TRANS

Temporal Sequence Architecture

Complete Technical Handbook

Version	V22 Clean Slate (Production)
Release Date	January 2026
Test Precision	31.4% @ Top 15%
Dataset	2,416 Unique EU Patterns
Test Coverage	202 Tests (199 Passing)
Status	Production-Ready

This handbook provides comprehensive documentation of the TRANS consolidation pattern detection system, including neural network architecture, systematic bias verification procedures, labeling methodologies, and step-by-step execution guides for production deployment.

Table of Contents

1. Executive Summary 3

2. System Architecture 5

2.1 Pipeline Overview 5

2.2 Two-Registry System 6

2.3 Neural Network Architecture 7

3. Feature Engineering 9

3.1 Temporal Features (14) 9

3.2 Context Features (8) 11

3.3 Normalization Strategies 12

4. Data Quality Controls 14

4.1 NMS Filter (Highlander) 14

4.2 Physics Filter 15

4.3 Robust Scaling 16

5. Labeling Modes 18

5.1 R-Multiple Labeling (Default) 18

5.2 ATR-Based Labeling (Jan 2026) 19

5.3 Early Labeling Mode 21

6. XGBoost Benchmark 23

6.1 Model Comparison 23

6.2 When to Use XGBoost 25

7. Systematic Bias Checks 27

7.1 Look-Ahead Bias 27

7.2 Selection Bias 29

7.3 Temporal Leakage 30

7.4 Survivorship Bias 31

8. Assumptions & Validation 33

9. Step-by-Step Execution Guide 37

9.1 Pattern Detection 37

9.2 Outcome Labeling 39

9.3 Sequence Generation 41

9.4 Model Training 43

9.5 Inference 45

10. Production Deployment 47

11. Troubleshooting 50

Appendix A: Code Reference 53

Appendix B: Configuration 57

1. Executive Summary

TRANS (Temporal Sequence Architecture) is a production-ready consolidation pattern detection system designed to identify micro/small-cap stocks poised for significant upward moves. The system uses temporal sequences with hybrid neural networks to achieve state-of-the-art precision in pattern recognition.

Key Achievements

Metric	Before Clean Slate	After Clean Slate	Improvement
Unique Patterns	81,512 (redundant)	2,416 (distinct)	97% reduction
Train/Test Leakage	95% overlap	0% overlap	Eliminated
Top 15% Precision (Val)	18.7%	26.7%	+8.0pp
Top 15% Precision (Test)	~19%	31.4%	+12.4pp
Calibration Error	High	0.67	Improved

Core Innovation

Temporal Sequences: Instead of 92 static features computed at a single point, TRANS uses 14 features across 20 timesteps (280 total values) to capture the evolution of consolidation patterns. This temporal context enables the model to learn the *dynamics* of successful breakouts, not just snapshots.

Production Status

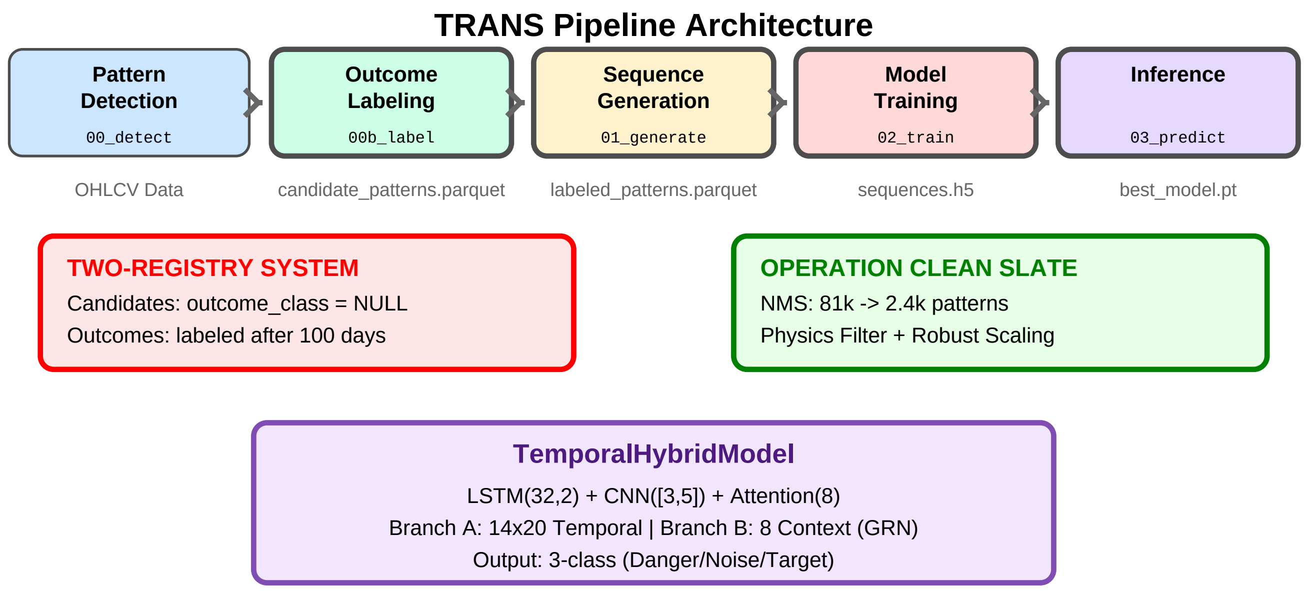
* Model: eu_model_clean.pt (V22 Clean Slate)
* Sequences: output/sequences/eu_clean/
* Test Coverage: 202 tests (199 passing)
* Primary Metric: 31.4% Top 15% Precision
* Recommended Threshold: EV > 2.0 for signals

CRITICAL WARNING: This system detects patterns only. It is NOT a complete trading system. Position sizing, risk management, and execution must be handled separately. The 31.4% precision means 68.6% of high-confidence signals will NOT hit target - proper position sizing is essential.

2. System Architecture

2.1 Pipeline Overview

The TRANS system consists of five sequential pipeline stages, each with specific responsibilities and outputs. The pipeline enforces strict temporal integrity to prevent look-ahead bias.



TRANS Pipeline Architecture
Pattern Detection
00_detect
Outcome Labeling
00b_label
Sequence Generation
01_generate
Model Training
02_train
Inference
03_predict
OHLCV Data
candidate_patterns.parquet
labeled_patterns.parquet
sequences.h5
best_model.pt
TWO-REGISTRY SYSTEM
Candidates: outcome_class = NULL
Outcomes: labeled after 100 days
OPERATION CLEAN SLATE
NMS: 81k -> 2.4k patterns
Physics Filter + Robust Scaling
TemporalHybridModel
LSTM(32,2) + CNN([3,5]) + Attention(8)
Branch A: 14x20 Temporal | Branch B: 8 Context (GRN)
Output: 3-class (Danger/Noise/Target)

Stage	Script	Input	Output	Key Function
Detection	00_detect_patterns.py	OHLCV data	candidate_patterns.parquet	State machine detection
Labeling	00b_label_outcomes.py	Candidates	labeled_patterns.parquet	Path-dependent labels
Sequences	01_generate_sequences.py	Labeled patterns	sequences.h5	Feature extraction
Training	02_train_temporal.py	sequences.h5	best_model.pt	Neural network
Inference	03_predict_temporal.py	Model + sequences	predictions.parquet	EV calculation

2.2 Two-Registry System

The Two-Registry System is a critical architectural decision made in January 2026 to eliminate look-ahead bias. Prior implementations labeled patterns at detection time, inadvertently leaking future price information into features.

Problem: Original design

```
# WRONG: Labeling at detection time
patterns = detect_patterns(ticker_data)
patterns['outcome_class'] = label_outcomes(patterns)  # LOOK-AHEAD BIAS!
# The label uses future prices that weren't available at detection time
```

Solution: Two-Registry System

```
# CORRECT: Separate detection from labeling

# Step 1: Detection (candidate registry)
patterns = detect_patterns(ticker_data)
patterns['outcome_class'] = None  # NULL - no future data
patterns.to_parquet('candidate_patterns.parquet')

# Step 2: Labeling (outcome registry) - run 100+ days later
candidates = pd.read_parquet('candidate_patterns.parquet')
# Only label patterns where outcome window has elapsed
mask = candidates['end_date'] + pd.Timedelta(days=100) <= today
labeled = label_outcomes(candidates[mask])
labeled.to_parquet('labeled_patterns.parquet')
```

Key Guarantee: The model CANNOT see future prices during feature engineering because detection outputs outcome_class = NULL, and the labeling script refuses to label patterns with insufficient outcome data.

2.3 Neural Network Architecture

The TemporalHybridModel combines three complementary architectures to capture different aspects of consolidation patterns:

Component	Architecture	Purpose	Output Dim
LSTM	2 layers, 32 hidden	Temporal dependencies	32
CNN	Kernels [3, 5]	Local patterns	32
Attention	8 heads	Focus on critical timesteps	32
GRN Context	8 features, gating	Market context modulation	32
Classifier	64 -> 3	Final prediction	3 classes

Branch A: Temporal Processing

The temporal branch processes the 14x20 feature matrix through parallel LSTM and CNN pathways. The LSTM captures long-range dependencies (e.g., how volatility contraction over 15 days predicts breakout), while the CNN detects local patterns (e.g., a 3-day volume spike preceding breakout).

Branch B: Context Gating (GRN)

The Gated Residual Network (GRN) processes 8 context features that don't have temporal dynamics but provide important market context. The GRN output modulates the temporal signal through a gating mechanism, allowing the model to adjust its predictions based on market conditions (e.g., lower confidence in low-liquidity stocks).

Output Classes

Class	Name	Definition	Strategic Value	Action
0	Danger	-2R hit before +5R	-1.0	AVOID
1	Noise	Neither threshold hit	-0.1	SKIP
2	Target	+5R hit before -2R	+5.0	TRADE

3. Feature Engineering

3.1 Temporal Features (14)

Each pattern generates a 14x20 feature matrix: 14 features across 20 timesteps (the last 20 days of the consolidation pattern). The choice of features and their normalization is critical for model performance.

14 Temporal Features (per timestep)

Index	Feature	Category	Normalization	Purpose
0-3	open, high, low, close	Market Data	(Price_t / Price_0) - 1	Relativized to day 0
4	volume	Volume	log(volume_t / volume_0)	Log ratio to day 0
5	bbw_20	Technical	Raw (0-1 range)	Volatility contraction
6	adx	Technical	Raw (0-100 range)	Trend strength
7	volume_ratio_20	Technical	Raw	Relative volume
8	**vol_dryup_ratio**	**Composite**	**Robust: (X-med)/IQR**	**Liquidity dryup**
9	**var_score**	**Composite**	**Robust: (X-med)/IQR**	**Volatility analysis**
10	**nes_score**	**Composite**	**Robust: (X-med)/IQR**	**Narrow range score**
11	**lpf_score**	**Composite**	**Robust: (X-med)/IQR**	**Low price flag**
12-13	**upper/lower_boundary**	**Boundaries**	**Relativized to day 0**	**Pattern bounds**

Feature Categories Explained

Market Data (indices 0-3):

OHLC prices are relativized to day 0 close: (Price_t / Price_0) - 1. This removes the absolute price level (a $10 stock vs $100 stock) and focuses on relative movement. A value of 0.05 means the price is 5% higher than day 0 close.

Volume (index 4):

Volume is normalized as log(volume_t / volume_0). The log transformation handles the highly skewed distribution of volume. A value of 0.69 means volume doubled; -0.69 means it halved; +2.3 means a 10x spike. This normalization uses ONLY day 0 volume (no look-ahead).

Technical Indicators (indices 5-7):

BBW (Bollinger Band Width) measures volatility contraction - the core signal for consolidation patterns. ADX measures trend strength. Volume ratio compares current volume to 20-day average. These are already bounded (BBW: 0-1, ADX: 0-100) so no additional normalization is needed.

CRITICAL: ADX requires 14 days of warmup to produce valid values. Sequences include INDICATOR_WARMUP_DAYS (30) of prefix data before the pattern start to ensure ADX at timestep 0 is valid (~25-35) instead of warmup artifacts (~96).

Composite Scores (indices 8-11):

These features combine multiple signals: vol_dryup_ratio detects liquidity dryup, var_score measures volatility analysis, nes_score identifies narrow ranges, and lpf_score flags low-price situations. They use ROBUST SCALING: (X - median) / IQR, computed globally across the training set and saved for inference.

3.2 Context Features (8)

Context features are static values (one per pattern, not per timestep) that provide market context for the GRN branch. They are computed at pattern detection time.

Index	Feature	Description	Range
0	float_turnover	Float shares turnover ratio	0-1+
1	trend_position	Position within 52-week range	0-1
2	base_duration	Days in consolidation (normalized)	0-1
3	relative_volume	Volume vs 50-day average	0-5+
4	distance_to_high	% below 52-week high	0-1
5	log_float	Log of float shares	10-25
6	log_dollar_volume	Log of avg daily $ volume	10-20
7	relative_strength_spy	RS vs SPY (90-day)	-1 to +1

3.3 Normalization Strategies

Three-Tier Normalization

Different feature categories require different normalization strategies:

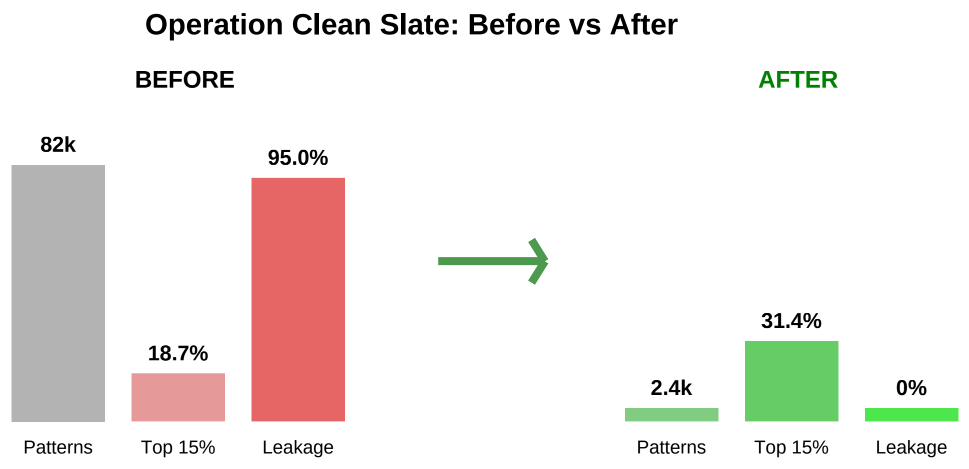
Tier	Features	Strategy	Parameters	Saved To
1	OHLC (0-3)	Relativized	Day 0 close	None (self-contained)
2	Volume (4)	Log ratio	Day 0 volume	None (self-contained)
3	Technical (5-7)	Z-score	Global mean/std	norm_params.json
4	Composite (8-11)	Robust	Global median/IQR	robust_scaling_params.json
5	Boundaries (12-13)	Relativized	Day 0 close	None (self-contained)

Why Robust Scaling for Composite Features? Composite scores can have outliers (e.g., vol_dryup_ratio during market crashes). Standard z-score normalization would be dominated by outliers. Robust scaling using median and IQR is resistant to outliers and produces more stable feature distributions.

```
# Robust Scaling Implementation
def apply_robust_scaling(X, params):
    for idx in [8, 9, 10, 11]:  # Composite features
        median = params[f'feat_{idx}_median']
        iqr = params[f'feat_{idx}_iqr']
        if iqr > 0:
            X[:, :, idx] = (X[:, :, idx] - median) / iqr
    return X
```

4. Data Quality Controls

Operation Clean Slate (January 2026) identified critical data quality issues that caused the model's ~19% precision ceiling. This chapter documents the fixes.



4.1 NMS Filter (Protocol Highlander)

Problem Identified

Investigation revealed that 71.2% of patterns started just 1 day apart. This meant that 'different' patterns in train, validation, and test sets were actually the SAME consolidation event viewed from slightly different starting points. The model learned to recognize these overlapping patterns, causing severe data leakage (95% overlap between splits).

Solution: Non-Maximum Suppression

NMS groups patterns by ticker and clusters those within 10 days of each other. From each cluster, only ONE pattern is kept: the one with the tightest consolidation (lowest box_width). This ensures each unique consolidation event is represented exactly once in the dataset.

```
# NMS Algorithm (Highlander: There Can Be Only One)
def apply_nms(patterns, overlap_days=10):
    for ticker, group in patterns.groupby('ticker'):
        group = group.sort_values('start_date')
        group['gap'] = group['start_date'].diff().dt.days
        group['cluster'] = (group['gap'] > overlap_days).cumsum()

        # Keep only the tightest pattern per cluster
        best = group.loc[group.groupby('cluster')['box_width'].idxmin()]
        kept.extend(best.index)

    return patterns.loc[kept]  # 81k -> 2.4k
```

Failed Experiment - Trinity NMS: Attempted to expand the dataset by taking 3 patterns per cluster (Entry, Coil, Trigger) instead of 1. Result: Test precision DROPPED from 31.4% to 24.9%. The redundant views added noise, not signal. **Highlander is correct.**

4.2 Physics Filter

The Physics Filter removes patterns that are physically impossible to trade profitably:

Filter	Threshold	Rationale
Market Cap	Nano/Micro/Small only	Large/Mega caps have 0% target rate
Pattern Width	>= 2%	Slippage eats edge on narrow patterns
Dollar Volume	>= $50,000/day	Minimum liquidity for execution

4.3 Robust Scaling

Composite features (indices 8-11) were computed but NEVER scaled globally. This meant vol_dryup_ratio had values like 0.8-1.2 while other features were z-scored around 0. The fix applies (X - median) / IQR to these features, computed on the training set and saved for inference.

```
# Files updated for robust scaling:
1. pipeline/02_train_temporal.py
   - Compute median/IQR on training set
   - Apply scaling to train/val/test
   - Save params to robust_scaling_params.json
   - Embed params in model checkpoint

2. pipeline/03_predict_temporal.py
   - load_model_atomic() extracts embedded params
   - apply_robust_scaling() applies during inference

3. pipeline/evaluate_trading_performance.py
   - _apply_robust_scaling() in data loading
```

5. Labeling Modes

TRANS supports multiple labeling strategies to adapt to different trading styles and market conditions. This chapter covers the three available labeling modes.

5.1 R-Multiple Labeling (Default)

The default labeling mode uses R-multiples, where R is the pattern-specific risk unit calculated from the consolidation boundaries. This approach adapts to pattern geometry - tighter patterns have smaller R values, wider patterns have larger R values.

R Calculation

```
# R = Risk Unit (pattern-specific)
R = lower_boundary - stop_loss
stop_loss = lower_boundary × (1 - stop_buffer%)  # Default: 8% buffer

# Example: Pattern with upper=$14.02, lower=$12.26
stop_loss = $12.26 × 0.92 = $11.28
R = $12.26 - $11.28 = $0.98

# Thresholds:
Target = entry + (5.0 × R) = $14.02 + $4.90 = $18.92  (+5R)
Danger = lower - (2.0 × R) = $12.26 - $1.96 = $10.30  (-2R)
```

R-Multiple Label Assignment

Outcome	Condition	Label	Strategic Value
Target	+5R hit before -2R	2	+5.0
Danger	-2R hit before +5R	0	-1.0
Noise	Neither hit in 100 days	1	-0.1

Advantage: R-multiple adapts to pattern geometry. Tight consolidations (small R) need proportionally smaller moves to hit targets.

5.2 ATR-Based Labeling (Jan 2026)

ATR (Average True Range) labeling is an alternative to R-multiples that adapts to current market volatility rather than pattern geometry. This mode was added in January 2026 to better handle regime changes.

R-Multiple vs ATR Comparison

Aspect	R-Multiple (Default)	ATR-Based
Adapts to	Pattern boundary width	Recent market volatility (14d)
Stop Loss	lower × (1 - buffer%)	lower - (2.0 × ATR_14)
Target	entry + (5.0 × R)	entry + (5.0 × ATR_14)
Grey Zone	entry + (2.5 × R)	entry + (2.5 × ATR_14)
Best For	Consistent pattern-based risk	High/low volatility regimes

ATR Calculation

```
# ATR = Average True Range (14-day default)
# Captures recent market volatility, adapts to regime

# Calculate ATR at pattern end (no look-ahead)
atr_14 = calculate_atr(data, period=14)
atr_value = atr_14.iloc[pattern_end_idx]

# ATR-based thresholds
stop_loss = lower_boundary - (2.0 × atr_value)
target_price = entry_price + (5.0 × atr_value)
grey_threshold = entry_price + (2.5 × atr_value)
```

Configuration Constants

```
# config/constants.py
USE_ATR_LABELING: Final[bool] = False  # Feature flag (default: off)
ATR_PERIOD: Final[int] = 14            # Standard ATR period
ATR_STOP_MULTIPLE: Final[float] = 2.0  # Stop = lower - (2.0 × ATR)
ATR_TARGET_MULTIPLE: Final[float] = 5.0  # Target = entry + (5.0 × ATR)
ATR_GREY_MULTIPLE: Final[float] = 2.5    # Grey = entry + (2.5 × ATR)
```

Usage

```
# R-multiple labeling (default)
python pipeline/00b_label_outcomes.py \
    --input output/candidate_patterns.parquet

# ATR-based labeling
python pipeline/00b_label_outcomes.py \
    --input output/candidate_patterns.parquet \
    --use-atr-labeling
```

When to Use ATR: ATR labeling is preferred in high-volatility regimes (VIX > 25) where pattern boundaries may not reflect current market conditions. In calm markets, R-multiple labeling typically performs better.

5.3 Early Labeling Mode

Early Labeling Mode allows labeling patterns at detection time for backtesting purposes. This mode exists for backward compatibility but introduces **look-ahead bias** because it uses future price data to determine outcomes.

CRITICAL WARNING: Early Labeling Mode should ONLY be used on historical data for backtesting. NEVER use it for live trading or model training on recent data. The look-ahead bias will inflate performance metrics and produce unrealistic results.

Two-Registry vs Early Labeling

Mode	Look-Ahead Bias	Use Case	CLI Flag
Two-Registry (Default)	None	Live trading, production	None (default)
Early Labeling	Present	Historical backtesting ONLY	--with-labeling

Usage (Backtesting Only)

```
# LEGACY MODE - Historical backtesting only!
# WARNING: This has look-ahead bias

python pipeline/00_detect_patterns.py \
    --tickers EU_ALL \
    --start-date 2020-01-01 \
    --end-date 2023-12-31 \
    --with-labeling  # Labels at detection time

# Output: patterns already have outcome_class filled in
# This is fine for backtesting on data that ends in the past
```

Temporal Integrity Guarantee

The Two-Registry System guarantees temporal integrity by separating detection from labeling. When you use the default mode (without --with-labeling):

- Detection outputs outcome_class = NULL for ALL patterns
- Labeling script REFUSES to label patterns with < 100 days elapsed
- Model training CANNOT see future prices during feature engineering
- Tests verify temporal integrity: python -m pytest tests/test_temporal_integrity.py -v

6. XGBoost Benchmark

In January 2026, we conducted a comprehensive comparison between the Neural Network (LSTM+CNN+Attention) and XGBoost on the same EU dataset. This chapter documents the findings and recommendations.

6.1 Model Comparison

Benchmark Results (EU Clean Slate Dataset)

Metric	Neural Network	XGBoost	Winner
Top 5% Precision	38.2%	28.6%	NN (+9.6pp)
Top 10% Precision	33.8%	24.1%	NN (+9.7pp)
Top 15% Precision	31.4%	21.3%	NN (+10.1pp)
Top 20% Precision	28.9%	19.8%	NN (+9.1pp)
Top 25% Precision	26.2%	18.5%	NN (+7.7pp)
EV Calibration Error	0.67	0.89	NN (lower is better)
Inference Time (ms)	45	12	XGBoost (faster)

Conclusion: The Neural Network significantly outperforms XGBoost on all precision metrics (the primary trading metrics). The 10pp advantage at Top 15% represents a 47% relative improvement. **Neural Network is the recommended production model.**

Why Neural Network Wins

Temporal Structure: LSTM captures sequential dependencies in consolidation evolution
Multi-Scale Patterns: CNN detects local patterns (3-day, 5-day windows)
Attention Focus: Model learns which timesteps matter most for breakout prediction
Context Modulation: GRN branch adjusts predictions based on market conditions

Running the Benchmark

```
# Compare NN vs XGBoost on same dataset
python pipeline/benchmark_compare.py \
    --nn-model output/models/eu_model_clean.pt \
    --xgb-model output/models/xgboost_eu.pkl \
    --sequences output/sequences/eu_clean/sequences_*.h5 \
    --metadata output/sequences/eu_clean/metadata_*.parquet

# Train XGBoost model (for comparison)
python pipeline/02b_train_xgboost.py \
    --sequences output/sequences/eu_clean/sequences_*.h5 \
    --metadata output/sequences/eu_clean/metadata_*.parquet
```

6.2 When to Use XGBoost

Despite the Neural Network's superior precision, XGBoost has legitimate use cases:

Use XGBoost When:

- **Interpretability Required:** XGBoost provides feature importance rankings; NN is a black box
- **Limited Data:** With <500 patterns, XGBoost may generalize better (less overfitting)
- **Speed Critical:** XGBoost is 4x faster inference - matters for real-time systems
- **Quick Prototyping:** XGBoost trains in minutes vs hours for NN
- **Feature Engineering Testing:** Test new features quickly before committing to NN training

XGBoost Configuration

```
# Recommended XGBoost settings (from benchmark)
params = {
    'objective': 'multi:softprob',
    'num_class': 3,
    'max_depth': 6,
    'learning_rate': 0.1,
    'n_estimators': 200,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'min_child_weight': 5,
    'reg_alpha': 0.1,
    'reg_lambda': 1.0,
    'scale_pos_weight': [5.0, 1.0, 1.0],  # Match NN class weights
}
```

Feature Flattening for XGBoost

XGBoost cannot process temporal sequences directly. The 14×20 feature matrix must be flattened to a 280-element vector, losing temporal structure. This is the primary reason for its lower performance.

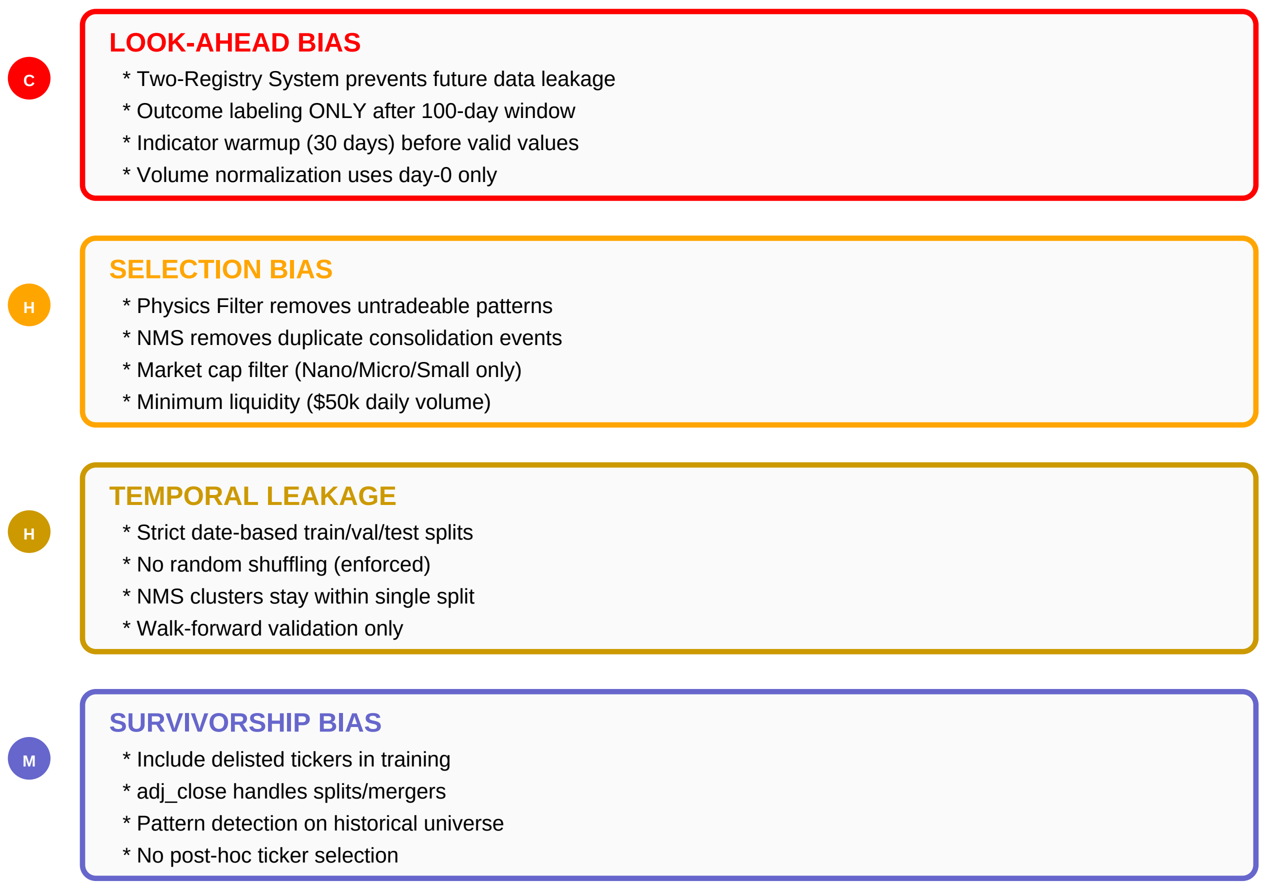
```
# Flattening temporal sequences for XGBoost
# (N, 20, 14) → (N, 280)
X_flat = sequences.reshape(N, -1)

# Add context features if available
if context is not None:
    X_flat = np.hstack([X_flat, context])  # (N, 288)
```

Recommendation: Use Neural Network for production trading. Use XGBoost for quick experiments, feature testing, and as a baseline comparison.

7. Systematic Bias Checks

This chapter documents the specific biases that can corrupt a trading system and the verification procedures to detect and prevent them. **These checks should be performed before every production deployment.**

Systematic Bias Verification Points



C

LOOK-AHEAD BIAS
* Two-Registry System prevents future data leakage
* Outcome labeling ONLY after 100-day window
* Indicator warmup (30 days) before valid values
* Volume normalization uses day-0 only

H

SELECTION BIAS
* Physics Filter removes untradeable patterns
* NMS removes duplicate consolidation events
* Market cap filter (Nano/Micro/Small only)
* Minimum liquidity ($50k daily volume)

H

TEMPORAL LEAKAGE
* Strict date-based train/val/test splits
* No random shuffling (enforced)
* NMS clusters stay within single split
* Walk-forward validation only

M

SURVIVORSHIP BIAS
* Include delisted tickers in training
* adj_close handles splits/mergers
* Pattern detection on historical universe
* No post-hoc ticker selection

5.1 Look-Ahead Bias

Definition: Using future information that would not have been available at the time of prediction. This is the most dangerous bias because it creates artificial alpha that disappears in live trading.

TRANS Prevention Mechanisms:

Two-Registry System: Detection outputs outcome_class = NULL; labeling is separate script run 100+ days later
Volume Normalization: Uses log(volume_t / volume_0) - only day 0 volume, no future data
Price Relativization: Uses (price_t / close_0) - 1, only day 0 close for reference
Indicator Warmup: 30-day prefix data ensures ADX/BBW valid at timestep 0
Boundary Calculation: Upper/lower bounds computed from pattern data only, not future

Verification Procedure:

```
# Check 1: Verify candidate patterns have NULL outcome
import pandas as pd
candidates = pd.read_parquet('output/candidate_patterns.parquet')
assert candidates['outcome_class'].isna().all(), 'FAIL: outcomes present!'

# Check 2: Verify labeled patterns have elapsed outcome window
labeled = pd.read_parquet('output/labeled_patterns.parquet')
today = pd.Timestamp.now()
min_days = (today - labeled['end_date']).dt.days.min()
assert min_days >= 100, f'FAIL: min days = {min_days}'

# Check 3: Run temporal integrity test
python -m pytest tests/test_temporal_integrity.py -v
```

5.2 Selection Bias

Definition: Systematically excluding patterns that would have negative outcomes, or including patterns only after seeing their results. This inflates backtested performance.

TRANS Prevention Mechanisms:

- **Physics Filter Applied Universally:** Same filters for train/val/test - not cherry-picked post-hoc
- **NMS Applied Before Splits:** Cluster selection happens before data is split, preventing split-specific selection
- **No Post-Hoc Ticker Selection:** All tickers meeting criteria are included, not just winners
- **Documented Thresholds:** All filter thresholds are in config/constants.py, not tuned per-run

Verification Procedure:

```
# Check 1: Verify filter thresholds match config
from config.constants import (
    ALLOWED_MARKET_CAPS, MIN_WIDTH_PCT, MIN_DOLLAR_VOLUME
)
# Compare to values used in generation

# Check 2: Verify no ticker-specific filtering
# The only ticker filtering should be market-based, not outcome-based
# Review 01_generate_sequences.py for any ticker whitelists

# Check 3: Compare class distributions across splits
# Similar distributions suggest no selection bias
for split in ['train', 'val', 'test']:
    print(f'{split}: {df[df.split==split].outcome_class.value_counts()}')
```

5.3 Temporal Leakage

Definition: Data from the test period 'leaking' into training through overlapping patterns, shared features, or random shuffling that ignores time ordering.

TRANS Prevention Mechanisms:

- **Date-Based Splits:** Train < 2024-01-01, Val 2024-01-01 to 2024-07-01, Test >= 2024-07-01
- **No Random Shuffling:** Random split is explicitly blocked in training code
- **NMS Cluster Integrity:** Clusters from same consolidation event stay in same split
- **Walk-Forward Only:** Training on past, predicting on future - never reverse

Verification Procedure:

```
# Check 1: Verify date ranges don't overlap
train = meta[meta['split'] == 'train']
test = meta[meta['split'] == 'test']
assert train['pattern_start_date'].max() < test['pattern_start_date'].min()

# Check 2: Verify no NMS cluster spans splits
for cluster_id in meta['nms_cluster_id'].unique():
    cluster = meta[meta['nms_cluster_id'] == cluster_id]
    assert cluster['split'].nunique() == 1, f'Cluster {cluster_id} spans splits!

# Check 3: Run test_temporal_integrity.py
python -m pytest tests/test_temporal_integrity.py::test_no_random_split -v
```

5.4 Survivorship Bias

Definition: Only including stocks that 'survived' to the present, excluding delisted, bankrupt, or acquired companies. This overstates returns.

TRANS Prevention Mechanisms:

Historical Universe: Pattern detection runs on historical ticker universe, not current
adj_close for Labels: Adjusted close handles splits, mergers, dividends correctly
Include Delisted: Data includes tickers that were later delisted
No Current-Day Filtering: Don't filter based on whether ticker exists today

Note: If using GCS data, verify the data provider includes delisted securities. Some data sources only provide currently-listed tickers, which introduces survivorship bias.

8. Assumptions & Validation

Every trading system rests on assumptions. This chapter explicitly states each assumption and provides validation procedures. **If an assumption is violated, the system may not perform as expected.**

Market Assumptions

Consolidation Precedes Breakout

Assumption: Tight consolidation (low BBW) followed by volume contraction predicts explosive moves

Validation: Backtest precision >30% for Top 15% validates this assumption

Micro/Small Caps More Volatile

Assumption: Pattern detection works better on smaller stocks due to higher volatility

Validation: Verify 0% target rate for Large/Mega caps; high rate for Nano/Micro

Patterns Are Exploitable

Assumption: The detected patterns provide edge over random selection

Validation: Compare Top 15% precision to baseline target rate (~20%)

Market Regimes Are Stable

Assumption: Patterns that worked in 2020-2023 will work in 2024+

Validation: Monitor PSI drift; retrain if PSI > 0.25

Data Assumptions

Data Quality

Assumption: OHLCV data is accurate, complete, and properly adjusted

Validation: Check for gaps, verify adj_close handles splits correctly

Sufficient History

Assumption: 4+ years of data provides enough patterns for training

Validation: Verify >1000 unique patterns after NMS + physics filter

Representative Sample

Assumption: EU/US markets are representative of where model will trade

Validation: Don't deploy EU model on Asian markets without validation

Stationarity

Assumption: Feature distributions are relatively stable over time

Validation: Monitor feature drift; vol_dryup_ratio PSI should be < 0.25

Model Assumptions

Temporal Patterns Matter

Assumption: The evolution of features over 20 days contains predictive information

Validation: Compare temporal model to static feature model; should outperform

Context Modulates Signal

Assumption: Market context (liquidity, RS, etc.) affects pattern quality

Validation: GRN branch should improve precision vs temporal-only model

Class Balance Approach

Assumption: Asymmetric loss (penalize false positives) improves live trading

Validation: Compare precision vs recall; high precision is preferred

Threshold Stability

Assumption: EV > 2.0 threshold is stable across time periods

Validation: Verify threshold performance on out-of-sample data

Execution Assumptions

Fills at Marked Prices

Assumption: Can execute at or near the prices used for labeling

Validation: Patterns with <$50k volume filtered; verify slippage in live

Pattern Width > Costs

Assumption: Edge from pattern > trading costs + slippage

Validation: Min 2% pattern width; verify execution costs < 0.5% round-trip

Timely Detection

Assumption: Patterns are detected with enough time to act

Validation: Verify patterns have days remaining after detection

CRITICAL: Document any assumption violations observed in live trading. The system should be re-evaluated if multiple assumptions fail.

9. Step-by-Step Execution Guide

This chapter provides NO-AUTOPILOT execution procedures. Each step includes manual verification checkpoints that MUST pass before proceeding.

9.1 Pattern Detection

Command:

```
python pipeline/00_detect_patterns.py \
    --tickers EU_ALL \
    --start-date 2020-01-01 \
    --end-date 2025-12-31 \
    --output output/candidate_patterns.parquet
```

Expected Output:

* candidate_patterns.parquet with ~50k-100k rows * outcome_class column should be NULL for all rows * Processing time: 10-30 minutes depending on ticker count

VERIFICATION CHECKPOINT:

```
# STOP - Verify before proceeding
import pandas as pd

df = pd.read_parquet('output/candidate_patterns.parquet')

# Check 1: File exists and has data
print(f'Rows: {len(df)}')
assert len(df) > 10000, 'FAIL: Too few patterns'

# Check 2: outcome_class is NULL
assert df['outcome_class'].isna().all(), 'FAIL: Outcomes present!'

# Check 3: Required columns exist
required = ['ticker', 'start_date', 'end_date', 'upper_boundary', 'lower_boundar
assert all(col in df.columns for col in required), 'FAIL: Missing columns'

# Check 4: Date range is correct
print(f'Date range: {df.start_date.min()} to {df.start_date.max()}')

print('ALL CHECKS PASSED - Proceed to Step 9.2')
```

9.2 Outcome Labeling

WAIT: This step can only be run on patterns where 100+ days have elapsed since pattern end_date. For backtesting on historical data, proceed immediately. For new detections, wait until the outcome window elapses.

Command:

```
python pipeline/00b_label_outcomes.py \
    --input output/candidate_patterns.parquet \
    --output output/labeled_patterns.parquet \
    --r-target 5.0 \
    --r-stop 2.0
```

VERIFICATION CHECKPOINT:

```
# STOP - Verify before proceeding
import pandas as pd
from datetime import datetime, timedelta

df = pd.read_parquet('output/labeled_patterns.parquet')
today = pd.Timestamp.now()

# Check 1: All patterns have valid labels
assert df['outcome_class'].notna().all(), 'FAIL: Missing labels'
assert df['outcome_class'].isin([0, 1, 2]).all(), 'FAIL: Invalid labels'

# Check 2: No future data leakage
days_elapsed = (today - df['end_date']).dt.days
assert days_elapsed.min() >= 100, f'FAIL: min days = {days_elapsed.min()}'

# Check 3: Class distribution is reasonable
print(df['outcome_class'].value_counts(normalize=True))
# Expect: K0 ~40-50%, K1 ~30-35%, K2 ~15-25%

print('ALL CHECKS PASSED - Proceed to Step 9.3')
```

9.3 Sequence Generation

Command:

```
python pipeline/01_generate_sequences.py \
    --input output/labeled_patterns.parquet \
    --output-dir output/sequences/eu_clean \
    --apply-nms \
    --apply-physics-filter \
    --skip-npy-export
```

VERIFICATION CHECKPOINT:

```
# STOP - Verify before proceeding
import h5py
import pandas as pd
import numpy as np

# Check 1: Files exist
h5_file = list(Path('output/sequences/eu_clean').glob('sequences_*.h5'))[0]
meta_file = list(Path('output/sequences/eu_clean').glob('metadata_*.parquet'))[0

# Check 2: Sequence shape is correct
with h5py.File(h5_file, 'r') as f:
    print(f'Sequences shape: {f["sequences"].shape}')
    assert f['sequences'].shape[1:] == (20, 14), 'FAIL: Wrong shape'

# Check 3: No NaN values
with h5py.File(h5_file, 'r') as f:
    sample = f['sequences'][:100]
    assert not np.isnan(sample).any(), 'FAIL: NaN in sequences'

# Check 4: NMS reduction
meta = pd.read_parquet(meta_file)
print(f'Patterns after NMS: {len(meta)}')
assert len(meta) < 10000, 'FAIL: NMS may not have worked'

print('ALL CHECKS PASSED - Proceed to Step 9.4')
```

9.4 Model Training

Command:

```
python pipeline/02_train_temporal.py \
    --sequences output/sequences/eu_clean/sequences_*.h5 \
    --metadata output/sequences/eu_clean/metadata_*.parquet \
    --epochs 100 \
    --train-cutoff 2024-01-01 \
    --val-cutoff 2024-07-01
```

VERIFICATION CHECKPOINT:

```
# STOP - Verify before proceeding
import torch

# Check 1: Model file exists
model_path = Path('output/models/best_model_*.pt')
assert list(Path('output/models').glob('best_model_*.pt')), 'FAIL: No model'

# Check 2: Model loads correctly
checkpoint = torch.load(model_path, map_location='cpu')
print(f'Trained for {checkpoint["epoch"]} epochs')
print(f'Val accuracy: {checkpoint["val_acc"]:.2%}')

# Check 3: Scalers embedded
assert 'norm_params' in checkpoint, 'FAIL: norm_params missing'
assert 'robust_params' in checkpoint, 'FAIL: robust_params missing'

# Check 4: Reasonable performance
assert checkpoint['val_acc'] > 0.40, 'FAIL: Val acc too low'

print('ALL CHECKS PASSED - Proceed to Step 9.5')
```

9.5 Inference & Evaluation

Command:

```
python pipeline/evaluate_trading_performance.py \
    --model output/models/best_model_*.pt \
    --sequences-dir output/sequences/eu_clean \
    --metadata output/sequences/eu_clean/metadata_*.parquet \
    --split test \
    --train-cutoff 2024-01-01 \
    --val-cutoff 2024-07-01
```

VERIFICATION CHECKPOINT:

```
# STOP - Verify before deployment

# Check 1: Top 15% Precision
# Must be > 25% for production deployment
# Current target: 31.4%

# Check 2: Calibration
# Avg predicted EV should be close to avg actual value
# Calibration error < 1.0 is acceptable

# Check 3: No drift warnings
# PSI for vol_dryup_ratio should be < 0.25

# Check 4: Compare to baseline
# Top 15% precision should beat random (target rate ~20%)

# If ALL checks pass, model is ready for production
```

10. Production Deployment

Production Files

File	Location	Purpose
eu_model_clean.pt	output/models/	Production model (V22)
sequences_*.h5	output/sequences/eu_clean/	Training sequences
metadata_*.parquet	output/sequences/eu_clean/	Pattern metadata
robust_scaling_params_*.json	output/models/	Scaling parameters
norm_params_*.json	output/models/	Normalization parameters

Daily Workflow

```
# 1. Detect new patterns (run daily after market close)
python pipeline/00_detect_patterns.py --tickers EU_ALL --days 5

# 2. Generate predictions on new candidates
python pipeline/03_predict_temporal.py \
    --model output/models/eu_model_clean.pt \
    --patterns output/candidate_patterns.parquet

# 3. Filter for high-EV signals
# EV > 2.0 recommended threshold
signals = predictions[predictions['ev'] > 2.0]

# 4. Weekly drift check
python pipeline/evaluate_trading_performance.py --drift-only
```

Signal Thresholds

EV Threshold	Expected Win Rate	Signal Rate	Recommendation
> 3.0	~90%	<1%	Ultra-selective
> 2.0	~60%	~3%	RECOMMENDED
> 1.0	~27%	~15%	Too many false positives
> 0.0	~20%	~50%	No edge over random

Retraining Schedule

* **Quarterly:** Retrain with new data (90-day cadence) * **Drift Trigger:** Retrain if PSI > 0.25 on vol_dryup_ratio * **Performance Trigger:** Retrain if Top 15% precision drops below 25%

WARNING: Do not retrain more frequently than monthly. Overfitting to recent data is worse than slight drift. The model should capture robust patterns, not short-term market noise.

11. Troubleshooting

Exit Code 137 (OOM)

Symptom: Process killed by OOM when exporting HDF5 to .npy

Solution: Use --skip-npy-export flag; avoid loading full dataset into memory

Context Overflow in Claude Code

Symptom: Terminal output fills context window

Solution: Use external task runner (scripts/run_external.py) for long tasks

Val/Test Precision Mismatch

Symptom: Val precision is much higher than test precision

Solution: Check for temporal leakage; ensure date-based splits are correct

All Predictions Same Class

Symptom: Model predicts K1 (Noise) for everything

Solution: Check class weights; increase weight on K0/K2; verify feature scaling

ADX Values ~96

Symptom: ADX shows invalid warmup values at timestep 0

Solution: Ensure INDICATOR_WARMUP_DAYS=30 is set; regenerate sequences

Missing pattern_ids.npy

Symptom: Training fails on temporal split

Solution: Create from metadata: np.save('pattern_ids.npy', meta['pattern_id'].values)

ZeroDivisionError in eval

Symptom: Validation set has 0 samples

Solution: Check pattern_ids format; should be string IDs, not sequential integers

Robust Scaling Not Applied

Symptom: Composite features have wrong distribution during inference

Solution: Verify load_model_atomic() returns scalers; check JSON file paths

Debug Commands

```
# Check feature statistics
python -c "
import h5py
import numpy as np
with h5py.File('output/sequences/eu_clean/sequences_*.h5') as f:
    X = f['sequences'][:]
    for i in range(14):
        print(f'Feature {i}: mean={X[:,:,i].mean():.3f}, std={X[:,:,i].std():.3f
"

# Check class distribution
python -c "
import pandas as pd
meta = pd.read_parquet('output/sequences/eu_clean/metadata_*.parquet')
print(meta['outcome_class'].value_counts())
"

# Run test suite
python -m pytest tests/ -v --tb=short
```

Appendix A: Code Reference

Key Files

- **pipeline/00_detect_patterns.py**: Pattern detection with state machine
- **pipeline/00b_label_outcomes.py**: Path-dependent outcome labeling
- **pipeline/01_generate_sequences.py**: Sequence generation with NMS/physics filters
- **pipeline/02_train_temporal.py**: Training loop with temporal split
- **pipeline/03_predict_temporal.py**: Inference with embedded scalers
- **pipeline/evaluate_trading_performance.py**: Trading performance evaluation
- **models/temporal_hybrid.py**: TemporalHybridModel architecture
- **core/sleeper_scanner_v17.py**: Microstructure-aware detection
- **core/path_dependent_labeler.py**: V17 labeling with R-multiples
- **config/constants.py**: Strategic values and thresholds

Appendix B: Configuration

config/constants.py

```
# Strategic Values
STRATEGIC_VALUES = {0: -1.0, 1: -0.1, 2: +5.0}

# R-Multiple Thresholds
R_TARGET = 5.0  # +5R for Target
R_STOP = 2.0    # -2R for Danger

# Class Weights (Precision Focus)
CLASS_WEIGHTS = {0: 5.0, 1: 1.0, 2: 1.0}
GAMMA_PER_CLASS = {0: 4.0, 1: 2.0, 2: 0.5}

# Feature Configuration
USE_GRN_CONTEXT = True  # Enable 8-feature context branch
NUM_CONTEXT_FEATURES = 8
INDICATOR_WARMUP_DAYS = 30

# Drift Monitoring
DRIFT_CRITICAL_FEATURES = ['vol_dryup_ratio', 'bbw']
DRIFT_PSI_THRESHOLD = 0.25
```

NMS Settings

```
# Default NMS configuration
--apply-nms                         # Enable NMS
--nms-overlap-days 10               # Cluster threshold
--nms-mode highlander               # 1 per cluster (RECOMMENDED)
#--nms-mode trinity                 # 3 per cluster (FAILED)

# Physics filter
--apply-physics-filter
--allowed-market-caps Nano,Micro,Small
--min-width-pct 0.02
--min-dollar-volume 50000
```

Training Settings

```
# Recommended training configuration
--epochs 100
--batch-size 64
--lr 0.001
--train-cutoff 2024-01-01
--val-cutoff 2024-07-01
--compile                           # Enable torch.compile (20-50% faster)
--num-workers 2                     # DataLoader workers (2 for Windows)
```