Gated Transformer Network (GTN) - Project Documentation: Version 4

Document Version: 1.0

Model Script: gtn_v10.4.py

Date: June 18, 2025

1. High-Level Overview

1.1. Purpose

The GTN V4 model is a deep learning system designed for **time-series classification**, specifically for predicting directional movements in the Forex market. Given a sequence of historical market data, the model classifies the most likely future market state as one of three signals: buy, sell, or keep. Its goal is to provide stable, reliable, and interpretable trading signals to inform decision-making.

1.2. Core Architecture: The Two-Analyst Approach

At its heart, the GTN employs a dual-tower Transformer architecture. This can be understood with an analogy of two expert analysts working together:

1. **The "Temporal" Analyst (step-wise tower):** This analyst examines the data as a story unfolding over time. It looks at each data point (e.g., each 4-hour candle) in sequence, paying attention to the temporal patterns and the order of events to understand the market's momentum.
2. **The "Feature" Analyst (channel-wise tower):** This analyst ignores the timeline and instead focuses on the relationships between all the different features at every single moment. For example, it might learn that a specific combination of low volatility, high RSI, and certain candlestick patterns is a powerful indicator, regardless of when it happens.

A **"Manager" (Gating Mechanism)** then intelligently combines the insights from both analysts. It dynamically decides whose opinion is more relevant for the current market context. In Version 4, this manager also incorporates its own high-level context (**Residual Connection**) to ensure no critical information from the original input is lost, leading to a more robust final decision.

1.3. Key Enhancements in Version 4

This version represents a major leap forward in stability, performance, and feature representation:

- **Learnable Time Embeddings:** Instead of using static representations of time (like sine/cosine waves), the model now *learns* the unique characteristics and importance of different times of day, minutes, and days of the week.
- **Stationary Price Features:** Raw price data is replaced with price *returns*, a more stable, "stationary" representation that significantly improves model trainability and performance.
- **Enhanced Training Stability:** The architecture incorporates modern best practices like **Pre-Layer Normalization** and **Residual Connections**, which help gradients flow more effectively and prevent the model from "forgetting" important information during training.
- **Sophisticated Model Selection:** A new **Composite Score** is used to select the best model during training. This score rewards models that are not only precise but also balanced in their ability to predict both buy and sell signals, promoting reliability.

1.4. System Workflow

The project follows a standard, end-to-end machine learning workflow:

1. **Data Ingestion:** Loads raw .csv data containing historical market data.
2. **Temporal Split:** Automatically partitions the data into a training set (older data) and a test set (most recent data) to simulate real-world performance.
3. **Feature Engineering:**
    - Calculates stationary price returns.
    - Generates integer-based time features (e.g., hour=14, dayofweek=2).
4. **Preprocessing:**
    - Scales numerical features to a common range using RobustScaler.
    - Encodes the text-based target labels (buy, sell, keep) into numbers.
    - Transforms the data into overlapping sequences (lookback windows) for the model.
5. **Model Training:**
    - Trains the GTN V4 model using the prepared sequences.
    - Employs a cost-sensitive loss function, an adaptive learning rate, and early stopping based on the composite score to find the optimal model.
6. **Evaluation & Interpretability:**
    - Evaluates the best model on unseen test data using a suite of metrics (F1-score, precision, etc.).
    - Generates SHAP (SHapley Additive exPlanations) reports to understand which features are driving the model's decisions.
7. **Artifact Storage:** Saves the trained model, data scaler, and evaluation reports for deployment or further analysis.

2. Technical Deep Dive

2.1. Data Pipeline

The data pipeline is designed to be robust and reproducible.

- **Input Data:** The system expects a CSV file with at a minimum:
    - A datetime column (e.g., Date).
    - OHLC (Open, High, Low, Close) price columns.
    - Binary indicator columns (Is_Doji, gap, etc.).
    - A target column with string labels (e.g., signal).
- **Feature Engineering:**
    - add_stationary_features(): This function is critical. Time-series models work best when the statistical properties of their inputs (like mean and variance) are not time-dependent. Raw prices are non-stationary (they tend to trend up or down). This function converts them into percentage returns, which are generally stationary.
    - **Integer Time Features:** The datetime column is used to generate integer representations for hour, minute, and dayofweek. These integers serve as direct inputs to the model's embedding layers.
- **Preprocessing (preprocess_and_split_data_temporal)**:
    - **Temporal Split:** Data is always split by time to prevent the model from seeing future data during training, which would lead to unrealistic performance estimates.
    - **Scaling:** Numerical features are scaled with RobustScaler. This scaler is less sensitive to outliers than StandardScaler, which is common in financial data.
    - **Sequencing (create_sequences)**: A sliding window approach is used to create input sequences. For a sequence_length of 120, the first sample will be data points 1-120, the second will be 2-121, and so on. The target for each sequence is the label corresponding to the *last* data point in that sequence.

2.2. Model Architecture: GTN_v10_4

The GTN_v10_4 class is the core of the system.

- **Learnable Time Embeddings:**
    - **Concept:** Traditional machine learning sees "14:00" and "15:00" as numerically close. An embedding layer learns that these hours might have vastly different market behaviors (e.g., London close vs. NY open). It maps each integer (e.g., hour 14) to a dense vector of learnable parameters, capturing the "personality" of that time feature.

Implementation: Python
self.hour_embedding = nn.Embedding(24, hour_emb_dim)
self.minute_embedding = nn.Embedding(60, minute_emb_dim)
self.day_embedding = nn.Embedding(7, day_emb_dim)

- **Step-wise Tower & Stability Enhancements:**
    - **Pre-Layer Normalization (norm_first=True):** In a traditional Transformer, normalization is applied *after* the main operations. In Pre-LN, it's applied *before*. This ensures the inputs to the attention and feed-forward layers are always

well-scaled, which dramatically stabilizes the training process and allows for more effective learning.

Implementation (nn.TransformerEncoderLayer): Python
step_encoder_layer = nn.TransformerEncoderLayer(d_model, n_heads, d_ff, dropout, 'gelu', batch_first=True, norm_first=True)

- **Gating and Residual Connection:**
  - **Concept:** After the two "analysts" (towers) produce their outputs, the gating mechanism decides how to blend them. The residual connection adds the model's initial interpretation of the input *back* into this final blend. This acts as a "skip-path" for gradients, ensuring that even in a deep network, the model doesn't lose the original signal. It combats the vanishing gradient problem.

Implementation: Python
# Gated blending of the two towers
gate_vals = torch.sigmoid(self.gate_linear(concat_features))
gated_output = gate_vals * h_step_pooled + (1 - gate_vals) * h_chan_pooled

# Add the residual from the input and normalize
input_residual = x_projected.mean(dim=1)
final_repr = self.final_norm(gated_output + input_residual)

2.3. Training Process

The training loop is optimized for performance and stability.

- **Loss Function:**
  - FocalLoss: Addresses class imbalance (e.g., fewer buy/sell signals than keep signals) by making the model focus more on hard-to-classify examples.
  - CostSensitiveRegularizedLoss: This wrapper adds a penalty term based on a "cost matrix." This allows you to define the business cost of a mistake. For example, incorrectly predicting sell when the market was a buy (a missed opportunity) can be assigned a higher cost than incorrectly predicting keep.
  - AdaptiveLambda: This helper class intelligently tunes the strength of the cost-sensitive penalty during training, preventing it from overpowering the primary learning objective.
- **Model Selection (composite_score):** The best model checkpoint is saved based on a composite score, not just raw precision.
  - $\text{composite\_score} = \text{avg\_precision} - (\text{penalty} \times |\text{buy\_precision} - \text{sell\_precision}|)$
  - This encourages the model to be equally good at identifying both buy and sell opportunities, leading to more reliable and balanced performance.

- **Gradient Clipping:** To prevent unstable training from "exploding gradients" (a common issue in deep networks), gradients are clipped by both their overall magnitude (clip_grad_norm_) and their individual values (clip_grad_value_).

2.4. Evaluation & Interpretability

- evaluate_model_enhanced: Provides a comprehensive report including per-class precision, recall, and F1-score. It specifically calculates and logs the precision for buy and sell classes, which are critical business metrics.
- analyze_model_with_shap: After training, SHAP analysis is run on the best model. It produces plots and a text report that ranks every feature by its overall importance to the model's predictions. This is invaluable for:
    - **Building trust:** Understanding *why* the model makes its decisions.
    - **Feature selection:** Identifying and removing features that provide little to no value.
    - **Debugging:** Spotting if the model is relying on spurious or undesirable features.

3. Strengths and Limitations

Strengths

- **Architectural Stability:** The use of Pre-Layer Normalization and residual connections makes the model easier to train and less sensitive to hyperparameter choices.
- **Advanced Feature Representation:** Learnable embeddings and stationary features allow the model to capture complex temporal and market dynamics more effectively than previous versions.
- **High Interpretability:** Built-in SHAP analysis provides clear, actionable insights into model behavior.
- **Business-Aware Training:** The cost-sensitive loss function allows the training objective to be aligned with real-world financial risk and opportunity costs.
- **Balanced Model Selection:** The composite score metric ensures the selected model is not just accurate but also reliable across different signal types.

Limitations

- **Computational Complexity:** The dual-tower architecture is computationally expensive to train compared to simpler models like LSTMs.
- **Data Requirements:** Like all Transformer-based models, the GTN is data-hungry and performs best with large datasets.
- **Risk of Overfitting:** Although highly regularized, the model's complexity means it can still overfit to training data if not carefully monitored with a proper validation set.
- **Hyperparameter Sensitivity:** While more stable, finding the optimal set of hyperparameters (e.g., sequence_length, d_model) still requires experimentation.

4. Parameter Tuning and Configuration Guide

The following are key parameters from the argparse configuration with tuning recommendations.

Parameter	Default (V4)	Recommendation & Rationale
sequence_length	120	**Tune this first.** This is the lookback window. A value of 120 for 4H data means looking back 20 days. This should be guided by market knowledge. Shorter values capture fast dynamics; longer values capture macro trends but may introduce noise.
d_model	256	**Model Capacity.** The default is a good starting point. Increase (e.g., to 512) if the model is underfitting (both train and validation scores are low and not improving). Decrease (e.g., to 128) if it's overfitting quickly or for faster experimentation.
num_layers	3	**Model Depth.** Deeper models can learn more complex patterns. The combination of d_model=256 and num_layers=3 is robust. Adjust in tandem with d_model.
dropout	0.15	**Regularization.** A crucial parameter to prevent overfitting. Increase (e.g., to 0.2-0.3) if the training accuracy is high but validation accuracy is poor. Decrease if the model is struggling to learn at all.
learning_rate	5e-5	**Optimizer Speed.** The default works well with the onecyclescheduler. For other schedulers, you may need to experiment in the range of 1e-5 to 1e-4.

precision_deviation_penalty	0.25	**Model Balance.** If your final model is excellent at predicting 'buy' but poor at 'sell' (or vice versa), increase this penalty (e.g., to 0.5 or 0.75) to force the model to balance its performance.
cost_* parameters	(various)	**Business Logic.** These are highly domain-specific. They should be configured based on the calculated financial risk of making different types of prediction errors. Start with the defaults and adjust based on backtesting results.

---

6. Comparison: Version 4 vs. Version 3

6.1. Executive Summary

Version 3 (gtn_v10.3.py) was an experiment to test the value of grouped linear projections for different feature types (numerical, binary, time). It used a simplified feature set and a more traditional Transformer architecture.

Version 4 (gtn_v10.4.py) is a significant, production-focused evolution. It abandons the simplistic approach of Version 3 and incorporates several state-of-the-art techniques. It fundamentally changes how features are generated (stationarity) and how time is represented (learnable embeddings), while dramatically improving training stability and introducing a more intelligent model selection criterion.

6.2. Detailed Comparison

Aspect	Version 3 (gtn_v10.3.py)	Version 4 (gtn_v10.4.py)	Impact and Rationale

Input Features	Raw Prices (Open, High, etc.), Cyclical Time (e.g., hour_sin)	**Stationary Price Returns**, **Integer Time** (e.g., hour)	**V4 is more robust.** Stationary features are a best practice for financial time series, leading to more stable training. Integer time is a prerequisite for the more powerful embedding layers.
Time Handling	Simple nn.Linearprojection on sine/cosine features.	**Learnable nn.Embeddinglayers.**	**V4 is far more expressive.** It allows the model to learn a rich, nuanced representation for each time step, capturing complex seasonalities that a linear layer cannot.
Architecture	Standard Gating Mechanism.	Gating Mechanism + **Final Residual Connection.**	**V4 has better gradient flow.** The residual connection helps prevent vanishing gradients, allowing for deeper or more complex models to be trained effectively.

Normalization	Post-Layer Normalization (norm_first=False).	**Pre-Layer Normalization (norm_first=True).**	**V4 has superior training stability.** Pre-LN is a modern standard that reduces the need for learning rate warm-up and makes training less volatile.
Model Selection	Based on average Buy/Sell precision.	Based on a **Composite Score**(penalizes deviation between buy/sell precision).	**V4 selects for more reliable models.** It explicitly avoids models that are good at one signal type but poor at another, promoting balanced performance.
Default Size	Smaller model (d_model=128, n_heads=4, num_layers=2).	Larger model (d_model=256, n_heads=8, num_layers=3).	V4's architectural improvements make it possible to reliably train larger, more capable models without them becoming unstable.

6.3. Performance Trade-offs

- **Accuracy:** Version 4 is expected to achieve **higher accuracy and better generalization** due to its superior feature representation (stationarity, embeddings) and more stable, robust architecture.
- **Stability:** Version 4 is **significantly more stable** to train. The combination of Pre-LN, residual connections, and advanced gradient clipping makes the training process less likely to diverge and less sensitive to the choice of learning rate.
- **Interpretability:** Both versions use SHAP, making them interpretable. V4's feature importance report may show that the new embedding dimensions are important, which, while abstract, confirms that the model is effectively learning from time.
- **Training Speed:** For the same set of hyperparameters, V4 might be negligibly slower due to the added embedding lookups and residual connection. However, its stability

often allows for faster convergence, potentially leading to **shorter overall training times**.

6.4. Recommendation: When to Choose Which Version?

- **Choose Version 3 for:**
    - **Academic Comparison:** To serve as a simpler baseline to demonstrate the specific uplift provided by V4's enhancements.
    - **Resource-Constrained Prototyping:** If you need an absolute bare-bones model for initial testing on very limited hardware, though V4 is still recommended.
- **Choose Version 4 for:**
    - **All Practical and Production Use Cases.**
    - **Achieving the Best Possible Performance.**
    - **Ensuring Reliable and Stable Training.**
    - **Building a Model Aligned with Business Metrics (via cost-sensitive loss and composite score).**

In conclusion, **Version 4 should be considered the new standard for this project.** It is a direct and substantial upgrade over Version 3, incorporating multiple modern deep-learning practices that lead to a more powerful, stable, and ultimately more useful model.