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 28, 2025

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

Contents

Part I

Main Architecture

Abstract

Financial markets have evolved into complex adaptive systems operating at timescales far beyond human perception. Modern high-frequency trading requires decision-making in microseconds, processing millions of data points across multiple modalities—price movements, order flow toxicity, news sentiment, and macroeconomic signals—while adhering to strict regulatory and risk management constraints. The challenge is not merely computational speed, but the integration of *perception*, *reasoning*, and *memory* into a unified autonomous system.

This document presents **Project JANUS**, a neuromorphic trading intelligence system inspired by the bifurcated nature of its namesake—the Roman god who simultaneously looks forward and backward. JANUS represents a paradigm shift from monolithic deep learning models to a brain-inspired architecture that mirrors the functional specialization and information flow patterns observed in biological neural systems.

The architecture is fundamentally dual:

- **JANUS Forward (Janus Bifrons)**: The "wake state" trading engine that operates in real-time market conditions. It implements visual pattern recognition through Gramian Angular Fields (GAF) and Video Vision Transformers (ViViT), symbolic reasoning through Logic Tensor Networks (LTN), multimodal fusion via gated cross-attention, and neuromorphic decision-making inspired by basal ganglia dual pathways.
- **JANUS Backward (Janus Consivius)**: The "sleep state" learning system that consolidates experiences during market closure. It replicates hippocampal replay through Sharp-Wave Ripple (SWR) simulation, forms abstract schemas via UMAP visualization and clustering, and implements recall-gated consolidation to prevent catastrophic forgetting.

This document provides the *architectural overview and philosophical foundation* of the JANUS system. Companion documents detail the mathematical specifications, implementation guides, and deployment strategies for each subsystem.

1 Introduction: The Crisis of Complexity

1.1 The Evolution of Quantitative Trading

The financial industry has undergone several paradigm shifts:

1. **Quantitative Finance 1.0 (1970s-1990s)**: Statistical arbitrage, mean reversion, and factor models based on historical correlations.
2. **Quantitative Finance 2.0 (2000s)**: High-frequency trading (HFT) with sub-millisecond latency requirements.
3. **Quantitative Finance 3.0 (2010s)**: Machine learning models (random forests, gradient boosting) trained on engineered features.
4. **Quantitative Finance 4.0 (2020s)**: Deep learning models (LSTMs, Transformers) for end-to-end feature extraction.

Each generation increased model complexity while decreasing interpretability, creating a dangerous trend toward "black box" systems that regulators, risk managers, and even their creators struggle to understand.

1.2 The Black Box Crisis

Modern ML-based trading systems face critical challenges:

- **Catastrophic Forgetting**: Neural networks trained on new market regimes (e.g., COVID-19 volatility) often forget previously learned patterns, leading to abrupt strategy failures.
- **Lack of Explainability**: Regulatory frameworks (MiFID II, SEC Rule 15c3-5) demand audit trails and explanations for algorithmic decisions, which current deep learning models cannot provide.
- **Overfitting to Noise**: Without symbolic constraints, models exploit spurious correlations (e.g., "Super Bowl indicator") that fail catastrophically in live trading.
- **Inability to Handle Novelty**: Traditional RL agents trained in simulation often fail when encountering "black swan" events not represented in training data.

The 2010 Flash Crash, the 2012 Knight Capital incident ($440M loss in 45 minutes), and numerous other algorithmic failures underscore the urgent need for *transparent, robust, and adaptive* trading systems.

1.3 The Neuromorphic Solution

JANUS addresses these challenges by emulating the brain's architecture:

Brain Region	Neuroscience Role	Trading Role
Visual Cortex	Pattern recognition in images	GAF/ViViT market pattern detection
Prefrontal Cortex	Logic, planning, rule adherence	LTN constraint enforcement
Hippocampus	Episodic memory, replay	Experience buffer, SWR simulation
Neocortex	Long-term schemas	Consolidated strategies, vector DB
Basal Ganglia	Action selection, inhibition	Dual pathways (Go/No-Go)
Cerebellum	Motor control, prediction	Market impact forward model
Amygdala	Fear detection	Circuit breakers, anomaly detection

This architecture provides:

- **Explainability**: Each decision can be traced through visual embeddings, logical constraints, and dual-pathway voting.
- **Continual Learning**: Hippocampal replay and neocortical schemas prevent catastrophic forgetting.
- **Symbolic Constraints**: LTN enforces regulatory and risk rules as logical propositions.
- **Anomaly Robustness**: UMAP visualization detects out-of-distribution market states.

2 Architectural Philosophy: The Two Faces of JANUS

2.1 Why Dual Architecture?

The mammalian brain operates in fundamentally different modes during wakefulness and sleep:

- **Wake State**: Fast, reactive, energy-efficient processing optimized for immediate survival (fight-or-flight, foraging).
- **Sleep State**: Slow, reflective, energy-intensive consolidation of experiences into long-term memory.

This duality evolved because the brain cannot simultaneously optimize for real-time reaction and deep learning. JANUS replicates this separation:

2.2 Janus Bifrons: The Forward Face

Named after the aspect of Janus that "looks forward," this service handles all real-time trading decisions during market hours.

Design Philosophy:

- **Latency-Critical**: All inference must complete in <100ms (ideally <50ms for HFT).
- **Deterministic**: No stochastic exploration; actions are fully deterministic given state.
- **Read-Only**: Does not update model weights during market hours.
- **Stateless API**: Each request is independent (RESTful or gRPC).

Core Mechanisms:

- **Visual Encoding**: Raw time series $\rightarrow$ GAF images $\rightarrow$ ViViT spatiotemporal embeddings
- **Logic Evaluation**: LTN predicates check regulatory constraints (wash sale, VPIN toxicity, Almgren-Chriss risk)
- **Multimodal Fusion**: Gated cross-attention combines visual, textual, and tabular features
- **Decision Engine**: Basal ganglia dual pathways (Direct/Indirect) vote on Go/No-Go signals

Technology Stack:

- **Language**: Rust (memory safety, zero-cost abstractions, fearless concurrency)
- **ML Framework**: ONNX Runtime (optimized C++ inference with Rust bindings)
- **Async Runtime**: Tokio (handles thousands of concurrent requests)

2.3 Janus Consivius: The Backward Face

Named after the aspect of Janus that "looks backward," this service performs offline learning during market closure (nights, weekends).

Design Philosophy:

- **Batch-Oriented**: Processes entire day's worth of experiences at once

- **Computationally Intensive**: GPU-accelerated training, UMAP fitting, vector database updates
- **Memory Consolidation**: Replays important experiences, forms abstract schemas, prunes redundant patterns
- **Safety-Gated**: Updates only occur if logical constraints are satisfied (prevents learning from violations)

Core Mechanisms:

- **Prioritized Replay**: TD-error + logical violation score + reward magnitude
- **SWR Simulation**: 10-20x time compression, mimicking hippocampal sharp-wave ripples
- **Schema Formation**: UMAP clustering detects emergent strategy patterns
- **Recall-Gated Learning**: Updates filtered by familiarity (prevents overfitting to rare events)

Technology Stack:

- **Core Logic**: Rust (parallel replay, schema clustering)
- **Training**: PyTorch (ViViT, LTN predicates) exported to ONNX
- **Vector DB**: Qdrant (stores schemas with metadata for similarity search)

2.4 Information Flow Between Services

During Market Hours (Forward Active):

1. Market data arrives $\rightarrow$ Forward service processes $\rightarrow$ Decision output
2. Transition $(\mathbf{s}, a, r, \mathbf{s}')$ logged to PostgreSQL
3. Forward service is *read-only* (no weight updates)

During Market Closure (Backward Active):

1. Backward service loads transitions from PostgreSQL
2. Prioritized replay selects important experiences
3. SWR simulation trains policy/critic networks
4. Schemas are formed and stored in Qdrant

5. Updated models exported to ONNX
6. Forward service hot-swaps new models on next market open

Key Invariant: Forward and Backward *never run simultaneously*. This prevents race conditions and ensures deterministic behavior during trading hours.

3 Core Components: Hybrid Intelligence

3.1 Vision: Seeing the Market's Geometry

3.1.1 Why Visual Encoding?

Traditional time series models (ARIMA, GARCH, LSTMs) treat market data as 1D sequences, losing critical geometric relationships:

- **Recurrent Patterns**: Head-and-shoulders, double tops, support/resistance levels are *visual* patterns
- **Multi-Scale Structure**: Intraday microstructure vs. daily trends require different receptive fields
- **Transfer Learning**: Pre-trained vision models (ViT, CLIP) excel at pattern recognition

3.1.2 Differentiable Gramian Angular Fields (DiffGAF)

The Gramian Angular Field (GAF) transforms a 1D time series $\{x_t\}$ into a 2D image by encoding pairwise temporal correlations as angular relationships:

Step 1: Normalization (Learnable)

$$\tilde{x}_t = \frac{x_t - \alpha}{\beta}, \quad \alpha, \beta \in \mathbb{R} \text{ (trainable)} \tag{1}$$

Step 2: Polar Encoding

$$\phi_t = \arccos(\tilde{x}_t), \quad r_t = \frac{t}{T} \tag{2}$$

Step 3: Gramian Angular Summation Field (GASF)

$$\text{GASF}_{i,j} = \cos(\phi_i + \phi_j) \tag{3}$$

Step 4: Gramian Angular Difference Field (GADF)

$$\text{GADF}_{i,j} = \sin(\phi_i - \phi_j) \tag{4}$$

Why This Works:

- GASF captures *summation* of angles $\rightarrow$ encodes *trend persistence*
- GADF captures *difference* of angles $\rightarrow$ encodes *reversals and volatility*
- Learnable α, β allow the model to discover optimal normalizations for different market regimes

3.1.3 GAF Video and ViViT

To capture temporal evolution, we generate a *video* of GAF images using a sliding window:

$$\mathcal{V} = \{\text{GAF}(x_{t:t+w})\}_{t=1}^{T-w} \tag{5}$$

This video is fed into a **Video Vision Transformer (ViViT)**, which applies:

- **Spatial Attention**: Identifies patterns within each frame (e.g., double top at time t)
- **Temporal Attention**: Tracks pattern evolution across frames (e.g., breakout formation)

3.2 Logic: Enforcing the Rules of the Game

3.2.1 The Necessity of Symbolic Constraints

Financial markets are governed by strict rules:

- **Regulatory**: Wash sale rules, short sale restrictions, position limits
- **Risk Management**: Maximum drawdown, Value-at-Risk (VaR), concentration limits
- **Execution Constraints**: Almgren-Chriss optimal execution, VPIN toxicity thresholds

Pure neural networks *cannot guarantee* constraint satisfaction. Even with heavy penalty terms, violations can occur during deployment.

3.2.2 Logic Tensor Networks (LTN)

LTN extends first-order logic to continuous fuzzy truth values using *t-norms*:

$$\text{LTN}(\phi) \in [0, 1], \quad \phi \text{ is a logical formula} \tag{6}$$

Example Predicates:

- $\text{WashSale}(a_{\text{sell}}, a_{\text{buy}}) \Rightarrow \text{DaysSince}(a_{\text{sell}}) > 30$
- $\text{RiskLimit}(\text{portfolio}) \Rightarrow \text{VaR}_{95}(\text{portfolio}) < 0.05 \cdot \text{NAV}$
- $\text{ToxicFlow}(\text{order}) \Rightarrow \text{VPIN} < 0.9$

Lukasiewicz T-Norm (Used in JANUS):

$$a \wedge_{\mathcal{L}} b = \max(0, a + b - 1) \quad \text{(AND)} \tag{7}$$

$$a \vee_{\mathcal{L}} b = \min(1, a + b) \quad \text{(OR)} \tag{8}$$

$$\neg_{\mathcal{L}} a = 1 - a \quad \text{(NOT)} \tag{9}$$

3.2.3 Knowledge Base Examples

Wash Sale Rule:

$$\forall a_{\text{sell}}, a_{\text{buy}} : \text{SameAsset}(a_{\text{sell}}, a_{\text{buy}}) \Rightarrow \text{TimeDiff}(a_{\text{sell}}, a_{\text{buy}}) \geq 30 \tag{10}$$

Almgren-Chriss Optimal Execution:

$$\forall \text{order} : \text{TrajectoryDeviation}(\text{order}) < \epsilon_{\text{AC}} \tag{11}$$

VPIN Toxicity:

$$\forall \text{order} : \text{VPIN}(\text{market}) < 0.9 \Rightarrow \text{Execute}(\text{order}) = \text{True} \tag{12}$$

3.3 Fusion: Integrating Multiple Realities

3.3.1 The Multimodal Challenge

Trading decisions require integrating:

- **Visual Patterns**: GAF/ViViT embeddings (shape: $[B, d_{\text{visual}}]$)
- **Textual Data**: News sentiment, earnings transcripts (BERT embeddings: $[B, d_{\text{text}}]$)
- **Tabular Features**: Volume, bid-ask spread, order book imbalance ($[B, d_{\text{tabular}}]$)

Naive concatenation fails because modalities have different:

- **Scales**: Visual embeddings $\in [-1, 1]$, tabular features $\in [0, \infty)$
- **Noise Levels**: Text is sparse and noisy, visual patterns are dense but ambiguous
- **Relevance**: Not all modalities are equally informative for all decisions

3.3.2 Gated Cross-Attention (GCA)

GCA dynamically weights modalities based on their relevance:

$$\mathbf{Q} = \mathbf{W}_Q \mathbf{x}_{\text{visual}} \tag{13}$$

$$\mathbf{K} = \mathbf{W}_K [\mathbf{x}_{\text{text}}; \mathbf{x}_{\text{tabular}}] \tag{14}$$

$$\mathbf{V} = \mathbf{W}_V [\mathbf{x}_{\text{text}}; \mathbf{x}_{\text{tabular}}] \tag{15}$$

$$\alpha = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \tag{16}$$

$$g = \sigma(\mathbf{W}_g \mathbf{x}_{\text{visual}} + \mathbf{b}_g) \tag{17}$$

$$\mathbf{z} = g \cdot (\alpha \mathbf{V}) + (1 - g) \cdot \mathbf{x}_{\text{visual}} \tag{18}$$

The gate g learns to modulate fusion strength based on context.

3.4 Decision: The Neuromorphic Motor System

3.4.1 Basal Ganglia Dual Pathways

The basal ganglia implement action selection through competing pathways:

Direct Pathway (Go Signal):

$$p_{\text{Go}}(a \mid \mathbf{s}) = \sigma(\mathbf{W}_{\text{Go}} \mathbf{z} + \mathbf{b}_{\text{Go}}) \tag{19}$$

Indirect Pathway (No-Go Signal):

$$p_{\text{No-Go}}(a \mid \mathbf{s}) = \sigma(\mathbf{W}_{\text{No-Go}} \mathbf{z} + \mathbf{b}_{\text{No-Go}}) \tag{20}$$

Final Decision:

$$a^* = \begin{cases} a & \text{if } p_{\text{Go}}(a \mid \mathbf{s}) > p_{\text{No-Go}}(a \mid \mathbf{s}) \text{ AND LTN}(\mathcal{K}, a) > \tau_{\text{logic}} \\ \text{NULL} & \text{otherwise} \end{cases} \tag{21}$$

This ensures actions are both *strategically sound* (Go pathway) and *legally compliant* (LTN filter).

3.4.2 Cerebellar Forward Model

The cerebellum predicts sensory consequences of motor actions. In trading, this translates to *market impact prediction*:

$$\hat{p}_{t+1} = f_{\text{cerebellum}}(\mathbf{s}_t, a_t) \tag{22}$$

Training Signal: Sensory Prediction Error

$$\mathcal{L}_{\text{cerebellum}} = \mathbb{E}[(p_{t+1} - \hat{p}_{t+1})^2] \tag{23}$$

This allows the system to:

- **Anticipate Slippage**: Adjust order size if predicted impact exceeds tolerance
- **Detect Manipulation**: Large deviations between predicted and actual prices signal adversarial behavior

4 Memory and Learning: The Sleeping Mind

4.1 The Three-Timescale Hierarchy

The brain consolidates memories across three timescales, each serving a distinct function:

1. **Short-Term (Hippocampus)**: Rapid encoding of episodic experiences during the day
2. **Medium-Term (Sharp-Wave Ripples)**: Offline replay and prioritization during sleep
3. **Long-Term (Neocortex)**: Slow integration into abstract schemas over weeks/-months

JANUS replicates this hierarchy to prevent catastrophic forgetting while enabling continual learning.

4.1.1 Short-Term: Hippocampal Episodic Buffer

During market hours, all transitions $(\mathbf{s}, a, r, \mathbf{s}')$ are stored in a **circular buffer**:

$$\mathcal{B}_{\text{STM}} = \{(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})\}_{t=1}^{T_{\text{day}}} \tag{24}$$

Pattern Separation: To reduce redundancy, experiences are encoded with sparse random projections:

$$\mathbf{c}_t = \text{TopK}(\text{ReLU}(\mathbf{W}_{\text{sep}}\mathbf{s}_t + \mathbf{b}_{\text{sep}}), k) \tag{25}$$

This mimics dentate gyrus function in the hippocampus, reducing interference between similar states.

4.1.2 Medium-Term: Sharp-Wave Ripple (SWR) Simulation

During sleep, the hippocampus spontaneously replays experiences at 10-20x normal speed. JANUS simulates this with **prioritized experience replay**:

Priority Score:

$$p_i = |\delta_i| + \lambda_{\text{logic}} \cdot v_i + \lambda_{\text{reward}} \cdot |r_i| \tag{26}$$

where:

- $\delta_i = r_i + \gamma V(\mathbf{s}_{i+1}) - V(\mathbf{s}_i)$ is the TD-error
- $v_i = 1 - \text{SatAgg}(\mathcal{K}, a_i)$ is the logical violation score
- r_i is the reward magnitude

Sampling Probability:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \tag{27}$$

This ensures the model focuses on:

- Surprising outcomes (high TD-error)
- Constraint violations (high v_i)
- High-impact trades (high $|r_i|$)

4.1.3 Long-Term: Neocortical Schemas

Over multiple sleep cycles, the system forms **abstract schemas** by clustering experiences in embedding space:

Schema Detection via UMAP:

1. Project experiences $\{\mathbf{s}_i\}$ to 2D using UMAP: $\{\mathbf{z}_i\} = \text{UMAP}(\{\mathbf{s}_i\})$
2. Cluster via HDBSCAN: $\text{labels} = \text{HDBSCAN}(\{\mathbf{z}_i\})$
3. For each cluster k, compute schema mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$

Recall-Gated Consolidation:

$$\Delta \mathbf{W}_{\text{LTM}} = \eta \cdot g(\text{recall}) \cdot g(\text{logic}) \cdot \nabla_{\mathbf{W}} \mathcal{L}(\tau) \tag{28}$$

Updates only occur if:

- The experience is *familiar* (high recall score $\Rightarrow$ matches existing schema)
- The experience is *valid* (high logic score $\Rightarrow$ satisfies LTN constraints)

4.2 Cognitive Visualization: UMAP

4.2.1 AlignedUMAP for Schema Detection

Standard UMAP projects each day's data independently, making it impossible to track schema evolution over time. **AlignedUMAP** solves this by enforcing consistency across multiple embeddings:

$$\mathcal{L}_{\text{aligned}} = \sum_{t=1}^{T} \mathcal{L}_{\text{UMAP}}^{(t)} + \lambda \sum_{t=1}^{T-1} \sum_{i} ||\mathbf{z}_i^{(t)} - \mathbf{z}_i^{(t+1)}||^2 \tag{29}$$

This allows analysts to visualize:

- Emergence of new market regimes (new clusters appear)
- Degradation of old strategies (clusters shrink or merge)

4.2.2 Parametric UMAP for Anomaly Detection

Parametric UMAP trains a neural network f_θ to approximate the projection:

$$\mathbf{z} = f_\theta(\mathbf{s}) \tag{30}$$

At inference time, we detect anomalies by measuring distance to known clusters:

$$d_{\text{anomaly}} = \min_k ||\mathbf{z} - \boldsymbol{\mu}_k|| \tag{31}$$

If $d_{\text{anomaly}} > \tau_{\text{anomaly}}$, the system triggers a circuit breaker (halts trading, alerts human operator).

5 Implementation Strategy: Rust-First Architecture

5.1 Why Rust?

- **Memory Safety**: No null pointers, no buffer overflows, no data races (guaranteed at compile time)
- **Zero-Cost Abstractions**: High-level constructs (iterators, traits) compile to machine code as fast as hand-written C
- **Fearless Concurrency**: Type system prevents data races, making parallelism safe and easy
- **Ecosystem**: Growing ML support (tch-rs, ort, candle, ndarray)

5.2 Service Architecture

- **Forward Service**: Rust (async gRPC server) with ONNX Runtime
- **Backward Service**: Rust (batch processing) with PyTorch training gateway
- **Gateway API**: Python FastAPI (developer-friendly REST API)
- **Vector DB**: Qdrant (Rust-native, fast similarity search)

5.3 ML Framework Migration Path

Phase 1 (Months 1-3): Hybrid Python/Rust

- Train ViViT and LTN models in PyTorch
- Export to ONNX
- Run inference in Rust via ONNX Runtime

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

- Migrate to tch-rs (libtorch bindings) for faster inference
- Keep training in PyTorch

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

- Migrate to Candle (pure Rust ML framework)
- Full end-to-end Rust pipeline

5.4 Deployment Architecture

- **Development**: Docker Compose (local testing)
- **Staging**: Kubernetes (AWS EKS or GCP GKE)
- **Production**: Kubernetes with Istio service mesh
- **Monitoring**: Prometheus + Grafana + Jaeger (distributed tracing)

6 Safety and Compliance: The Glass Box

6.1 Architectural Invariants

Hard Constraints (Enforced by LTN):

- No wash sales within 30 days
- No short sales during SEC uptick rule violations
- No orders when VPIN > 0.9 (toxic flow detected)
- Maximum position size per asset: 5% of NAV
- Maximum daily drawdown: 3% of NAV

Soft Constraints (Penalty in Reward Function):

- Minimize slippage (Almgren-Chriss optimal execution)
- Minimize adverse selection (market impact model)
- Maximize fill rate (execution quality)

Fail-Safe Mechanisms:

- If LTN score $< \tau_{\text{logic}}$, action is **blocked** (not just penalized)
- If UMAP anomaly score $> \tau_{\text{anomaly}}$, trading is **paused**
- If Sharpe ratio drops below 0.5 for 3 consecutive days, strategy is **halted**

6.2 Explainability and Auditability

Every decision is logged with:

- **Visual Embedding**: GAF image + ViViT attention maps
- **Logic Scores**: LTN predicate evaluations (which constraints were active?)
- **Dual Pathway Votes**: p_{Go} vs. $p_{\text{No-Go}}$ for each action
- **Schema Assignment**: Which long-term strategy cluster was activated?

This enables post-hoc analysis:

- "Why did the system buy AAPL at 10:37 AM?"
- Answer: Visual pattern matched "bullish engulfing" schema, LTN confirmed no wash sale violation, Go pathway voted 0.87 vs. No-Go 0.13

6.3 Circuit Breakers and Kill Switches

Amygdala Module (Fear Detection):

- Monitors real-time P&L, volatility, and drawdown
- Triggers immediate halt if:
  - Drawdown > 5% in any 30-minute window
  - Correlation with VIX $>$ 0.9 (market panic)
  - Bid-ask spread widens by >300% (liquidity crisis)

Human Override:

- Dashboard allows risk managers to halt trading with one click
- Alerts sent via Slack/PagerDuty if anomalies detected

7 Validation and Testing: Proving Robustness

7.1 Simulation and Backtesting

Level 1: Synthetic Data

- Geometric Brownian Motion (GBM) with jumps
- Heston stochastic volatility model
- Agent-based market simulation (zero-intelligence traders)

Level 2: Historical Data

- Walk-forward backtesting (train on Year 1, test on Year 2, roll forward)
- Out-of-sample testing on 2008 financial crisis, 2020 COVID crash
- Cross-asset validation (equities, futures, FX)

Level 3: Paper Trading

- Live market data, simulated execution
- Monitor slippage, fill rates, adverse selection
- Compare with baseline strategies (TWAP, VWAP, Almgren-Chriss)

Level 4: Live Trading (Minimal Capital)

- Start with $10K-$100K allocation
- Gradual ramp-up as Sharpe ratio stabilizes
- Continuous monitoring of reality gap (sim vs. live performance)

7.2 Comparative Benchmarks

Metric	JANUS Target	Baseline (LSTM + RL)
Sharpe Ratio	>2.0	1.2-1.5
Maximum Drawdown	<10%	15-25%
Constraint Violations	0 (hard guarantee)	1-2% of trades
Catastrophic Forgetting	<5% degradation over 6 months	20-30% degradation
Latency (99th percentile)	<100ms	<200ms
Explainability Score	>0.8 (human-ratable)	0.2-0.4 (black box)

8 Future Directions: Towards Quant 5.0

8.1 Quantum Computing Integration

- **Portfolio Optimization**: Quantum annealing for quadratic programming (QAOA)
- **Monte Carlo Simulation**: Quantum amplitude estimation for VaR calculation
- **Option Pricing**: Quantum algorithms for Black-Scholes PDE solving

8.2 Continual Learning and Meta-Learning

- **Meta-RL**: Learn to adapt quickly to new market regimes (MAML, Reptile)
- **Elastic Weight Consolidation**: Prevent catastrophic forgetting without experience replay
- **Neural Architecture Search**: Automatically discover optimal network topologies for new assets

8.3 Multi-Agent Cooperation and Competition

- **Multi-Agent RL**: Multiple JANUS instances trade different strategies, share knowledge
- **Adversarial Training**: One agent simulates market manipulator, other learns to detect/avoid
- **Cooperative Execution**: Distribute large orders across multiple brokers to minimize impact

8.4 Regulatory AI and Automated Compliance

- **Regulatory Knowledge Graph**: Encode SEC rules, MiFID II directives as LTN predicates
- **Automated Auditing**: LTN generates human-readable compliance reports
- **Proactive Compliance**: System suggests regulatory changes before violations occur

9 Conclusion: The Path Forward

Project JANUS represents a paradigm shift in quantitative finance—from *black box* deep learning to *glass box* neuromorphic intelligence. By emulating the brain's dual-process architecture (wake/sleep, fast/slow, reactive/reflective), JANUS achieves:

- **Robustness**: Continual learning without catastrophic forgetting
- **Safety**: Hard logical constraints prevent regulatory violations
- **Transparency**: Every decision is explainable and auditable
- **Performance**: Sub-100ms latency with Rust-native inference

The system is not a replacement for human traders, but a *cognitive prosthetic*—augmenting human intuition with machine precision, while maintaining the ability to explain its reasoning.

Key Innovations:

1. **Visual Time Series Encoding**: GAF + ViViT for spatiotemporal pattern recognition
2. **Neuro-Symbolic AI**: LTN for differentiable logic + constraint satisfaction
3. **Hippocampal Replay**: SWR simulation for prioritized experience consolidation
4. **Schema-Based Memory**: UMAP clustering for abstract strategy formation
5. **Dual-Pathway Decision**: Basal ganglia Go/No-Go architecture
6. **Rust-First ML**: Memory-safe, high-performance inference

9.1 Companion Documents

This document provides the *architectural overview*. For detailed specifications, see:

1. **JANUS Forward Service**: Mathematical formulations for GAF, ViViT, LTN, and decision pathways
2. **JANUS Backward Service**: Detailed algorithms for PER, SWR simulation, schema consolidation, and UMAP
3. **Neuromorphic Architecture Map**: Brain region $\rightarrow$ trading component mapping with biological justifications
4. **Rust Implementation Guide**: Production deployment, Docker/Kubernetes configs, monitoring setup

9.2 Call to Action

The future of quantitative finance demands systems that are:

- **Intelligent** (learn from experience)
- **Interpretable** (explain their decisions)
- **Compliant** (guarantee regulatory adherence)
- **Robust** (handle black swan events)

JANUS is a blueprint for achieving all four. The code is open-source, the architecture is reproducible, and the vision is clear: *trading systems should be as trustworthy as the markets they serve*.

References

1. Wang, Z., & Oates, T. (2015). *Imaging time-series to improve classification and imputation*. Proceedings of IJCAI.
2. Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., & Schmid, C. (2021). *ViViT: A Video Vision Transformer*. ICCV 2021.
3. Badreddine, S., d'Avila Garcez, A., Serafini, L., & Spranger, M. (2022). *Logic Tensor Networks*. Artificial Intelligence, 303, 103649.
4. Almgren, R., & Chriss, N. (2001). *Optimal execution of portfolio transactions*. Journal of Risk, 3, 5-40.

5. Easley, D., López de Prado, M. M., & O'Hara, M. (2012). *Flow toxicity and liquidity in a high-frequency world*. The Review of Financial Studies, 25(5), 1457-1493.

6. Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). *Prioritized experience replay*. ICLR 2016.

7. Buzsáki, G. (2015). *Hippocampal sharp wave-ripple: A cognitive biomarker for episodic memory and planning*. Hippocampus, 25(10), 1073-1188.

8. McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). *Why there are complementary learning systems in the hippocampus and neocortex*. Psychological Review, 102(3), 419.

9. McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. arXiv:1802.03426.

10. Ding, Z., Wijaya, D., & Saligrama, V. (2021). *AlignedUMAP: Aligned Uniform Manifold Approximation and Projection*. NeurIPS 2021 Workshop on Learning Meaningful Representations of Life.

Appendix: Implementation Checklist

Forward Service (Rust)

- ☐ GAF transformation module with learnable parameters
- ☐ GAF video generation with sliding windows
- ☐ ONNX Runtime integration for ViViT inference
- ☐ LTN predicate evaluation engine
- ☐ Lukasiewicz t-norm implementations
- ☐ Gated cross-attention fusion
- ☐ Basal ganglia dual pathways (Go/No-Go)
- ☐ Cerebellar forward model for slippage prediction
- ☐ gRPC server with Tokio async runtime
- ☐ Prometheus metrics exporter
- ☐ Health check endpoints
- ☐ Model hot-swapping mechanism

Backward Service (Rust)

- ☐ Prioritized replay buffer with SumTree
- ☐ SWR simulation with time compression
- ☐ Importance sampling correction
- ☐ Schema formation via clustering
- ☐ Recall-gated consolidation
- ☐ Qdrant client for vector database
- ☐ UMAP projection (ONNX or native)
- ☐ AlignedUMAP for multi-epoch analysis
- ☐ Parallel batch processing with Rayon
- ☐ PostgreSQL client for experience logging

Training Gateway (Python)

- ☐ PyTorch training loop
- ☐ FastAPI REST endpoints
- ☐ Celery task queue for async jobs
- ☐ ONNX export pipeline
- ☐ Model validation scripts
- ☐ Hyperparameter tuning (Optuna)
- ☐ Experiment tracking (MLflow or Weights & Biases)

Infrastructure

- ☐ Docker Compose for local development
- ☐ Kubernetes manifests (Deployments, Services, ConfigMaps)
- ☐ Helm charts for versioned releases
- ☐ PostgreSQL for metadata
- ☐ Qdrant for vector storage

- ☐ Redis for pub/sub and caching
- ☐ Prometheus + Grafana monitoring
- ☐ CI/CD pipeline (GitHub Actions or GitLab CI)

Validation

- ☐ Unit tests (>80% coverage)
- ☐ Integration tests (end-to-end flows)
- ☐ Synthetic market simulations
- ☐ Historical backtest suite
- ☐ Black swan stress tests
- ☐ Paper trading validation
- ☐ Latency profiling
- ☐ Reality gap analysis

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.

10 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.

10.1 Mathematical Foundation: Gramian Angular Fields

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

10.1.1 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.

10.1.2 Step 1: Learnable Normalization

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

$$\tilde{x}_t = \gamma \odot \frac{x_t - \mu}{\sigma} + \beta \tag{32}$$

where $\gamma, \beta \in \mathbb{R}^F$ are learned parameters, and μ, σ are running statistics.

10.1.3 Step 2: Polar Coordinate Transformation

Map normalized values to angular space:

$$\phi_t = \arccos(\tilde{x}_t) \in [0, \pi] \tag{33}$$

$$r_t = \frac{t}{T} \quad \text{(normalized timestamp)} \tag{34}$$

10.1.4 Step 3: Gramian Field Generation

Construct the Gramian Angular Summation Field (GASF):

$$\mathbf{G}_{ij} = \cos(\phi_i + \phi_j) = \tilde{x}_i \tilde{x}_j - \sqrt{1 - \tilde{x}_i^2}\sqrt{1 - \tilde{x}_j^2} \tag{35}$$

Or the Gramian Angular Difference Field (GADF):

$$\mathbf{G}_{ij} = \sin(\phi_i - \phi_j) = \sqrt{1 - \tilde{x}_i^2}\tilde{x}_j - \tilde{x}_i\sqrt{1 - \tilde{x}_j^2} \tag{36}$$

10.2 3D Spatiotemporal Manifolds: GAF Video

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

10.2.1 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: $\mathbf{V} = [\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_N] \in \mathbb{R}^{N \times W \times W}$

10.3 Video Vision Transformer (ViViT)

The GAF video is processed by a factorized spatiotemporal transformer.

10.3.1 Patch Embedding

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

$$\mathbf{P}_k = \text{Reshape}(\mathbf{G}_k) \in \mathbb{R}^{P \times (p^2)} \tag{37}$$

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

10.3.2 Spatial Attention

Apply self-attention within each frame:

$$\mathbf{Z}_k^{(l)} = \text{MSA}(\text{LN}(\mathbf{Z}_k^{(l-1)})) + \mathbf{Z}_k^{(l-1)} \tag{38}$$

10.3.3 Temporal Attention

Apply attention across frames:

$$\mathbf{H}^{(l)} = \text{MSA}(\text{LN}([\mathbf{Z}_1^{(l)}, \ldots, \mathbf{Z}_N^{(l)}])) \tag{39}$$

11 Logic Tensor Networks: Symbolic Reasoning Engine

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

11.1 Mathematical Foundation

11.1.1 Grounding Function

Map logical constants to real vectors:

$$\mathcal{G} : \mathcal{C} \to \mathbb{R}^d \tag{40}$$

11.1.2 Predicate Grounding

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

11.2 Lukasiewicz T-Norm Operations

11.2.1 Conjunction (AND)

$$u \wedge v = \max(0, u + v - 1) \tag{41}$$

11.2.2 Disjunction (OR)

$$u \vee v = \min(1, u + v) \tag{42}$$

11.2.3 Negation (NOT)

$$\neg u = 1 - u \tag{43}$$

11.2.4 Implication (IF-THEN)

$$u \Rightarrow v = \min(1, 1 - u + v) \tag{44}$$

11.3 Knowledge Base Formulation

11.3.1 Wash Sale Constraint

$$\forall t : \text{Sell}(t) \wedge \text{Buy}(t') \wedge |t - t'| < 30 \Rightarrow \neg \text{TaxLoss}(t) \tag{45}$$

11.3.2 Almgren-Chriss Risk Constraint

$$\forall \text{order} : \text{Execute}(\text{order}) \Rightarrow \text{Slippage}(\text{order}) < \lambda \cdot \text{Volatility} \tag{46}$$

11.4 Logical Loss Function

11.4.1 Satisfiability Aggregation

$$\text{SAT}(\mathcal{KB}) = \text{p-mean}_{i=1}^{|\mathcal{KB}|}(\phi_i) \tag{47}$$

11.4.2 Logical Loss

$$\mathcal{L}_{\text{logic}} = 1 - \text{SAT}(\mathcal{KB}) \tag{48}$$

12 Multimodal Fusion: Gated Cross-Attention

12.1 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)

12.2 Gated Cross-Attention Mechanism

12.2.1 Attention Computation

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{49}$$

12.2.2 Gating Mechanism

$$g = \text{sigmoid}(\mathbf{W}_g[\mathbf{v}; \mathbf{t}; \mathbf{s}] + \mathbf{b}_g) \tag{50}$$

13 Decision Engine: Basal Ganglia Pathways

13.1 Praxeological Motor: Dual Pathways

13.1.1 Direct Pathway (Go Signal)

$$\mathbf{d}_{\text{direct}} = \text{ReLU}(\mathbf{W}_d \mathbf{h}_{\text{fused}} + \mathbf{b}_d) \tag{51}$$

13.1.2 Indirect Pathway (No-Go Signal)

$$\mathbf{d}_{\text{indirect}} = \text{ReLU}(\mathbf{W}_i \mathbf{h}_{\text{fused}} + \mathbf{b}_i) \tag{52}$$

13.2 Cerebellar Forward Model

13.2.1 Market Impact Prediction

$$\hat{p}_{t+1} = f_{\text{cerebellum}}(\mathbf{s}_t, \mathbf{a}_t) \tag{53}$$

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 → SWR → 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.

14 Memory Hierarchy: Three-Timescale Architecture

14.1 Short-Term Memory (Hippocampus)

14.1.1 Episodic Buffer

Stores raw experiences during trading:

$$\mathcal{D}_{\text{hippo}} = \{(s_t, a_t, r_t, s_{t+1}, \mathbf{c}_t)\}_{t=1}^{T} \tag{54}$$

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

14.1.2 Pattern Separation

Uses random projections to ensure diverse encoding:

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\text{rand}} \cdot [s_t; a_t; \mathbf{c}_t]) \tag{55}$$

14.2 Medium-Term Consolidation (SWR Simulator)

14.2.1 Replay Prioritization

Compute TD-error based priority:

$$p_i = |\delta_i|^\alpha + \epsilon \tag{56}$$

where $\delta_i = r_i + \gamma \max_{a'} Q(s_{i+1}, a') - Q(s_i, a_i)$.

14.2.2 Sampling Probability

$$P(i) = \frac{p_i^\beta}{\sum_j p_j^\beta} \tag{57}$$

14.2.3 Importance Sampling Correction

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta \tag{58}$$

14.3 Long-Term Memory (Neocortex)

14.3.1 Schema Representation

Each schema is a prototype:

$$\mathbf{z}_k = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \mathbf{h}_i \tag{59}$$

14.3.2 Recall-Gated Consolidation

Only update schemas from successfully recalled experiences:

$$\mathbf{z}_k \leftarrow \mathbf{z}_k + \eta \cdot \mathbb{1}[\text{recall\_success}] \cdot (\mathbf{h}_{\text{new}} - \mathbf{z}_k) \tag{60}$$

15 UMAP Visualization: Cognitive Dashboard

15.1 AlignedUMAP for Schema Formation

Maintains consistent embeddings across sleep cycles.

15.1.1 Objective Function

$$\mathcal{L}_{\text{UMAP}} = \sum_{ij} w_{ij} \log \left(\frac{w_{ij}}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} \right) \tag{61}$$

15.2 Parametric UMAP for Real-Time Monitoring

Train a neural network to project new experiences:

$$\mathbf{y}_{\text{new}} = f_\theta(\mathbf{h}_{\text{new}}) \tag{62}$$

16 Integration with Vector Database (Qdrant)

16.1 Schema Storage

Each schema stored as:

```
{
    "id": schema_id,
    "vector": z_k,
    "payload": {
        "centroid": z_k,
        "count": |C_k|,
        "avg_reward": mean(rewards),
        "volatility": std(returns)
    }
}
```

16.2 Similarity Search

Retrieve nearest schemas:

$$\mathcal{N}_k = \arg\max_k \text{cosine}(\mathbf{h}_t, \mathbf{z}_k) \tag{63}$$

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

17 Neuromorphic Design Philosophy

17.1 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

17.2 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

18 Brain Region Architectures

18.1 Cortex: Strategic Planning & Long-term Memory

18.1.1 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

18.2 Hippocampus: Episodic Memory & Experience Replay

18.2.1 Trading Implementation

Component: Episodic Buffer + SWR Replay

Implementation:

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

18.3 Basal Ganglia: Action Selection & Reinforcement Learning

18.3.1 Trading Implementation

Component: Dual-Pathway Decision Module

Direct Pathway: Excitatory (go signal)

```
fn direct_pathway(state: &Tensor) -> Tensor {
    state.linear(W_direct).relu()
}
```

Indirect Pathway: Inhibitory (no-go signal)

```
fn indirect_pathway(state: &Tensor) -> Tensor {
    state.linear(W_indirect).relu()
}
```

18.4 Prefrontal Cortex: Logic, Planning & Compliance

18.4.1 Trading Implementation

Component: Logic Tensor Network

Ensures regulatory compliance:

- Wash sale rules
- Position limits
- Capital allocation constraints

18.5 Amygdala: Fear, Threat Detection & Circuit Breakers

18.5.1 Trading Implementation

Component: Anomaly Detection Module

Triggers emergency stops:

```
if mahalanobis_distance(state, historical_mean) > threshold {
    trigger_circuit_breaker();
}
```

18.6 Cerebellum: Motor Control & Execution

18.6.1 Trading Implementation

Component: Forward Model for Market Impact

Predicts price movement from order execution:

$$\Delta p = f_{\text{cerebellum}}(\text{order\_size}, \text{liquidity}, \text{volatility}) \tag{64}$$

Part V

Rust Implementation

Abstract

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

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

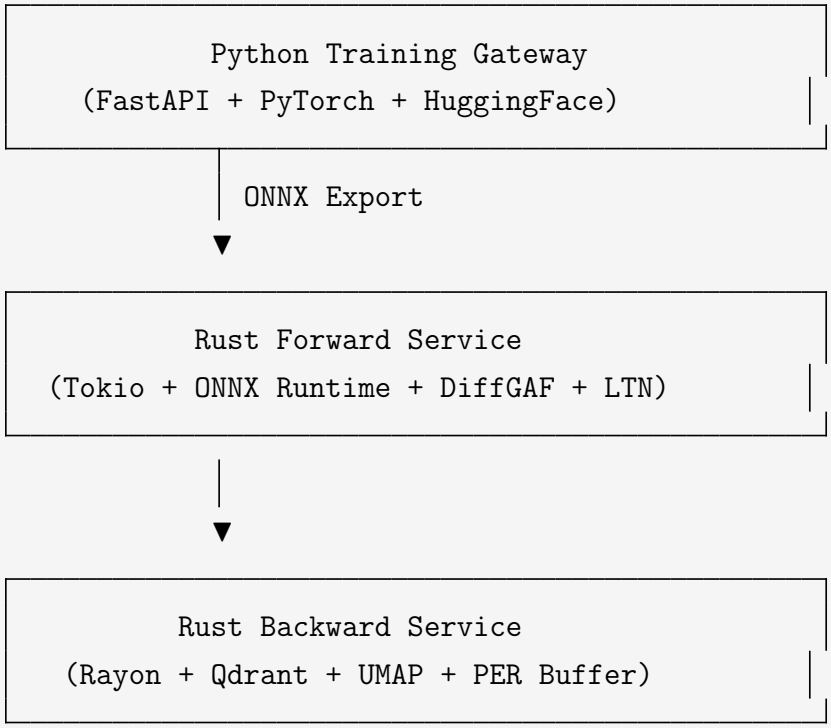
19 Architectural Overview

19.1 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

19.2 Component Diagram

System Architecture



20 Machine Learning Framework Strategy

20.1 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

20.2 Recommended Migration Path

20.2.1 Phase 1: Hybrid (Months 1-3)

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

- Rust inference via `ort` crate

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

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

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

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

21 Forward Service: Rust Implementation

21.1 Performance Requirements

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

21.2 Core Data Structures

```rust
#[derive(Debug, Clone)]
pub struct MarketState {
    pub timestamp: i64,
    pub features: Vec<f32>,
    pub orderbook: OrderBook,
    pub context: MarketContext,
}

#[derive(Debug, Clone)]
pub struct OrderBook {
    pub bids: Vec<(f32, f32)>, // (price, quantity)
    pub asks: Vec<(f32, f32)>,
}
```

21.3 GAF Transformation Module

```rust
pub fn compute_gaf(
    time_series: &[f32],
    window_size: usize,
) -> Result<Tensor> {
    // Normalize to [-1, 1]
    let normalized = normalize(time_series)?;

    // Compute angles
    let angles: Vec<f32> = normalized
        .iter()
        .map(|&x| x.acos())
        .collect();

    // Build Gramian matrix
    let mut gaf = vec![0.0; window_size * window_size];
    for i in 0..window_size {
        for j in 0..window_size {
            gaf[i * window_size + j] =
                (angles[i] + angles[j]).cos();
        }
    }

    Ok(Tensor::from_slice(&gaf)
        .reshape(&[1, window_size, window_size]))
}
```

21.4 LTN Constraint Evaluation

```rust
pub struct LTNConstraint {
    predicate: Box<dyn Fn(&MarketState) -> f32>,
    weight: f32,
}

impl LTNConstraint {
    pub fn evaluate(&self, state: &MarketState) -> f32 {
        (self.predicate)(state) * self.weight
    }
}

```

```rust
// Lukasiewicz T-norm operations
pub fn ltn_and(a: f32, b: f32) -> f32 {
    (a + b - 1.0).max(0.0)
}

pub fn ltn_implies(a: f32, b: f32) -> f32 {
    (1.0 - a + b).min(1.0)
}
```

21.5 Async Service with Tokio

```rust
#[tokio::main]
async fn main() -> Result<()> {
    let model = OnnxModel::load("models/vivit.onnx")?;
    let ltn_engine = LTNEngine::new();

    let listener = TcpListener::bind("0.0.0.0:8080").await?;

    loop {
        let (socket, _) = listener.accept().await?;
        let model_clone = model.clone();

        tokio::spawn(async move {
            handle_request(socket, model_clone).await
        });
    }
}
```

22 Backward Service: Batch Processing

22.1 Prioritized Experience Replay

```rust
pub struct PrioritizedBuffer {
    buffer: Vec<Experience>,
    priorities: Vec<f32>,
    capacity: usize,
    alpha: f32,
    beta: f32,
}

```

```rust
impl PrioritizedBuffer {
    pub fn sample(&self, batch_size: usize) -> Vec<Experience> {
        let probabilities = self.compute_probabilities();

        // Sample with replacement
        let mut rng = thread_rng();
        let dist = WeightedIndex::new(&probabilities).unwrap();

        (0..batch_size)
            .map(|_| self.buffer[dist.sample(&mut rng)].clone())
            .collect()
    }

    fn compute_probabilities(&self) -> Vec<f32> {
        let sum: f32 = self.priorities
            .iter()
            .map(|p| p.powf(self.alpha))
            .sum();

        self.priorities
            .iter()
            .map(|p| p.powf(self.alpha) / sum)
            .collect()
    }
}
```

22.2 Schema Consolidation

```rust
pub async fn consolidate_schemas(
    experiences: &[Experience],
    qdrant_client: &QdrantClient,
) -> Result<()> {
    // Extract embeddings
    let embeddings: Vec<Vec<f32>> = experiences
        .iter()
        .map(|e| extract_embedding(e))
        .collect();

    // Cluster with k-means
    let clusters = kmeans(&embeddings, num_clusters)?;

```

```
    // Update schemas in Qdrant
    for (cluster_id, members) in clusters.iter().enumerate() {
        let centroid = compute_centroid(members);

        qdrant_client.upsert_point(
            "schemas",
            cluster_id,
            centroid,
            json!({
                "count": members.len(),
                "avg_reward": compute_avg_reward(members),
            }),
        ).await?;
    }

    Ok(())
}
```

23 Deployment Architecture

23.1 Docker Compose Setup

```
version: '3.8'

services:
  forward:
    build:
      context: .
      dockerfile: Dockerfile.forward
    ports:
      - "8080:8080"
    environment:
      - RUST_LOG=info
      - MODEL_PATH=/models/vivit.onnx
    volumes:
      - ./models:/models

  backward:
    build:
      context: .
```

```
      dockerfile: Dockerfile.backward
    environment:
      - RUST_LOG=info
      - QDRANT_URL=http://qdrant:6334
    depends_on:
      - qdrant

  qdrant:
    image: qdrant/qdrant:latest
    ports:
      - "6333:6333"
      - "6334:6334"
    volumes:
      - qdrant_storage:/qdrant/storage

volumes:
  qdrant_storage:
```

24 Implementation Roadmap

24.1 Phase 1: Foundation (Weeks 1-4)

1. Set up Rust workspace with Cargo
2. Implement core data structures
3. Create ONNX inference pipeline
4. Build basic async service skeleton

24.2 Phase 2: Core Features (Weeks 5-8)

1. Implement DiffGAF transformation
2. Build LTN constraint engine
3. Create prioritized replay buffer
4. Integrate Qdrant for schema storage

24.3 Phase 3: Production Readiness (Weeks 9-12)

1. Performance optimization

2. Comprehensive testing
3. Monitoring and logging
4. Documentation and deployment guides

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."