

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	3
1.1	Mathematical Foundation: Gramian Angular Fields	3
1.1.1	Input Preprocessing	3
1.1.2	Step 1: Learnable Normalization	3
1.1.3	Step 2: Polar Coordinate Transformation	3
1.1.4	Step 3: Gramian Field Generation	3
1.1.5	Implementation Algorithm	4
1.2	3D Spatiotemporal Manifolds: GAF Video	4
1.2.1	Sliding Window GAF Video Generation	4
1.2.2	Mathematical Formulation	5
1.3	Video Vision Transformer (ViViT)	5
1.3.1	Architecture Overview	5
1.3.2	Patch Embedding	5
1.3.3	Spatial Attention	6
1.3.4	Temporal Attention	6
1.3.5	Output Embedding	6
2	Logic Tensor Networks: Symbolic Reasoning Engine	6
2.1	Mathematical Foundation	6
2.1.1	Grounding Function	6
2.1.2	Predicate Grounding	6
2.2	Lukasiewicz T-Norm Operations	7
2.2.1	Conjunction (AND)	7
2.2.2	Disjunction (OR)	7
2.2.3	Negation (NOT)	7
2.2.4	Implication (IF-THEN)	7
2.2.5	Universal Quantification (FOR ALL)	7
2.2.6	Existential Quantification (EXISTS)	7
2.3	Knowledge Base Formulation	7
2.3.1	Wash Sale Constraint	7
2.3.2	Almgren-Chriss Risk Constraint	8
2.3.3	VPIN Toxicity Constraint	8
2.4	Logical Loss Function	8
2.4.1	Satisfiability Aggregation	8
2.4.2	Logical Loss	8
2.4.3	Combined Loss	8

3	Multimodal Fusion: Gated Cross-Attention	9
3.1	Input Modalities	9
3.2	Gated Cross-Attention Mechanism	9
3.2.1	Attention Computation	9
3.2.2	Gating Mechanism	9
3.2.3	Fused Representation	9
3.3	Multi-Modal Fusion Pipeline	10
4	Decision Engine: Basal Ganglia Pathways	10
4.1	Praxeological Motor: Dual Pathways	10
4.1.1	Direct Pathway (Go Signal)	10
4.1.2	Indirect Pathway (No-Go Signal)	10
4.1.3	Final Action	10
4.2	Cerebellar Forward Model	10
4.2.1	Market Impact Prediction	11
4.2.2	Sensory Prediction Error	11
4.2.3	Trajectory Adjustment	11
5	Implementation Checklist	12
5.1	Core Components	12
5.2	Integration & Testing	13
5.3	Deployment Readiness	14
6	Rust Implementation Considerations	14
6.1	Hot Path Optimization	14
6.2	ML Framework Integration	15
6.2.1	Option 1: PyTorch via tch-rs	15
6.2.2	Option 2: ONNX Runtime via ort	15
6.2.3	Option 3: Candle (Hugging Face)	15
6.3	Error Handling Strategy	15

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 :

$$X = \{x_1, x_2, \dots, x_N\} \in \mathbb{R}^N \quad (1)$$

1.1.2 Step 1: Learnable Normalization

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

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

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:

$$\alpha_0 = 1.0 \quad (3)$$

$$\beta_0 = 0.0 \quad (4)$$

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:

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

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):

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

Gramian Angular Difference Field (GADF):

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

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 $\in \mathbb{R}^{N \times N}$

```

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

```

1.2 3D Spatiotemporal Manifolds: GAF Video**1.2.1 Sliding Window GAF Video Generation**

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

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

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:

$$\mathcal{V} \in \mathbb{R}^{F \times 2 \times N \times N} \quad (9)$$

1.2.2 Mathematical Formulation

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

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

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 :

$$N_p = \frac{H}{P} \times \frac{W}{P} \quad (11)$$

patches per frame.

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

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

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:

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

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:

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

1.3.5 Output Embedding

The final visual embedding vector is:

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

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

$$\mathcal{G} : \text{Constants} \cup \text{Predicates} \cup \text{Functions} \rightarrow \mathbb{R}^n \quad (16)$$

2.1.2 Predicate Grounding

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

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

where d is the embedding dimension.

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

$$\mathcal{G}(IsVolatile)(e_{market}) = \text{sigmoid}(\mathbf{W}_v e_{market} + b_v) \quad (18)$$

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)

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

2.2.2 Disjunction (OR)

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

2.2.3 Negation (NOT)

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

2.2.4 Implication (IF-THEN)

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

2.2.5 Universal Quantification (FOR ALL)

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

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

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

2.2.6 Existential Quantification (EXISTS)

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

2.3 Knowledge Base Formulation

2.3.1 Wash Sale Constraint

The wash sale rule is encoded as:

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

In grounded form:

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

2.3.2 Almgren-Chriss Risk Constraint

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

Grounded:

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

where:

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

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

2.3.3 VPIN Toxicity Constraint

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

Grounded:

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

2.4 Logical Loss Function

2.4.1 Satisfiability Aggregation

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

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

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

2.4.2 Logical Loss

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

2.4.3 Combined Loss

The total loss combines predictive and logical components:

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

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:

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

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

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

3.2 Gated Cross-Attention Mechanism

3.2.1 Attention Computation

For primary modality m and auxiliary modality n :

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

where:

$$\mathbf{Q}_m = \mathbf{H}_m \mathbf{W}_Q \quad (39)$$

$$\mathbf{K}_n = \mathbf{H}_n \mathbf{W}_K \quad (40)$$

$$\mathbf{V}_n = \mathbf{H}_n \mathbf{W}_V \quad (41)$$

3.2.2 Gating Mechanism

The gating scalar is computed as:

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

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

3.2.3 Fused Representation

$$\mathbf{H}_{\text{fused}} = \mathbf{H}_m + \lambda_{\text{gate}} \cdot \alpha_{m \rightarrow n} \mathbf{V}_n \quad (43)$$

3.3 Multi-Modal Fusion Pipeline

The complete fusion process:

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

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

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

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:

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

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:

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

4.1.3 Final Action

The final action is gated by the risk veto:

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

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:

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

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

4.2.2 Sensory Prediction Error

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

4.2.3 Trajectory Adjustment

The execution trajectory is adjusted:

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

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

```

1  // Custom error types for Forward service
2  #[derive(Debug, thiserror::Error)]
3  pub enum ForwardError {
4      #[error("GAF transformation failed: {0}")]
5      GafError(String),
6
7      #[error("LTN constraint violation: {constraint}")]
8      ConstraintViolation { constraint: String },
9
10     #[error("Model inference failed: {0}")]
11     InferenceError(String),
12
13     #[error("Risk threshold exceeded: {risk_score}")]
14     RiskVeto { risk_score: f64 },
15 }
16
17 // Result type alias
18 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.