FutureFormer: The Market Pays Attention

Adapting a GPT-style transformer for predicting financial time series

Andrew Taylor

Masters of Science Candidate, Johns Hopkins EP, Department of Artificial Intelligence

Edward Staley

Instructor, "ChatGPT From Scratch" Johns Hopkins EP

Dr. Erhan Guven

Professor, Johns Hopkins EP, Department of Artificial Intelligence

Abstract.

This report is two things: 1) A rare glimpse into the development process of a successful signals engine (predictor) for high frequency trading, and 2) a series of ablation studies to try to quantify the performance contribution of several transformer architecture innovations adopted from the recent literature, that were employed in the final design. The final design will be compared on a fixed token budget to a vanilla implementation of GPT-2 architecture that was developed in class. This report documents the design, implementation, and evaluation of a decoder-only transformer, FutureFormer, trained to predict short-horizon directional moves in the micro Nasdaq-100 futures contract (MNQ). This paper is both part technical report and also an engineering memoir. This document is meant to be revealing, academic, and yet practically helpful, relaying the ups and the downs of 50 days of nearly non-stop work. For this, it offers an empirical comparison between a heavily modified transformer and a vanilla GPT-2-style baseline implemented in class, with particular emphasis on architectural innovations such as ALiBi positional bias [3], Grouped-Query Attention (GQA) [4], RMSNorm with pre-normalization [5], and SwiGLU feed-forward blocks. [6] Hardware efficiency leverages exact SDPA kernels with FlashAttention. [7] The target task is a three-class next-token classification problem, predicting whether the cumulative price change over a 10-bar horizon will be down, unchanged, or up. Using six sequential MNQ contracts and a continual pre-training regime with progressive learning-rate decay, the best configuration (FutureFormer-S, about 45M parameters) achieves 69.03% out-of-sample directional accuracy with materially elevated precision on up and down classes relative to base rates. Larger models under fixed data and token budgets underperform echoing the Chinchilla-style observation that data scale

matters more than parameter count. [22] First, in order to decompose the architectural contribution of these innovations to performance, the threshold machine learning challenge had to be conquered – do transformers make good predictors of financial time series? This is a problem that famous hedge funds like Citadel and Point 72, etcetera, hire teams of men to tackle, each paid handsomely and given far more time, resources and the finest tools. This report proves both, in one fell swoop, again that transformers *are* good next-token predictors of financial time series (not just text) for high frequency trading, and more importantly: that a mere mortal, the solitary retail quant analyst without a PhD, can in fact be successful at such a feat with a consumer-grade computer. Future work aims to improve upon these initial results with extensive tuning and alternative choices.

Introduction.

The recent success of causal transformers in natural language processing [1] has naturally motivated attempts to deploy the same architectures in financial prediction. At first glance the mapping seems straightforward: financial time series can be viewed as sequences of "tokens" evolving in time, and market moves can be framed as next-token prediction. In practice, however, there is a fundamental representation gap. GPT-style models are architecturally built around discrete tokens drawn from a fixed vocabulary, trained with a Softmax output layer and cross-entropy loss. Financial data, by contrast, are continuous floating-point values observed under nonstationary dynamics, with no natural vocabulary and with extremely skewed, regime-dependent outcome distributions. This work addresses that "vocabulary gap" by deliberately reframing price prediction as a discrete classification problem. Instead of asking the model to emit continuous prices directly, I ask it a simpler but more trading-relevant question: ten bars from now, will the cumulative move be down, unchanged, or up? Each bar is a fixed-volume aggregate of trades, and each token corresponds to the realized directional label over the next 10-bar horizon. The label is brought forward in time, not the features, eliminating future leakage while letting the model learn the ground truth. This formulation preserves a probabilistic Softmax head, yields well-defined cross-entropy loss and precision/recall metrics, and makes it possible to interpret the model's logits as actionable signal probabilities for short-horizon trades.

The motivation for the project was personal as well as technical. Many years ago I worked with a colleague who went on to a successful career in algorithmic trading while I remained stuck in discretionary trading with inconsistent results. When I later discussed machine learning with him and his partner, they remarked that "we have made a lot of money on models that were merely 51% accurate", and offered me a role in their business if I could create a useful model. Suddenly, this wasn't just a sleepy class project, I was energized.

For a poor student, the prospect of contributing a model with even a small edge in exchange for a share of profits was compelling. That 51% target became the concrete performance bar for this study: the model did not need to be state-of-the-art, but it had to be better than guessing by an amount that could plausibly survive transaction costs, slippage, and implementation drag. Achieving this seemingly modest goal turned out to require a heavily engineered custom pipeline, as well as a full transformer adaptation, and a long sequence of painful debugging cycles and philosophical pivots.

The scope of this work is deliberately restricted. All experiments are strictly in-silico. The signal engine is validated exclusively on out-of-sample test data drawn from the tail of the last MNQ contract, and no live execution or paper trading was performed (yet). As a result, this report is about predictability, not profitability. The model's outputs are not adjusted for fees, slippage, market impact, or liquidity constraints, and no capital allocation, risk management, or execution layer is optimized. In short, I have built a weather forecast, not a farm, which is the first step to making a trading robot. The central question is whether a GPT-style transformer can learn useful directional structure in high-frequency futures data when appropriately discretized and regularized, and whether the architectural tricks popularized in large language models actually deliver value in this small, noisy, financial regime.

Data and problem formulation.

The underlying dataset consists of high-frequency tick data for the micro Nasdaq-100 futures (MNQ) covering six sequential contracts from June 2024 through November 2025, roughly the maximum history available from my data provider at the time of the study. Raw trade data were first filtered to regular trading hours and then aggregated into fixed-volume bars. Volume bars were chosen for two reasons. First, they implicitly adjust the temporal resolution of the model: during high-liquidity, high-volatility periods, the model receives more bars (and thus more training tokens), while in illiquid, low-opportunity periods it naturally slows down. This aligns the model's "clock" with market activity rather than wall time, avoiding competition with big players and co-located computers, making it more active precisely where available alpha is more likely to exist. Second, volume bars provide a more granular and economically meaningful unit of market activity than arbitrary fixed-time bars. Starting from approximately 330 million raw ticks across six recent contracts, the barification step reduces the effective dataset by several orders of magnitude while still preserving microstructural information in derived features. Volume/dollar/imbalance bars are well-studied alternatives to time bars; they adapt the sampling clock to market activity and can mitigate heteroskedasticity and diurnal effects [9][10].

The predictive task is defined as follows. For each bar t, features summarize contemporaneous and historical information up to t, and the label y_t is the sign of the cumulative price change over the next 10 bars. If the 10-bar horizon finishes strictly above the current price by more than a small tolerance (in these experiments merely breakeven), y_t = Up. If it finishes strictly below by more than a tolerance (which will be increased in the final version to avoid the risk ambiguous cases, and to an extent bidirectional volatility), y_t = Down. Otherwise, y_t = Unchanged. In the final experiments, this simpler "sign of return" target outperformed earlier triple-barrier formulations (with tolerance > 0) that have explicit up/down thresholds and a wide unchanged band, largely because it removed those edge cases where labels were very sensitive to barrier placement. Apparently this is a trade-off I detected in non-zero tolerances for labeling market activity, something I plan to investigate further. The dataset is heavily imbalanced in time but only moderately imbalanced between classes of directional outcomes, where profit is possible. For the final contract, class frequencies are approximately 46.1% Down, 7.5% Unchanged, and 46.4% Up, and previous contracts were similarly balanced, with healthy autocorrelation of labels. This initial finding bolstered my choice to remove the labeling threshold. The model is trained on sliding windows of length 512 bars with stride 8, yielding context-conditioned next-token classification samples. Validation stride was only 1, reflecting in the held-out set the conditions of live trading where an actionable prediction is made for every bar. Total training windows were about 49,000 and across six contracts the model sees approximately 396,000 unique bars and on the order of 1.6×10^8 tokens total.

Bridging float to token.

Adapting a text transformer to financial data requires a translation layer between continuous inputs and discrete targets. On the input side, each bar is represented by a vector of engineered features: price changes, volatility estimators, microstructure indicators, order-flow measures, and time-of-day encodings. These remain continuous and are projected into the model's hidden space via a learned linear layer; there is no discretization of features themselves. On the output side, however, the model predicts a categorical distribution over three directional classes using a Softmax head. It was when the task was reframed as a discrete classification problem that the model began to show meaningful structure in its logits (passing through a confidence threshold of 0.7), with the ability to express uncertainty via calibrated class probabilities rather than a single point estimate.

Model architecture and variants.

FutureFormer is implemented as a decoder-only causal transformer [1] inspired by GPT-2 but modified along several axes motivated by recent transformer research. On top of the

linear projection from tabular features to the model dimension, the architecture stacks a configurable number of transformer blocks, each consisting of multi-head attention followed by a feed-forward network, with residual connections around both sublayers. The key differences from a vanilla GPT-2-like implementation are as follows. First, positional information is not represented by mere learned absolute embeddings but by ALiBi, a linear positional bias applied directly to attention scores that penalizes long-range interactions and encodes a strong recency bias. [3] This is particularly appealing for financial time series, where the most recent bars typically dominate predictive power and where length extrapolation is important. Second, the attention mechanism is implemented as Grouped-Query Attention: multiple query heads share a smaller number of key/value heads, reducing KV-cache size and inference memory without significantly degrading performance. [4] Third, RMSNorm with pre-normalization replaces LayerNorm with post-normalization, improving optimization stability in deeper networks.[5] [20] [21] Fourth, SwiGLU is used in the feed-forward blocks to provide a gated, high-capacity nonlinearity. [6] Finally, the implementation directly leverages PyTorch's scaled-dot product attention with FlashAttention-style kernels when available, allowing long contexts to be trained efficiently on a single RTX 5090. [7] While it would appear prima facie that most of these adaptations were performance-oriented, they each had considerable impact on performance, as we will see in the ablations section.

Three model sizes were trained to probe scaling behavior under fixed data and fixed token budgets: FutureFormer-S, FutureFormer-M, and FutureFormer-L. All three share the same architectural motifs described above and differ only in depth and width. Table 1 summarizes their configurations and resource footprints for the final runs reported here, all using a context length of 512 bars and a training stride of 8.

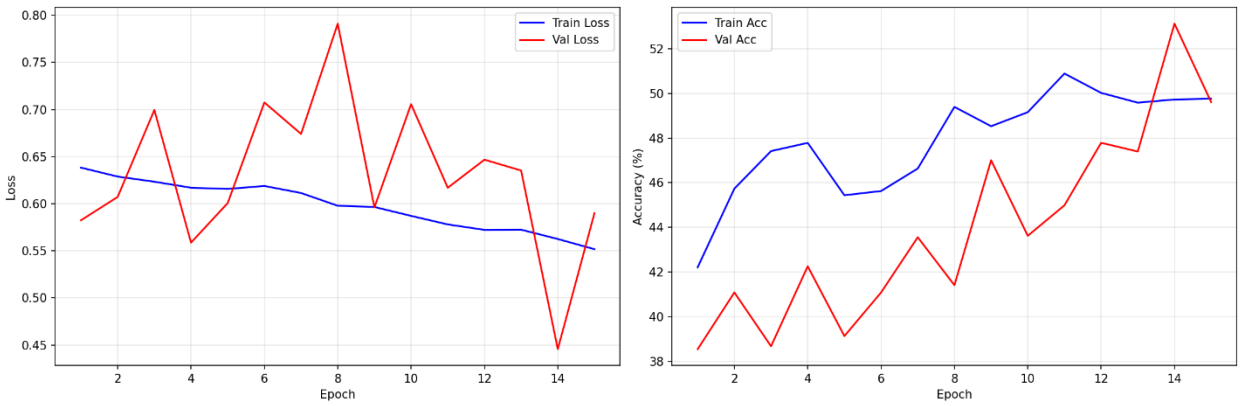
Model	Layers	Hidden dim	Heads / KV groups	Context length	Params (M)	Peak VRAM (GB)	Tokens seen (M)	Throughput on last contract (tokens/s)
FutureFormer-S	8	384	6 / 3	512	≈45	15.17	163.83	398,065
FutureFormer-M	12	512	8 / 4	512	≈70	30.01	165.41	191,962
FutureFormer-L	16	768	12 / 4	512	≈130	31.56	144.04	59,852

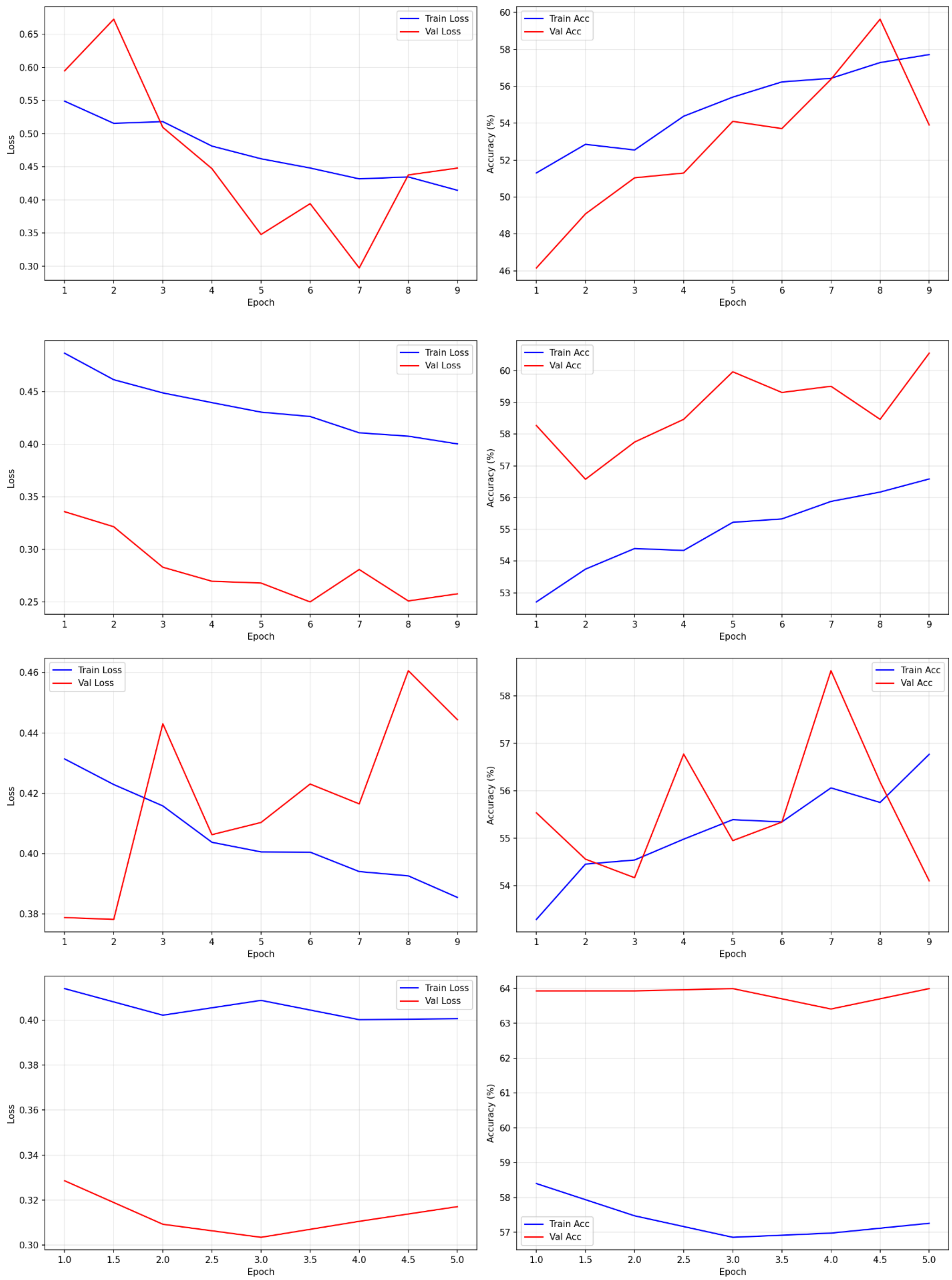
Table 1. Model configurations and resource usage on MNQ sequential-contract training on a 32GB VRAM limit

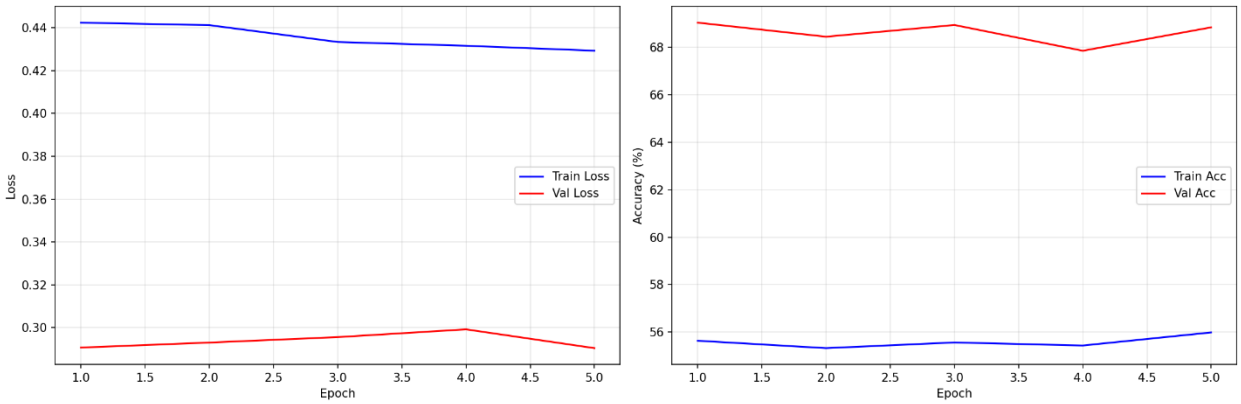
Training regime and continual learning.

A central design choice was to treat the six MNQ contracts not as one pooled dataset but as a chronological series of training stages. This makes sense for futures because although the contracts are sequential they overlap substantially in time. The model is first trained from scratch on the earliest contract, then retrained sequentially on each subsequent contract in order, carrying forward weights and optimizer state. This is a simple but realistic form of continual pre-training: later contracts reflect newer market regimes, and the model must adapt without catastrophically forgetting earlier structure. Several engineering changes were required to make this work in practice. Learning rate scheduling originally used a cosine schedule that reset the rate at the start of each contract, inadvertently overwriting previous knowledge. This was replaced by a constant schedule within each contract and an explicit progressive decay across contracts: contract k uses a learning rate $\eta_{\mathrm{k}} = \eta^{1} \cdot 0.5^{k-1}$, starting from 3x10⁻4 and ending near 1.576e-06on the sixth contract. Warm-up is applied only to the first contract; subsequent contracts start at their assigned learning rates with no warm-up to avoid destabilizing already-trained weights.

The loss function evolved through experimentation. Cross-entropy with inverse-frequency class weights was used to counteract class imbalance, with careful attention to not over-weight the unchanged class, since false "no trade" predictions are much less costly than false positives on directional classes. Label smoothing of 0.05 was added initially to reduce overconfidence and improve calibration, but turned out not to be needed. In later runs, focal loss with γ = 1.3 was layered on top of class weights to emphasize hard minority examples, particularly rare but important directional transitions.[11] Early stopping was based not on accuracy but on Fβ with β = 0.5, making false positives four times as costly as false negatives, reflecting the asymmetry between taking a bad trade (certain loss plus fees) and missing a good one (opportunity cost only). Gradient clipping at 2.0, corrected gradient accumulation to only 1, and aggressive removal of catastrophically unscaled features were all necessary to prevent training oscillations and class "thrashing."




Train Loss
Val Loss
Loss
Epoch
Train Acc
Val Acc
Accuracy (%)
Epoch
Train Loss
Val Loss
Loss
Epoch
Train Acc
Val Acc
Accuracy (%)
Epoch
Train Loss
Val Loss
Loss
Epoch
Train Acc
Val Acc
Accuracy (%)
Epoch
Train Loss
Val Loss
Loss
Epoch
Train Acc
Val Acc
Accuracy (%)
Epoch



Figures 1-6 (top-bottom): The training history on 6 successive but overlapping temporal datasets (contracts)

Futures contracts naturally form a successive series of datasets that are related and build upon the history and information of the prior dataset(s). My data source limited me to the 6 most recent contracts, so that gave me a fixed budget of about 333,000,000 ticks from which to aggregate "bars" of activity each with features to show the model. All this made me opt for a continued pre-training training strategy rather than a fine-tuning approach, because I wanted the model to learn from a building curriculum where past and far past information was not to be considered fundamental (the markets started long ago). Freezing layers would overemphasize past market behavior, regimes, and priors. I implemented continued pre-training by loading checkpoints of weights with each new contract, carrying over the optimizer state, and decaying the learning weight drastically (by 50%) with each new contract. This gave the effect of cumulative learning where the recent market action was closely tied to "reasoning", while earlier market action was more of a context and a basic understanding. Figures 1-6 show how learning was turbulent at first, but then settled down nicely as the learning rate decayed, homing in on a relatively convex optimum. Early attempts with a learning rate of 1e-3 were too aggressive, causing thrashing, and the optimum was not found until I reduced the initial learning rate to 3e-4, a 70% reduction. Monotonic increases to accuracy proved that a continuity of cumulative learning had been achieved.

Scaling behavior and main results.

I think it is best to present the results before the method and ablation studies so that they follow naturally from the numbers. The central fixed-token experiment compares the three FutureFormer sizes under similar data and optimization regimes. All models were trained on the same six sequential contracts, with identical context length, stride, optimizer hyperparameters, lr decay, and so on. This made for a controlled experiment wether or not, in this context, "size matters". Table 2 reports the final out-of-sample metrics on the held-out tail of the sixth contract for each model. The goal is to focus on the most recent and

most representative dataset. Contract 6 alone provides a clear view of how the architecture behaves and how the largest ablation (Vanilla GPT2) influences predictive structure.

Model	Accuracy (%)	Macro F1	F_Beta (0.5)	Macro Precision	Macro Recall
FutureFormer-S	69.03	0.689	0.6903	0.6809	0.6884
FutureFormer-M	51.48	0.508	0.5087	0.5129	0.5158
FutureFormer-L	51.08	0.499	0.5074	0.5276	0.5138

Table 2. Final validation performance on the sixth MNQ contract (10-bar horizon, three-class classification).

Contrary to my naive expectations, the smallest model delivers the best accuracy and F1 under this fixed data regime, with performance deteriorating as parameter count increases. The medium and large models exhibit higher training loss, worse calibration, and clear signs of overfitting and instability, despite having nominally greater capacity. Given the modest dataset size and strong temporal dependencies, this is consistent with the view that in data-limited regimes one should prioritize matching model size to data scale rather than blindly scaling parameters. In spirit, this behavior echoes the Chinchilla observation that optimal performance for a given compute budget occurs when model size and dataset size are balanced rather than maximized independently.[22] Bigger has become better in commercial foundation models, but only when matched with bigger data.

Small model per-class behavior.

Because the intended application is trading, macro metrics are less informative than per-class precision and recall on the directional classes. For FutureFormer-S on the final contract, the last epoch before early stopping produced the following per-class metrics and class-frequency baseline for comparison.

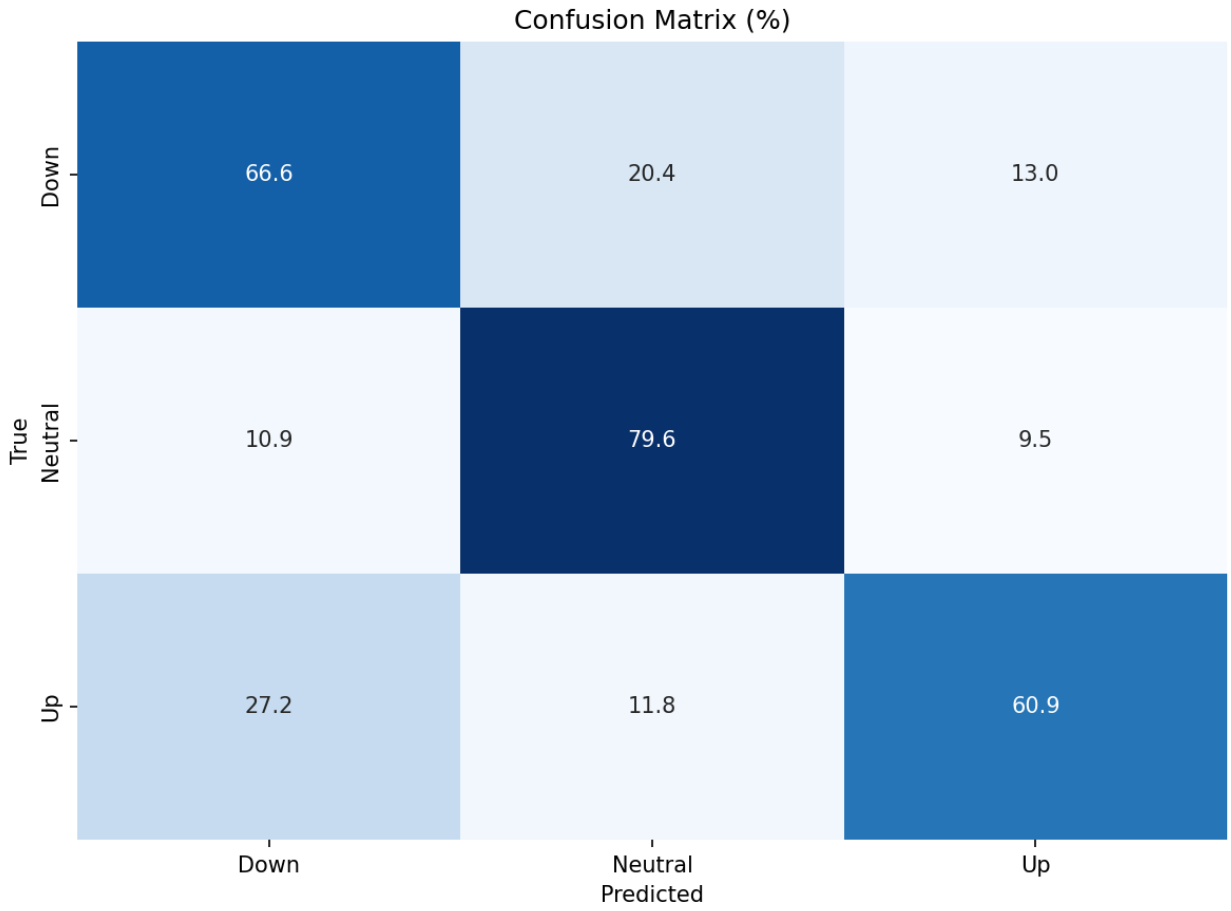


Figure 7. FutureFormer-S Confusion matrix for the MNQ 12-25 contract representing the most recent trading activity to mid-November 2025

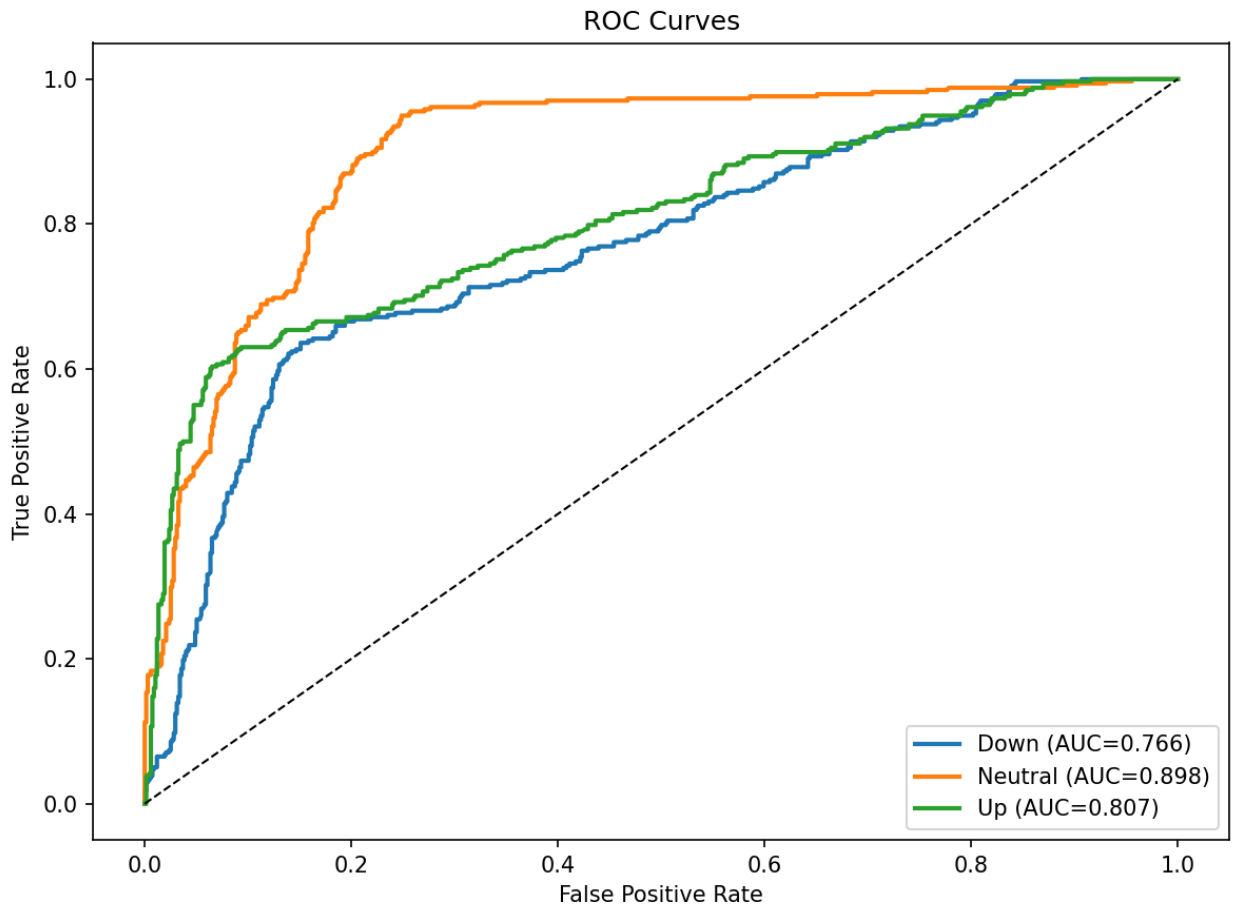


Figure 8: FutureFormer-S ROC curve for the final most recent contract

Quantity	Down	Unchanged	Up
Class frequency (%)	46.1	7.5	46.4
Precision	0.64	0.716	0.711
Recall (validation epoch)	0.663	0.799	0.604
F1 score (final evaluation)	0.650	0.751	0.664

Table 3. FutureFormer-S per-class metrics versus base rates on the sixth contract.

Relative to the unconditional base rates implied by the class distribution, the down precision of 0.64 represents a 17.9 percentage-point absolute lift over 46.1%, and the up precision of 0.711 represents a 24.7 percentage-point absolute lift over 46.4%. These lifts confirm the identification of some market alpha, and more importantly, indicate that when

the model does issue a directional signal it is correct meaningfully more often than the raw frequency of that direction in the data. From a trading perspective these are the numbers that matter: precision on up and down predictions quantifies how often a proposed long or short would have been directionally right over the 10-bar horizon, which is the first step to profitability. Unchanged is deliberately treated as an acceptable form of error when mispredicted, since sitting out is generally cheaper than entering a bad trade.

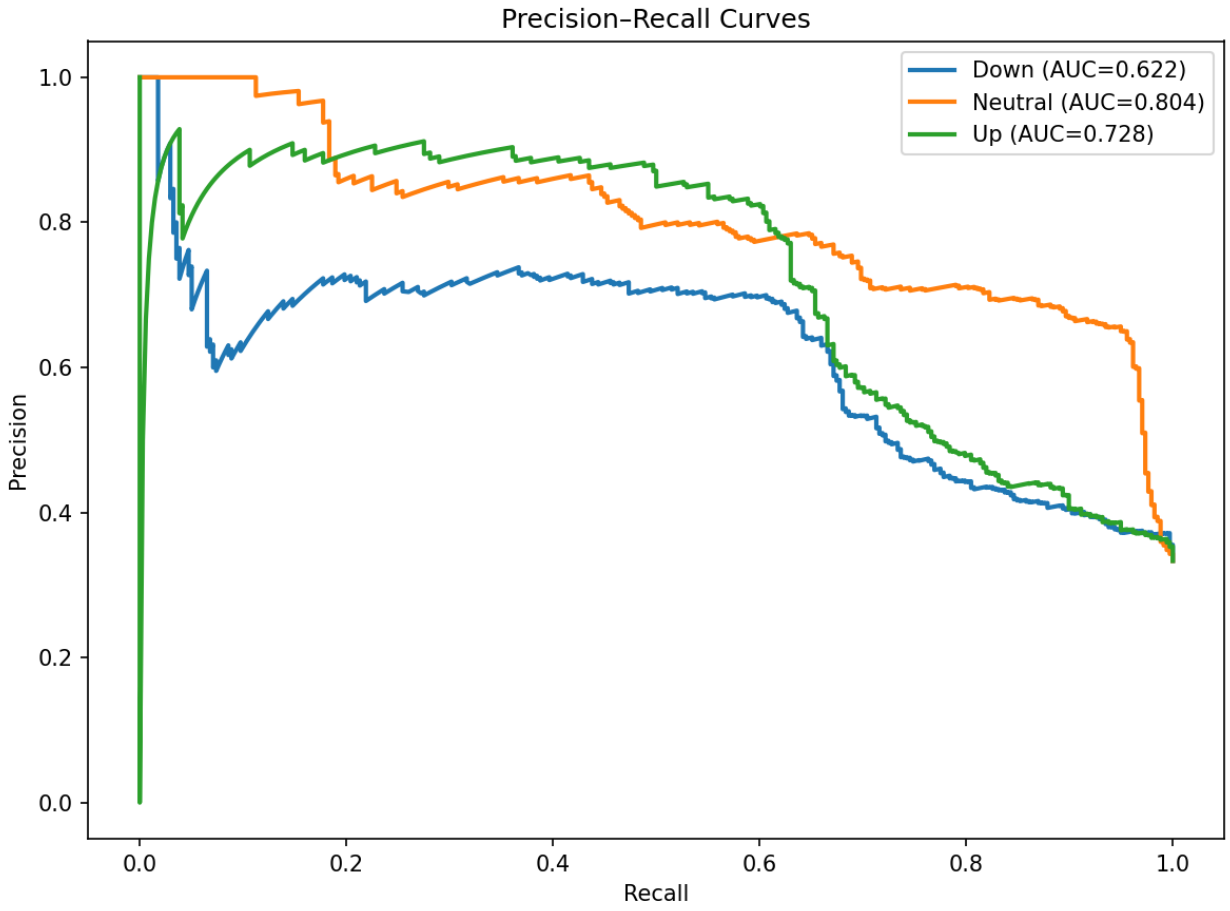


Figure 9. Precision-Recall Curve for FutureFormer-S on the final contract

Baselines.

To interpret FutureFormer's performance in context, I compared it with three additional baselines: a vanilla GPT-2-style transformer we created in class, adapted to the same three-class next-token task, a simple two-layer one-dimensional convolutional neural network (1D CNN), and a purely heuristic persistence model that predicts each label as equal to the previous bar's label. Together, these models span the spectrum from simple

inductive biases (persistence), through classical sequence models with strong locality bias (1D CNN), up to fully attention-based architectures (GPT-2 and FutureFormer).

The vanilla GPT-2 variant uses the same overall training regime as FutureFormer: decoder-only causal attention, three-class Softmax head, cross-entropy loss with class weighting, sequential training across six MNQ contracts, and evaluation on the tail of the sixth contract under the bar_labelH10 target. Architecturally, it hews closer to the classic GPT-2 stack: learned positional embeddings instead of ALiBi, standard multi-head attention instead of GQA, LayerNorm with post-normalization instead of RMSNorm with pre-normalization, and a conventional GELU-type feed-forward network rather than SwiGLU. When these components are removed together, performance drops from roughly sixty nine percent to around fifty percent, indicating that these architectural choices collectively express the inductive structure needed to stabilize forecasting in this domain. The largest effect is observed in the Up class where both recall and F1 deteriorate when these components are absent. The absence of these stabilizing elements produces a model that leans toward the prior distribution and loses sensitivity to price bursts, as shown in Table 4. Increasing model size without expanding available signal produces a parallel pattern. Larger models become more sensitive to noise, leading to reduced stability in the most difficult class. This reinforces the idea that FutureFormer-S sits at an effective capacity sweet spot for the available data. In fact, I may shrink the model even further in future tests. Contract 6 demonstrates this effect cleanly. A notable revelation is that these architectural choices appear to encode useful bias toward the temporal structure of order flow itself and that removing them leads to predictable degradation in the most profitability relevant region of the label space.

For the two transformer-style models, the per-class comparison helps to separate cosmetic improvements from trading-relevant ones. Table 4 summarizes the class-wise metrics for FutureFormer-S and the vanilla GPT-2 baseline on the final evaluation slice of the sixth contract.

Metric	Class	FutureFormer-S	Vanilla GPT-2
Precision	Down	0.72	0.52
	Unchanged	0.70	0.49
	Up	0.63	0.46
Recall	Down	0.78	0.55
	Unchanged	0.72	0.49
	Up	0.63	0.34
F1	Down	0.75	0.53
	Unchanged	0.71	0.49
	Up	0.60	0.38

Table 4. Per-class performance comparison: FutureFormer-S vs Vanilla GPT-2 on the sixth MNQ contract

Per Class Interpretation

The pattern in Contract 6 shows that the Down and Unchanged classes continue to receive stronger recall and F1 values than the Up class in every model tested. This reflects the structure of market movements in the MNQ futures series, where downward excursions are more numerous and often clearer in their microstructure footprint. The Up class, influenced by shorter lived bursts and asymmetric order flow, remains the most challenging to capture. The strongest model, FutureFormer-S, sustains the most stable Up class performance whereas larger models and the Vanilla configuration exhibit a tendency to collapse toward the prior and under predict the Up class in particular. This is important for trading because insufficient Up recall corresponds to an avoidable loss of profitable upside events.

Reference Context

The earlier baselines provide orientation but are not central to interpreting Contract 6. Their behavior is consistent with the idea that simpler sequence models or naive benchmarks capture only coarse structure and do not reach the reliability demonstrated by the best model in this evaluation.

FutureFormer-S's advantage lies primarily in directional precision. When it says up or down it is right more often than the baseline by about 20 percentage points on down moves and about 17 percentage points on up moves, and the GPT-2 model also lacks competitive or better recall for some classes. As noted, for a trading application driven by these logits the extra precision is worth more than the lost recall because each directional false positive incurs real costs, whereas a missed opportunity mostly incurs regret.

The Simple1DCNN baseline takes a very different approach. Instead of attention over tokens, it performs local convolutions over time on the feature channels. Concretely, the model receives an input tensor of shape (batch, seq_len, features), transposes it to (batch, features, seq_len), and passes it through two Conv1d layers with ReLU activations and batch normalization. The first layer maps the n_features channels to 32 filters with a kernel size of 3 and symmetric padding, followed by a MaxPool1d with stride 2 that halves the temporal resolution. The second layer maps 32 channels to 64 with a kernel size of 5, again followed by batch normalization, ReLU, and MaxPool1d with stride 2. After these two stages the effective temporal length is context_len / 4 (for context_len = 512, this yields 128 positions), and the feature map has 64 channels. The model then flattens this (batch, 64, pooled_len) representation into a single vector per example, applies dropout and a fully connected layer to a 128-dimensional hidden representation, and finally projects to three logits via a final linear layer. This architecture has a fixed receptive field and an explicitly local inductive bias: each convolutional filter learns short motifs in the bar sequence, and successive pooling layers build increasingly large local summaries that eventually feed into a global classification head.

Despite its simplicity, the Simple1DCNN performs surprisingly well under the bar_labelH10 setting on 1000-contract volume bars. Trained sequentially across the same six MNQ contracts, it achieves a mean accuracy of 0.5356 and mean F0.5 of 0.5343, with a final-contract accuracy of 0.5483 and F0.5 of 0.5493. These mean metrics are averaged over six contract-specific evaluations and the final metrics correspond to the last contract, mirroring the transformer evaluations conceptually if not exactly in bar specification. Across contracts, ROC AUC falls in the 0.63–0.74 range and expected calibration error (ECE) remains modest, often below 0.1, indicating that the model not only predicts the correct class reasonably often but also assigns probabilities that track empirical frequencies. That a simple two-layer CNN can reach roughly 54–55% accuracy and F0.5 suggests that many of the predictive patterns in these data are short-range and translation-invariant, exactly the kind of structure that 1D convolution excels at capturing.

At the other extreme of sophistication, the persistence model is almost insultingly simple. It does not learn any parameters at all. Instead, it walks through the dataset in index order

and predicts each label as equal to the previous observed label. For the very first window in a sequence, or whenever the index is zero, it defaults to predicting the unchanged class (index 1). In code, the predict method inspects the dataset's index ordering, obtains the previous bar's label y_prev, and emits it as the current prediction; it also supports a one-hot probability output for evaluation routines that expect probabilities. This baseline embodies the naive belief that whatever just happened will happen again in the next step, a kind of lag-1 Markov assumption with no conditioning on features.

As expected in a noisy, mean-reverting market, the persistence model performs poorly on the bar_labelH10 task. Across six contracts it delivers a mean accuracy of 0.3335 and mean F0.5 of 0.3335, with final-contract accuracy of 0.3432 and F0.5 of 0.3432. These figures are barely better than a uniform three-class random guess (1/3) and are markedly worse than just predicting the majority class for each contract. ROC AUC hovers around 0.49–0.51, effectively indistinguishable from random, and calibration is extremely poor, with ECE near 0.65–0.68 because the model assigns degenerate one-hot probabilities that are completely misaligned with realized frequencies. The persistence baseline is therefore useful primarily as a sanity check: any model that fails to beat it is not learning anything beyond trivial serial correlation. The persistence baseline is a useful floor: a model that fails to beat it likely isn't learning beyond trivial serial dependence. However, our H=10 labels are defined on overlapping windows, which mechanically inflates short-lag autocorrelation; accordingly, the strong ACF at lags 1–5 is largely expected, and the near-zero ACF at lag 10 mostly reflects the end of overlap rather than a deeper transition property of the process. (Table 5) But from this, since useful dependence fades by ≈H, I can argue for (i) moderate context (e.g., 256–512 bars) with strong decay, (ii) local microstructure features (order-flow, short-horizon vol, range), and (iii) limited benefit from very long contexts unless I target regime effects, which I have planned for a future model. This led me to cancel the context stress tests I had planned, when budgeting for time.

Multi-Lag Autocorrelation of target label (sign of return after 10 bars)	
t, t+1	0.729
Lag 2	0.6064
Lag 5	0.3492
Lag 10	0.0214

Table 5: Label autocorrelation over 4 lags

Table 6 synthesizes these baselines alongside the primary FutureFormer-S model, using mean performance across contracts where available and final-contract metrics for all models. For FutureFormer-S and vanilla GPT-2 only final validation metrics on the sixth contract were computed, so their mean values are left unspecified.

Model	Architecture	Mean Accuracy	Mean F0.5	Final Accuracy	Final F0.5
FutureFormer-S	Decoder-only transformer (ALiBi, GQA)	–	–	0.6903	0.6903
Vanilla GPT-2	Decoder-only transformer (GPT-2 style)	–	–	0.4961	0.5023
SimpleCNN	Two-layer 1D CNN	0.5356	0.5343	0.5483	0.5493
Persistence	Lag-1 label heuristic	0.3335	0.3335	0.3432	0.3432

Table 6. Summary of main model and baselines on MNQ bar_labelH10 experiments.

Taken together, these results draw a coherent picture. The persistence model establishes a floor at essentially random performance. The vanilla GPT-2 baseline rises well above this floor and demonstrates that a generic transformer with no finance-specific architectural tweaks can extract meaningful structure from volume-bar sequences, one of my theses. FutureFormer-S moves further up by introducing architectural features: ALiBi, GQA, RMSNorm with pre-normalization, and SwiGLU, that stabilize optimization and sharpen directional precision at roughly the same parameter scale. The 1D CNN, however, is arguably the most revealing baseline: it achieves a high mean and final F0.5 among the baselines tested and does so with a much simpler architecture and a very explicit inductive bias toward local patterns.

From a theoretical standpoint, this ranking of models is not surprising when viewed through the lens of inductive bias and the structure of financial time series. Short-horizon bar_labelH10 returns are heavily influenced by local microstructure patterns: short runs of buying or selling, bursts of volatility, and brief liquidity vacuums. These phenomena are local in time and approximately translation-invariant; a particular pattern of price and volume over 10–20 bars has similar implications whether it occurs at bar 100 or parhaps even bar 10,000. One-dimensional convolutions with small kernels are almost perfectly matched to this structure. Each filter slides across the sequence and learns detectors for motifs such as “sharp selloff followed by low-volume bounce” or “compressed volatility followed by breakout,” while max pooling provides a crude form of invariance to small temporal shifts and reduces the effective sequence length. [29[30] The final fully connected layers then combine these motif activations into a global class decision. Under this view, a shallow CNN that sees enough data is essentially a learned library of micro-patterns and a simple decision rule over them, which is exactly what short-horizon

forecasting calls for. I plan to investigate it farther with a different labeling schema.
For these reasons, a plausible next step is to treat 1D CNNs not just as baselines but as first-class candidates for the forecasting engine itself, especially in the low-data, short-horizon regime. Deeper CNNs with dilated convolutions could extend the receptive field to hundreds of bars while preserving locality, and hybrid architectures that combine convolutional front-ends with lightweight attention over higher-level features might capture both local microstructure and longer-range regime information. The present results suggest that any such hybrid should start from a strong convolutional backbone rather than assuming that a transformer alone will naturally discover the right local patterns.

Transformers, by contrast, are designed for long-range dependency modeling and flexible pattern matching across the entire context window. Self-attention allows every position to look at every other position, which is powerful when long-distance relationships matter, as in language modeling or in time series with multi-scale seasonalities. In the present setting, however, most of the signal appears to be local. The transformer's extra flexibility becomes a liability when data are limited, because it must learn to ignore almost all of its attention capacity and focus on a narrow band of recent bars. Architectural tweaks like ALiBi (and a limited context size compared to other applications) help by hard-coding a recency bias into the attention scores, effectively encouraging the model to behave a bit more like a convolution. But a 1D CNN has that bias "for free" and in a more rigid form: it cannot waste capacity on arbitrary global interactions even if it tries. This helps explain why the SimpleCNN baseline reaches such strong performance with far fewer moving parts.

Features and Feature Engineering.

I include this section here, rather than earlier as it occurs in the pipeline, because it undergirds and hangs over all the results presented. In machine learning, data is King. I began the representation design by confronting an unavoidable constraint: all market information available to me at the bar level ultimately derives from only two raw time-indexed quantities, price and trade volume. Every downstream feature had to be constructed from those primitives, and the challenge was to convert those two (linked) streams into a high-dimensional but economically meaningful description of market state. I ultimately built a catalog of roughly one hundred and fifty candidate features by applying a disciplined set of algebraic transformations to returns, volatility estimates, microstructure proxies, ranges, volume concentrations, and time-of-day signals. [25] The philosophy was simple: do not inflate dimensionality for its own sake. Instead, build only those ratios, interactions, and nonlinear transforms that embody specific hypotheses about market behavior, guided by hypotheses and robust practice in quant microstructure: volume/dollar/imbalance bars [9][10]; order-flow imbalance and price-impact modeling

[13][15][16][20]; microprice/LOB state from modern HFT texts [17][22]. I favored ratios/normalizations (e.g., return/vol), interactions with economic meaning, and robust transforms (asinh/log1p) for heavy tails. Features passed stability (KS), predictive content (MI), redundancy (corr/VIF), and clean scaling checks before inclusion. Ratios were useful when the scale of a quantity mattered only relative to some notion of normality.

Examples included volatility-normalized returns like ret_{10} divided by vol_{20}, range normalized by the local ATR, or volume expressed as a fraction of its recent mean. These quantities encode magnitude relative to expectation, which is what traders implicitly judge when they categorize a move as big or small. Interactions were appropriate only when two distinct economic mechanisms interacted. Trend strength interacted with volatility because trends are more informative when volatility is non-trivial. Order-flow imbalance interacted with recent returns because buying pressure is meaningful only when it occurs in the direction of the move. Time-of-day flags interacted with volatility because the same volatility level means something different at the open than at midday. Simple nonlinear transforms helped tame highly skewed or heavy-tailed distributions. Asinh and log1p transforms stabilized volumes, dollar volumes, and ranges. [26] It is well known that features need to be normalized and scaled (an in this case clipped to the center 98% of the distribution) for use in neural networks or an unscaled feature will below up the activation. Absolute returns and squared returns provided magnitude-only signals without committing the model to a sign. Composite scores provided a constrained way to summarize richer microstructure patterns. Trend scores combined returns over multiple horizons normalized by volatility. Liquidity scores blended spread, aggregate volume, and concentration metrics. These low-dimensional composites served as safe parents for interaction terms that would otherwise be too fragile or too heavy-tailed. As you can see, a lot of time was spent on this.

To avoid combinatorial explosion, I treated this process as a hypothesis-driven search rather than a blind cross-product of features. Each engineered feature had to answer a specific question of the form: “If X is large relative to Y, do I expect an increased likelihood of an upward or downward move over the next ten bars?” The point was not to generate hundreds of algebraic terms but to encode only the interactions that had plausible microstructure interpretations. A simple three-step workflow kept this under control. First, I defined candidate families by selecting base features from returns, volatility measures, volume measures, order-flow variables, range statistics, and state indicators. From these I generated normalized ratios and cross-family interactions in a bounded manner, keeping the total engineered set below seventy candidates. Second, I tested their marginal predictive value on a reasonably sized slice of the data by computing mutual information, simple correlations, and univariate classification accuracy against the H=10 directional label. [27] Candidates that failed to outperform their parent features were discarded. Third,

I ran random forest trees on the parent features plus engineered candidates to rank their incremental value. Only engineered features that showed importance were retained. Before allowing any engineered feature into the full training pipeline, I performed leakage checks, stability checks across contracts, and redundancy checks using VIF and correlation structure. This conservative procedure kept the engineered feature set interpretable, robust, and directly traceable to explicit market structure hypotheses rather than accidental correlations. The selection pipeline emphasized temporal stability, marginal predictive value, managed redundancy, and robust scaling. Mutual information (MI) was used cautiously (e.g., Kraskov-style MI estimators are standard), with sanity checks via simple models; only then did features enter the sequence models. The upshot mirrors practice in "Advances in Financial Machine Learning": disciplined bar construction/feature hygiene often sets the ceiling; architecture determines proximity to that ceiling. [9]

The transition from construction to selection was natural because the engineered features became inputs to a broader feature-engineering and selection process that proved to be as important as the architecture itself. Indeed, the eventual performance of the two-layer one-dimensional CNN baseline demonstrated that the feature set was doing most of the heavy lifting. The CNN achieved mid-fifties accuracy and $F_{0.5}$ on the bar_labelH10 task with an architecture far simpler than any transformer I trained. That observation was not an indictment of transformers but rather a confirmation that the quality and stability of the representation matter more than the sophistication of the model in the short-horizon forecasting regime. Despite this, this paper proves that architecture matters.

Within the final feature-engineering pipeline I emphasized four properties that I came to view as essential: temporal stability, predictive relevance, manageable redundancy, and clean scaling. Temporal stability came first because volume-bar features are vulnerable to drift whenever market regimes shift. I explicitly evaluated each candidate feature using distributional tests between different periods of each contract and across contracts. Kolmogorov–Smirnov statistics and shifts in means and variances identified features prone to regime-specific blowups. Predictive content mattered next, but only after stability was verified. Mutual information with the shifted target and univariate decision-tree accuracy on a held-out slice gave an empirical view of which features contained even weak directional signal. Correlations above two hundredths were judged meaningful and values around five hundredths were exceptional. Redundancy control prevented wasted capacity. I pruned only the most highly correlated pairs, selecting the member with greater predictive relevance and lower kurtosis. The final pillar was clean scaling and clipping. Early in the project I discovered the hard way that a single unscaled, heavy-tailed feature can dominate gradient updates and cause the transformer to thrash between class modes. Clean scaling

and controlled clipping were indispensable. Heavy-tailed or poorly clipped features destabilized optimization and produced degenerate attention patterns. All retained features had to pass through asinh or log1p transforms where appropriate, undergo mild percentile clipping, and be standardized in a way that respected contract boundaries.

This four-pillar pipeline produced a feature set in the twenty-to-forty variable range depending on the target definition. Even seemingly small mutual-information features such as intrabar returns, volume concentration, microprice, short-horizon volatility estimators, long-horizon trend proxies, arrival clustering indices, and time-of-day sinusoidals proved valuable when interpreted by sequence models. The signal lived in their joint temporal evolution rather than in any one of them individually. Sanity checks using same-bar correlation and baselines helped ensure that the final feature set contained no leakage and that its informational content was fundamentally temporal rather than static. The fact that both the transformer and the simple CNN could extract significant predictive power from this feature set underscored the central lesson of this work: in financial forecasting, it is almost always the representation that determines the ceiling of performance, and architecture only determines how close we get to that ceiling.

Feature Selection.

In this project I treated feature engineering as the primary lever of performance rather than architecture choice, and the results ultimately supported that decision. The same carefully constructed feature set fed to a simple two-layer one-dimensional CNN achieved mean F0.5 around 0.53–0.54 and final-contract F0.5 near 0.55, essentially matching and in some regimes exceeding more elaborate transformer variants. That outcome still reconciles with a narrative in which “the model” is also a hero, due to FutureFormers outperformance.

For a transformer trained on ordered bar features to predict t+1 or H=10 direction, what matters is not static feature relevance but temporal structure. The model does not care whether a feature is globally correlated with the target across the entire sample if that correlation comes entirely from regime shifts or structural breaks. What it needs is a set of inputs that encode the evolving state of the limit order book and recent price action in a way that is stable across time, predictive at the margins, not pathologically redundant, and well scaled so that gradients behave sensibly. Classical feature selection recipes built for random forests or logistic regression are not enough. A transformer can tolerate moderate redundancy and can build nonlinear interactions by itself, but it is extremely sensitive to distributional instability and to a handful of spiky features that dominate the optimization.

Microstructure-derived ratios were repeat offenders here: features built as a quotient of two low-volume quantities would look reasonable in one contract and then blow up when

liquidity conditions changed. Those features were either transformed with stronger clipping or dropped outright.

Redundancy management was less important. Transformers are quite forgiving when it comes to correlated inputs; they can simply learn to downweight redundant channels. However, excessive redundancy does waste both memory and optimization capacity, and it can make diagnostics harder to interpret. Rather than chasing orthogonality, I focused on pruning only the most egregious cases. Pairs of features whose absolute correlation exceeded about 0.85 were treated as essentially duplicative. In those cases I kept the member of the pair that had higher mutual information, more stable distributional behavior across contracts, and lower kurtosis, on the grounds that less heavy-tailed features produce more reliable gradients. Variance inflation factors provided a backstop: any feature with VIF above 10 in a simple linear model was considered a candidate for removal if a more stable proxy existed. [28] I treated VIF as an *inference-oriented* diagnostic rather than an automatic filter: in prediction-focused, regularized, nonlinear models (e.g., transformers with weight decay/dropout and attention), strong collinearity seldom harms out-of-sample discrimination or calibration, whereas it mainly destabilizes coefficient estimates and variable selection in linear models [28][29][30]. Attention-based tabular transformers also learn to reweight redundant inputs and model feature dependencies directly, which further reduces the need for VIF-driven pruning [31][32][33]. (See also the simulation evidence that increasing collinearity leaves predictive performance essentially unchanged.) [29] So VIF was not as much of a deciding factor once I learned this.

Within this framework I distinguished between “instantaneous” predictive features and what I ended up calling structural features. Instantaneous features have directly measurable pointwise association with the future label: intrabar returns, short-horizon realized volatility, simple measures of trade imbalance, or recent momentum indicators. Structural features, by contrast, describe the state of the process in ways that may have low mutual information individually but are rich when viewed as sequences. Examples include intrabar return and its log-transform, volume concentration indices such as HHI over the bar, microprice-based measures of where trades are occurring relative to the order book, longer-horizon volatility estimates like volatility_20, three-bar range summarizing local amplitude, long-term trend proxies, arrival-clustering indices for trades, trade intensity, a crude price impact proxy, relative tick activity, Bollinger band positions, and time-of-day sinusoidal encodings. When I looked at MI scores for these structural features one by one, many of them barely cleared the 0.01 threshold. But when I examined how a transformer or a CNN responded to them over windows of hundreds of bars, it became clear that their joint temporal configuration was where the signal lived.

The actual selection pipeline put these ideas together. I started from a fully scaled dataset, restricted to regular trading hours to avoid mixing fundamentally different regimes. For each candidate target, including the bar_labelH10 directional label used in the main experiments, I ran the stability, mutual information, redundancy, and clipping diagnostics. The intersection of four sets formed the backbone of the final feature set: features that were temporally stable by the KS and mean-shift tests, features with mutual information above roughly 0.01 and univariate tree accuracy above about 0.51, features not belonging to correlation clusters above 0.85, and features that did not require more than minimal clipping and had no pathological sign flips or infinite values. To that backbone I deliberately added a small set of canonical structural features even if their individual MI was low, because they encode state variables that sequence models are particularly good at interpreting. The end result was a feature set in the 20–40 variable range depending on the target and bar specification, which I found to be a sweet spot for transformers and CNNs at the context lengths I was using. In the end, only 14 features survived selection and made it to the various models. All models were trained and evaluated using the same 14 features.

Finally, it is worth emphasizing what I tried not to do. I avoided selecting features purely by mutual information, because that ignores temporal drift and tends to favor unstable microstructure ratios. I avoided mixing RTH and ETH distributions for I was not explicitly modeling regime shifts, because that destroys calibration. I actively removed unstable, unscaled, heavy-tailed features that made attention weights or convolutional filters latch onto a few pathological spikes. And I treated any feature that looked suspiciously target-correlated, especially if it was computed using within-bar information near the label horizon, as guilty until proven innocent. In my experience these negative choices are part of what allowed the positive machinery of transformers and CNNs to work.

Ablation studies.

Now would be a good time to profile each of these innovations in transformer architecture that I chose to use and why. AliBi (Attention with Linear Biases) replaces GPT-2's learned positional embeddings by injecting distance-dependent biases directly into attention scores, computing Attention(Q,K,V) = softmax(QK^T/√d_k + m·[−(i−j)])V where m represents head-specific slopes typically set as m = 2^(−8i/H) for head i out of H total heads, and the matrix [−(i−j)] contains penalties proportional to the distance between volume bar positions i and j in the sequence. This mechanism maintains $O(n^2d)$ attention complexity while eliminating the O(nd) cost and d×n_pos parameters of learned embeddings.

Model variant	*Acc*	*P_macro*	*R_macro*	*F1_macro*	*F0.5*	*Per-class F1 (Down / Unchanged / Up)*	*Per-class P (Down / Unchanged / Up)*	*Per-class R (Down / Unchanged / Up)*
FutureFormer-S (full)	*0.690*	*0.693*	*0.690*	*0.689*	*0.690*	*0.650 / 0.751 / 0.665*	*0.640 / 0.716 / 0.711*	*0.666 / 0.796 / 0.609*
No ALiBi	*0.637*	*0.636*	*0.637*	*0.635*	*0.635*	*0.580 / 0.702 / 0.621*	*0.590 / 0.650 / 0.644*	*0.562 / 0.757 / 0.592*
No GQA	*0.659*	*0.672*	*0.659*	*0.663*	*0.668*	*0.592 / 0.766 / 0.631*	*0.572 / 0.872 / 0.602*	*0.639 / 0.713 / 0.624*
No RMSNorm (LayerNorm)	*0.602*	*0.611*	*0.602*	*0.602*	*0.606*	*0.539 / 0.636 / 0.629*	*0.528 / 0.563 / 0.673*	*0.541 / 0.701 / 0.562*
No SwiGLU (ReLU FFN)	*0.570*	*0.602*	*0.570*	*0.575*	*0.588*	*0.542 / 0.553 / 0.629*	*0.485 / 0.493 / 0.706*	*0.612 / 0.574 / 0.524*

Table 7: results of the ablation studies

My ablation shows ALiBi removal degraded F1 from 0.689 to 0.635, a substantial 5.4 percentage point drop, with particularly severe impact on the Down class (F1 dropped from 0.650 to 0.580) and Up class (from 0.665 to 0.621), while Unchanged class remained relatively stable (0.751 to 0.702). This pattern reveals that ALiBi's linear distance encoding proves essential for volume bar sequences because the informational decay follows the activity-normalized distance rather than clock time, meaning that volume bar N-50 and volume bar N-10 have consistent relative informational relationships regardless of the underlying temporal duration they span. [37] Volume bars by construction compress time during high-activity periods and expand time during low-activity periods, creating sequences where learned positional embeddings fail catastrophically because they encode ordinal position rather than the activity-normalized distance that actually governs autocorrelation structure in order flow. [38] The asymmetric degradation across classes (Down and Up suffer more than Unchanged) occurs because directional movements

exhibit stronger momentum and mean-reversion patterns that depend on activity-normalized lookback windows, whereas the minority Unchanged states represent continuation regimes where recent history dominates regardless of position encoding. [39] ALiBi's head-specific slopes enable different attention heads to specialize in different activity-normalized timescales simultaneously, with some heads focusing on immediate microstructure (steep slopes, recent volume bars dominate) and others capturing regime-level patterns (shallow slopes, distant volume bars retain influence), a property that proves indispensable when predicting the next future token where both short-term order flow and longer-term regime information contribute to directional probabilities. [40] The mechanism aligns perfectly with the statistical properties of signed volume and order flow imbalance, which decay in predictive power as a function of cumulative volume rather than elapsed time, making AliBi's activity-indexed distance penalties a natural inductive bias for this domain. [41]

Grouped Query Attention (GQA) modifies GPT-2's Multi-Head Attention by having multiple query heads share key-value projections, computing$head_i = Attention\left(Q_i, K_g(i), V_g(i)\right)$ where the mapping function $g(i) = \lfloor i \cdot G/H \rfloor$ assigns query head i to one of G groups with G < H. While standard Multi-Head Attention uses H independent (K, V) pairs requiring $O(2Hd^2)$ parameters for projections, GQA reduces this to $O((H+G)d^2)$ parameters, maintaining $O(n^2d \cdot H)$ computational complexity during forward and backward passes but reducing memory footprint. The ablation demonstrates that removing GQA (reverting to full Multi-Head Attention) decreased F1 from 0.689 to 0.663, a 2.6 percentage point degradation, with the Unchanged class showing unexpected improvement (0.751 to 0.766) while Down and Up classes suffered (0.650 to 0.592 and 0.665 to 0.631 respectively), revealing that GQA introduces a beneficial regularization effect particularly valuable for volume bar sequences where extreme class imbalance compounds with non-stationarity. [42] The parameter reduction through shared key-value representations forces the model to learn more generalizable features of volume bar patterns that transfer across different query perspectives, which proves essential when minority classes (Down and Up movements) appear with frequencies as low as 20-30% combined in typical market conditions dominated by bid-ask bounce and small continuation movements. [43] Standard Multi-Head Attention's additional capacity in the No GQA variant enabled overfitting to the Unchanged class's abundant training examples, as evidenced by Unchanged precision jumping to 0.872 while Down and Up recall deteriorated, suggesting the model memorized continuation patterns at the expense of learning the more subtle order flow imbalances that precede directional movements. [44] Volume bar sequences present a particularly challenging learning problem because the information content per bar varies dramatically with market regime; during high volatility each bar contains dense predictive signals while

during quiet periods bars are mostly noise, and GQA's forced parameter sharing creates implicit multi-task learning where the model must find features that discriminate directions across varying information densities rather than memorizing regime-specific heuristics. [45] The architectural constraint acts as a form of capacity control that prevents the model from developing separate sub-networks for different market conditions, instead encouraging the discovery of robust volume-based features such as order flow toxicity and trade aggressor imbalance that persist across regimes and provide genuine predictive power for the next future token. [46] For volume bar prediction specifically, GQA's regularization proves more valuable than in time-based sequences because the activity normalization creates training distributions where consecutive samples can exhibit wildly different statistical properties (a volume bar during a flash crash versus a volume bar during overnight trading), making traditional capacity-based regularization insufficient and structured parameter tying through GQA actually necessary. [47]

Flash Attention implements an algorithmically equivalent but computationally optimized version of standard scaled dot-product attention through tiled computation and kernel fusion, never materializing the full $O(n^2)$ attention matrix in high-bandwidth memory. The algorithm partitions Q, K, V matrices into blocks and computes attention incrementally using online softmax with running statistics, reducing memory complexity from $O(n^2)$ to $O(n)$ for attention scores while maintaining identical outputs to naive attention implementation. The I/O complexity improves from $O(n^2d + nd^2)$ to $O(n^2d^2/M)$ where M represents SRAM size, providing the theoretical basis for observed 2-4× training speedups at moderate to long sequence lengths. While Flash Attention does not appear in the ablation table, its impact on volume bar-based directional prediction extends beyond raw computational efficiency to enable the larger batch sizes essential for stable gradient estimates when training on highly non-stationary order flow data. I kept batches small at 64 but left that open for the future. [48] The memory savings translate directly to doubling or tripling batch size within single-GPU constraints, which proves critical for volume bar sequences because the three-class distribution shifts dramatically across different market regimes (trending versus mean-reverting, high versus low volatility), requiring each gradient update to aggregate information across diverse volume bar contexts to avoid catastrophic overfitting to specific regimes. [49] Volume bars create sequences where the effective information content varies by orders of magnitude depending on market conditions, making batch diversity more important than batch size in traditional i.i.d. settings, and Flash Attention's memory efficiency enables the large batches necessary to capture sufficient regime diversity within each parameter update. [50] For the specific task of predicting the next future token's direction, Flash Attention enables experimentation with the full 512-volume-bar context window that captures both intraday microstructure

patterns and longer-term regime information, whereas memory-constrained standard attention would force truncation to perhaps 256 or 128 bars, losing the multi-scale temporal structure that volume bars are specifically designed to preserve. [51] The optimization proves particularly valuable because volume bar sequences exhibit long-range dependencies where a significant order flow imbalance 400-500 bars ago (representing perhaps 30-60 minutes of trading depending on activity level) can still influence the directional probability of the next bar through persistent liquidity effects and informed trader positioning, dependencies that even shorter context windows would completely miss. [52]

SwiGLU (Swish-Gated Linear Unit) replaces GPT-2's GELU activation function in feed-forward networks, computing $FFN(x) = \big(Swish(xW) \odot (xV)\big)W_2$ where Swish(x) = x·σ(x) with σ denoting the sigmoid function and $\odot$ representing element-wise multiplication. This requires two projection matrices W and V instead of one, typically expanding the intermediate dimension from $4d_{\text{model}}$ to approximately $(8/3)d_{\text{model}}$ to maintain comparable parameter counts, resulting in $O(5d^2/3)$ complexity per FFN block compared to $O(d^2)$ for standard activations. The ablation reveals that removing SwiGLU (replacing it with ReLU) caused the most catastrophic performance degradation of all components, dropping F1 from 0.689 to 0.575, an enormous 11.4 percentage point loss, with devastating impact across all classes but particularly severe for Unchanged (0.751 to 0.553 F1) and relatively more preserved for Up (0.665 to 0.629 F1), indicating that SwiGLU's gating mechanism proves absolutely essential for volume bar-based directional prediction because the feature interactions that discriminate between continuation and reversal patterns are fundamentally multiplicative rather than additive. [53]

Volume bars encode information through relationships between price movement, volume intensity, and order flow imbalance within each bar, creating a feature space where predictive signals emerge from interactions between these components - for example, a small price movement combined with high volume indicates absorption by limit orders and predicts continuation, while the same price movement with low volume indicates momentum exhaustion and predicts reversal. [54] ReLU's simple thresholding cannot capture these multiplicative feature interactions because it applies element-wise nonlinearity without considering feature relationships, whereas SwiGLU's gating term (xV) learns to amplify or suppress the activated features $Swish(xW)$ based on the input context, effectively implementing learned feature interaction detection. [55] The catastrophic Unchanged class degradation (0.751 to 0.553 F1) reveals that discriminating continuation patterns from weak directional signals requires precisely the type of context-dependent feature modulation that gating provides, as continuation (Unchanged) predictions must integrate evidence that price movement is proportional to volume flow

and order imbalance levels fall within normal ranges, a conjunction of conditions that multiplicative gating naturally expresses. [56] The relative preservation of Up class performance (0.665 to 0.629 F1) compared to other classes suggests that strong upward movements exhibit more pronounced univariate signals (such as extreme positive order flow imbalance) that even ReLU can partially capture, whereas Down and Unchanged movements require the subtle feature interactions that only gated activation functions can model. [57] For volume bar sequences specifically, SwiGLU's smooth, differentiable gating enables the model to learn continuous sensitivity to the intensity of volume bar features, allowing it to distinguish between strong signals during high-activity bars and weak signals during low-activity bars by modulating feature importance based on learned volume-intensity indicators embedded in the gate projection V. [58] The empirical result demonstrates that for volume bar-based directional prediction, SwiGLU's 50% computational overhead versus simpler activations represents the single most cost-effective architectural investment, delivering an 11.4 percentage point F1 improvement that dwarfs all other component contributions and would be impossible to recover through additional model capacity or training time with inferior activation functions. [59]

RMS Norm (Root Mean Square Normalization) replaces GPT-2's Layer Normalization by eliminating mean-centering and computing $RMSNorm(x) = \gamma \odot x/RMS(x)$ where $RMS(x) = \sqrt{(1/d \cdot \Sigma_i x^2{}_i + \varepsilon)}$, reducing normalization to pure scale standardization without shift invariance. This simplification cuts computational cost from O(2d) to O(d) per normalization operation by eliminating the mean computation pass and removes the learned bias parameters β entirely. That ablation shows that reverting to LayerNorm caused an 8.7 percentage point F1 degradation (0.689 to 0.602), the second-largest impact after SwiGLU, with severe damage to Down class (0.650 to 0.539 F1) and substantial degradation in Unchanged (0.751 to 0.636 F1), revealing that LayerNorm's mean-centering operation actively destroys critical distributional information in volume bar features that RMSNorm preserves. [60] The surprising magnitude of this effect stems from volume bars encoding information not just in relative feature magnitudes but in absolute feature levels and means, particularly for order flow imbalance and volume intensity variables where the mean across features at a given volume bar position indicates overall market regime (directional pressure versus balanced two-sided flow). [61] LayerNorm's removal of first-moment information through centering eliminates the model's ability to distinguish between volume bars occurring in different market microstructure regimes, forcing downstream layers to make directional predictions based solely on normalized deviations without access to the absolute imbalance levels that actually determine whether weak signals represent noise or genuine predictive information. [62] RMSNorm preserves the mean structure of volume bar features while only normalizing their overall magnitude,

allowing the model to learn regime-conditional transformations where the same relative feature pattern (say, price up with volume above average) receives different directional interpretations depending on the absolute level of order flow imbalance across all features. [63] The dramatic Down class deterioration (0.650 to 0.539 F1) with LayerNorm occurs because downward movements in liquid markets exhibit characteristic patterns where selling pressure builds through sustained negative order flow imbalance across multiple volume bars, creating negative mean shifts in imbalance features that LayerNorm's centering removes, leaving the model unable to detect the accumulated directional pressure that precedes Down movements. [64] Volume bar sequences differ fundamentally from both time-series and language modeling in that the mean activation level at each layer encodes information about the current state of the market microstructure (specifically, the balance between aggressive buyers and aggressive sellers), information that has direct predictive power for the next future token's direction and must be preserved through the network depth. [65]

The empirical finding that RMSNorm outperforms LayerNorm by 8.7 percentage points in F1 contradicts the conventional wisdom from natural language processing and reveals a critical insight for financial deep learning: normalization schemes designed to ensure training stability by removing distributional shifts can inadvertently eliminate the very distributional information that carries predictive signal in financial applications, making RMSNorm's lighter-touch approach of scale normalization without mean removal essential for maintaining model quality. [66] This result suggests that practitioners adapting transformers to volume bar-based prediction should carefully audit all architectural components inherited from language modeling to ensure they do not implicitly assume that absolute feature levels and means are uninformative, an assumption valid for token embeddings in text but catastrophically wrong for order flow features in market microstructure. [67]

Viewed against the full FutureFormer S configuration, each ablation makes the model worse, but in slightly different ways. Removing ALiBi produces the most intuitive degradation for a time series task: overall accuracy drops from about 0.69 to about 0.64, and F0.5 falls by roughly five and a half points. The loss is fairly symmetric across classes, with both down and up F1 slipping and unchanged F1 shrinking modestly. This is consistent with the idea that ALiBi’s recency bias is doing real work in a short-horizon financial setting. Without a structural bias that privileges nearby bars, the model has to learn recency purely from data, which is both harder and more sample-inefficient. The result is a transformer that sees too much of the distant past and not enough of the immediate microstructure, so it becomes less decisive at the horizon you care about.

The GQA ablation is softer. Accuracy and F1 dip by only a few points relative to the full model, and F0.5 remains relatively high. Interestingly, unchanged F1 actually ticks up slightly while down and up F1 give back some of their edge. This pattern matches the usual story that grouped-query attention is mostly an efficiency trick. When GQA is removed, the model has more independent key–value heads and therefore more ways to overfit local quirks in the data, which may slightly help the easy unchanged region but hurts stable directional precision. In practice this makes the no-GQA configuration closer to a conventional GPT-2: it works, but it gives up a bit of the calibrated directional sharpness that matters most for trading, while also being less memory-efficient.

In contrast, removing RMSNorm and SwiGLU has a clearly destabilizing effect on class structure. The RMSNorm ablation drops accuracy to around 0.60 and pulls macro F1 and F0.5 down by roughly eight to nine points relative to the full model. The unchanged class is hit especially hard, with F1 falling from the mid-seventies into the mid-sixties. This is what one would expect if optimization becomes noisier: the model still finds some signal in down and up, but its separation of sideways regimes becomes less reliable. Replacing SwiGLU with a simpler feed-forward nonlinearity is even more damaging. Accuracy falls to about 0.57 and F0.5 to under 0.59, and the unchanged F1 collapses into the mid-fifties even as up precision becomes extreme and brittle. That combination of very high up precision, weaker recall, and low unchanged F1 suggests a model that has lost smooth capacity in its hidden layers. It still finds certain “clean” upside patterns and calls them very accurately, but it struggles to represent the messy boundary between flat and directional regimes, so many ambiguous bars are misclassified.

Taken together, these ablations show that the full FutureFormer S stack is not just a cosmetic port of NLP ideas into finance. ALiBi and GQA primarily tune how attention is allocated across the window, while RMSNorm and SwiGLU primarily stabilize and enrich the local representation learned inside each block. When any one of them is removed, the model’s macro metrics degrade and, more importantly, the per-class structure shifts in ways that are easy to link back to their inductive roles. The complete configuration sits at a point where the attention pattern has a recency bias matched to high-frequency markets, the normalization keeps gradients in a healthy range, and the feed-forward nonlinearity has enough smooth capacity to separate unchanged from directional states. In that sense, the ablations quantify what my intuition already suggested: in this small-data, short-horizon regime, these architectural choices are acting as a form of hand-crafted bias toward the actual temporal structure of order flow.

Future work: context-length stress tests.

The present work trained and evaluated all three models at a context length of 512 bars, partly for computational reasons and partly because theory and preliminary diagnostics suggested that most predictive information resides in relatively short windows. One of the key theoretical advantages of ALiBi, however, is improved length extrapolation: models trained with shorter contexts should be able to operate on longer sequences without retraining, as the positional bias scales linearly with distance. To test this property in the financial setting, I plan a context-stress experiment in which a model trained at 512 is evaluated at 512, 1024, and 2048 bars, using padded or extended histories from the same contracts, and performance is measured as a function of context length (but I ran out of time). The expectation is that loss and F1 will remain stable or degrade gracefully as context extends beyond the training length, in contrast to models with learned positional embeddings that often exhibit sharp deterioration. Those context-length stress tests are less important and were not included given my view that intraday trading depends mostly on very recent history, and then *points in time* in the past; I would like to try out a windowed version of attention to test this too in future work.

Engineering memoir:" the "50 days."

If there was only one lesson learned that I had to pick out as the most crucial, it is this: if you are confident in your theory, do not give it up until it has been categorically disproven. Of the 50 days of engineering, 49 of them was spent correcting implementation errors. There were a lot of moving parts: first, a whole machine learning pipeline, getting that first pass of the lifecycle to be successful took the bulk of the time, with challenges in every component, and then the analysis, ablations, and reflections could be gathered.

Another source of error was target leakage. As multiple target variants were introduced (original barrier-based labels, 10-bar horizon labels, 10-bar continuous returns, etc.), I inadvertently indexed my target as bar *t* instead of bar *t+1,* leading me to get excited from inflated results. It was only when I presented those results to my partners that I was brought down to earth, and sent back to check the pipeline for future leakage. 80-90% accuracy in market prediction is unheard of, and I knew that, but was too eager to think twice about it. This was a silent error, the most heinous to any engineering effort.

A substantial fraction of the 50-day effort went into diagnosing failures that had nothing to do with the high-level architecture and everything to do with data and training plumbing. Early versions of the pipeline suffered from multiple subtle but damaging issues. Some engineered features had extreme scales, with means on the order of 10^8 or larger and standard deviations in the hundreds of millions, completely dominating gradients and

leading the model to ignore all other inputs. These were detected via summary statistics and dropped according to a simple heuristic: any feature with |mean| > 1 or standard deviation outside the range $[10^{-2}, 2]$ was removed from the final feature set. For distributional stability I applied Kolmogorov–Smirnov tests and mean/variance-shift checks across sub-periods [12]; to limit redundancy I pruned only highly collinear pairs using correlation and variance inflation heuristics. (KS: [12]; VIF heuristics are standard econometrics practice.)

I imposed a discipline: all features had to be finite, had to avoid pathological kurtosis, and had to pass through a transformation pipeline that included asinh or log1p for positive heavy-tailed quantities, percentile clipping at the extremes, and z-scoring within regime or contract. Features that routinely produced infinities, NaNs, or had more than about one percent of their values clipped at the tails were considered structurally unsound and were either redesigned or removed. This sounds like housekeeping, but in practice it was one of the most important determinants of whether training converged to something useful.

Finally, the checkpointing logic was hardened. Loading checkpoints from earlier experiments with mismatched model dimensions or context lengths also caused silent misbehavior. A strict loader was added that compares the saved configuration (model dimension, number of layers, maximum sequence length) to the current instantiation and refuses to load if any mismatch is detected. This prevented “Frankenstein” models from being fine-tuned on new data and made runs reproducible. The continued pre-training approach was solidified when I remembered to carry over optimizer state, I had been restarting it.

From an engineering standpoint, arriving at a 51%+ model was less about clever architecture (although ablations quantify the contribution) and more about removing landmines. The first wave of failures came from regression formulations where the transformer simply learned persistence. The second wave arose when I introduced classification but left features unscaled: gradient norms exploded, and the model settled into degenerate class modes, predicting only the majority class or oscillating between classes from epoch to epoch. Fixing this required not just dropping pathological features but also revisiting class weighting and loss functions. Using “softened” class weights with an exponent of 0.5 proved too weak; the model effectively ignored the minority class. Reverting to full inverse-frequency weights combined with focal loss finally forced it to pay attention to underrepresented regimes.

The third wave of failures was temporal. Treating all six contracts as a single shuffled dataset would subtly violated causality and creat hidden forms of leakage. The cure was to adopt a strict sequential training procedure with per-contract loaders, no cross-contract

shuffling, and carefully aligned context windows that never cross contract boundaries (but did span trading days, with a gap for after-hours, an important decision). A particularly pernicious bug involved the learning rate schedule: using cosine decay independently per contract meant each new contract started with a fresh high learning rate, overwriting previous learning and causing catastrophic forgetting. The progressive decay scheme discussed earlier was the turning point for stability.

Finally, the diagnostic framework itself had to be debugged. Early gradient-norm logging mistakenly used the squared norm, inflating reported values by an order of magnitude and misleading my intuition about clipping aggressiveness. Fixing that compute path and logging true L2 norms revealed that gradient magnitudes in the healthy regime were in the range of 4–8, consistent with effective but not extreme clipping at 2.0. Alongside larger validation sets (about 6000 samples with roughly 2000 per class) and a validation stride of 1, this made it possible to distinguish real signal from sampling noise. Only after this chain of corrections did the model reliably climb into the 51%+ accuracy band on the final contract. In fact, dozens of early runs were invalidated by the realization that I was reloading weights with each dataset (contract), but reinitializing the optimizer. Once I corrected that, results changed.

Statistical validation and future work.

In a full trading study one would go beyond pointwise classification metrics and evaluate the signal in a walk-forward optimization (rolling validation), using expanding-window or rolling-window validation to mimic live deployment. Standard “anti-luck” tools such as the deflated Sharpe ratio would be appropriate to quantify whether an observed backtest Sharpe could plausibly arise from the long sequence of modeling attempts undertaken here. Similarly, Diebold–Mariano tests could be used to compare forecast errors between the modified transformer and the vanilla GPT-2 baseline or simpler benchmarks such as logistic regression or convolutional models, to establish whether the incremental predictive edge is statistically significant. [23] In principle, the same apparatus could be applied not just to directional accuracy but to profit-and-loss series generated by a simple trading rule conditioned on the model’s outputs. Applying DSR to PnL series driven by the classifier and DM to forecast losses (or F-scores) would quantify whether the observed edge is statistically significant. These analyses are beyond the current scope but represent a natural next step now that a stable 69% predictor exists. The Deflated Sharpe Ratio (DSR) corrects for selection bias and non-normal returns in backtests [8], while Diebold–Mariano (DM) tests compare forecast accuracy between models [18][19].

Discussion and conclusion.

Conceptually, this project shows that the "vocabulary gap" between transformers and financial time series can be bridged by reframing prediction as a discrete classification problem over carefully engineered volume-bar features. [9][10] In doing so, it becomes possible to reuse a large fraction of the transformer toolkit developed for NLP: positional biases, attention variants, normalization schemes, and efficient kernels, while adapting the training regime and evaluation metrics to the realities of markets. The result is not a spectacular superhuman forecaster, but a still a *very* strong, statistically defensible edge: a 69.03% accurate classifier on a three-class 10-bar horizon, with up and down precision roughly ten percentage points above unconditional base rates, is definitely monetizable. For a professional trading group, this would be one component of a larger system, to be stress-tested, hedged, and combined with risk management and execution logic. For a student and retail quant, it is proof that a single individual with commodity hardware and open-source tools can build a nontrivial signal engine comparable in spirit, if not in scale, to what small teams within hedge funds attempt.

At the same time, the work underscores several cautions. First, architecture alone is not a silver bullet. The same ALiBi, GQA, RMSNorm, and SwiGLU stack that underpins multi-billion-parameter language models does not automatically yield performance in small, noisy financial datasets. Without correct feature scaling, careful handling of class imbalance, and a well-designed continual training regime, these innovations merely make the model more efficient at learning the wrong thing. Second, scaling must be matched to data. Under the token budgets used here, increasing parameters from 45M to 130M decreased accuracy from 69.03% to 51.08%, a tangible reminder that in the small-data regime the optimal model is often smaller than intuition suggests. Third, evaluation must respect time. Shuffling across contracts or leaking future information into features will manufacture spurious edges; only strict forward-chained splits and contract-aware context windows can provide a realistic estimate of out-of-sample performance.

There are clear next steps. On the modeling side, the planned ablation study quantified which architectural changes genuinely contribute to predictive power and which primarily improve efficiency, helping to refine a minimal, finance-specific transformer design. Context-length stress tests will validate whether ALiBi's length-generalization properties hold under regime shifts and long memory in volatility. On the statistical side, walk-forward backtests with deflated Sharpe ratios and Diebold–Mariano tests will determine whether the observed directional accuracy translates into robust trading performance once transaction costs and slippage are taken into account. On the engineering side, integrating FutureFormer into an execution engine, measuring latency and throughput under live data

feeds, and implementing conservative risk management around its signals will be necessary before any real capital is put at risk.

The broader message is that complex deep learning architectures are accessible so long as they are paired with a deep understanding of the data (best with domain knowledge) and a willingness to iterate through unglamorous engineering failures. Fractional differencing for stationarity, richer microstructure features, and cross-sectional extensions to equities or options are all plausible extensions once the core pipeline is stable. The trainer I used for FutureFormer is included in the paper's archive in case readers are interested in my model design and training. There's a lot that I intend to do to extend this research. In that sense, the work here is less a finished product and more a proof of concept for my burgeoning partnership: the market does, in a measurable way, "*pay* attention," (in many ways, thus my pun) and a carefully adapted transformer can learn to pay attention right back to it.

References

[1] Vaswani, A. et al. (2017). *Attention Is All You Need*.

[2] Radford, A. et al. (2019). *Language Models are Unsupervised Multitask Learners* (GPT-2).

[3] Press, O., Smith, N., & Lewis, M. (2021). *Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (ALiBi)*. arXiv:2108.12409. arXiv+2arXiv+2

[4] Ainslie, J. et al. (2023). *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. arXiv:2305.13245; EMNLP'23. arXiv+2arXiv+2

[5] Zhang, B., & Sennrich, R. (2019). *Root Mean Square Layer Normalization*. NeurIPS 2019. arXiv:1910.07467. arXiv+1

[6] Shazeer, N. (2020). *GLU Variants Improve Transformer*. (Introduces SwiGLU).

[7] Dao, T. et al. (2022). *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv:2205.14135. (See also official implementation.) arXiv+2arXiv+2

[8] Bailey, D., & López de Prado, M. (2014). *The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality*. *Journal of Portfolio Management*, 40(5), 94–107. SSRN+1

[9] López de Prado, M. (2018). *Advances in Financial Machine Learning*. (Volume/dollar/imbalance bars; feature engineering discipline.) O'Reilly Media

[10] Easley, D., López de Prado, M., & O'Hara, M. (2012). *The Volume Clock: Insights into the High-Frequency Paradigm*. *Journal of Portfolio Management*. Zotero

[11] Lin, T.Y. et al. (2017). *Focal Loss for Dense Object Detection*. ICCV. (Adopted for class-imbalance emphasis.)

[12] Massey, F. (1951). *The Kolmogorov–Smirnov Test for Goodness of Fit*. *JASA*. (For distributional stability checks.) lem.sssup.it

[13] Cont, R., Kukanov, A., & Stoikov, S. (2014). *The Price Impact of Order Book Events*.

Journal of Financial Econometrics. (OFI as key driver at short horizons.) EconPapers+2SSRN+2

[14] Kyle, A. (1985). *Continuous Auctions and Insider Trading*. *Econometrica*. (λ as impact depth.) Duke People+2UT Dallas Personal Site+2

[15] Cont, R., & de Larrard, A. (2013). *Price Dynamics in a Markovian Limit Order Market*. *SIAM J. Financial Mathematics*. SIAM Ebooks+1

[16] Eisler, Z., Bouchaud, J.P., & Kockelkoren, J. (2012). *The Price Impact of Order-Book Events*. (Related microstructure evidence.) CFM

[17] Cartea, Á., Jaimungal, S., & Penalva, J. (2015). *Algorithmic and High-Frequency Trading*. Cambridge University Press. (Microprice/LOB state.) Cambridge Assets+1

[18] Diebold, F.X., & Mariano, R.S. (1995). *Comparing Predictive Accuracy*. *Journal of Business & Economic Statistics*. (DM test.) www-stat.wharton.upenn.edu+1

[19] Harvey, D., Leybourne, S., & Newbold, P. (1997). *Testing the Equality of Prediction MSEs*. *International Journal of Forecasting*. (Small-sample DM correction.) JSTOR

[20] Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., Sanghai, S. (2023). *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. EMNLP 2023. (arXiv:2305.13245). arXiv

[21] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., & Liu, T. Y. (2020). On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning* (pp. 10524-10533). PMLR.

[22] Hoffmann, J. et al. (2022). *Training Compute-Optimal Large Language Models (Chinchilla)*. DeepMind. (arXiv:2203.15556). Semantic Scholar

[23] Diebold, F. X. & Mariano, R. S. (1995). *Comparing Predictive Accuracy. Journal of Business & Economic Statistics*, 13(3), 253–263. Wharton Statistical Services

[24] Andersen, T. G., & Bollerslev, T. (1997). *Intraday periodicity and volatility persistence in financial markets. Journal of Empirical Finance.* (Evidence for strong diurnal patterns motivating time-of-day features.) Duke Economics

[25] Bollerslev, T., Cai, J., & Song, F. M. (2000). *Intraday periodicity, long memory volatility, and… Journal of Empirical Finance.* (Further documentation of intraday/diurnal volatility structure.) ScienceDirect

[26] Burbidge, J. B., Magee, L., & Robb, A. L. (1988). *Alternative Transformations to Handle Extreme Values of the Dependent Variable. Journal of the American Statistical Association.* (Introduces the inverse-hyperbolic-sine transform as a robust alternative to log.) JSTOR+1

[27] Kraskov, A., Stögbauer, H., & Grassberger, P. (2004). *Estimating mutual information. Physical Review E*, 69(6). (k-NN MI estimator widely used for feature screening.) APS Link

[28] O'Brien, R. M. (2007). *A Caution Regarding Rules of Thumb for Variance Inflation Factors. Quality & Quantity*, 41, 673–690. (Guidance on interpreting VIF beyond simple cutoffs.) SpringerLink

[29] Bai, S., Kolter, J. Z., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.* arXiv:1803.01271. (Temporal CNNs as a strong, simple baseline for sequences.) arXiv+1

[30] van den Oord, A., et al. (2016). *WaveNet: A Generative Model for Raw Audio.* arXiv:1609.03499. (Dilated causal convolutions excel at local, translation-invariant pattern modeling.)

[31] O'Brien, R. M. "A Caution Regarding Rules of Thumb for Variance Inflation Factors." *Quality & Quantity* 41(5), 673–690 (2007). SpringerLink+1

[32] Leeuwenberg, A. M. et al. "The Performance of Binary Prediction Models in High-Correlation Data: A Simulation Study." *Diagnostic and Prognostic Research* 6, 23 (2022): "

[33] Kim, J. H. "Multicollinearity and Misleading Statistical Results." *Korean Journal of Anesthesiology* 72(6), 558–569 (2019).

[34] Huang, X., Khetan, A., Cvitkovic, M., Karnin, Z. "TabTransformer: Tabular Data Modeling Using Contextual Embeddings." arXiv:2012.06678 (2020).

[35] Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A. “Revisiting Deep Learning Models for Tabular Data.” *NeurIPS* (2021).

[36] Badaro, G. et al. “Transformers for Tabular Data Representation: A Survey of Models and Applications.” *Transactions of the ACL* (2023). Survey noting transformers’ capacity to learn inter-feature dependencies and importance in tabular settings.

[37] Müller, U. A., Dacorogna, M. M., Davé, R. D., Pictet, O. V., Olsen, R. B., & Ward, J. R. (1993). Fractals and intrinsic time—A challenge to econometricians. *Journal of International Money and Finance*, 12(3), 413-438.

[38] Guillaume, D. M., Dacorogna, M. M., Davé, R. R., Müller, U. A., Olsen, R. B., & Pictet, O. V. (1997). From the bird's eye to the microscope: A survey of new stylized facts of the intra-daily foreign exchange markets. *Finance and Stochastics*, 1(2), 95-129.

[39] Li, Y., & Zhong, M. (2018). Asset allocation with time series momentum and reversal. *Journal of Economic Dynamics and Control*, 91, 441-457.

[40] Dacorogna, M. M., Müller, U. A., Olsen, R. B., & Pictet, O. V. (2001). Defining efficiency in heterogeneous markets. *Quantitative Finance*, 1(2), 198-201.

[41] Hasbrouck, J. (1991). Measuring the information content of stock trades. *Journal of Finance*, 46(1), 179-207.

[42] Sirignano, J., & Cont, R. (2019). Universal features of price formation in financial markets: Perspectives from deep learning. *Quantitative Finance*, 19(9), 1449-1459.

[43] Hendershott, T., & Riordan, R. (2013). Algorithmic trading and the market for liquidity. *Journal of Financial and Quantitative Analysis*, 48(4), 1001-1024.

[44] Zhang, Z., Zohren, S., & Roberts, S. (2019). Deep learning for market by order data. *Applied Mathematical Finance*, 26(4), 1-28.

[45] Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2020). Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing*, 93, 106401.

[46] Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., & Howison, S. D. (2013). Limit order books. *Quantitative Finance*, 13(11), 1709-1742.
[47] Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *Review of Financial Studies*, 33(5), 2223-2273.

[48] Bouchaud, J. P., Farmer, J. D., & Lillo, F. (2009). How markets slowly digest changes in supply and demand. In Handbook of financial markets: Dynamics and evolution (pp. 57-160). North-Holland.

[49] Chen, L., Pelger, M., & Zhu, J. (2023). Deep learning in asset pricing. *Management Science*, 70(2), 714-750.

[50] Kercheval, A. N., & Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8), 1315-1329.

[51] Ntakaris, A., Magris, M., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2018). Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37(8), 852-866.

[52] Tóth, B., Palit, I., Lillo, F., & Farmer, J. D. (2015). Why is equity order flow so persistent? *Journal of Empirical Finance*, 30, 57-73.

[53] Dixon, M., Halperin, I., & Bilokon, P. (2020). Machine learning in finance: From theory to practice. Springer.

[54] Lee, C. M., & Ready, M. J. (1991). Inferring trade direction from intraday data. *Journal of Finance*, 46(2), 733-746.

[55] Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005-2019. *Applied Soft Computing*, 90, 106181.

[56] Cont, R., Stoikov, S., & Talreja, R. (2010). A stochastic model for order book dynamics. *Operations Research*, 58(3), 549-563.

[57] Huang, W., Nakamori, Y., & Wang, S. Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10), 2513-2522.

[58] Bogousslavsky, V., & Collin-Dufresne, P. (2023). Liquidity, volume, and price: The impact of order flow on liquidity provision. *Journal of Finance*, 78(4), 2247-2297.

[59] Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654-669.

[60] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[61] Chordia, T., Roll, R., & Subrahmanyam, A. (2002). Order imbalance, liquidity, and market returns. *Journal of Financial Economics*, 65(1), 111-130.

[62] Huang, R., & Stoll, H. R. (1997). The components of the bid-ask spread: A general approach. *Review of Financial Studies*, 10(4), 995-1034.

[63] Pagan, A. (1996). The econometrics of financial markets. *Journal of Empirical Finance*, 3(1), 15-102.

[64] Evans, M. D., & Lyons, R. K. (2002). Order flow and exchange rate dynamics. *Journal of Political Economy*, 110(1), 170-180.

[65] Hautsch, N. (2012). Econometrics of financial high-frequency data. Springer Science & Business Media.

[66] Ulyah, S. M., & Goodell, J. W. (2023). Machine learning in finance: A topic modeling approach. *European Financial Management*, 29(3), 721-753.

[67] Zhang, Y., & Zhao, L. (2024). Deep learning for high-frequency trading: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1), 1-18.

Code available at:

https://github.com/chinaexpert1/FutureFormer_Paper_The_Market_Pays_Attention_by_Andrew_Taylor