JANUS FORWARD

Wake State: Logic Trading Algorithm

Real-Time Decision Making, Pattern Recognition, and Trade Execution

Classification: Technical Implementation Guide

Version: 1.0 (Implementation-Ready)

Author: Jordan Smith
github.com/nuniesmith

Date: December 25, 2025

JANUS Forward Overview:

- **Purpose:** Real-time trading decisions during market hours
- **Hot Path:** Low-latency, high-throughput execution
- **Components:** Visual pattern recognition, symbolic reasoning, multimodal fusion, execution control
- **Goal:** Neuro-symbolic trading that combines deep learning with logical constraints

Abstract

JANUS Forward represents the "wake state" of the JANUS trading system, responsible for all real-time decision-making during market hours. This document provides a comprehensive mathematical and implementation specification for the Forward service, which 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.

Contents

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

1.1.1 Input Preprocessing

Given a univariate time series of length N:

$$X = \{x_1, x_2, \ldots, x_N\} \in \mathbb{R}^N \tag{1}$$

1.1.2 Step 1: Learnable Normalization

The time series is normalized using learnable parameters $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$:

$$\tilde{x}_i = \tanh\left(\frac{x_i - \min(X)}{\max(X) - \min(X) + \epsilon} \cdot \alpha + \beta\right) \tag{2}$$

where $\epsilon = 10^{-8}$ prevents division by zero. The normalized values $\tilde{x}_i \in [-1, 1]$ are constrained to the domain of the cosine function.

Implementation Note: α and β are learnable parameters initialized as:

$$\alpha_0 = 1.0 \tag{3}$$

$$\beta_0 = 0.0 \tag{4}$$

These are optimized via backpropagation through the entire pipeline.

1.1.3 Step 2: Polar Coordinate Transformation

Each normalized value is mapped to polar coordinates:

$$\begin{cases} \phi_i = \arccos(\tilde{x}_i), & \tilde{x}_i \in [-1, 1] \\ r_i = \frac{i}{N}, & i \in \{1, 2, \ldots, N\} \end{cases} \tag{5}$$

where $\phi_i \in [0, \pi]$ is the angular component and $r_i \in [0, 1]$ is the radial component (normalized time index).

1.1.4 Step 3: Gramian Field Generation

Two Gramian Angular Fields are computed:

Gramian Angular Summation Field (GASF):

$$\text{GASF}_{i,j} = \cos(\phi_i + \phi_j) = \tilde{x}_i \tilde{x}_j - \sqrt{1 - \tilde{x}_i^2}\sqrt{1 - \tilde{x}_j^2} \tag{6}$$

Gramian Angular Difference Field (GADF):

$$\text{GADF}_{i,j} = \sin(\phi_i - \phi_j) = \sqrt{1 - \tilde{x}_i^2}\tilde{x}_j - \tilde{x}_i\sqrt{1 - \tilde{x}_j^2} \tag{7}$$

The result is two $N \times N$ matrices (images) where:

- The main diagonal ($i = j$) contains the original normalized values
- Off-diagonal elements encode temporal correlations
- GASF captures summation relationships
- GADF captures difference relationships

1.1.5 Implementation Algorithm

Algorithm 1 DiffGAF Transformation

Require: Time series $X \in \mathbb{R}^N$, learnable params α, β
Ensure: GASF and GADF matrices $\in \mathbb{R}^{N \times N}$

$X_{\min} \leftarrow \min(X)$, $X_{\max} \leftarrow \max(X)$
$\tilde{X} \leftarrow \tanh\left(\frac{X - X_{\min}}{X_{\max} - X_{\min} + \epsilon} \cdot \alpha + \beta\right)$
$\Phi \leftarrow \arccos(\tilde{X})$ ▷ Element-wise arccos
Initialize GASF $\leftarrow \mathbf{0}_{N \times N}$, GADF $\leftarrow \mathbf{0}_{N \times N}$
for $i = 1$ to N **do**
    for $j = 1$ to N **do**
        $\text{GASF}_{i,j} \leftarrow \cos(\Phi_i + \Phi_j)$
        $\text{GADF}_{i,j} \leftarrow \sin(\Phi_i - \Phi_j)$
    end for
end for
return GASF, GADF

1.2 3D Spatiotemporal Manifolds: GAF Video

1.2.1 Sliding Window GAF Video Generation

Given a time series $X = \{x_1, x_2, \ldots, x_T\}$ of length T, we generate a sequence of overlapping GAF frames:

$$\mathcal{V} = \{GAF(X_{t:t+w}), GAF(X_{t+s:t+w+s}), \ldots, GAF(X_{t+(F-1)s:t+w+(F-1)s})\} \tag{8}$$

where:

- w = window size (e.g., 60 timesteps)
- s = stride (e.g., 10 timesteps)
- F = number of frames (e.g., 16 frames)

Each frame is a $2 \times N \times N$ tensor (GASF + GADF channels). The complete video tensor is:

$$\mathcal{V} \in \mathbb{R}^{F \times 2 \times N \times N} \tag{9}$$

1.2.2 Mathematical Formulation

For frame $f \in \{0, 1, \ldots, F-1\}$:

$$\mathcal{V}_f = \begin{bmatrix} \text{GASF}(X_{t+fs:t+w+fs}) \\ \text{GADF}(X_{t+fs:t+w+fs}) \end{bmatrix} \tag{10}$$

1.3 Video Vision Transformer (ViViT)

1.3.1 Architecture Overview

ViViT processes the 3D tensor $\mathcal{V}$ using factorized spatial-temporal attention.

1.3.2 Patch Embedding

Each frame is divided into patches. For a frame of size $H \times W$ with patch size P:

$$N_p = \frac{H}{P} \times \frac{W}{P} \tag{11}$$

patches per frame.

The patch embedding for patch (i, j) in frame f is:

$$\mathbf{z}_{f,i,j}^{(0)} = \mathbf{E} \cdot \text{flatten}(\mathcal{V}_{f,i:i+P,j:j+P}) + \mathbf{p}_{f,i,j} \tag{12}$$

where:

- $\mathbf{E} \in \mathbb{R}^{d \times (2P^2)}$ is the embedding matrix
- $\mathbf{p}_{f,i,j}$ is the positional encoding (spatial + temporal)

1.3.3 Spatial Attention

Within each frame f, spatial self-attention is computed:

$$\text{Attention}(\mathbf{Q}_s, \mathbf{K}_s, \mathbf{V}_s) = \text{softmax}\left(\frac{\mathbf{Q}_s \mathbf{K}_s^\top}{\sqrt{d_h}}\right)\mathbf{V}_s \tag{13}$$

where $\mathbf{Q}_s, \mathbf{K}_s, \mathbf{V}_s$ are queries, keys, and values from spatial patches.

1.3.4 Temporal Attention

Across frames, temporal attention captures evolution:

$$\text{Attention}(\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t) = \text{softmax}\left(\frac{\mathbf{Q}_t \mathbf{K}_t^\top}{\sqrt{d_h}}\right)\mathbf{V}_t \tag{14}$$

1.3.5 Output Embedding

The final visual embedding vector is:

$$\mathbf{e}_{\text{visual}} = \text{MLP}(\text{GlobalPool}(\mathbf{Z}^{(L)})) \in \mathbb{R}^{d_{\text{embed}}} \tag{15}$$

where L is the number of transformer layers and d_{embed} is the embedding dimension (e.g., 768).

2 Logic Tensor Networks: Symbolic Reasoning Engine

The LTN subsystem ensures that all trading decisions satisfy regulatory and risk management constraints through differentiable first-order logic.

2.1 Mathematical Foundation

2.1.1 Grounding Function

Let $\mathcal{G} : \mathcal{S} \to \mathbb{R}^n$ be a grounding function that maps logical symbols to tensors:

$$\mathcal{G} : \text{Constants} \cup \text{Predicates} \cup \text{Functions} \to \mathbb{R}^n \tag{16}$$

2.1.2 Predicate Grounding

A predicate P with arity k is grounded as a neural network:

$$\mathcal{G}(P) : \mathbb{R}^{k \times d} \to [0, 1] \tag{17}$$

where d is the embedding dimension.

For example, *IsVolatile*$(market)$ is implemented as:

$$\mathcal{G}(\textit{IsVolatile})(\mathbf{e}_{\text{market}}) = \text{sigmoid}(\mathbf{W}_v \mathbf{e}_{\text{market}} + b_v) \tag{18}$$

where $\mathbf{W}_v \in \mathbb{R}^{1\times d}$ and $b_v \in \mathbb{R}$ are learnable parameters.

2.2 Lukasiewicz T-Norm Operations

2.2.1 Conjunction (AND)

$$\mathcal{G}(A \wedge B) = \max(0, \mathcal{G}(A) + \mathcal{G}(B) - 1) \tag{19}$$

2.2.2 Disjunction (OR)

$$\mathcal{G}(A \vee B) = \min(1, \mathcal{G}(A) + \mathcal{G}(B)) \tag{20}$$

2.2.3 Negation (NOT)

$$\mathcal{G}(\neg A) = 1 - \mathcal{G}(A) \tag{21}$$

2.2.4 Implication (IF-THEN)

$$\mathcal{G}(A \to B) = \min(1, 1 - \mathcal{G}(A) + \mathcal{G}(B)) \tag{22}$$

2.2.5 Universal Quantification (FOR ALL)

For a formula $\phi(x)$ with free variable x:

$$\mathcal{G}(\forall x : \phi(x)) = \min_{x\in\mathcal{D}} \mathcal{G}(\phi(x)) \tag{23}$$

where $\mathcal{D}$ is the domain of x.

2.2.6 Existential Quantification (EXISTS)

$$\mathcal{G}(\exists x : \phi(x)) = \max_{x\in\mathcal{D}} \mathcal{G}(\phi(x)) \tag{24}$$

2.3 Knowledge Base Formulation

2.3.1 Wash Sale Constraint

The wash sale rule is encoded as:

$$\forall t, \forall k \in [1, 30] : \neg(\textit{SaleAtLoss}(t) \wedge \textit{Buy}(t + k)) \tag{25}$$

In grounded form:

$$\mathcal{G}(\text{WashSale}) = \min_{t,k\in[1,30]} [1 - \max(0, \mathcal{G}(\textit{SaleAtLoss})(t) + \mathcal{G}(\textit{Buy})(t+k) - 1)] \tag{26}$$

2.3.2 Almgren-Chriss Risk Constraint

$$\forall v : \textit{Volatile}(Market) \rightarrow \textit{Impact}(v) < \textit{Threshold}(\sigma) \tag{27}$$

Grounded:

$$\mathcal{G}(\text{ACRisk}) = \min_{v} [\min(1, 1 - \mathcal{G}(\textit{Volatile}) + \mathcal{G}(\textit{Impact}(v) < \textit{Threshold}(\sigma)))] \tag{28}$$

where:

$$\textit{Threshold}(\sigma) = \eta \cdot \sigma \cdot \sqrt{\frac{v}{V}} \tag{29}$$

with η = impact coefficient, σ = volatility, v = trade size, V = average volume.

2.3.3 VPIN Toxicity Constraint

$$\forall t : \textit{VPIN}_t > \tau_{\text{VPIN}} \rightarrow \textit{HaltTrading}(t) \tag{30}$$

Grounded:

$$\mathcal{G}(\text{VPIN}) = \min_{t} [\min(1, 1 - \mathcal{G}(\textit{VPIN}_t > \tau_{\text{VPIN}}) + \mathcal{G}(\textit{HaltTrading})(t))] \tag{31}$$

2.4 Logical Loss Function

2.4.1 Satisfiability Aggregation

For a knowledge base $\mathcal{K} = \{\phi_1, \phi_2, \ldots, \phi_m\}$:

$$\text{SatAgg}(\mathcal{K}) = \left(\frac{1}{m} \sum_{i=1}^{m} \mathcal{G}(\phi_i)^p \right)^{1/p} \tag{32}$$

where p is the generalized mean parameter (typically $p = 2$ for quadratic mean).

2.4.2 Logical Loss

$$\mathcal{L}_{\text{logic}}(\theta) = 1 - \text{SatAgg}(\mathcal{K}) \tag{33}$$

2.4.3 Combined Loss

The total loss combines predictive and logical components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{predictive}} + \lambda_{\text{logic}} \cdot \mathcal{L}_{\text{logic}} \tag{34}$$

where λ_{logic} is a hyperparameter (typically 0.1 to 1.0).

3 Multimodal Fusion: Gated Cross-Attention

The fusion subsystem integrates visual, temporal, and textual market information into a unified representation.

3.1 Input Modalities

The system receives three input streams:

$$\mathbf{H}_{\text{TS}} \in \mathbb{R}^{L_{\text{TS}} \times d} \quad \text{(Time Series Tokens from Chronos-Bolt)} \tag{35}$$

$$\mathbf{H}_{\text{Vis}} \in \mathbb{R}^{L_{\text{Vis}} \times d} \quad \text{(Visual Embeddings from ViViT)} \tag{36}$$

$$\mathbf{H}_{\text{Text}} \in \mathbb{R}^{L_{\text{Text}} \times d} \quad \text{(Text Embeddings from FinBERT)} \tag{37}$$

3.2 Gated Cross-Attention Mechanism

3.2.1 Attention Computation

For primary modality m and auxiliary modality n:

$$\alpha_{m \to n} = \text{softmax}\left(\frac{\mathbf{Q}_m \mathbf{K}_n^\top}{\sqrt{d_h}}\right) \tag{38}$$

where:

$$\mathbf{Q}_m = \mathbf{H}_m \mathbf{W}_Q \tag{39}$$

$$\mathbf{K}_n = \mathbf{H}_n \mathbf{W}_K \tag{40}$$

$$\mathbf{V}_n = \mathbf{H}_n \mathbf{W}_V \tag{41}$$

3.2.2 Gating Mechanism

The gating scalar is computed as:

$$\lambda_{\text{gate}} = \text{sigmoid}(\mathbf{W}_g [\mathbf{H}_m; \mathbf{H}_n] + b_g) \tag{42}$$

where $[\cdot;\cdot]$ denotes concatenation.

3.2.3 Fused Representation

$$\mathbf{H}_{\text{fused}} = \mathbf{H}_m + \lambda_{\text{gate}} \cdot \alpha_{m \to n} \mathbf{V}_n \tag{43}$$

3.3 Multi-Modal Fusion Pipeline

The complete fusion process:

$$\mathbf{H}_1 = \mathbf{H}_{\text{TS}} + \lambda_{\text{vis}} \cdot \text{CrossAttn}(\mathbf{H}_{\text{TS}}, \mathbf{H}_{\text{Vis}}) \tag{44}$$

$$\mathbf{H}_2 = \mathbf{H}_1 + \lambda_{\text{text}} \cdot \text{CrossAttn}(\mathbf{H}_1, \mathbf{H}_{\text{Text}}) \tag{45}$$

$$\mathbf{e}_{\text{final}} = \text{GlobalPool}(\mathbf{H}_2) \tag{46}$$

4 Decision Engine: Basal Ganglia Pathways

The decision engine implements a dual-pathway architecture inspired by the basal ganglia, with separate "go" and "no-go" pathways.

4.1 Praxeological Motor: Dual Pathways

4.1.1 Direct Pathway (Go Signal)

The direct pathway generates action proposals:

$$a_{\text{proposed}} = \arg\max_{a \in \mathcal{A}} \left[\mathbf{W}_{\text{alpha}} \mathbf{e}_{\text{final}} + b_{\text{alpha}} \right]_a \tag{47}$$

where $\mathcal{A}$ is the action space (BUY, SELL, HOLD, or continuous trade sizes).

4.1.2 Indirect Pathway (No-Go Signal)

The indirect pathway computes risk veto:

$$v_{\text{risk}} = \text{sigmoid}(\mathbf{W}_{\text{risk}} [\mathbf{e}_{\text{final}}; \text{VPIN}; \sigma_{\text{market}}] + b_{\text{risk}}) \tag{48}$$

4.1.3 Final Action

The final action is gated by the risk veto:

$$a_{\text{final}} = \begin{cases} a_{\text{proposed}} & \text{if } v_{\text{risk}} < \tau_{\text{risk}} \text{ AND } \mathcal{L}_{\text{logic}} < \tau_{\text{logic}} \\ \text{HOLD} & \text{otherwise} \end{cases} \tag{49}$$

where τ_{risk} is the risk threshold (e.g., 0.7) and τ_{logic} is the logical constraint threshold (e.g., 0.1).

4.2 Cerebellar Forward Model

4.2.1 Market Impact Prediction

The forward model predicts execution price:

$$\hat{p}_{\text{exec}} = f_{\text{forward}}(\mathbf{s}_{\text{LOB}}, v, a_{\text{final}}) \tag{50}$$

where $\mathbf{s}_{\text{LOB}}$ is the limit order book state.

4.2.2 Sensory Prediction Error

$$\text{SPE} = |p_{\text{actual}} - \hat{p}_{\text{exec}}| \tag{51}$$

4.2.3 Trajectory Adjustment

The execution trajectory is adjusted:

$$v_{\text{adjusted}} = v_{\text{original}} - \eta_{\text{cerebellar}} \cdot \text{SPE} \cdot \nabla_v \text{SPE} \tag{52}$$

5 Implementation Checklist

sec:checklist

This section provides a sequential checklist for implementing JANUS Forward.

5.1 Core Components

1. **Visual Pattern Recognition Module**
   - ☐ Implement DiffGAF normalization with learnable α, β
   - ☐ Implement polar coordinate transformation
   - ☐ Implement GASF and GADF computation
   - ☐ Implement sliding window GAF video generation
   - ☐ Implement ViViT patch embedding
   - ☐ Implement spatial attention mechanism
   - ☐ Implement temporal attention mechanism
   - ☐ Test on sample time series data
   - ☐ Benchmark latency (target: <50ms for inference)
2. **Logic Tensor Networks Module**
   - ☐ Implement grounding function framework
   - ☐ Implement predicate neural networks
   - ☐ Implement Lukasiewicz T-norm operations
   - ☐ Implement universal/existential quantification
   - ☐ Encode wash sale constraint
   - ☐ Encode Almgren-Chriss constraint
   - ☐ Encode VPIN constraint
   - ☐ Implement satisfiability aggregation
   - ☐ Implement logical loss function
   - ☐ Test constraint satisfaction with edge cases
   - ☐ Benchmark latency (target: <10ms for evaluation)
3. **Multimodal Fusion Module**
   - ☐ Implement time series tokenization (Chronos-Bolt integration)
   - ☐ Implement visual embedding extraction

- ☐ Implement text embedding (FinBERT integration)
- ☐ Implement gated cross-attention mechanism
- ☐ Implement multi-modal fusion pipeline
- ☐ Test fusion on sample multimodal data
- ☐ Validate attention weight distributions

4. **Decision Engine Module**

- ☐ Implement direct pathway (alpha motor)
- ☐ Implement indirect pathway (risk motor)
- ☐ Implement risk-gated action selection
- ☐ Implement logic-gated action selection
- ☐ Implement cerebellar forward model
- ☐ Implement sensory prediction error computation
- ☐ Implement trajectory adjustment
- ☐ Test end-to-end decision making
- ☐ Validate constraint adherence in production scenarios

5.2 Integration & Testing

1. **End-to-End Pipeline**

- ☐ Connect all modules into unified forward pass
- ☐ Implement gradient flow verification
- ☐ Test backpropagation through entire pipeline
- ☐ Validate differentiable constraint satisfaction

2. **Performance Optimization**

- ☐ Profile latency bottlenecks
- ☐ Optimize tensor operations for GPU
- ☐ Implement model quantization (INT8/FP16)
- ☐ Add batching support for parallel inference
- ☐ Target: <100ms end-to-end latency

3. **Safety & Validation**

- ☐ Add input validation and sanitization

- ☐ Implement kill switch integration
- ☐ Add logging for all trading decisions
- ☐ Implement emergency halt on constraint violation
- ☐ Test failure modes (network outage, invalid data, etc.)

5.3 Deployment Readiness

1. **Production Hardening**
   - ☐ Remove all panic!() calls
   - ☐ Replace unwrap() with proper error handling
   - ☐ Add comprehensive error types
   - ☐ Implement graceful degradation
   - ☐ Add health check endpoints
2. **Monitoring & Observability**
   - ☐ Add metrics export (Prometheus format)
   - ☐ Implement distributed tracing
   - ☐ Log all constraint violations
   - ☐ Monitor inference latency
   - ☐ Track prediction accuracy

6 Rust Implementation Considerations

sec:rust

6.1 Hot Path Optimization

The Forward service must maintain low latency (<100ms end-to-end). Key Rust optimizations:

- **Zero-copy operations:** Use `ndarray` views instead of clones
- **SIMD acceleration:** Leverage `packed_simd` for GAF computation
- **Async runtime:** Use `tokio` for non-blocking I/O
- **Memory pooling:** Pre-allocate tensor buffers to avoid allocation overhead

6.2 ML Framework Integration

6.2.1 Option 1: PyTorch via tch-rs

- Pros: Full PyTorch ecosystem, easy model export
- Cons: Requires LibTorch, larger binary size

6.2.2 Option 2: ONNX Runtime via ort

- Pros: Lightweight, cross-platform, optimized inference
- Cons: Limited to inference, requires model conversion
- **Recommended for production**

6.2.3 Option 3: Candle (Hugging Face)

- Pros: Pure Rust, no C++ dependencies
- Cons: Younger ecosystem, fewer pre-trained models
- **Recommended for future migration**

6.3 Error Handling Strategy

```rust
// Custom error types for Forward service
#[derive(Debug, thiserror::Error)]
pub enum ForwardError {
    #[error("GAF transformation failed: {0}")]
    GafError(String),

    #[error("LTN constraint violation: {constraint}")]
    ConstraintViolation { constraint: String },

    #[error("Model inference failed: {0}")]
    InferenceError(String),

    #[error("Risk threshold exceeded: {risk_score}")]
    RiskVeto { risk_score: f64 },
}

// Result type alias
pub type ForwardResult<T> = Result<T, ForwardError>;
```

References

[1] Jordan Smith, "Project JANUS: Implementation Guide v1.0," 2025.

[2] Wang, Oates, "Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks," AAAI 2015.

[3] Arnab et al., "ViViT: A Video Vision Transformer," ICCV 2021.

[4] Badreddine et al., "Logic Tensor Networks," Artificial Intelligence, 2022.

[5] Ansari et al., "Chronos: Learning the Language of Time Series," arXiv:2403.07815, 2024.

[6] Araci, "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models," arXiv:1908.10063, 2019.

[7] Almgren, Chriss, "Optimal Execution of Portfolio Transactions," Journal of Risk, 2000.

[8] Easley et al., "Flow Toxicity and Liquidity in a High-frequency World," Review of Financial Studies, 2012.