_t = \{(p_i, q_i) : p_i \in \mathbb{R}^+, q_i \in \mathbb{R}^+\}_{i=1}^{N_{\text{bid}}} \tag{44}$$

$$\mathcal{A}_t = \{(p_j, q_j) : p_j \in \mathbb{R}^+, q_j \in \mathbb{R}^+\}_{j=1}^{N_{\text{ask}}} \tag{45}$$

12.3 GAF Transformation Algorithm

The GAF transformation converts time series to 2D images via the following algorithm:

Algorithm: GAF Computation

```
Input: Time series X = {x_1, ..., x_W}, window size W
Output: Gramian matrix G ∈ ℝ^{W×W}

X̃ ← Normalize(X) to [−1, 1]
φ_i ← arccos(x̃_i) for i = 1, ..., W
for i = 1 to W do
    for j = 1 to W do
        G_ij ← cos(φ_i + φ_j)
    end for
end for
return G reshaped to [1, W, W] tensor
```

Computational Complexity: $\mathcal{O}(W^2)$ for matrix construction, where W is the window size.

12.4 LTN Constraint Evaluation

Each constraint is represented as a weighted predicate function:

Constraint Structure:

$$\mathcal{C}_k = (P_k, w_k) \tag{46}$$

where $P_k : \mathcal{S} \to [0, 1]$ is a predicate and $w_k \in \mathbb{R}^+$ is the weight.

Evaluation Function:

$$\text{Eval}(\mathcal{C}_k, \mathcal{S}_t) = w_k \cdot P_k(\mathcal{S}_t) \tag{47}$$

T-norm Operations (already defined in Part 2):

$$a \wedge_{\mathcal{L}} b = \max(0, a + b - 1) \quad \text{(Conjunction)} \tag{48}$$

$$a \Rightarrow_{\mathcal{L}} b = \min(1, 1 - a + b) \quad \text{(Implication)} \tag{49}$$

Total Constraint Satisfaction:

$$\mathcal{L}_{\text{constraint}} = 1 - \frac{1}{K}\sum_{k=1}^{K} \text{Eval}(\mathcal{C}_k, \mathcal{S}_t) \tag{50}$$

12.5 Async Service Architecture

The service follows an event-driven architecture with the following characteristics:

Request Processing Pipeline:

1. **Initialization:** Load ONNX model $\mathcal{M}_{\text{ViViT}}$ and LTN engine $\mathcal{E}_{\text{LTN}}$
2. **Connection Handling:** Bind TCP listener on port 8080
3. **Concurrent Processing:** For each incoming request:
   - Spawn asynchronous task with model clone
   - Process request independently (non-blocking)
   - Return prediction and constraint satisfaction scores

Concurrency Model:

$$\text{Throughput} = \frac{N_{\text{workers}} \times 1000}{T_{\text{avg}}} \tag{51}$$

where N_{workers} is the thread pool size and T_{avg} is average processing time in ms.

Performance Characteristics: - Non-blocking I/O via async/await - Zero-copy model sharing across tasks - Bounded memory through connection limiting

13 Backward Service: Batch Processing

13.1 Prioritized Experience Replay

The replay buffer maintains experiences with importance-based sampling.

Buffer State:

$$\mathcal{B} = \{(e_i, p_i)\}_{i=1}^{N} \tag{52}$$

where e_i is an experience and $p_i \in \mathbb{R}^+$ is its priority.

Hyperparameters:

- $\alpha \in [0, 1]$: Priority exponent (0 = uniform, 1 = full prioritization)
- $\beta \in [0, 1]$: Importance sampling correction
- C: Buffer capacity

Sampling Algorithm:

Input: Buffer $\mathcal{B}$, batch size B
Output: Sampled batch $\{e_{i_1}, \ldots, e_{i_B}\}$

Compute probabilities: $P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}$
for $k = 1$ to B **do**
    Sample index $i_k \sim$ Categorical(P)
    Add e_{i_k} to batch
end for
return batch

Importance Weights:

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^\beta \tag{53}$$

These weights correct for the non-uniform sampling distribution.

13.2 Schema Consolidation Algorithm

Schemas are formed by clustering experience embeddings and storing centroids.

Algorithm: Schema Update

Input: Experiences $\mathcal{E} = \{e_1, \ldots, e_N\}$, number of clusters K
Output: Updated schema database

Extract embeddings: $\mathbf{h}_i =$ Embed(e_i) for $i = 1, \ldots, N$
Cluster: $\mathcal{C} = \{C_1, \ldots, C_K\} \leftarrow$ K-means$(\{\mathbf{h}_i\}, K)$
for $k = 1$ to K **do**
    Compute centroid: $\mathbf{z}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{h}_i$
    Compute statistics:
        $n_k = |C_k|$
        $\bar{r}_k = \frac{1}{|C_k|} \sum_{i \in C_k} r_i$ (average reward)
    Upsert schema k with vector $\mathbf{z}_k$ and metadata $(n_k, \bar{r}_k)$
end for

Schema Metadata: Each schema k stores:

- Centroid vector $\mathbf{z}_k \in \mathbb{R}^d$
- Member count n_k
- Average reward $\bar{r}_k$
- Volatility σ_k (standard deviation of returns)

K-means Objective:

$$\min_{\mathcal{C}} \sum_{k=1}^{K} \sum_{i \in C_k} \|\mathbf{h}_i - \mathbf{z}_k\|^2 \tag{54}$$

14 Deployment Architecture

14.1 Service Orchestration

The system deploys as three independent services:

Service Topology:

1. **Forward Service:**
   - Port: 8080 (HTTP API)
   - Dependencies: ONNX model files
   - Environment: Logging level, model paths
   - Resource limits: 2GB memory, 2 CPU cores
2. **Backward Service:**
   - Internal service (no external ports)
   - Dependencies: Qdrant vector database
   - Environment: Qdrant connection URL
   - Scheduling: Triggered during market close
3. **Qdrant Vector Database:**
   - Ports: 6333 (HTTP), 6334 (gRPC)
   - Persistent storage for schemas
   - Vector similarity search engine

Service Communication:

$$\text{Forward} \xrightarrow{\text{experiences}} \text{Buffer} \xrightarrow{\text{nightly}} \text{Backward} \xrightarrow{\text{schemas}} \text{Qdrant} \tag{55}$$

Volume Management: - Model artifacts: Shared read-only volume - Schema database: Persistent volume with backups - Experience buffer: Ephemeral storage (daily rotation)

Conclusion

Project JANUS represents a paradigm shift in quantitative trading: from opaque black boxes to transparent, brain-inspired systems that combine the best of deep learning and symbolic reasoning.

Key Innovations

1. **Neuromorphic Architecture:** Biologically plausible design with modular brain regions
2. **Neuro-Symbolic Fusion:** LTNs bridge neural networks and logical constraints
3. **Multi-Timescale Memory:** Three-tier hierarchy mirrors hippocampal-neocortical consolidation
4. **Production-Ready Rust:** High-performance, safe, and maintainable implementation

Future Work

- Quantum computing integration for portfolio optimization
- Continual learning without catastrophic forgetting
- Multi-agent systems for distributed trading
- Regulatory AI for automated compliance

Repository & Contact

GitHub: `https://github.com/nuniesmith/technical_papers`

For implementation code, updates, and discussions, visit the repository.

"The god of beginnings and transitions, looking simultaneously to the future and the past."