A Appendix

A.1 Foundation Models for Structured Data

This work draws inspiration from the success of foundation models beyond NLP, in domains with structured sequential data. This paradigm has been successfully adapted by treating domain-specific sequences as a form of "language." In genomics, for example, models treat DNA or protein sequences as sentences to learn fundamental biological patterns (Ji et al., 2021). For general-purpose time-series forecasting, large pre-trained models have demonstrated strong zero-shot performance on unseen series (Garza et al., 2023; Zhou et al., 2021). Similarly, for tabular data, Transformers pre-trained on a diverse collection of tables can perform inference on new, smaller tables without task-specific fine-tuning (Badaro et al., 2023; Hollmann et al., 2025). In finance, by framing order flow as a structured language of market events, our approach aligns with this proven paradigm, arguing for its direct applicability to the unique challenges of financial data.

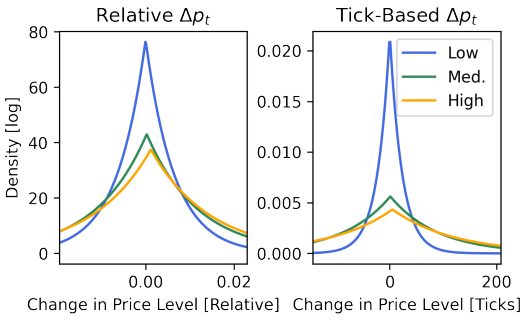

Figure 8: Properties of tick-based vs. relative feature construction for the sample feature Δp_t, across liquidity profiles. We find that relative features generalize better across assets than absolute, tick-based features.

A.2 The Transformer as a Natural Fit

The Transformer architecture is uniquely suited to address these challenges. Its core components map naturally to the fundamental properties of order flow data:

- **Self-Attention**: The attention mechanism is designed to capture complex, long-range dependencies within a sequence. This makes it an ideal tool for modeling the long memory and intricate, non-linear interactions inherent in order flow.
- **Sequence-to-Sequence Framework**: As an autoregressive, sequence-based model, the Transformer inherently handles the asynchronous, event-driven nature of the data, where the time between events is itself a feature to be learned.
- **Adapting to Multi-feature Sequences**: While Transformers excel at processing univariate text, our trade events are multi-feature tuples. A key challenge is thus to effectively discretize and tokenize these mixed-type features into a processable sequence, which motivates our novel tokenization and embedding methodology.

We develop an end-to-end methodology to build TradeFM, a generative foundation model for market microstructure. The following four sections detail each component of our pipeline: our data processing and scale-invariant feature engineering (Section 5), our universal tokenization scheme (Section 6), the TradeFM model architecture (Section 7), and the closed-loop market simulator used for evaluation (Section 8).

A.3 Mid-Price Estimation

A robust estimate of the true market mid-price (p_t^{mid}) is critical for normalizing price-related features. While dedicated market data sources for this exist, they are often expensive. Given our access to raw transaction data, we seek to estimate this value directly. In our partial-information setting, we primarily observe the execution prices (p_t^{exec}) for consummated trades. The raw stream of p_t^{exec} is a noisy version of the true mid-price p_t^{mid}.

A naive approach, such as a simple rolling average of execution prices, is insufficient. A fixed-width window of trades is not comparable across assets with different liquidity levels; a 50-trade window may span less than a second for a highly liquid asset but several hours for an illiquid one. A time-based window (e.g., 2 seconds) is more relevant, but a simple average still fails to account for trade

Time-step	Time-stamp	Asset	Avg. Daily Vol. (shs)	Midprice ($)	Action	Side	Order Price ($)	Vol. (shs)
				⋮				
42	09:45:30	AAPL	53,496,022	182.45	ADD	BUY	182.44	500
43	09:45:38	AAPL	53,496,022	182.48	ADD	SELL	182.50	750
44	09:45:52	AAPL	53,496,022	182.50	CANCEL	BUY	182.49	300
				⋮				

Table 3: Toy example of trading activity for an imaginary AAPL trade sequence, demonstrating the multifeature and heterogeneous nature of our data pre-tokenization.

volume. For example, an average that gives equal weight to a 1,000-share trade at $10.00 and a 1-share trade at $9.00 would produce a misleading estimate.

The conventional solution to this problem is the volume-weighted average price (VWAP), which is

$$\hat{p}_t^{\text{VWAP}} = \frac{\sum_{i=0}^{W} v_{t-i} p_{t-i}^{\text{exec}}}{\sum_{i=0}^{W} v_{t-i}} \tag{2}$$

To make this estimator more reactive to recent information, we introduce **Exponentially-Weighted Volume-Weighted Average Price (EW-VWAP)**. This is calculated by maintaining two separate exponential moving averages (EMAs): one for the volume-weighted price and one for the volume itself. For each incoming trade with execution price e_t and volume v_t, we update the EMAs for the numerator (N_t) and denominator (D_t) as follows:

$$N_t = \alpha \cdot (p_t^{\text{exec}} \cdot v_t) + (1 - \alpha) \cdot N_{t-1}$$
$$D_t = \alpha \cdot v_t + (1 - \alpha) \cdot D_{t-1}$$

The EW-VWAP at time t is then the ratio of these two values:

$$\hat{p}_t^{\text{EW-VWAP}} = \frac{N_t}{D_t} \tag{3}$$

The smoothing factor α is determined by a time-based halflife, ensuring that the estimate gives more weight to larger and more recent trades in a temporally consistent manner. This provides a stable and representative price benchmark that reflects the price at which the bulk of recent market activity has occurred.

A.4 Tokenization Example

Given the imaginary sequence of trade events e_t constructed in Table 3, our features for timestep $t = 43$ are as follows:

- $\Delta t_t = w_t - w_{t-1} = 09{:}45{:}38 - 09{:}45{:}30 = 8\text{sec}$
- $\delta p_t = \frac{o_t - p_t}{p_t} = \frac{\$182.50 - \$182.48}{\$182.48} = \frac{\$0.02}{\$182.48} = 0.011\% = +1.1\text{bps}$
- $v_t = 750\text{shs}$
- $a_t = \text{Add Order}$
- $s_t = \text{Sell}$

Using our calibrated bins, we would discretize these features to the bin indices:

- $i_{\Delta t_t} = \text{bin } 11$
- $i_{\delta p_t} = \text{bin } 7$
- $i_{v_t} = \text{bin } 7$
- $i_{a_t} = \text{bin } 0$

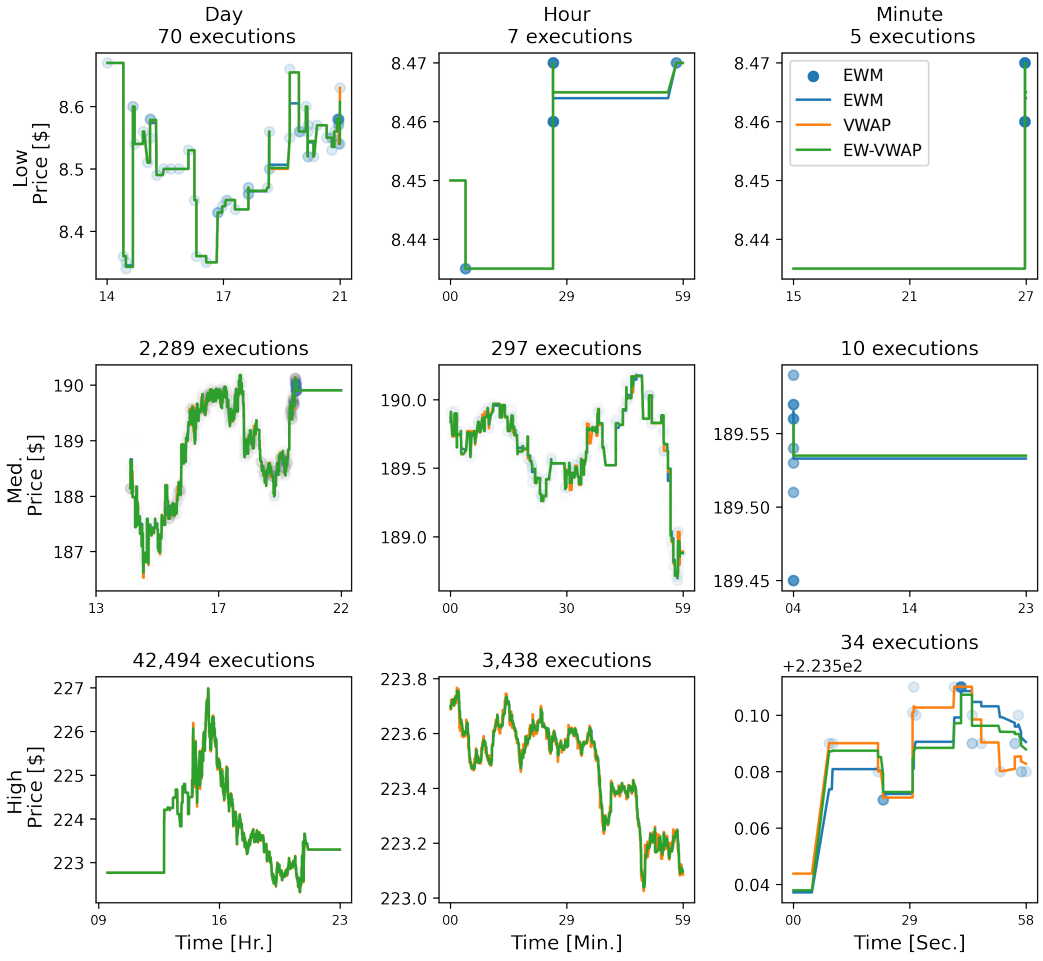

Figure 9: Comparison of mid-price estimators across time scales and liquidity levels. Our proposed EW-VWAP provides a more stable and responsive estimate than standard VWAP or EWM, closely tracking the executed fill prices across different time scales and liquidity regimes.

- $i_{s_t} =$ bin 1

Using Eq. 1 would yield:

$$i_{trade} = 6,011$$

Assuming an opening price $p_0 = \$179.50$, we would have change in price level feature:

$$\Delta p_t = \frac{p_t - p_0}{p_0} = \frac{\$182.50 - \$179.50}{\$179.50} = 0.017\% = +1.7\text{bps}$$

Discretizing this using our bins would give $i_\Delta p_t = 19$. Based on the asset's average daily volume of 53 million, which falls into the high liquidity bin, our liquidity indicator i_l would be 2. If this sequence was a market-level sequence, we would have market-participant indicator $I_{MP} = 0$.

Our final model input would then be:

$$[i_l, I_{MP}, i_{\Delta p_t}, i_{\text{trade}}] = [2, 0, 19, 6011].$$

B Reproducibility Guide

B.1 Model Backbone

TradeFM is a decoder-only Transformer, trained from scratch with a custom configuration. The architecture is based on the Llama family (Touvron et al., 2023) and incorporates modern enhancements for efficiency and performance, including:

- **Grouped-Query Attention (GQA)**: Balances the performance of Multi-Head Attention with the reduced memory bandwidth of Multi-Query Attention, enabling faster inference and larger batch sizes.
- **Rotary Position Embeddings (RoPE)**: Encodes relative positional information by applying a rotation to query and key vectors, which has been shown to improve generalization for long sequences.

B.2 Model Hyperparameters and Scaling

The model size is guided by the Chinchilla scaling laws, which suggest a compute-optimal ratio of approximately 20 training tokens per model parameter (Kaplan et al., 2020; Hoffmann et al., 2022). Given our dataset of 10.7 billion tokens, this implies a target model size of around 525 million parameters. Our final hyperparameters are as follows:

- **Context Length**: 1,024 tokens
- **Hidden Layers**: 32
- **Embedding Dimension**: 1,024
- **Intermediate MLP Size**: 4,096
- **Attention Heads**: 32
- **Key-Value Heads (GQA)**: 8 heads, 4 groups
- **Total Parameters**: 524.4 Million

The 1:4 ratio between the embedding dimension and the intermediate MLP size is chosen in accordance with best practices for Transformer models (Petty et al., 2024).

B.3 Training Configuration

We train the model on an AWS instance with 3 Nvidia A100 GPUs, each with 80GB of RAM. All training is performed in `fp16` half-precision. To achieve an effective batch size of 4,032, we use a per-device batch size of 24 and gradient accumulation over 56 steps. For further memory

Model Size	Num. Train Tokens	Batch Size	GPUs	Train Time / Epoch (hrs)	Optimization
125M	2.6B	24	3xA100	17	Accelerate
250M	6.4B	32	4xA10G	29	DeepSpeed
500M	10.7B	24	3xA100	125	Accelerate

Table 4: Training configuration for different model sizes.

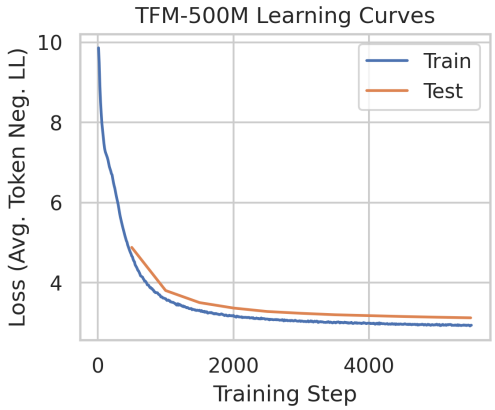

Figure 10: Learning curve for TradeFM-500M, reporting the average token negative log likelihood over the test set constituting Jan. through Sept. 2025 as loss.

optimization and training acceleration, we use the Accelerate library. The model is trained using the AdamW optimizer with a linear learning rate schedule, a learning rate of 5×10^{-5}, and 500 steps of warmup. Following recommendations for training on large datasets (Muennighoff et al., 2025), we train for a total of 4 epochs.

Due to compute constraints, different model sizes in memory, and different dataset sizes implied by Chinchilla scaling laws, our training setups vary slightly between model sizes. We summarize these variations in Table 4.

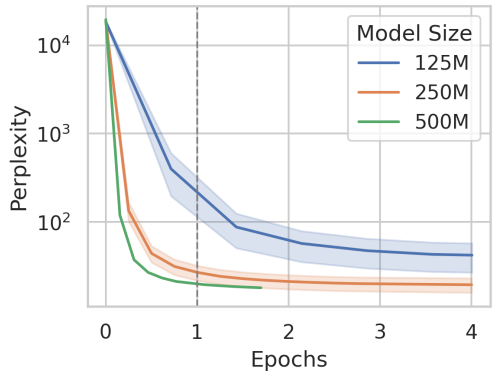

Figure 11: Out-of-sample Perplexity vs Epochs for all TradeFM models. Epoch 1 represents one pass over the entire dataset; subsequent training is on repeated data.

B.4 TOKENIZER PSEUDOCODE

Algorithm 1 Tokenizer Pseudocode

Input: Flattened trading data with features: price, volume, time, action, side, trader, account
Input: Bin counts: N_{price}, N_{depth}, N_{volume}, N_{time}, $N_{\text{side}} = 2$, $N_{\text{action}} = 2$
for each feature f in {price, depth, volume, time} **do**
  Remove NaN and infinite values from f
  Compute upper outlier threshold $u = \text{percentile}(f, 99)$
  if feature is double-sided **then**
    Compute lower outlier threshold $l = \text{percentile}(f, 1)$
    Assign values outside $[l, u]$ to lower / upper outlier bins
  else
    Assign values above u to upper outlier bin
  end if
  if using equal-frequency bins **then**
    Compute bin edges B_f using quantile binning with N_f bins
  else
    Compute bin edges B_f using histogram binning with N_f bins
  end if
  Digitize f into bin indices I_f using B_f
  Handle outliers: assign out-of-range values to edge bins as needed
end for
for each categorical feature c in {action, side, trader, account} **do**
  Map each category to a unique integer index I_c
end for
for each order o in the dataset **do**
  for each feature f **do**
    if $o[f]$ is NaN **then**
      Impute $o[f]$ with a random valid bin index or default value
    end if
  end for
  Compute token index for o:
  $T_o = I_{\text{action}} \times N_{\text{side}} \times N_{\text{depth}} \times N_{\text{volume}} \times N_{\text{time}}$
    $+ I_{\text{side}} \times N_{\text{depth}} \times N_{\text{volume}} \times N_{\text{time}}$
    $+ I_{\text{depth}} \times N_{\text{volume}} \times N_{\text{time}}$
    $+ I_{\text{volume}} \times N_{\text{time}}$
    $+ I_{\text{time}}$
  Assign T_o to order o
end for

B.5 MARKET SIMULATION PSEUDOCODE

Algorithm 2 Market Simulator: Part 1 - Initialization and Utilities

Input: Sequence of order transactions, initial price p_0, simulation parameters

Initialize Exchange:
Set initial price p_0
Initialize order book, midprice, fills, deletes, spreads, bid/ask volumes

Function: INITIALIZEORDERBOOK(order_columns)
Reset order book, midprice, fills, deletes, spreads, bid/ask volumes
Set initial bid/ask to p_0

Function: GETORDERPRICE(transaction)
if order is market **then**
  if side is Sell **then**
    price ← lowest ask
  else
    price ← highest bid
  end if
else
  price ← (order price depth / 10,000) × current midprice + current midprice
end if
Return price

Function: GENERATEFILL(best_past_order, order, quantity)
Compute time since best_past_order
Determine match price:
if both orders are market **then**
  price ← current midprice
else if order is market **then**
  price ← best_past_order price
else if best_past_order is market **then**
  price ← order price
else
  price ← best_past_order price
end if
Return fill record with IDs, sides, prices, depths, volume, time delta

Algorithm 3 Market Simulator: Part 2 - Simulation Step Functions

Function: StepOrderBook(order)
Extract side and price from order
while order volume > 0 **do**
  if side is Sell **then**
    Find matching Buy orders with price $\geq$ order price
  else
    Find matching Sell orders with price $\leq$ order price
  end if
  if no matching orders **then**
    break
  end if
  Select best matching order (highest bid or lowest ask)
  if best_past_order volume $>$ order volume **then**
    Reduce best_past_order volume by order volume
    Record fill and **return**
  else if best_past_order volume $<$ order volume **then**
    Reduce order volume by best_past_order volume
    Remove best_past_order from book
    Record fill
  else
    Record fill
    Remove best_past_order from book
    return
  end if
end while
if order volume > 0 **then**
  Add partially filled order to book
end if

Function: StepMidprice(transaction)
if transaction is Delete **then**
  Use current order book
else
  Add transaction to temporary order book
end if
Update highest bid and lowest ask from book
Compute midprice as average of highest bid and lowest ask
Record midprice and bid/ask volumes

Function: StepSim(transaction)
Update transaction midprice
if action is Add **then**
  Compute order price
  Update midprice
  Step order book
else if action is Delete **then**
  Match on order ID
  Remove matching orders and record deletes
  Update midprice
end if
Record simulation time for profiling

Function: RunSimulation(data)
Initialize order book
for each transaction in data **do**
  StepSim(transaction)
end for
Return fills and midprice history

B.6 Zero-Intelligence Baseline

The Zero-Intelligence (ZI) agent is a canonical null model used to test whether a model learns complex, conditional dynamics beyond the market's basic structural properties (Gode & Sunder, 1993; Farmer et al., 2005). To provide a fair baseline, our ZI agent generates orders stochastically by sampling from distributions calibrated to match the marginals of key features in a 450-million-trade sample of the training data.

Specifically, side and action type are sampled from their empirical categorical distributions; interarrival time is sampled from a fitted Exponential distribution; order volume from a fitted Exponential distribution; and price depth is drawn from a Gaussian Mixture Model (GMM).

The resulting ZI agent orders are processed through the identical market simulator and evaluation pipeline as TradeFM to ensure a direct and fair comparison. We compute 2,048 rollouts of 1,024 events, and compute the same stylized facts.

B.7 Compound Hawkes Baseline

Hawkes Processes are commonly applied to market data for their ability to robustly model interarrival times of self-exciting events (Bacry et al., 2015; Jain et al., 2024). We adopt the Compound Hawkes model which combines a Hawkes process for modeling interarrival times with empirical distributions for modeling additional event features like volume and price depth. We use the same 450-million-trade data as is used to train our zero-intelligence baseline, and separate the data based on action and side.

We then fit a Hawkes process using a sum of exponential kernel, with 4 dimensions, one for each combination of action and side (buy-delete, buy-add, sell-delete, sell-add). For each of these action-side combinations we calibrate a Gaussian Mixture Model for price depths, and an Exponential for volume.

C Scaling Analysis

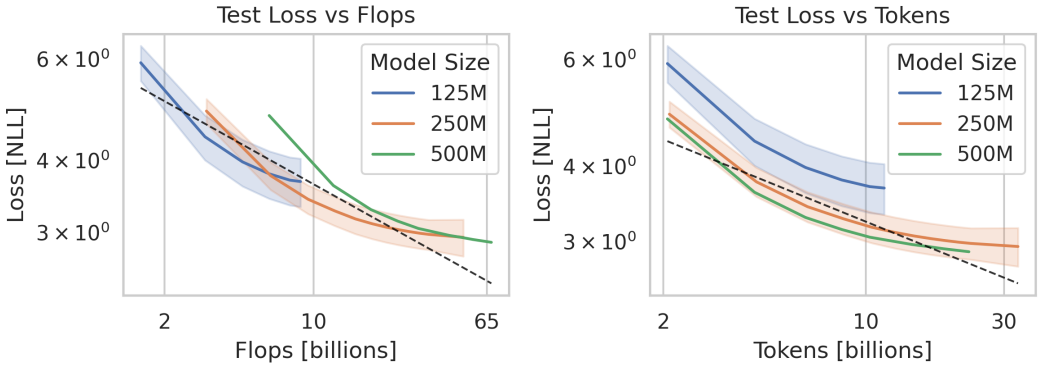

Figure 12: Scaling law results reporting test loss (negative log likelihood) on held-out data one month in advance of the training data cutoff. The black dashed line represents the power law fit to the minimum loss frontier.

To substantiate the *Foundation Model* claim, we conducted a comprehensive scaling analysis of our approach. We trained models $\in\approx [125M, 250M, 500M]$.

Our **500M parameter model** is still training, and while currently at an early checkpoint, its performance aligns with our scaling projections. The scaling law plots in Figure 12 demonstrate the expected power-law relationships between compute, dataset size, and test loss. These plots include repeated data, as we train for four epochs. We verify this by computing the minimum loss frontier in terms of both compute and dataset size, and fitting power laws to find that the test loss $L(C)$ with respect to compute in Flops C, and $L(D)$ with respect to dataset size in tokens D, follow:

$$L(C) \propto C^{-\alpha_C}; \alpha_C \approx 0.21$$
$$L(D) \propto D^{-\alpha_D}; \alpha_D \approx 0.19$$

While 500M is small relative to general purpose LLMs (Llama-3 8B, GPT-OSS 20B), it is large for the Financial Microstructure domain, and similar to other domain-specific models such as MaRS. Standard SOTA models in this field typically have $10M$ parameters (e.g., DeepLOB (60K params) and LOBS5 (6.3 M params)). TradeFM represents a $> 50x$ **increase in model capacity** over existing domain-specific baselines.

D Extended Experimental Results

D.1 Dataset Details

Table 5 contains details on the various held-out datasets used for evaluation.

Country	Number of Assets	Date-Asset Pairs	Tokens
US	6,885	81,203	857,017,219
China	4,926	68,925	37,408,529
Japan	2,932	37,235	286,476,052

Table 5: Dataset statistics for US, China, and Japan held-out data. All geographies are evaluated on Jan. 2025 data.

D.2 Simulator Validation

In order to evaluate the simulator, we replay sequences of real orders through it and compare the statistical properties of the resulting simulated trade fills against the real fills from our historical data. We focus on two key metrics: the cumulative distribution function (CDF) of fill volumes and the CDF of lot counts (the number of discrete fills required to complete a single order). As shown in Figure 13, we find a strong correspondence between the real and simulated distributions across assets of varying liquidity, confirming that our simulator is a high-fidelity environment for evaluation. We find correlations of 0.91 and 0.98 between sim and real volumes and lot counts, respectively.

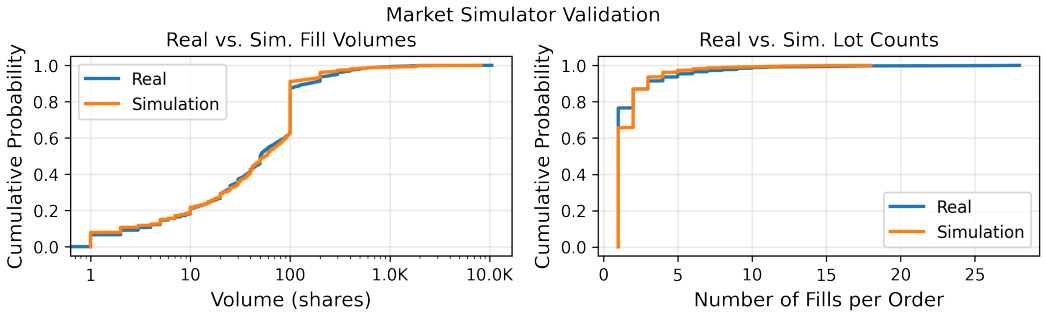

Figure 13: Stylized facts of market simulator fills: (left) fill volumes; (right) lot counts, or number of separate fills taken to fulfill an order. We see good correspondence between simulated and actual fills, with a correlation of 0.91 for volumes and 0.98 for lot counts, respectively.

D.3 Temporal Drift

As financial markets are dynamic and market regimes are constantly changing, we investigate the tendency of model performance to drift over time. Our tokenizer's main contribution is to standardize representations of market features over both the liquidity and time regime.

In Fig. 14 we demonstrate the universality of these features by exploring the distribution of our relative price level, relative price depth, interarrival time, and volume features in both the month used to calibrate our tokenizer, Feb. 2024, and one year later in Feb. 2025. We observe that our features are stationary over this period even as volatility, price level, and other market conditions vary. Fig. 15 shows the Kolmogorov-Smirnov and Wasserstein distance of each of these features between the tokenizer calibration month and each of 9 held-out months. We include a non-stationary feature, raw midprice, to contextualize the stationarity of these metrics.

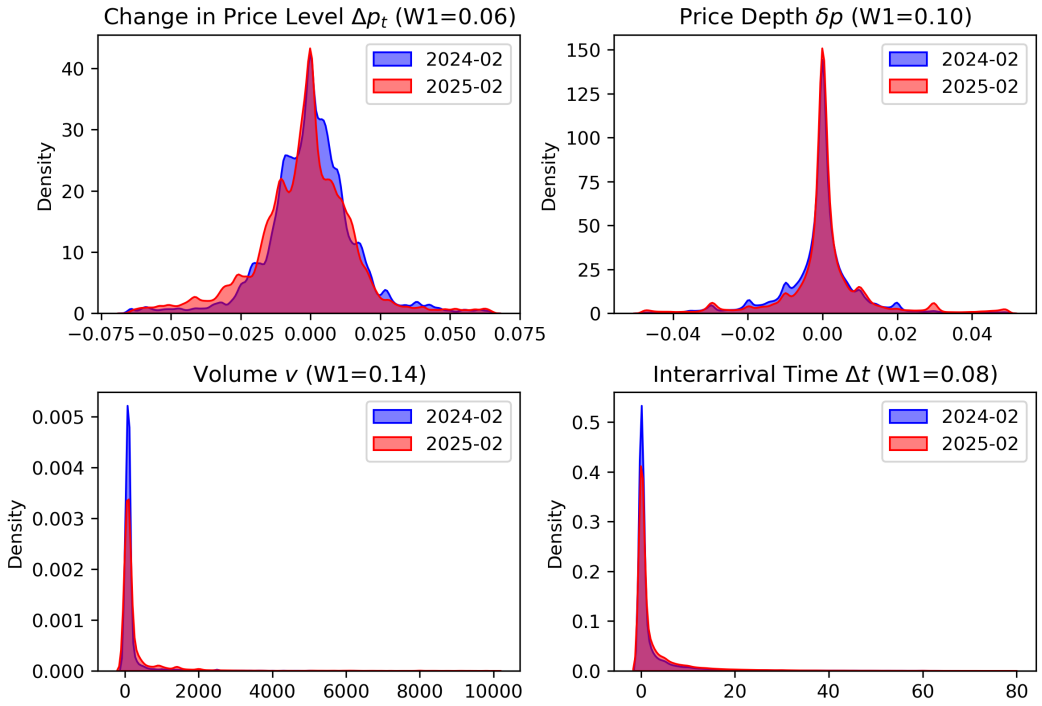

Figure 14: Kernel-density estimation of feature distributions from the tokenizer calibration period of Feb. 2024 to one year later in Feb. 2025. Our feature engineering successfully makes these features stationary over time, allowing our model to generalize to out of distribution temporal regimes.

Figure 15: Kolomogorov-Smirnov and Wasserstein distances between distributions of our features during our tokenizer calibration month and held-out months. We include raw midprice, a non-stationary feature, for context.

In Fig. 16 we extend the aggregated results in Table 2 for all quantities of interest. We observe that while these metrics do vary within a range, the variance is small and our method mostly achieves higher fidelity than baselines.

D.4 Market Simulation & Stress Testing

The integrated TradeFM-simulator system functions as a high-fidelity environment for complex "what-if" analyses and stress testing. This allows for the study of systemic risk and market sta-

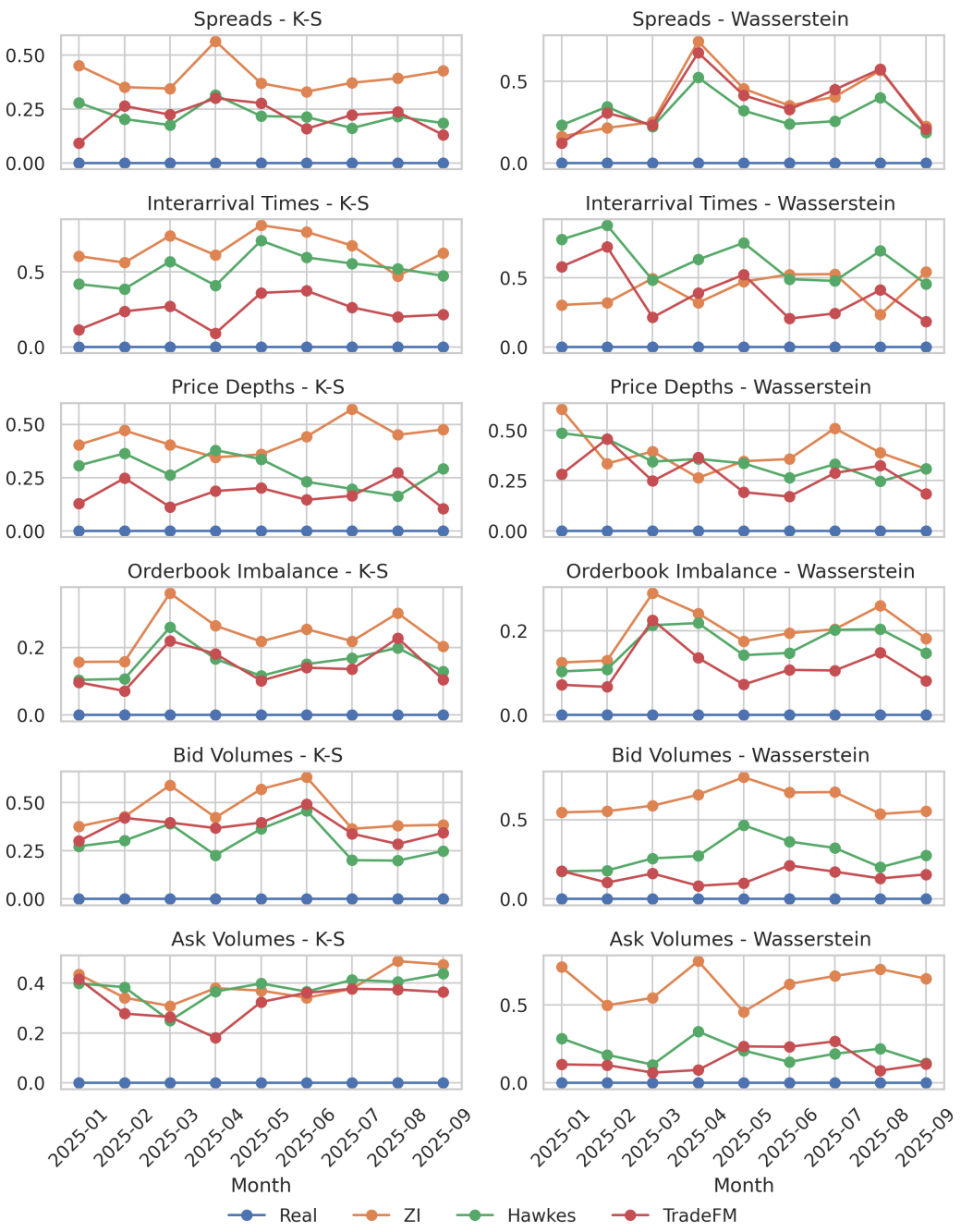

Figure 16: Wasserstein distance and Kolmogorov-Smirnov statistic of feature distributions and emergent market factors from various methods over nine held-out months.

bility in a controlled environment. The ability to generate plausible, multi-step forecasts of future market trajectories, as illustrated in Figure 17, is a direct outcome of this closed-loop simulation capability.

Such systems are also useful to regulators and risk managers (Dwarakanath et al., 2024), who can use this system to simulate the market's response to extreme or counterfactual scenarios, such as by injecting large, anomalous orders into a historical context and observing the resulting price tra-

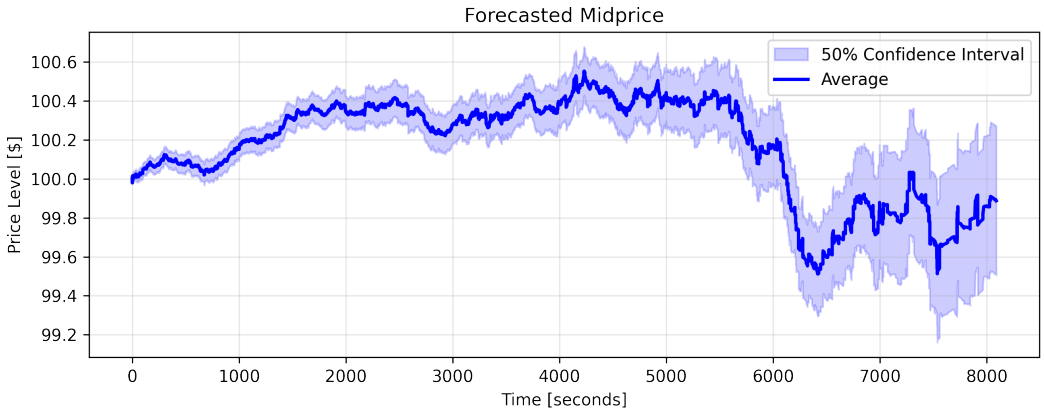

Figure 17: Multi-step mid-price forecast generated via rollouts for an imaginary asset initially priced at $100. The average trajectory and 50% confidence interval over 256 simulations show the model generates plausible, non-stationary market paths.

jectory. Fig. 18 demonstrates this capability – for a sample asset, we artificially inject buy or sell orders at 10x the frequency found in the real context, and average midprice forecasts over 10 rollouts. When we inject artificial sell orders, the midprice drops, and when we inject buy orders, the midprice rises, illustrating realistic behavior.

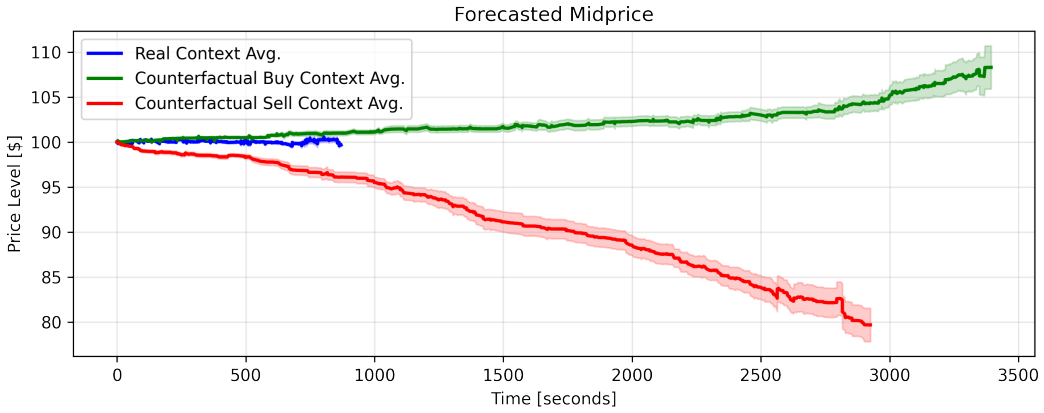

Figure 18: Stress testing via counterfactual simulation. The model's generated price path responds realistically to injected anomalous order flow (10x normal frequency), demonstrating its utility for impact analysis.

D.5 Multi-Agent Modeling & RL Fine-Tuning

Our system provides a high-fidelity, interactive environment for training and evaluating sophisticated, learning-based agents. The pre-trained TradeFM can serve as a realistic "background" market, generating plausible and reactive counterparty order flow. This creates a dynamic training ground for:

- **Reinforcement Learning (RL) for Optimal Execution**: RL agents can be trained to learn optimal strategies for executing large orders by interacting with the simulated market, minimizing costs such as price impact and the bid-ask spread.
- **Multi-Agent Systems (MAS)**: The simulator can be populated with multiple, heterogeneous learning-based agents to study the emergent, collective behaviors and potential instabilities that arise from their interactions. The participant-level conditioning of our model provides a natural and powerful mechanism for initializing and fine-tuning diverse agent policies within such a system.