TRANS

Temporal Sequence Architecture

System Handbook & Technical Reference

PRODUCTION MODEL DASHBOARD

eu_model.pt (2026-01-12) | V18 + Coil Focal Loss | 13 Context Features

32.1%	1.67x	13.6%	2,416
Top 15% Precision	Lift vs Random	K2 Recall	Clean Patterns
Primary Metric	Signal Quality	Target Detection	After NMS Filter

Production Model	eu_model.pt (V18 + Coil Focal Loss)
Top 15% Precision	32.1% (1.67x lift)
Context Features	13 (8 market + 5 coil)
Training Data	2,416 clean patterns (after NMS)
Document Version	2.0 (2026-01-12)

Table of Contents

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 hybrid neural networks combining LSTM, CNN, and attention mechanisms with a novel Coil-Aware Focal Loss function.

32.1%	1.67x	13	2,416
Top 15% Precision	Lift vs Random	Context Features	Clean Patterns
Primary Metric	Signal Quality	8 Market + 5 Coil	After NMS Filter

Key Innovations

• **Two-Registry System:** Separates pattern detection from outcome labeling to guarantee NO look-ahead bias. Candidates are detected with outcome_class=NULL, then labeled only after 100 days have elapsed.

• **13-Feature Context Branch:** The GRN (Gated Residual Network) processes 13 context features: 8 original market features plus 5 NEW coil features that capture pattern state at detection time without any look-ahead bias.

• **Coil-Aware Focal Loss:** Custom loss function that amplifies gradients for K2 (Target) patterns with high coil_intensity scores. This solved the critical K2 learning problem where standard loss resulted in near-0% K2 predictions.

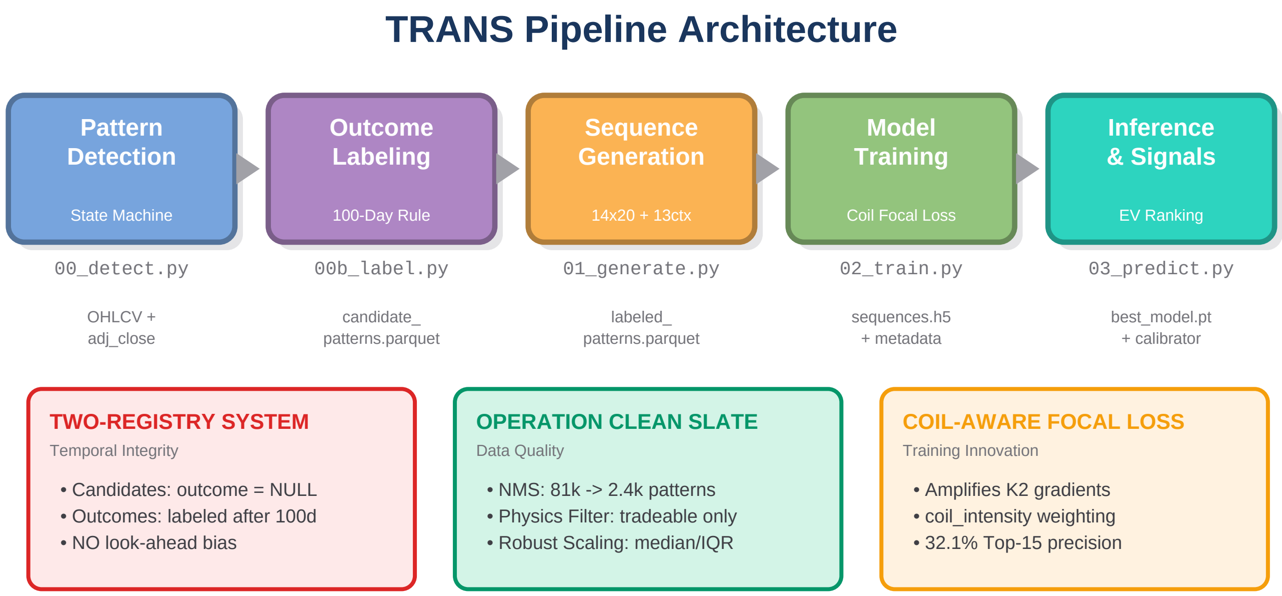
• **Operation Clean Slate:** NMS filter reduces 81k overlapping patterns to 2.4k unique consolidation events, eliminating severe data leakage between train/val/test splits.

Production Model Performance

Metric	Value	Benchmark	Status
Top 5% Precision	22.2%	>20%	**PASS**
Top 10% Precision	30.8%	>25%	**PASS**
Top 15% Precision	32.1%	>30%	**PASS**
Top 20% Precision	29.0%	>25%	**PASS**
K2 Recall	13.6%	>10%	**PASS**
Lift vs Random	1.67x	>1.5x	**PASS**

2. Pipeline Architecture

The TRANS pipeline consists of five main stages, each implemented as a separate Python script for modularity and debugging. The architecture enforces strict temporal integrity to prevent any look-ahead bias.



2.1 Two-Registry System

The Two-Registry System is the foundation of temporal integrity in TRANS. It ensures that pattern detection and outcome labeling are completely separate processes, eliminating any possibility of future data leaking into training features.

Registry	Script	Output	Key Guarantee
Candidate	00_detect_patterns.py	candidate_patterns.parquet	outcome_class = NULL
Outcome	00b_label_outcomes.py	labeled_patterns.parquet	Labels only after 100 days

Critical Rule: The labeling script (00b) ONLY processes patterns where end_date + 100 days <= today. This ensures we never use future price data to determine outcomes.

2.2 Pipeline Scripts

00_detect_patterns.py - Pattern Detection

Runs the state machine (QUALIFYING → ACTIVE → COMPLETED/FAILED) to identify consolidation patterns. Outputs candidates with outcome_class=NULL.

00b_label_outcomes.py - Outcome Labeling

Labels patterns using path-dependent logic: Class 0 (Danger) if -2R hit first, Class 2 (Target) if +5R hit first, Class 1 (Noise) otherwise.

01_generate_sequences.py - Sequence Generation

Extracts 14×20 temporal features and 13 context features. Applies NMS filter and physics filter for clean data.

02_train_temporal.py - Model Training

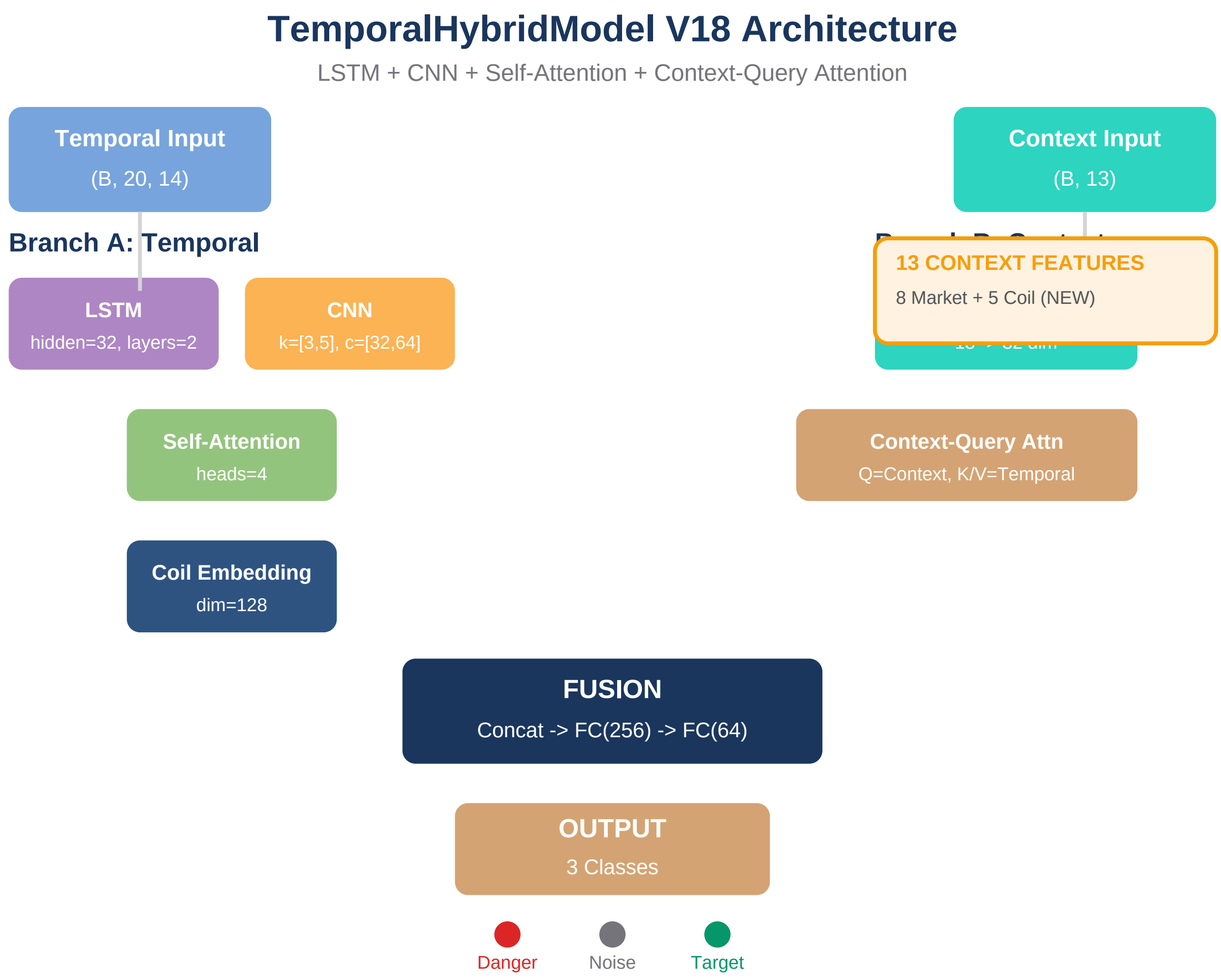
Trains TemporalHybridModel with Coil-Aware Focal Loss. Enforces temporal split (no random shuffle) with early stopping.

03_predict_temporal.py - Inference

Generates predictions, applies probability calibration, calculates Expected Values, and assigns trading signals (STRONG/GOOD/MODERATE/AVOID).

3. Neural Network Architecture

TemporalHybridModel V18 combines four processing branches: LSTM for sequence evolution, CNN for micro-patterns, Self-Attention for geometric structure, and Context-Query Attention for regime-guided search. The model processes 14×20 temporal features plus 13 context features.



3.1 Branch A: Temporal Processing

Branch A processes the 14×20 temporal sequence through three parallel paths:

Component	Architecture	Output Dim	Purpose
LSTM	hidden=32, layers=2, dropout=0.2	32	Sequence evolution & narrative state
CNN	kernels=[3,5], channels=[32,64]	96	Multi-scale micro-pattern detection
Self-Attention	heads=4, dropout=0.1	96	Geometric structure (coil shape)

3.2 Branch B: Context Processing (13 Features)

Branch B processes 13 context features through a Gated Residual Network (GRN) followed by Context-Query Attention. This allows the model to search for regime-specific patterns in the temporal sequence.

Context Features (13 Total)

Original Market Context (8)

Idx	Feature	Description	Range
0	float_turnover	Accumulation activity	0-1+
1	trend_position	200-SMA relative position	0-1
2	base_duration	Days in consolidation (norm)	0-1
3	relative_volume	Vol vs 50-day avg	0-5+
4	distance_to_high	% below 52-week high	0-1
5	log_float	Log10 shares outstanding	10-25
6	log_dollar_volume	Log10 avg daily $ vol	10-20
7	relative_strength_spy	RS vs SPY (90-day)	-1 to +1

NEW: Coil Features (5) - Bias-Free Pattern State

8	**price_position_at_end**	(close - lower) / width	0-1
9	**distance_to_danger**	(close - stop) / close	0-0.2
10	**bbw_slope_5d**	BBW regression slope	-0.1 to +0.1
11	**vol_trend_5d**	Volume regression slope	-1 to +1
12	**coil_intensity**	Composite spring tension	0-1

3.3 Fusion Mechanism

KEY INSIGHT:
price_position_at_end < 0.4 has 5.5x better K2 hit rate

The four branch outputs are concatenated and processed through a fusion network:

models/temporal_hybrid_v18.py

```
# Fusion Architecture
temporal = concat(lstm_out, cnn_out, self_attn_out)  # 32+96+96 = 224
context = concat(grn_out, cross_attn_out)            # 32+96 = 128
fused = concat(temporal, context)                     # 352 total

# Classification head
x = Linear(352 -> 256) -> ReLU -> Dropout(0.3)
x = Linear(256 -> 64)  -> ReLU -> Dropout(0.3)
output = Linear(64 -> 3)  # Danger, Noise, Target
```

5. Coil-Aware Focal Loss

The Coil-Aware Focal Loss is the key innovation that enabled the model to learn K2 (Target) patterns effectively. Without it, standard loss functions caused the model to predict K2 only 0.04% of the time despite K2 being 21% of the data.

Coil-Aware Focal Loss

Custom loss function that amplifies gradients for high-coil K2 patterns

THE PROBLEM

Standard loss: K2 predictions = 0.04%
Model learned to avoid K2 due to
precision penalties (False Positives)

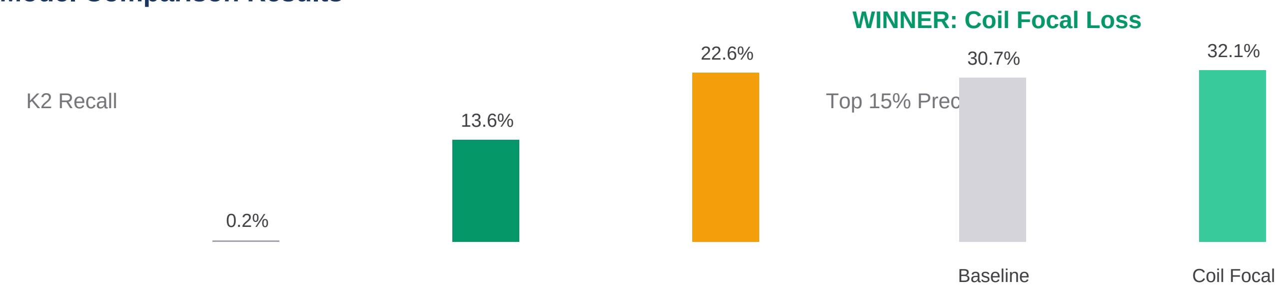
THE SOLUTION

Coil Focal Loss: K2 predictions = 6.4%
Amplify gradients for K2 patterns
with high coil_intensity score

```
loss_K2 = focal_loss * (1 + coil_intensity * 3.0)
```

Where coil_intensity = (1 - price_pos) * (1 - bbw_pctl) * vol_dryup

Model Comparison Results



5.1 The K2 Learning Problem

With standard focal loss or cross-entropy, the model learned to maximize overall accuracy by predicting only Danger (Class 0) and Noise (Class 1). The precision penalty for False Positive K2 predictions was too high, so the model avoided predicting K2 entirely.

Metric	Standard Loss	Coil Focal Loss	Change
K2 Predictions	1 (0.04%)	154 (6.4%)	**+160x**
K2 Recall	0.2%	13.6%	**+13.4pp**
Top 15% Precision	30.7%	32.1%	**+1.4pp**

5.2 Solution: Coil Intensity Weighting

The solution amplifies gradients for K2 patterns that have high coil_intensity scores. This tells the model: "Pay extra attention to these high-coil K2 patterns because they represent the best trading opportunities."

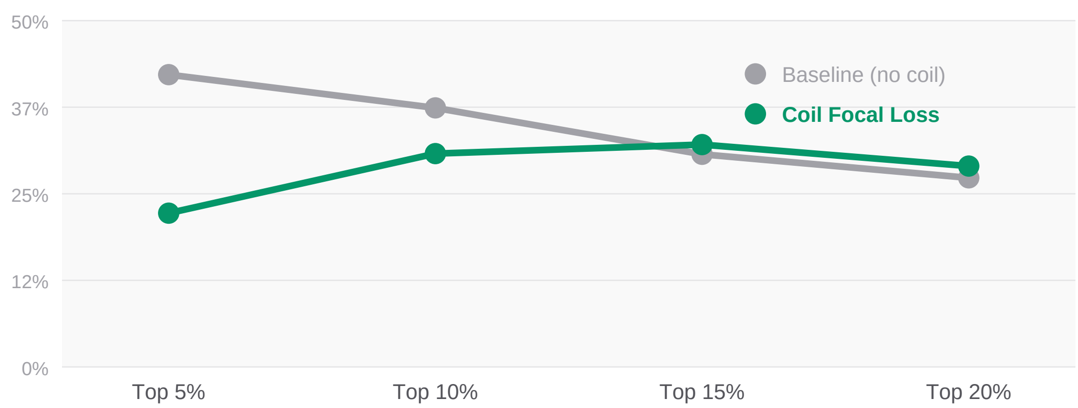
losses/coil_focal_loss.py

```python
class CoilAwareFocalLoss(nn.Module):
    def __init__(self, gamma=2.0, coil_weight=3.0):
        self.gamma = gamma
        self.coil_weight = coil_weight

    def forward(self, inputs, targets, coil_intensity):
        # Standard focal loss
        ce = F.cross_entropy(inputs, targets, reduction='none')
        pt = torch.exp(-ce)
        focal = (1 - pt) ** self.gamma * ce

        # Coil boost for K2 patterns
        is_k2 = (targets == 2)
        boost = 1.0 + coil_intensity * self.coil_weight
        focal[is_k2] *= boost[is_k2]

        return focal.mean()
```

5.3 Why Coil Focal Loss Beats ASL+Coil

We tested combining Coil-Aware Focal Loss with Asymmetric Loss (ASL) for hard negative mining. Surprisingly, the combination performed WORSE than Coil Focal alone. The ASL's aggressive gamma_neg=4.0 pushed the model too far toward K2, sacrificing precision.

Precision @ Top K% (Test Set)



Conclusion: Use --use-coil-focal alone for training. Do NOT combine with --use-asl as it reduces Top 15% precision from 32.1% to 26.3%.

6. Data Quality Controls

Operation Clean Slate (January 2026) identified critical data quality issues that caused a ~19% precision ceiling. The fixes improved Top 15% precision to 32.1%.

6.1 NMS Filter (Protocol Highlander)

Problem: 71.2% of patterns started just 1 day apart, meaning 'different' patterns in train/val/test were actually the SAME consolidation event. This caused 95% data overlap and severe leakage.

Solution: Non-Maximum Suppression groups patterns by ticker, clusters those within 10 days, and keeps only ONE pattern per cluster (the tightest consolidation). "There can be only one."

Metric	Before NMS	After NMS	Reduction
Total Patterns	81,512	2,416	97%
1-Day Gaps	71.2%	0%	100%
Data Overlap	95%	0%	100%
Top 15% Precision	~19%	32.1%	+13pp

6.2 Physics Filter

Removes patterns that are physically impossible to trade profitably:

• Large/Mega caps excluded (0% target rate at scale)

• Pattern width < 2% excluded (edge eaten by slippage)

• Dollar volume < $50k excluded (can't execute without market impact)

6.3 Robust Scaling

Composite features (indices 8-11) use robust scaling with median and IQR instead of z-score normalization. This prevents outliers from dominating the feature distribution.

Robust Scaling Implementation

```
# Robust Scaling for Composite Features
for idx in [8, 9, 10, 11]:  # vol_dryup, var, nes, lpf scores
    median = robust_params[f'feat_{idx}_median']
    iqr = robust_params[f'feat_{idx}_iqr']
    X[:, :, idx] = (X[:, :, idx] - median) / iqr
```

Appendix A: CLI Reference

Training Pipeline

Core Pipeline Commands

```
# Generate sequences with 13 context features
python pipeline/01_generate_sequences.py \
    --input output/detected_patterns.parquet \
    --apply-nms \
    --apply-physics-filter \
    --skip-npy-export

# Train with Coil-Aware Focal Loss (RECOMMENDED)
python pipeline/02_train_temporal.py \
    --sequences output/sequences/eu_13ctx/sequences_*.h5 \
    --metadata output/sequences/eu_13ctx/metadata_*.parquet \
    --epochs 100 \
    --use-coil-focal \
    --coil-strength-weight 3.0 \
    --train-cutoff 2024-01-01 \
    --val-cutoff 2024-07-01

# Generate predictions
python pipeline/03_predict_temporal.py \
    --model output/models/eu_model.pt \
    --sequences output/sequences/eu_13ctx/sequences_*.h5
```

Training Flags

Flag	Default	Description
`--use-coil-focal`	Off	Enable Coil-Aware Focal Loss (RECOMMENDED)
`--coil-strength-weight`	3.0	Coil intensity multiplier for K2 boost
`--use-asl`	Off	Asymmetric Loss (NOT recommended with coil)
`--compile`	Off	torch.compile for 20-50% speedup
`--num-workers`	Auto	DataLoader workers (2 Win, 4 Linux)
`--train-cutoff`	None	Date for train/val split
`--val-cutoff`	None	Date for val/test split