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

February 4, 2026

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

Contents

Part I

Main Architecture

Overview

The Epistemological Transition to Quant 4.0

The trajectory of algorithmic trading has historically been defined by a tension between interpretability and capability. We are currently witnessing a phase transition from the "black box" empiricism of deep learning (**lecun2015deep**) toward a new paradigm of Neuro-Symbolic integration (**garcez2024mapping**). Project JANUS stands at the vanguard of this transition, termed **Quant 4.0**. This architecture does not merely iterate on existing statistical methods but fundamentally reimagines the financial agent as a biological entity—one that perceives, reasons, remembers, and fears.

Historical Evolution of Quantitative Finance

- **Quant 1.0 (1980s-1990s):** Era of heuristics and expert systems with high interpretability but extreme rigidity
- **Quant 2.0 (1990s-2000s):** Statistical rigour through mean reversion, cointegration, and factor models (**fama1993common**)
- **Quant 3.0 (2010s-Present):** Deep learning hegemony with LSTMs, Transformers (**vaswani2017attention**), and Deep Reinforcement Learning
- **Quant 4.0 (JANUS):** Neuro-Symbolic AI achieving adaptability of deep learning with reliability of rule-based systems

The Dual-Process Architecture

The architectural philosophy of JANUS is strictly biomimetic, mirroring the Dual-Process Theory of cognition (**kahneman2011thinking**; **evans2008dual**). The system is bifurcated into two distinct but interacting services:

Service	Persona	Cognitive Role	Biological Analogue
Forward Service	Janus Bifrons	Perception & Action	Basal Ganglia & Thalamus
Backward Service	Janus Consivius	Memory & Learning	Hippocampus & Neocortex

This separation allows JANUS to optimize for latency on the hot path (Forward Service) while reserving heavy computational resources for consolidation and schema for-

mation on the cold path (Backward Service), implementing the Complementary Learning Systems theory (**mcclelland1995complementary**).

Note: The detailed mathematical specifications for each component are presented in Parts 2-5 below.

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) (**wang2015imaging**) and Video Vision Transformers (ViViT) (**arnab2021vivit**)
- **Symbolic Reasoning** via Logic Tensor Networks (LTN) (**badreddine2022logic**) for constraint satisfaction
- **Multimodal Fusion** integrating time series (**ansari2024chronos**), visual, and textual market data
- **Dual-Pathway Decision Making** inspired by basal ganglia architecture (**collins2014opponent**

The Forward service operates on a hot path with strict latency requirements, implementing end-to-end gradient flow through differentiable market simulation (**fu2024jaxlob**) while maintaining regulatory compliance through symbolic constraints. FPGA acceleration (**marino2023mevit**; **vemeko2023fpga**) enables nanosecond-level latency for high-frequency trading applications.

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. This approach is grounded in the work of **wang2015imaging**, who demonstrated that imaging time series significantly improves classification and imputation tasks by exposing temporal correlations to the inductive biases of convolutional neural networks.

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 (**wang2015imaging**). Recent research (**fusion2025gaf**) has validated that GAF encodings substantially outperform raw time-series inputs in classification tasks by capturing multi-scale temporal structures.

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), which JANUS employs to encode velocity of price changes as visual textures:

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

This transformation allows the system to visually perceive volatility regimes and microstructure dynamics (**wang2015imaging**).

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 (**arnab2021vivit**). Unlike standard Vision Transformers (**dosovitskiy2020image**) which process static images, ViViT factorizes attention across both space (price/volume levels of the limit order book) and time (sequences of LOB snapshots), enabling the system to track dynamic microstructure events.

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:

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

2 Logic Tensor Networks: Symbolic Reasoning Engine

LTNs bridge neural networks and first-order logic, enabling differentiable constraint satisfaction (**badreddine2022logic**). This neuro-symbolic approach allows JANUS to enforce regulatory and risk constraints directly within the gradient descent optimization, a capability absent in pure deep learning systems (**garcez2024mapping**).

2.1 Mathematical Foundation

The central innovation of LTNs (**badreddine2022logic**) is the ability to make Boolean logic differentiable using Real Logic, specifically Łukasiewicz t-norms (**lukasiewicz2024deep**).

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 $f_\theta : \mathbb{R}^d \to [0, 1]$, where truth values range continuously from 0 to 1 rather than being discrete binary values.

2.2 Lukasiewicz T-Norm Operations

Following **badreddine2022logic** and **lukasiewicz2024deep**, we employ fuzzy logic operators that are differentiable and thus compatible with backpropagation.

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

The knowledge base $\mathcal{KB}$ encodes regulatory constraints and risk management rules as logical predicates.

2.3.1 Wash Sale Constraint

The Wash Sale Rule (**irs2024wash**)—a critical regulatory constraint for active traders—prevents claiming tax losses on securities sold and repurchased within 30 days:

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

2.3.2 Almgren-Chriss Risk Constraint

Following the optimal execution framework of **almgren2001optimal**, we constrain market impact relative to volatility:

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

This ensures trades remain within the efficient frontier between expected cost and risk.

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

JANUS integrates multiple data modalities through gated cross-attention (**gatedattention2023**), allowing the system to dynamically weight inputs based on their predictive uncertainty. This is the machine-learning analogue of thalamic attentional gating in biological systems (**thalamus2018attention**).

3.1 Input Modalities

- Visual: $\mathbf{v} \in \mathbb{R}^{d_v}$ (from ViViT (**arnab2021vivit**))
- Temporal: $\mathbf{t} \in \mathbb{R}^{d_t}$ (from Chronos (**ansari2024chronos**) or Transformer (**vaswani2017attentior**
- Textual: $\mathbf{s} \in \mathbb{R}^{d_s}$ (from BERT embeddings for news/sentiment)

3.2 Gated Cross-Attention Mechanism

3.2.1 Attention Computation

Following the attention mechanism of **vaswani2017attention**:

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

3.2.2 Gating Mechanism

The gating function suppresses noise and amplifies signal, similar to thalamic regulation (**halassa2014thalamic**):

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

4 Decision Engine: Basal Ganglia Pathways

Action selection in JANUS is mediated by a model of the Basal Ganglia, comprising Direct ("Go") and Indirect ("No-Go") pathways (**collins2014opponent**; **foster2013hierarchical**). This architecture, known as Opponent Actor Learning (OpAL), creates dynamic risk tolerance that adapts to market conditions.

4.1 Praxeological Motor: Dual Pathways

4.1.1 Direct Pathway (Go Signal)

Encodes the benefits of an action, amplified by dopamine during high-confidence regimes (**dopamine2020reward**):

$$Q_\theta^+(s, a) = \mathbb{E}[\text{Reward} \mid s, a] \tag{22}$$

4.1.2 Indirect Pathway (No-Go Signal)

Encodes the costs and risks, amplified during uncertainty (**collins2014opponent**):

$$Q_\theta^-(s, a) = \mathbb{E}[\text{Risk} \mid s, a] \tag{23}$$

4.2 Cerebellar Forward Model

The Cerebellar module simulates market dynamics to predict execution outcomes before committing to a trade.

4.2.1 Market Impact Prediction

Following the Almgren-Chriss framework (**almgren2001optimal**; **markwick2023almgren**; **almgren2024deep**), the model predicts slippage and volatility. To detect predatory environments, JANUS employs the VPIN (Volume-Synchronized Probability of Informed Trading) metric (**easley2011vpin**; **easley2012flow**), which serves as a proxy for "Flow Toxicity"—the probability that the counterparty has superior information. High VPIN levels often precede flash crashes and feed into the Amygdala circuit for threat detection.

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

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 the Complementary Learning Systems (CLS) theory (**mcclelland1995complement** which posits that intelligent agents require two learning systems: a fast-learning hippocampus for episodic details and a slow-learning neocortex for statistical generalization. This service implements:

- **Three-Timescale Memory Hierarchy** (Hippocampus → SWR → Neocortex) following CLS architecture (**mcclelland1995complementary**)
- **Sharp-Wave Ripple Simulation** for prioritized experience replay (**buzsaki2015hippocampal**; **schaul2015prioritized**)
- **Schema Formation** via UMAP-based clustering (**mcinnes2018umap**; **alignedumap2023**)
- **Recall-Gated Consolidation** ensuring only successful patterns are promoted (**frank2006working**)

The Backward service runs on a cold path during off-market hours, performing computationally intensive operations to distill daily experiences into long-term knowledge, effectively replicating the biological process of memory consolidation during sleep (**buzsaki2015hippocampal**).

5 Memory Hierarchy: Three-Timescale Architecture

The three-tier memory architecture directly implements the Complementary Learning Systems theory (**mcclelland1995complementary**), preventing catastrophic forgetting while enabling rapid learning of new market patterns.

5.1 Short-Term Memory (Hippocampus)

The hippocampal buffer stores episodic memories of individual trading events, enabling fast learning without interfering with consolidated knowledge (**mcclelland1995complementary**).

5.1.1 Episodic Buffer

Stores raw experiences during trading, mirroring the role of biological hippocampus in episodic memory formation:

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

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)

Consolidation occurs during Sharp-Wave Ripples (SWRs)—high-frequency oscillations that replay compressed sequences of neural activity (**buzsaki2015hippocampal**). JANUS mimics this using Prioritized Experience Replay (**schaul2015prioritized**), modified to incorporate surprise, emotion, and logical violations.

5.2.1 Replay Prioritization

During "sleep" (post-market hours), experiences are replayed in priority order. Biological research shows that replay is biased towards salient and novel events (**kar2023selection**), which JANUS replicates through composite priority scoring. 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

To visualize and manage schema structures, JANUS employs Uniform Manifold Approximation and Projection (UMAP) (**mcinnes2018umap**), which preserves both local and global structure better than alternatives like t-SNE.

6.1 AlignedUMAP for Schema Formation

Track how internal representations evolve over time using AlignedUMAP (**alignedumap2023**), which aligns manifolds across different time steps to monitor representational drift. 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)

Schemas are stored in Qdrant (**qdrant2024**), a high-performance vector similarity search engine, enabling fast retrieval of relevant historical patterns.

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 is grounded in cognitive neuroscience (**buzsaki2015hippocampal**; **frank2006working**; **collins2014opponent**) and provides:

- **Modular Design** with clear functional boundaries mirroring brain organization
- **Biological Validation** of architectural decisions based on empirical neuroscience (**mcclelland1995complementary**)
- **Emergent Intelligence** through brain-inspired interactions and allostatic regulation (**sterling2012allostasis**)

8 Neuromorphic Design Philosophy

8.1 Why Brain-Inspired Architecture?

The brain efficiently solves problems similar to trading (**daw2006cortical**), demonstrating capabilities that map directly to trading challenges:

- Pattern recognition under uncertainty (visual cortex and hippocampus (**buzsaki2015hippocam**
- Fast decision-making with delayed rewards (basal ganglia (**collins2014opponent**))
- Continual learning without catastrophic forgetting (complementary learning systems (**mcclelland1995complementary**))
- Multi-timescale memory consolidation (hippocampal-neocortical transfer (**buzsaki2015hippoca**

8.2 Neuroscience-to-Trading Mapping

This mapping is grounded in empirical neuroscience and cognitive modeling (**frank2006working**; **collins2014opponent**; **foster2013hierarchical**).

Brain Region	Biological Function	Trading Function
Visual Cortex	Pattern recognition	GAF/ViViT chart analysis (**wang2015imaging**; **arnab2021vivit**)
Hippocampus	Episodic memory (**buzsaki2015hippocampal**)	Experience replay buffer (**schaul2015prioritized**)
Prefrontal Cortex	Logic and planning (**frank2006working**)	LTN constraint checking (**badreddine2022logic**)
Basal Ganglia	Action selection (**collins2014opponent**)	Buy/sell/hold decisions
Cerebellum	Motor prediction	Market impact forecasting (**almgren2001optimal**)
Amygdala	Threat detection (**amygdala2019fear**)	Risk circuit breakers

9 Brain Region Architectures

The following sections detail how each brain region's computational principles are implemented in JANUS.

9.1 Cortex: Strategic Planning & Long-term Memory

The neocortex implements slow, statistical learning of market schemas (**mcclelland1995complemer**

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

The hippocampus provides fast learning and episodic memory storage, with consolidation via Sharp-Wave Ripples (**buzsaki2015hippocampal**; **kar2023selection**).

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

The basal ganglia implements Opponent Actor Learning (OpAL) (**collins2014opponent**), balancing Go (Direct) and No-Go (Indirect) pathways modulated by dopamine (**dopamine2020rewar**

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

The prefrontal cortex provides working memory gating and logical reasoning capabilities (**frank2006working**).

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

The amygdala provides rapid threat detection and fear learning (**amygdala2019fear**), with connections to substantia nigra enabling fear extinction (**substantia2016connections**; **monfils2009extinction**).

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

The cerebellum provides forward models for motor prediction, adapted here for market impact forecasting (**almgren2001optimal**).

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. The choice of Rust is strategic, prioritizing memory safety and concurrency (**tradfi2023hft**), with zero-cost abstractions essential for nanosecond-critical high-frequency trading environments. This section includes:

- **ML Framework Strategy** leveraging tch-rs (**mazare2024tch**), Candle (**candle2024**), and ONNX Runtime
- **High-Performance Services** with async Tokio runtime (**tokio2024**)
- **Hybrid Training Pipeline** (Python training, Rust deployment)
- **Deployment Architecture** (Docker Compose + Kubernetes)

10 Architectural Overview

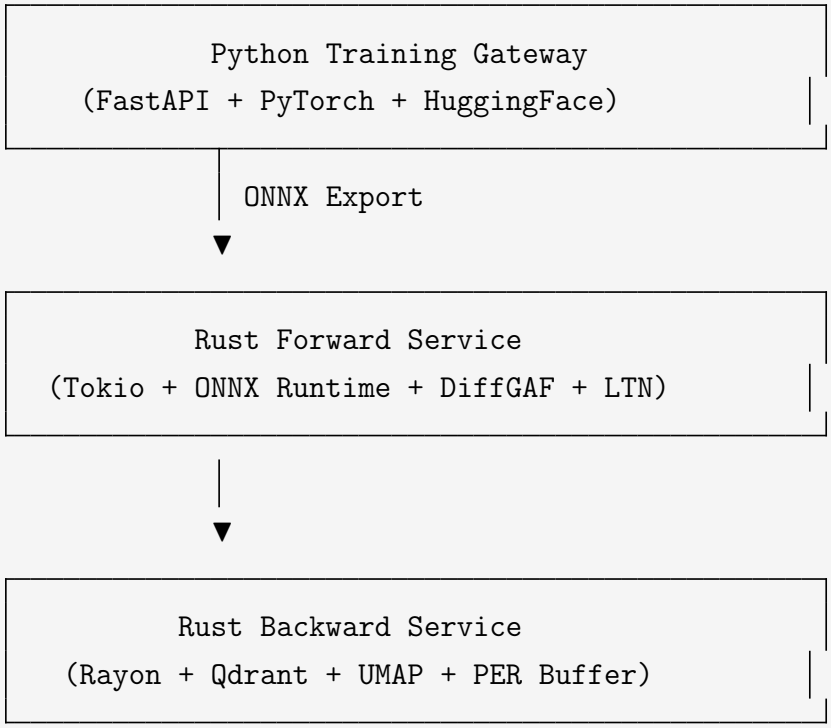
10.1 The Rust-First Philosophy

The move to Rust is justified by requirements in high-frequency trading systems (**tradfi2023hft**):

1. **Performance:** Zero-cost abstractions, no GC pauses—critical for sub-microsecond latency
2. **Safety:** Memory safety without runtime overhead, preventing undefined behavior
3. **Concurrency:** Fearless async/await with Tokio (**tokio2024**) for handling high-frequency websocket feeds
4. **Ecosystem:** Production-ready ML inference via tch-rs (**mazare2024tch**) and Candle (**candle2024**)

10.2 Component Diagram

System Architecture



11 Machine Learning Framework Strategy

11.1 Framework Comparison Matrix

The following frameworks are evaluated for JANUS deployment (**mazare2024tch**; **candle2024**):

Framework	Pros	Cons	Use Case
tch-rs (**mazare2024tch**)	Native PyTorch, GPU support	C++ deps, larger binary	Training & inference
ONNX Runtime	Universal, production-ready	No training	Inference only
Candle (**candle2024**)	Pure Rust, HF integration	Young ecosystem	Future migration
Polars (**polars2024**)	High-speed DataFrames	N/A	Data manipulation

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

FPGA acceleration using AMD Alveo U55C cards (**amd2023alveo**; **vemeko2023fpga**; **marino2023mevit**) enables the following targets:

- Latency: p99 < 10ms (target: < 1μs with FPGA)
- 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 (**wang2015imaging**). For training, JANUS utilizes JAX-LOB (**fu2024jaxlob**), a GPU-accelerated limit order book simulator that enables parallel simulation of thousands of order books, solving the data scarcity problem inherent in traditional trading systems.

Algorithm: GAF Computation

Input: Time series $X = \{x_1, \ldots, x_W\}$, window size W
Output: Gramian matrix $\mathbf{G} \in \mathbb{R}^{W \times W}$

$\tilde{X} \leftarrow$ Normalize(X) to $[-1, 1]$
$\phi_i \leftarrow \arccos(\tilde{x}_i)$ for $i = 1, \ldots, W$
for $i = 1$ to W **do**
    for $j = 1$ to W **do**
        $\mathbf{G}_{ij} \leftarrow \cos(\phi_i + \phi_j)$
    end for
end for
return $\mathbf{G}$ 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 $i_k \sim \text{Categorical}(P)$
    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: $\mathbf{h}_i = \text{Embed}(e_i)$ for $i = 1, \ldots, N$
Cluster: $\mathcal{C} = \{C_1, \ldots, C_K\} \leftarrow \text{K-means}(\{\mathbf{h}_i\}, K)$
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)

15 Advanced Neuroscience Integration

15.1 Thalamic Oscillations and Attentional Gating

The Thalamus functions as the "gatekeeper" of perception in JANUS, regulating the flow of visual and numerical data into the decision engine. This is implemented using Wilson-Cowan mean-field models (**wilson1972excitatory**; **wilson2024bidirectional**), which simulate the oscillatory dynamics of neural populations.

These models generate "attention masks" that suppress noise (irrelevant price ticks) and amplify signal (structural breaks). The Thalamic Reticular Nucleus provides attentional gating (**thalamus2018attention**; **halassa2014thalamic**), enabling JANUS to focus computational resources on the most informative market data streams. The integration of gated cross-attention (**gatedattention2023**) serves as the machine-learning equivalent of this biological process.

15.2 Dopaminergic Modulation and Market Regimes

The balance between Direct (Go) and Indirect (No-Go) pathways is modulated by a simulated dopamine signal (**dopamine2020reward**). High dopamine (bull market/high confidence) amplifies the Direct pathway; low dopamine (bear market/uncertainty) amplifies the Indirect pathway. This creates dynamic risk tolerance that adapts to market volatility, implementing allostatic regulation (**sterling2012allostasis**) rather than simple homeostatic feedback.

15.3 Fear Extinction and Adaptive Risk Management

The Amygdala module implements fear extinction mechanisms (**monfils2009extinction**; **substantia2016connections**), allowing the system to "unlearn" fear when threats have passed. This prevents the agent from becoming permanently paralyzed by a single traumatic market event (e.g., flash crash), while maintaining protective circuit breakers for genuine systemic risks.

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 (**quantum2024portfolio**)
- Continual learning without catastrophic forgetting using CLS principles (**mcclelland1995compl**
- Multi-agent systems for distributed trading using feudal hierarchies (**vezhnevets2017feudal**; **feudal2019multi**)
- Regulatory AI for automated compliance leveraging LTN constraint satisfaction (**badreddine2022logic**)
- Integration of foundation models for time series (**ansari2024chronos**; **ansari2024chronos2**)
- Hardware acceleration using FPGAs (**marino2023mevit**; **vemeko2023fpga**; **amd2023alveo**) and neuromorphic chips
- Enhanced market microstructure analysis using VPIN and flow toxicity metrics (**easley2011vpin**; **easley2012flow**)
- Generative diffusion models for synthetic market data generation (**diffusion2024lob**)
- Scalable training with GPU-accelerated simulators (**fu2024jaxlob**)

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."