State-Space Meets Attention: Modeling Financial Time-Series for Volatility Estimation

Junyao Duan (jd4024), Yingxin Zhang (yz3202), William Vietor (wgv2103)

1 Introduction

Short-term volatility forecasting is a cornerstone of modern quantitative trading, underpinning option pricing, risk management, and execution strategies in electronic markets. The Optiver Realized Volatility Prediction challenge [4] offers a rigorous testbed, requiring models that can handle high-resolution, irregularly sampled limit-order book data across hundreds of stocks. Classical approaches—including gradient-boosted trees (e.g., LightGBM [5]), shallow MLPs, and 1D convolutional networks—have demonstrated solid baseline performance but depend heavily on manual feature engineering and may falter on very long horizons or sparse observations.

Deep sequence models bring new capabilities to this problem. Transformer encoders [7] leverage self-attention for efficient, parallel modeling of long-range dependencies, while continuous-time state-space architectures such as Mamba [2, 1] combine structured latent dynamics with convolutional mixing to achieve linear-time inference and a reduced memory footprint. Beyond these pure paradigms, hybrid designs—such as appending attention layers to Mamba blocks, interleaving state-space and attention modules, or fusing them in parallel—promise to unite the best of both worlds, capturing local microstructure fluctuations and global sequence patterns in a unified model.

In this work, we benchmark a spectrum of architectures on the Optiver dataset: classical machine-learning baselines, pure Transformer and Mamba models, and three hybrid variants (Mamba with Attention, Interleaved Mamba–Attention, and Parallel Mamba–Attention). Our goal is to determine which design principles most effectively capture the complex temporal dynamics of high-frequency volatility data.

2 Related Work

Previous work on the Optiver volatility prediction task has relied largely on tree-based methods (e.g., LightGBM [5]), shallow feed-forward networks, and 1D convolutional architectures. These models are adept at capturing local non-linear relationships and short-term temporal patterns but often require extensive feature engineering and can struggle to generalize over long or irregularly sampled sequences.

Recurrent networks such as LSTMs [3] offer a more direct mechanism for modeling sequential dependencies by maintaining internal state across time steps. However, their inherently sequential processing limits parallelism and scalability, making them computationally expensive on high-resolution, long-horizon time series common in financial microstructure data.

Transformers [7] mitigate these limitations through self-attention, which computes pairwise interactions between all time steps in parallel. This allows them to capture both short- and long-range dependencies efficiently, and they have demonstrated state-of-the-art performance in diverse domains including natural language, computer vision, and time series analysis.

More recently, continuous-time state-space models like Mamba [2, 1] have gained traction for long sequence modeling. By combining structured state updates with convolutional mixing, Mamba achieves

[0] Our code is available at `https://github.com/wvietor/coms4995FinalProject`.

Preprint. Under review.

linear-time inference and a reduced memory footprint, while matching or exceeding Transformer accuracy in many sequence modeling benchmarks [6].

3 Method

3.1 Data Preprocessing

The dataset comprises one-second snapshots of high-frequency limit-order book activity and executed trades. Each record—indexed by `stock_id` and `time_id` —features normalized first- and second-level order-book entries (e.g., `bid_price[1/2]`, `ask_price[1/2]`, `bid_size[1/2]`, `ask_size[1/2]`), along with trade sizes and order counts. This multi-depth view of liquidity and price dynamics provides the fine-grained information necessary to forecast short-term realized volatility from market microstructure signals.

The prediction target for each observation at `time_id` $= t$ is the realized volatility over the 10-minute interval that follows, which we denote by `time_id` $= t'$. Since `time_id` values are not guaranteed to be sequential, we define t' as the next chronologically ordered interval following t, based on a reconstructed time axis. The realized volatility, denoted $\mathrm{RV}_{t'}$, is computed from mid-prices sampled at one-second resolution within the target interval:

$$\mathrm{RV}_{t'} = \sqrt{\sum_{s=1}^{599} \left(\log P_{t',s} - \log P_{t',s-1}\right)^2},$$

where $P_{t',s}$ denotes the mid-price at the s-th second of the interval t'. In our setting, mid-price is calculated as the average of the best bid and ask prices at the top of the order book:

$$P_{t',s} = \frac{\texttt{bid\_price1}_{t',s} + \texttt{ask\_price1}_{t',s}}{2}.$$

This non-overlapping estimator serves as a high-frequency proxy for true return volatility. Ground truth volatility values are available for training, while test targets are hidden and only evaluated during submission in a forecasting setup. As we do not have access to the actual test labels, we construct our own validation and test splits using the methodology described below to simulate the forecasting setting and assess model generalization.

Our initial models used only raw order-book and trade features. To accelerate convergence and embed market microstructure insights, we then extracted a richer set of derived descriptors. We computed multi-level spreads (e.g., `spread11`, `spread12`), VWAPs, log-scaled trade sizes and bid/ask imbalances, and rolling estimates of realized volatility. Additionally, we fitted short-window OLS regressions on key series (e.g., `bid_price1`, `ask_price2`, `vwap1`) to obtain local slope and intercept terms as indicators of immediate trends. These engineered features surface latent liquidity and volatility patterns, improving the model's ability to learn complex market dynamics.

To streamline training, we built a custom PyTorch data loader that constructs fixed ten-minute windows of observations and yields them as mini-batches of shape $(B, 600, F)$, where 600 is the number of one-second timesteps and F is the feature dimension. We forward-fill missing entries using the last observed values—since missing data generally indicates no change in the order book—and truncate or pad each window to maintain consistent length.

3.2 Baseline Architectures

RNN/LSTM We normalize input features and process them through a stack of LSTM layers to capture temporal dependencies. The final hidden and cell states are aggregated, flattened, and passed through a deep MLP head to produce the scalar prediction.

Transformer We project input features into a common model space, add positional information to capture sequence order, and then feed the sequence through a stack of self-attention encoder layers to model long-range dependencies. The resulting sequence representation is aggregated (e.g., by mean pooling) and passed through a final linear layer to produce the scalar prediction.

Mamba We project input features into a common model space and pass them through a sequence of Mamba state-space blocks, which combine continuous-time dynamics with convolutional mixing to capture both long- and short-range dependencies. After layer normalization, the final time-step embedding is projected to produce the scalar prediction.

3.3 Architectures to Be Evaluated

Mamba with Attention This architecture combines six sequential Mamba blocks—a linear-time state-space model designed for long sequences—with a single Transformer-style Multi-Head Attention (MHA) layer appended at the end. The Mamba blocks capture local and medium-range temporal patterns efficiently, while the final MHA layer aggregates global dependencies across the entire sequence. This hybrid design leverages the strengths of both paradigms: efficient sequence modeling from Mamba and long-range pattern recognition from attention.

InterleavedMambdaAttn The Interleaved architecture alternates Mamba and Attention blocks in a stacked configuration, allowing information to pass through both mechanisms at each depth. This structure promotes richer representations by interleaving localized modeling (Mamba) with global contextualization (Attention). Each layer is structured as: Input $\rightarrow$ Mamba $\rightarrow$ Multi-Head Attention $\rightarrow$ LayerNorm $\rightarrow$ Next Block This design encourages the model to capture both microstructure fluctuations and broader temporal context at multiple abstraction levels.

ParallelMambdaAttn The Parallel Mamba-Attention network processes inputs through Mamba and Attention modules in parallel at each layer. Their outputs are then combined via summation before passing through normalization. This fusion architecture allows the model to simultaneously leverage both localized state transitions and cross-token relationships, improving robustness to diverse temporal patterns. The shared input allows both branches to process identical signals through different lenses.

3.4 Model Evaluation

We evaluate model performance using Root Mean Squared Percentage Error (RMPSE), a scale-invariant metric well-suited to volatility forecasting. RMPSE penalizes relative deviations between predicted and actual values, making it more informative than absolute error metrics when targets span multiple orders of magnitude. Formally, it is defined as:

$$\text{RMPSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{y_i}\right)^2},$$

where $\hat{y}_i$ denotes the predicted realized volatility and y_i the ground truth.

4 Experiments

4.1 Experiment Setup

Hardware. The Transformer, Mamba, Mamba with Attention, Interleaved Mamba Attention, and Parallel Attention Mamba models were run on a Google Colab Pro+ VM equipped with a single NVIDIA Tesla T4 GPU (15 GB VRAM) and 2 vCPU Intel Xeon cores. The host machine provides 51 GB system RAM and 236 GB ephemeral disk (Ubuntu 22.04.4 LTS, Linux 6.1.123). We used PyTorch 2.4.0 + cu121 with CUDA 12.1, Python 3.11.12, and enabled automatic mixed precision (torch.cuda.amp.autocast). All runs use a fixed random seed 42 for reproducibility.

Target scaling. Raw realised-volatility values are $\sim 10^{-4}$, which led to vanishing gradients early in training. We therefore multiply the targets by 10^3 before computing the RMSPE loss and divide the model outputs by 10^3 at prediction time. This rescaling leaves RMSPE unchanged but improves numerical stability and accelerates convergence.

Table 1: Hardware and software environment used for all experiments.

Component	**Specification**
GPU	1 NVIDIA Tesla T4 (15 GB VRAM)
CPU	2 × Intel Xeon vCPUs (Colab VM)
System RAM	51 GB
Disk	236 GB ephemeral storage (188 GB free)
Operating System	Ubuntu 22.04.4 LTS, Linux 6.1.123
CUDA / cuDNN	CUDA 12.1, cuDNN bundled with PyTorch 2.4
PyTorch	2.4.0 +cu121
Python	3.11.12
Mixed Precision	`torch.cuda.amp.autocast` enabled
Random Seed	42

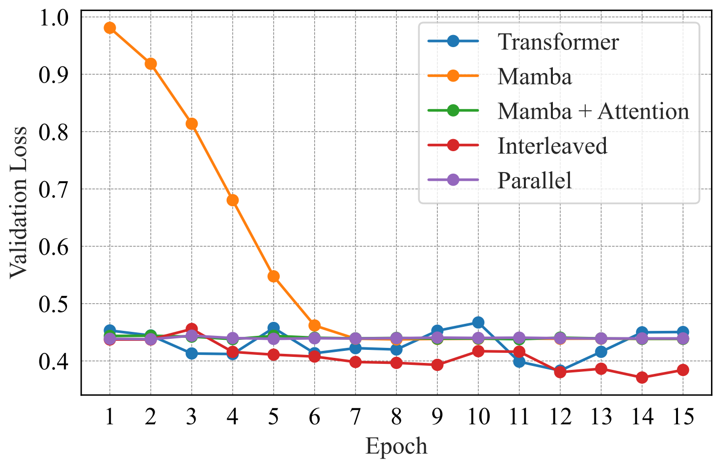
4.2 Main Results

Table 2: Capacity-matched models (≈12 M parameters, best of 15 epochs, single seed).

Model	**Params (M)**	**Best Val RMSPE ↓**	**Test RMSPE ↓**	**Epoch**
Transformer 6×256	12.9	0.3831	0.4502	12
Mamba 6×300	12.8	0.4376	0.4386	8
Mamba + Attention 6×280	12.3	0.4378	0.4385	11
Interleaved 6×256	12.4	**0.3711**	**0.3841**	14
Parallel 6×280	12.4	0.4379	0.4392	2

Capacity scaling. Previouly, we trained pure-Mamba, Mamba + Attn and Parallel models with 8 to 9 M parameters (results not included in the table). We increased the number of parameters to 12 M to match the scale of transformer and the interleaved hybrid model. The validation RMSPEs for the three enlarged models remain unchanged (<0.003). We attribute this to a dataa and feature limited regime and width-only scaling. Future work should explore deeper stacks or richer feature sets as well.

Figure 1: Validation loss per epoch for different models.



The results in Table 2 and Figure 1 reveal several key insights about how these capacity-matched (≈ 12 M-parameter) architectures perform on high-frequency volatility forecasting. First, the vanilla Transformer achieves a strong baseline with a best validation RMSPE of 0.3831 and a test RMSPE of 0.4502 (epoch 12). In contrast, the pure Mamba state-space model (6×300) converges much more rapidly—reaching its best validation RMSPE of 0.4376 by epoch 8—but plateaus at a higher error on both validation and test sets (0.4376 / 0.4386), indicating that its inductive bias toward continuous-time dynamics alone is insufficient to match the Transformer's representational power.

When we append a single self-attention layer after six Mamba blocks ("Mamba + Attention 6×280"), or instead fuse a lightweight attention branch inside every Mamba layer ("Parallel 6×280"), the validation and test RMSPE remain essentially unchanged from the pure-Mamba baseline (best val$\approx$0.438 in both cases). This indicates that neither adding attention only at the end nor shallow, per-layer fusion is sufficient to unlock a clear accuracy benefit on this dataset.

By contrast, the interleaved architecture—alternating Mamba blocks with Transformer-style multi-head attention layers (Interleaved 6×256)—delivers a substantial improvement. It attains the lowest validation RMSPE (0.3711) and the best test RMSPE (0.3841) of all models, albeit at a later convergence point (epoch 14). This demonstrates that deeply integrating state-space updates with global attention at each layer enables the model to capture both local microstructural fluctuations and long-range dependencies more effectively than either mechanism alone or when they are combined superficially.

5 Conclusion

We compared five capacity-matched sequence models for one-second–resolution volatility forecasting on the Optiver dataset: a 6-layer Transformer, a 6-layer Mamba state-space model, and three hybrids that combine Mamba and self-attention by (i) appending a single attention layer, (ii) interleaving blocks, and (iii) fusing both mechanisms in parallel.

Our results lead to two lessons. *First*, simply stacking an attention layer on top of a state-space backbone—or vice-versa—delivers only marginal accuracy gains; the two inductive biases do not automatically reinforce one another. *Second*, architectures that chain local (SSM) and global (attention) modules in alternating fashion throughout the network can outperform either paradigm in isolation, albeit at the cost of extra compute.

After equalising parameter budgets ($\approx$ 12 M) and training for 15 epochs, the models split into two accuracy bands. Mamba, two of the three Mamba-derived variants, and the Transformer cluster tightly at 0.437 ± 0.001 RMSPE, while the Interleaved hybrid achieves 0.371. Thus the overall spread is about 0.066 RMSPE ($\approx$15 % relative), but within the upper band the models differ by less than 0.001 ($\approx$ 0.2 % relative).

6 Future Work

While our preliminary results demonstrate the potential of state-space and attention-based models for high-frequency volatility forecasting, several practical limitations constrain the scope of our current study. We outline four primary directions for extending this work.

Lightweight Hybrid Designs One promising research direction is to make the hybrid architecture leaner without sacrificing the complementary strengths of state–space and attention. The first idea is that we can further investigate lighter interleaving schedules, such as adding attention layer every k (k > 1) mamba blocks instead of every block. The second idea is that instead of using 50/50 fixed fusion in the parallel model, we can add a learned gate to dynamically control how much of each branch's outout contributes to the final representation.

Scaling to the Full Dataset and Larger Models Due to the limited resources of our Colab environment (a single NVIDIA T4 GPU with 15 GB VRAM and 2 vCPUs), we restricted our experiments to only five of the 112 stocks in the Optiver challenge. In future work, we will utilize larger GPU clusters and longer training schedules to process the entire dataset with expanded model capacities. By training richer models across all stocks for more epochs—without premature plateaus or overfitting—we expect to capture a broader range of market behaviors and rare volatility events, thereby enhancing generalization and robustness under extreme conditions.

Deeper Feature Engineering Constrained by project deadlines, our current feature set is limited to basic order-book spreads, VWAPs, log-transformed trade imbalances, and rolling realized volatility.

With additional time, we will conduct a thorough exploratory analysis—leveraging microstructure theory and statistical diagnostics—to develop richer descriptors such as higher-order imbalance ratios, dynamic liquidity measures, and automated interaction terms. We will also investigate learned feature representations (e.g., embedding layers or automated search algorithms) to uncover latent patterns.

Benchmarking and Cross-Pollination with Top Kaggle Approaches We have yet to perform an in-depth comparison between our pipeline and the methodologies used by leading Kaggle competitors. Examining their published approaches—including preprocessing techniques, feature engineering strategies, and model ensembles—can both validate our design choices and reveal opportunities for improvement. By integrating selected best practices from those solutions, we expect to enhance preprocessing robustness, enrich feature representations, and refine ensembling strategies, ultimately boosting predictive performance.

References

[1] Albert Gu, Karan Goel, Tri Dao, Christopher Ré, and Kurt Keutzer. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. URL `https://arxiv.org/abs/2405.21060`.

[2] Albert Gu, Karan Goel, Atri Zhang, Tri Dao, Christopher Ré, and Kurt Keutzer. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2024. URL `https://arxiv.org/abs/2312.00752`.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

[4] Kaggle. Optiver realized volatility prediction. `https://www.kaggle.com/competitions/optiver-realized-volatility-prediction`, 2021. Accessed: 2025-05-11.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[6] Zichao Liu, Yuning Du, Xiaojie Ma, and Ji Lin. The hidden attention of mamba models. *arXiv preprint arXiv:2403.01590*, 2024. URL `https://arxiv.org/abs/2403.01590`.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL `https://arxiv.org/abs/1706.03762`.

A Appendix

A.1 Use of Generative AI Tools

We made limited use of generative AI tools (e.g., Windsurf, ChatGPT) to assist with low-level coding tasks and improve the clarity of our writing. These tools were used strictly to accelerate implementation and refine phrasing. All conceptual work, model architecture choices, experimental design, and analysis were independently developed by our team. In code, we reviewed and modified any AI-assisted suggestions to ensure correctness and understanding. All uses of generative tools were carefully constrained to preserve the originality and integrity of the work.