Learning the Market: Machine Learning and Generative Modelling for Limit Order Books



Peer Nagy

Reuben College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2025

Abstract

This integrated thesis advances the application of machine learning (ML) to limit-order book (LOB) markets, a central mechanism of financial market microstructure. The work spans reinforcement learning (RL) for trading and inventory management, hardware-accelerated LOB simulation, deep generative modelling of order-flow dynamics, and standardised evaluation of generative market models. Together, these components constitute an end-to-end pipeline for constructing, training, and assessing intelligent agents and generative world models in high-frequency financial environments. The thesis is structured around four research papers, each addressing a distinct challenge in this pipeline.

1. **Reinforcement learning for trading and inventory management.** We formulate intraday trading within a LOB as a Markov decision process and demonstrate that deep RL can transform directional alpha signals into effective execution strategies. An agent trained in a replay-based environment substantially outperforms heuristic baselines in both execution quality and inventory control.

2. **JAX-LOB: a hardware-accelerated, parallelisable LOB simulator.** To enable large-scale simulation and RL training, we develop JAX-LOB, a fully parallelisable LOB simulation engine built in the JAX ecosystem. Supporting thousands of concurrent simulations on GPUs, JAX-LOB achieves up to 75× speed-ups over conventional simulators, providing the computational substrate for scalable policy learning and rapid experimentation.

3. **LOB-S5: a deep autoregressive generative model for LOB dynamics.** Moving beyond replay-based simulation, we introduce LOB-S5, the first end-to-end deep autoregressive state-space model for generating realistic LOB message sequences. Using a custom encoder and structured state-space layers, LOB-S5 captures long-range dependencies in order flow and achieves state-of-the-art out-of-sample performance, establishing a foundation for black-box market simulation and RL environment generation.

4. **LOB-Bench: a benchmarking suite for generative LOB models.** To address the absence of standardised evaluation protocols, we propose LOB-Bench, the first comprehensive benchmarking framework for generative LOB

models. LOB-Bench assesses both unconditional and conditional distributional properties — including market spread, queue depth, and price impact response functions — and is released as an open-source Python package to promote reproducibility and community adoption.

The principal contributions of this thesis to the scientific literature are as follows. **(i)** It provides the first systematic demonstration that deep RL, when combined with structured LOB representations and synthetic alpha signals, can learn execution policies that meaningfully outperform rule-based baselines in realistic microstructure settings. **(ii)** It introduces a GPU-native, massively parallel LOB simulation architecture that reduces the computational barrier to RL research in financial microstructure by up to two orders of magnitude. **(iii)** It establishes deep autoregressive state-space modelling as a viable paradigm for generative modelling of high-frequency order flow, achieving state-of-the-art realism in a previously unsolved sequence generation task. **(iv)** It proposes and releases the first standardised, open-source evaluation framework for generative LOB models, enabling rigorous and reproducible comparison across methods. Collectively, these contributions lay the groundwork for future research in multi-agent market simulation, generative world models for finance, and the design of intelligent trading systems.

Statement of Originality

This thesis is submitted to the Department of Engineering Science, The University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.



Peer Nagy, October 2025

Acknowledgement

My path, and this thesis with it, have been shaped by countless people and serendipitous encounters. First and foremost, I am deeply grateful to Jakob Foerster, whose guidance, encouragement, and unwavering belief in the most ambitious ideas have been a constant source of motivation. I also thank Stefan Zohren for his continued supervision, sound advice, and valuable suggestions, as well as Jan-Peter Calliess, whose time, ideas, and discussions were instrumental in setting me on course during the early stages of my DPhil. My gratitude extends as well to Anisoara Calinescu for her support, thoughtful feedback, and unfailing kindness.

I am indebted to the *Oxford-Man-Institute of Quantitative Finance* for providing research funding, resources, and institutional support, and to the *Foerster Lab for AI Research* for embodying the collaborative spirit that has made my work possible. I am also grateful to have been part of the inaugural cohort at *Reuben College*, for the people I met there and the chance to contribute to shaping the college's future.

I am fortunate to have worked alongside talented collaborators and co-authors, especially Sascha Frey and Kang Li, who made the research process not only fruitful but also enjoyable. I am especially grateful to my many friends in Oxford and beyond who brought laughter, perspective, and companionship throughout these years.

I owe my deepest thanks to my parents and my whole family, whose unwavering support has carried me to this point. Above all, I am profoundly grateful to Annette – my partner, companion, and greatest supporter – whose kindness and belief in me have made this journey possible. With you on my team, I have always felt that I had already won, and that anything is possible.

Contents

1 Introduction **1**

1.1 Simulating Limit Order Books 1

1.2 Research Objectives and Contributions 5

1.3 Thesis Outline 8

1.4 Publications and Author Contributions 12

2 Background and Literature Review **14**

2.1 Machine Learning for Microstructure Data-Driven Execution Algorithms 14

2.1.1 Introduction to Order Execution Problems 14

2.1.2 Microstructure Data and Analysis 16

2.1.3 Predictive Machine Learning Signals for Execution Algorithms 20

2.1.4 From Integrating Predictive Signals into Execution Algorithms to Automatically Learning Policies Using Reinforcement Learning 23

2.1.5 Conclusions on Machine Learning for Execution Algorithms . 26

2.2 Model Architectures 28

2.2.1 State-Space Models and Transformers 28

2.2.2 Discretisation of State-Space Models 31

2.2.3 Receptance Weighted Key Value (RWKV) Models 33

3 Limit Order Books and Reinforcement Learning **39**

3.1 Preface and Contextual Overview 39

3.2 Asynchronous Deep Double Duelling Q-Learning for Trading-Signal Execution in Limit Order Book Markets 41

3.2.1 Introduction 42

3.2.2 Related Literature 45

3.2.3 Background 46

3.2.4 Framework 50

3.2.5 Experimental Results 53

3.2.6 Conclusions 60

3.3 Discussion and Integration 63

3.4 Author Contributions 64

4 Simulating Limit Order Books **65**

4.1 Preface and Contextual Overview 65

4.2 JAX-LOB: A GPU-Accelerated Limit Order Book Simulator to Unlock Large Scale Reinforcement Learning for Trading 68

4.2.1 Introduction 69

4.2.2 Related Literature 71

4.2.3 Background 73

4.2.4 The JAX-LOB Simulator 74

4.2.5 `Gymnax` Environments for the LOB 82

4.2.6 Using the `gymnax` Environment for Reinforcement Learning . . 88

4.2.7 Conclusions 91

4.3 Discussion and Integration 92

4.4 Author Contributions 94

5 Generating Limit Order Book Data **95**

5.1 Preface and Contextual Overview 95

5.2 Generative AI for End-to-End Limit Order Book Modelling 97
5.2.1 Introduction 98
5.2.2 Related Literature 100
5.2.3 Background 101
5.2.4 Data 103
5.2.5 Model 107
5.2.6 Results 111
5.2.7 Conclusions 115
5.3 Discussion and Integration 119
5.4 Author Contributions 122

6 Evaluating Generated Limit Order Book Data 124
6.1 Preface and Contextual Overview 124
6.2 LOB-Bench: Benchmarking Generative AI for Finance 127
6.2.1 Introduction 128
6.2.2 Related Literature 132
6.2.3 Background 133
6.2.4 Evaluation Framework 135
6.2.5 LOB-Bench Package 141
6.2.6 Results 143
6.2.7 Conclusions 147
6.3 Discussion and Integration 149
6.4 Author Contributions 151

7 Conclusion 152
7.1 Summary of Contributions 152
7.2 On the Design of RL Algorithms in Finance 154
7.3 “Causal” Sequence Modelling 157
7.4 Societal Impact of Large Financial Models 159

7.5 Future Work 161

References **164**

A Trading in Limit Order Book Markets using RL **187**

B LOB-Bench **189**

B.1 Benchmark Code 189

B.2 LOBS5 Training Details 190

B.3 RWKV Training Details 191

B.4 Sensitivity of Divergence Metrics to the Bin Size 192

B.5 Additional Figures 193

C Statement of Authorship **204**

List of Figures

2.1 **Illustration of an LOB with Five Bid and Ask Levels.** The arrival of a new limit sell order of size L^s at the best ask is added to the book by increasing the depth at that level. 18

2.2 **Simple TWAP Strategy.** This exhibit shows an illustration of an execution schedule with a linear TWAP target trajectory and executed discrete slices as blocks over time around the target trajectory. 23

2.3 **S5 schematics.** (a) The SSM is first diagonalised and then discretised. The latent state $\mathbf{x_{1:L}}$ is then computed using a parallel scan over the layer input $\mathbf{u_{1:L}}$, which in turn is used to compute the layer pre-output $\mathbf{y_{1:L}}$. A non-linear activation function then produces the final layer output $\mathbf{u'_{1:L}}$. (b) The layer computations are illustrated for a single input element $\mathbf{u_k} \in \mathbf{u_{1:L}}$ in matrix form. 31

2.4 **The RWKV block** is split between a time-mixing and a channel-mixing sub-module. The time-mixing component uses a decaying linear attention mechanism, resulting in a time complexity of $\mathcal{O}(BLd^2)$ per layer, for batch size B, sequence length L, and feature dimension d. μ describes the token shift mechanism, producing a linear combination of the current and previous step input. The Figures are taken from Figure 2 in Peng et al. [2023] and Figure 1 in Peng et al. [2024]. 34

3.1 **Example Limit Order Book Snapshot** of Apple Stock (AAPL) on 01 February 2012 at 10 am. Displayed are the 5 best bid (blue) and ask levels (red). Ask sizes are shown as negative values to indicate limit *sell* orders. AAPL is an example of a small-tick stock since the minimum tick size of ¢1 is small compared to the stock price. For a given number of orders, this results in sparser books with more empty price levels. (NASDAQ data from LOBSTER [R. Huang and Polak 2011]) 47

3.2 **A short snapshot of simulation results** (AAPL on 2012-06-14), comparing the RL policy (second panel) with the baseline (first panel). The first two panels plot the best bid, ask, and mid-price, overlaying trading events of buy orders (green) and sell orders (red). Circles mark new unmarketable limit orders entering the book. Crosses mark order executions (trades) and triangles order cancellations. Open orders are connected by lines to either cancellations or trades. Since we are simulating the entire LOB, trading activity can be seen to affect bid and ask prices. The third panel plots the evolution of the inventory position of both strategies, and the last panel the trading profits over the period in USD. 55

3.3 **Account curves**, trading the noisy oracle signal in the test set, comparing the learned RL policies (solid lines) with the baseline trading strategy (dashed). The black line shows the performance of the buy & hold strategy over the same period. Different colours correspond to different signal noise levels. The RL policy is able to improve the trading performance across all signal noise levels. 56

3.4 **Mean return and turnover** of the baseline and RL trading strategies. 57

3.5 **Robustness check:** account curves for second evaluation period 2022-11-14 to 2022-12-31, comparing RL performance to baselines for three noise levels. Across all noise levels, the RL strategies produce a similar level of out-performance during this period than during the original period in 2012. For a high noise signal ($a = 1.1$), neither the baseline nor the RL policies are profitable. In this case, our RL strategy performs similarly to the baseline and does not learn to stop trading completely. This supports the hypothesis that similar micro-structure effects can be utilised in more recent periods, during a different macroeconomic landscape. 59

4.1 **Structure of data** loaded into the *gymnax* environments. In our experiments, each step processes `N_messages` = 100 messages. The number of `N_steps` is variable with a fixed time of 30 minutes but the data is padded with zeros (blue boxes). There are `N_time_windows` per trading day of data. 83

4.2 **The architecture of the RNN-PPO actor and critic networks.** The hidden states of the RNN layer add the necessary recurrence for the network to have memory of previous states. The Boolean termination values ("Dones") are required to reset the memory of the RNN at episode termination. The policy π is a multivariate normal distribution $mvn(\boldsymbol{\mu}, \boldsymbol{\sigma})$ whereby $\boldsymbol{\sigma}$ is a trained network parameter and $\boldsymbol{\mu}$ is the output vector from the final dense network layer. 89

4.3 **Training steps per second** for gymnax environment based on JAX-LOB and our CPU gym environment implementation. The former is more than 75 times faster. 90

5.1 **Schematic of message tokenization** of a limit order deletion message. Each field of the pre-processed message at the top corresponds to a sequence of tokens in the encoded message at the bottom. The grey message part corresponds to the new message arriving at the exchange, while the blue part encodes the referenced message to be deleted. 106

5.2 **Schematic of model architecture.** Book sequences and flattened tokenized message sequences are initially processed separately before being processed by a combined S5 module (using 6 S5 layers, resulting in $\sim 6.3x10^6$ parameters). Sequence outputs from the last S5 layer are averaged along the sequence dimension to produce token logits. . 109

5.3 **Inference Loop.** Tokens are generated autoregressively from model logits until a message is complete (Token Generation Loop). Completed messages are error-corrected and input to the LOB simulator. The updated book state and the generated message are then added to the input sequences to start generation at $t+1$. (Message Generation Loop) 110

5.4 **Generated (red) and realized (blue) mid-price return distributions** show how well the model matches the unconditional distribution's mean (solid lines) and 95%ile intervals (shaded regions). Returns are calculated between the mid-price s messages after, and the mid-price just before the start of generation. (n=1000 test sequences) 113

5.5 **Comparison of order type frequency** between generated (red) and realized data. While relative frequencies produced by the model roughly match magnitudes in the data, there is still a significant mismatch, with both models overestimating the probability of execution messages *in test data*. n=(1000 sequences)×(100 events) from the test set. 114

5.6 **Generated decoded 4-token sequences of Δt** match message data counterparts well. First row: probability plots comparing generated and realized inter-arrival times. Second row: histograms of generated (red) and realized (blue) inter-arrival times on a log-x scale. The dashed vertical lines plot the mean and the dotted lines the median of the distributions. 114

5.7 **Per-token sequence perplexity:** the GOOG model exhibits better data fit on average and fewer extreme values. Both distributions appear benign without much mass in the right tail. An observation corresponds to the PPL of a single realized 100-message sequence from the test set. Dotted lines indicate medians and dashed lines the means. 116

5.8 ***Top:* Pearson correlation coefficient** ρ between generated and realized returns, indicating directional forecasting performance. The dashed line indicates a 0 correlation. ***Bottom:* corresponding p-values** of t-tests testing $H_1 : \rho > 0$ against $H_0 : \rho = 0$. The dotted line plots the 5% significance threshold. The GOOG model stays below or around 5% significance for up to 80 messages into the future, while correlation for INTC remains significant for at least 100 messages. 117

6.1 **Schematic of LOB-Bench methodology** for conditional distributional evaluation 128

6.2 **Schematic of the LOB.** Not immediately executable orders are placed in first-in, first-out (FIFO) queues at the specified limit price. Sell limit orders are placed on the Ask side (red) and Buy limit orders on the Bid side (blue). 129

6.3 **Model comparison spider plot:** the *LOBS5* model beats the *baseline* and *coletta* model on almost all scores. Note: the radial axis is inverted by plotting the negative loss (larger is better). 144

6.4 **Model score summaries** (lower is better). The *LOBS5* model achieves the lowest overall scores. *Coletta* beats the *baseline* on the Wasserstein metric, but not for L_1. Error bars are bootstrapped 99% CIs. 144

6.5 ***LOBS5* results** – (left): histogram matching of unconditional score distributions for real and generated data. (right): error accumulation – the further out the prediction horizon, the worse is the model performance – an important model characteristic to measure. 145

6.6 **L_1 distance** between real and generated data histograms (incl. 99% CIs). *baseline* performs well on LOB depth and level-related scores, and much worse on time and volume metrics. *LOBS5* dominates L1 loss for GOOG. 145

6.7 Comparison of tick-normalized **mid-price impact response functions** for different event types between real and generated datasets. Shaded regions are 99% confidence intervals. Compared are the *LOBS5* and stochastic *baseline* models. In contrast to the *baseline*, *LOBS5* reproduces most features of the expected impact response functions. 146

6.8 Comparison of **F1-scores for the MLP classification models** trained on historical (real) data only (solid lines), and trained on both real data and data generated (dashed lines) for a subset of the generated models. The error bars depict the 95% confidence intervals bootstrapped from five different training seeds. 147

A.1 **Confusion matrices** of the artificial oracle signal for three noise levels, from low to high noise. 188

B.1 **Test set loss curves (2023 data)** for the LOBS5 model, measuring the mean *per-message* negative log-likelihood for INTC (green) and GOOG (red) throughout 100 training epochs. Message cross-entropy after 100 epochs is 19.14 for INTC and 17.80 for GOOG. 190

B.2 **Training loss curves for RWKV model training.** The y-axis represents the average negative log likelihood of the messages being trained on, calculated as the sum of negative log likelihoods of each token in the message. The bold lines represent the exponential moving average of the true curve (presented with lower opacity) with an α of 0.1 to better highlight the trend. 191

B.3 **L1 Divergence Scores with half and double regular bin size.** Large bin size deviations (halfing or doubling) systematically affect the level of scores but not their ranking. 193

B.4 **L1 and Wasserstein-1 errors** of generated unconditional distributions for easy comparison between Alphabet (GOOG) and Intel (INTC). Error bars show 99% bootstrapped confidence intervals. . . . 194

B.5 ***LOBS5* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. Overall, the generative *LOBS5* model evaluated here, adapted from Nagy et al. [2023b], does a good job in matching data along various dimensions. Bigger errors in matching distributions are visible in e.g. spread (GOOG), orderbook imbalance (INTC) and time to cancel (GOOG and INTC). 195

B.6 ***baseline* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The [Rama Cont et al. 2010b] model does a decent job matching some of the scores, particularly discrete ones, such as depths and levels. Clear shortcomings are visible in scores such as orderbook imbalance or volumes. 196

B.7 ***rwkv4* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The model produces volatile data with larger spreads, missing correct order levels, leading to difficulty matching book volumes. 197

B.8 ***rwkv6* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The model has similar shortcomings to RWKV 4 (wrong price levels, mismatched book volumes etc.) due to tokenization of raw data and missing order book information. 198
B.9 ***coletta* - GOOG - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. 199
B.10 **L1 error divergence:** comparing the L1 errors of score distributions of real data with generated data distributions at a specific horizon into the future shows accumulating model errors. This is explainable due to snowballing errors caused by teacher forcing (conditional next token loss). A good model should be able to control errors for sequence lengths as long as possible. To provide a significance threshold over pure sampling noise, the dotted lines plot the 99. percentile of L1 error between bootstrapped samples of only real data. 200
B.11 **Histograms of conditional score distributions** for real (blue) and generated (orange) data for the Alphabet stock (GOOG). Weights w, expressing the share of data in the bin, measure the impact of the specific conditional distribution (row) on the total metric loss. 201
B.12 **Histograms of total book volume conditional on bid-ask spread for Alphabet stock (GOOG)**. Weights w, expressing the share of data in the bin, measure the impact of the specific conditional distribution (row) on the total metric loss. 202
B.13 **LOBS5 - ROC curve of the discriminator on test data (GOOG).** The discriminator represents a worst-case adversarial score function by learning to effectively differentiate between real and generated sequences of LOB states. 202
B.14 **LOBS5 - Histogram of logit scores for real and generated sequences** on held-out test data (GOOG). Matching this distribution well would indicate high model quality, as even a trained discriminator network would not be able to differentiate the distributions. 202
B.15 **Comparison of impact response functions** for different event types between real and generated data-sets, tick-normalized mid-price response. Shaded regions are 99% confidence intervals. There is a comparison between two select models: the LOBS5 and the stochastic baseline. We see that, in contrast to the baseline, the generative model is able to reproduce much more of the expected impact function, though not as well as for GOOG. 203

List of Tables

3.1 **Actions taken by RL policy** for the three different noise levels: the first row shows how often the policy chooses the "skip" action. Not choosing this action does however not necessarily result in an order being placed, e.g. if inventory constraints are binding. The last two rows show the relative proportion of limit order placement levels for sell orders, and buy orders, respectively. 58

4.1 **Average run-time** for the three basic operations that may be applied to the LOB for different sizes N. The tests are conducted on an Nvidia 2080 Ti and averaged across 1000 serial trials and 10 experiments. The book is initially filled to one-third capacity with random orders. The standard error over 10 experiments is given. 78

4.2 **Analysis of the effect of incoming order size on the time required to complete the matching operation.** As the size increases, more standing orders need to be considered to fully match the incoming order. The capacity for the book is set to $N = 100$. Testing is done on an Nvidia 2080 Ti GPU. The mean and standard error across 10 experiments is given. 79

4.3 **Time to process different message types.** The last column differs from the second column in that the price crosses the spread, requiring a matching operation. Tests are done on an Nvidia 2080 Ti GPU, the book capacity is $N = 100$. The mean and standard error across 10 experiments is given. 80

4.4 **Time to process different message types with `vmap`.** The difference to table 4.3 is that messages are processed by $N_Book = 1000$ books in parallel. Times measure the processing of a message in all books. The mean and standard error across 10 experiments is given. . 81

4.5 **Processing times** for messages on the CPU order book implementation based on *linked list* and *RB trees*, and the implementation used in RL4MM [Jerome et al. 2022a] based on *numpy* arrays, and for the JAX-LOB implementation based on *jax arrays* with capacity $N = 100$ and $N_Books = 1000$ books in parallel. A key difference between the CPU implementation and that in RL4MM is that data is pre-loaded, as opposed to being accessed from a database. Both CPU implementations are tested on a single core of an Apple M1 chip and the JAX-LOB on an Nvidia 2080 Ti. 82

4.6 **Average time per step** comparison between the RL4MM environment, our CPU-based *gym* environment, and the JAX-LOB based *gymnax* environment for a varying number of environments in parallel. The CPU environments are run on an Apple M1 processor and the *gymnax* environment is run on an Nvidia 2080 Ti. 88

5.1 **Token Perplexity Scores by Message Position** 116

6.1 Scoring functions computed as part of the standardised LOB-Bench suite. 142

A.1 **Selected RL parameters** for APEX algorithm using RLlib [Liang et al. 2018] library for training. 187

Chapter 1

Introduction

1.1 Simulating Limit Order Books

Limit order books (LOBs) are a central market mechanism used by financial exchanges to trade a wide range of assets [Roşu 2009; Luckock 2001; G.-F. Gu et al. 2008], including stocks and derivatives [Gould et al. 2013], ETFs [Deville 2008], electricity [Martin and Otterson 2018], and even cryptocurrencies [Schnaubelt et al. 2019]. They offer a rich source of high-frequency data that captures supply and demand dynamics, as well as price formation processes. The increasing availability of large-scale LOB data sets has spurred the development of novel empirical methods and modelling approaches [R. Huang and Polak 2011]. A solid understanding of LOB dynamics is critical for effective risk management and accurate performance estimation of trading strategies [J. A. Sirignano 2019]. Beyond their commercial relevance, LOB models are also vital for regulatory applications, such as detecting collusion and other illicit practices, and for safeguarding market stability [Cartea et al. 2022; Cartea et al. 2025].

LOB data accumulates billions of individual messages for highly traded securities, such as liquid stocks, over years of trading [R. Huang and Polak 2011]. Since the LOB state at any point in time is a high-dimensional object, which is updated with each new message, this type of data is well-suited for machine learning

(ML) models. However, financial data also presents several challenges for ML. The LOB environment is highly stochastic, driven by the interactions of many self-interested agents [Cristelli 2013]. Moreover, predictability is often low due to arbitrage mechanisms, which remove tradeable inefficiencies from the market. These factors complicate ML applications but also offer a demanding test bed for developing new methods with potential impact beyond finance.

A realistic model of LOB dynamics – down to the level of individual agents or agent types – would enable the aggregation of micro-level activity into a generative process that matches the real LOB data distribution. Such a model represents the holy grail of LOB research, as it would support a wide range of applications, including accurate trading strategy backtesting, distributional simulation, market impact optimisation, and the automation of high-frequency trading (HFT) strategy development. It would also allow for automated LOB trading via reinforcement learning (RL), where the model serves as a simulation environment, similar to how RL has solved games in recent years [Silver et al. 2017]. The motivation of this thesis is to advance this long-term agenda by addressing key open problems toward building an accurate, realistic, and ultimately multi-agentic LOB simulator.

LOB data captures a complex ecosystem, characterised by a wide range of statistical properties spanning multiple time scales. In the philosophy of science, the ontological and epistemological roles of scientific models have long been debated [Frigg and Hartmann 2025]. Models are typically designed for specific purposes, as they cannot capture every detail of the system they represent. In the context of LOBs, some models may approximate the probability of certain events, while others may estimate future mid-price movements. However, no practical model can predict the precise reason why a particular message is submitted to an exchange.

Nor should LOB models be expected to reproduce the exact future order flow, as forecasting chaotic systems is inherently impossible over extended time horizons [Lorenz 1996]. The value of a model lies in abstracting away from such details and focusing on the system's relevant properties, relative to its intended

use. We adopt an argument analogous to one used in the context of large language models (LLMs), tracing back to the Turing test [Turing 2009], where the goal is indistinguishability between generated and real data. For LOBs, this suggests a formal benchmark: model performance should be measured by how closely the distribution of generated data matches that of real data. As models advance, this benchmark must also evolve – demanding fidelity across increasingly complex distributional characteristics.

Current research identifies a wide range of so-called "stylised facts" of LOB data [Rama Cont 2011; Vyetrenko et al. 2020]. These include characteristic autocorrelation structures of mid-price returns – most notably, negative autocorrelation at the first lag – strong autocorrelation in traded volumes, and pronounced intra-day volume profiles, with higher activity at market open and close. Stylised facts also encompass typical volume shapes across the order book for different stock types, as well as signature plots, which relate volatility to the sampling frequency.

Beyond ML – and more recently, specifically deep learning – there exists a broad literature on statistical methods for modelling LOBs. For a comprehensive review of these approaches, see Jain et al. [2024]. Popular modelling paradigms typically fall into four categories: point process models, often based on Poisson or Hawkes processes; agent-based models (ABMs); stochastic differential equation models; and deep learning models[1].

Zero-intelligence point process models based on Poisson processes, such as those proposed by Bouchaud et al. [2002], E. Smith et al. [2003], Luckock [2003], Rama Cont et al. [2010b], Gao and S. Deng [2018], and Kelly and Yudovina [2018], are mathematically simple and elegant. However, they suffer from a poor fit to empirical autocorrelations due to their assumption of order independence. Slightly more complex extensions allow for variable order intensities [Hult and Kiessling 2010; W. Huang et al. 2015; Xiaofei Lu and Abergel 2018], offering modest improvements in realism.

[1]Deep learning models are also discussed in more detail in Chapter 5

Another class of statistical models for LOBs uses Hawkes processes, which are self-excitatory, meaning that event arrival probabilities increase with the number of prior events. For a comprehensive overview, see Bacry et al. [2015]. Hawkes models enhance the modelling of market impact and order clustering, although they still struggle to capture the full range of context-conditional distributions observed in real markets.

Stochastic differential equation models provide another avenue for modelling, relying on continuous approximations of LOB state changes. Favoured for their analytical tractability, these models typically focus on long-term dynamics and stability analysis rather than short-term order-level behaviour.

Agent-based models (ABMs) offer an alternative approach inspired by statistical physics and popularised in the social sciences through the econophysics literature. These models simulate and aggregate the behaviour of individual agents, such as traders submitting orders to a LOB, to generate entire LOB histories. For surveys of ABM approaches to LOB simulation, see Chakraborti et al. [2011] and Chapter 5 of [Abergel et al. 2016]. Notable examples of LOB-focused ABMs include ABIDES [Byrd et al. 2020] and the C++ LOB simulator by Belcak et al. [2022].

A key challenge for realism in this modelling class is the anonymity of LOB messages, which prevents identifying individual market participants. Additionally, defining plausible and effective trading policies for individual agents remains non-trivial. Nonetheless, recent work suggests that clusters of distinct agent types can be inferred from LOB data [Rama Cont et al. 2023a], which may guide future research in this space.

Common criticisms of ABMs include their high computational cost, limited analytical tractability, and the difficulty of calibrating models to real data. Despite these challenges, ABMs remain a promising avenue for simulating rich market dynamics in a bottom-up fashion.

Currently, perhaps the most promising approach for developing highly realistic LOB models lies in generative deep learning techniques that produce synthetic

LOB data. Most existing models in this space function as black-box approximators of aggregate order flow, employing architectures such as recurrent neural networks (RNNs) [Rumelhart and McClelland 1987; Hultin et al. 2023], generative adversarial networks (GANs) [Coletta et al. 2021; Coletta et al. 2022], or even diffusion models [Backhouse et al. 2025]. For a comparative survey of such machine learning approaches, see Briola et al. [2020].

A key limitation of these aggregate, black-box models is their lack of structural assumptions about individual agents and their decision-making processes. As a result, they may struggle to simulate counterfactual or out-of-distribution scenarios where market conditions change and agent incentives lead to changes in behaviour. Therefore, a long-term objective of this research agenda should be to integrate the strengths of both ABMs and ML-based generative models. This could be realised through multi-agent models of LOBs, where each agent acts according to an individually rational policy defined by its reward function.

Recent advances in multi-agent reinforcement learning (MARL) [Rutherford et al. 2024] and inverse reinforcement learning (IRL) [Song et al. 2018; L. Yu et al. 2019] offer promising foundations for such a synthesis. By combining behavioural realism with scalable learning methods, these tools could pave the way for a new generation of LOB simulators that are both interpretable and generative.

1.2 Research Objectives and Contributions

This thesis investigates machine learning (ML) problems in the context of limit order books (LOBs), which constitute one of the most widely used market mechanisms at financial exchanges worldwide [Gould et al. 2013]. LOBs serve as an efficient means of aggregating and matching incoming limit orders – buy or sell requests submitted at specific prices. Due to their central role in facilitating price discovery and executing trades, LOBs represent a highly consequential domain for both theoretical and applied research.

Applying machine learning to financial data presents a range of challenges, including asymmetric information, partial observability, a large number of interacting agents with heterogeneous incentives, and a high degree of aleatoric uncertainty, i.e. randomness inherent to the system itself. These and other factors have contributed to the comparatively slower pace of ML progress in finance relative to other domains. Nonetheless, the abundance of high-frequency data available on financial market activity, combined with recent advances in areas such as large language models (LLMs) [Achiam et al. 2023], reinforcement learning (RL) [Volodymyr Mnih et al. 2013; Vinyals et al. 2019; Silver et al. 2017; Schulman et al. 2017; Jumper et al. 2021; Degrave et al. 2022; Fawzi et al. 2022], and multi-agent learning [Tampuu et al. 2017], make this an increasingly promising and important area of research.

A well-calibrated generative LOB simulator that accurately reproduces the statistical properties of real-world LOB data is a crucial tool. It enables the training and evaluation of optimal execution, market making, and trading strategies in a controlled setting, with the potential for strong transfer performance to real-world markets.

A central open problem in the study of limit order books (LOBs) is the development of robust solutions to sequential decision-making tasks under realistic LOB dynamics. This thesis aims to make progress toward this goal by addressing several key challenges at the intersection of machine learning and financial market simulation.

First, it demonstrates that reinforcement learning (RL) can be effectively applied to derive optimal trading strategies from noisy, synthetic signals of future returns in a static LOB environment. This establishes a proof of concept for using RL beyond classical execution problems, bridging the gap between signal generation and position management.

Second, to support scalable RL research and experimentation, the thesis introduces JAX-LOB, a high-performance, hardware-accelerated LOB simulator built

using the JAX programming ecosystem [Bradbury et al. 2018]. JAX-LOB achieves speed-ups of up to 75x on parallelisable tasks compared to widely used alternatives [S. Y. Frey et al. 2023], unlocking new opportunities for computationally intensive simulation and learning workflows. As a practical application and stress test for the developed tools, the thesis applies RL in conjunction with the JAX-LOB simulator to address the optimal trade execution problem. This serves as a concrete demonstration of how the proposed methods can be used to design and evaluate algorithmic trading strategies in highly granular and realistic settings.

Third, to enhance the realism of simulated market environments, the thesis develops a state-of-the-art autoregressive generative model of LOB data [Nagy et al. 2023b]. This model captures the sequential and dynamic nature of market activity, paving the way for more realistic ML-driven agent-based simulations and data augmentation approaches.

Fourth, recognising the difficulty of objectively assessing the quality of synthetic LOB data, this work also addresses a major gap in the field by introducing a systematic evaluation methodology. This includes the development of a comprehensive, open-source benchmarking toolkit, LOB-Bench[2], to support reproducible research and fair comparison across models.

RL has proven to be a powerful paradigm for solving sequential decision-making problems across a wide range of domains—from gameplay [Volodymyr Mnih et al. 2013; Vinyals et al. 2019; Silver et al. 2017] to the natural sciences [Jumper et al. 2021; Degrave et al. 2022] and even in optimizing computing itself [Fawzi et al. 2022]. Financial decision-making tasks, such as optimal trade execution or portfolio management, are inherently sequential and path-dependent, often exhibiting strong feedback loops. For instance, submitting an order to a financial exchange not only impacts the market directly through the mechanics of the order-matching process (*direct* market impact) but also influences other agents' behaviours and future order flow (*indirect* market impact).

[2] https://lobbench.github.io/

These dynamics render static, one-shot prediction methods, such as those typical in supervised learning, insufficient for learning optimal strategies. Instead, approaches that explicitly model environment interactions are required. Generative models, such as those developed in this thesis, offer a promising solution by capturing the temporal and interactive nature of market behaviour. When used as simulators, they enable the training of RL agents in a controlled yet realistic environment.

The work presented in this thesis thus contributes foundational tools for the development of agentic AI in financial contexts, where learning and acting must occur under complex, dynamic, and strategic conditions.

1.3 Thesis Outline

This thesis is structured as an integrated thesis around 4 key papers presented in Chapters 3–6. Each of these chapters starts with a section titled "Preface and Contextual Overview" to guide the reader into the topics discussed in the respective papers. The main papers follow, and each main chapter concludes with "Author Contributions" and "Discussion and Integration" sections, which respectively discuss the author contributions to the work, and put the papers into context by also linking them to other sections of the thesis.

Chapter 2 starts by providing additional background which is not directly discussed in the respective papers. A practical application of the methods developed in this thesis is the improvement of trade execution algorithms in limit order books (LOBs) through fast, parallelisable LOB simulation, including the use of generative models. Given the significance of this area of application and the novel tools introduced in this thesis, Section 2.1 presents an overview of machine learning approaches to trade execution. Section 2.2 then treats the model architectures employed in the work later presented.

In the first separate paper of this thesis (Chapter 3) [Nagy et al. 2023a], we train

an agent to translate high-frequency trading signals into concrete individual limit orders, maximising trading return. Using the ABIDES [Byrd et al. 2020] limit order book simulator as a foundation, we build a reinforcement learning environment with the OpenAI gym interface [Brockman et al. 2016] to mimic a realistic NASDAQ equities trading setting, based on historic order book records. To maximise trading returns, the agent is trained using *Deep Duelling Double Q-learning* [Volodymyr Mnih et al. 2013; Hasselt 2010] and the *APEX* (*Asynchronous Prioritised Experience Replay*) [Horgan et al. 2018] architecture. The agent's decision-making process relies on the current state of the limit order book, its recent history, and a short-term directional forecast. To evaluate the performance of RL in adaptive trading, independently from any specific forecasting algorithm, we explore our approach using synthetic alpha signals. These signals are generated by incorporating varying levels of noise into forward-looking return signals. The findings indicate that the RL agent is capable of learning an efficient trading strategy for inventory control and order placement, subsequently outpacing a heuristic benchmark strategy even when both had access to identical signals.

To overcome computational constraints and to facilitate further progress in RL for acting in LOB environments, we develop the Jax-LOB simulator [S. Y. Frey et al. 2023] (Chapter 4). Jax-LOB, which is the first realistic GPU-accelerated LOB simulator capable of processing thousands of limit order books concurrently. It parallelises the simulation of multiple scenarios of either identical or distinct securities, thereby offering up to a 75x boost in per-message processing speed. Our Jax-LOB implementation hinges on strategic design choices that maximally utilise Jax's [Bradbury et al. 2018] capabilities without sacrificing LOB-related mechanism realism. Moreover, Jax-LOB seamlessly integrates with other Jax packages [Lange 2023; Rutherford et al. 2024]. In the paper we also illustrate how the simulator can be employed to solve optimal execution problems in a LOB environment using reinforcement learning. We also provide preliminary results demonstrating the advantages of end-to-end RL training on GPUs.

For RL to be useful for real-world applications, accurate simulation dynamics are

essential. Although we address part of the simulation task with the Jax-LOB simulator [S. Y. Frey et al. 2023], this alone cannot account for the entire causal and counterfactual market behaviour, as the simulator replays historical data, which are only changed by active injections into the order flow. The strategic effects, which actions have on the successive actions of other market participants, are therefore not considered. To address this shortcoming, we develop a generative model [Nagy et al. 2023b] for LOB message data, which acts as a probabilistic, conditional black-box market simulator, generating new message flow, conditioned on previous flow and LOB states.

Creating a precise generative model of realistic financial market order flow is a complex and yet-to-be-resolved task with numerous potential benefits. In our study [Nagy et al. 2023b], we introduce a novel end-to-end autoregressive generative model capable of producing tokenised limit order book (LOB) messages (Chapter 5). The JAX-LOB simulator interprets these messages to update the LOB state. For efficient handling of long sequences of up to tens of thousands of tokens, our model architecture utilises simplified structured state-space layers [J. T. Smith et al. 2022] to manage both LOB state and tokenised message sequences. Using LOBSTER [R. Huang and Polak 2011] data of NASDAQ equities, we have designed a custom, specialised tokeniser to convert sequences of digits into tokens, analogous to the tokenisation process in large language models.

Out-of-sample analysis of generated data reveals a promising performance in approximating data distributions, as shown by the model's low perplexity [S. F. Chen et al. 1998]. Notably, there is a significant correlation between the mid-price returns derived from the generated order flow and the actual data, indicating surprisingly good conditional forecasting performance. The model's accuracy combined with the granularity of the generated data, however, open new avenues for applications beyond forecasting, such as serving as a world model in high-frequency financial reinforcement learning applications.

A persistent challenge in developing generative LOB models lies in the robust

evaluation and comparison of different modelling approaches. Existing models in the literature often report ad-hoc performance metrics or stylised facts, limiting comparability and hindering progress in the field. To address this, we introduce *LOB-Bench* [Nagy et al. 2025], a distributional evaluation framework for generated LOB data (Chapter 6). LOB-Bench computes unconditional and conditional distributions of LOB statistics and quantifies differences in distribution between generated and real data. Furthermore, it also computes price impact response functions by evaluating correlations of specific message events, thereby providing another measure of model quality.

Building on the theoretical evaluation framework, we provide an open-source Python benchmark package that computes quantifiable statistics for model comparison. Using this framework, we evaluate the autoregressive model introduced in Chapter 5 and compare its performance with three alternatives: a recent conditional generative adversarial network (CGAN) [Coletta et al. 2022], a statistical point-process baseline model [Rama Cont et al. 2010a], and a new autoregressive model based on the *Receptance Weighted Key Value* (*RWKV*) large language model (LLM) architecture [Peng et al. 2023; Peng et al. 2024]. With the RWKV model, we assess whether off-the-shelf byte-pair tokenisation [Sennrich et al. 2016], applied without data preprocessing and combined with an LLM architecture, can match the performance of our bespoke generative model. Our findings show that this is not the case: the RWKV-based models fall short of the performance achieved by our considerably smaller, domain-specific architecture.

Chapter 7 concludes the thesis by summarising the contributions of each chapter and discussing their broader implications and directions for future research. Appendix C provides authorship statements for the included publications, detailing the specific contributions of each author.

1.4 Publications and Author Contributions

The following list enumerates the thesis sections, which have been published in peer-reviewed venues and the corresponding author contributions.

- **Section 2.1**

  Peer Nagy, James Powrie, and Stefan Zohren [2023c]. "Machine learning for microstructure data-driven execution algorithms". In: *Handbook of Artificial Intelligence and Big Data Applications in Investments*, pp. 100–106

  Peer Nagy wrote the manuscript with input from all authors. All authors devised the conceptual ideas, contributed to the article and approved the submitted version.

- **Section 3.2**

  Peer Nagy, Jan-Peter Calliess, and Stefan Zohren [2023a]. "Asynchronous Deep Double Dueling Q-learning for trading-signal execution in limit order book markets". In: *Frontiers in Artificial Intelligence* 6

  Peer Nagy performed the experiments and wrote the manuscript with input from all authors. All authors devised the project, the conceptual ideas, and the design of the experiments. All authors contributed to the article and approved the submitted version.

- **Section 4.2**

  Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu [2023]. "JAX-LOB: A GPU-Accelerated limit order book simulator to unlock large scale reinforcement learning for trading". In: *Proceedings of the Fourth ACM International Conference on AI in Finance*, pp. 583–591

  Sascha Frey, Kang Li, and Peer Nagy contributed equally to this work. All three contributed to the conception, implementation, and writing of the paper with input from all authors. Silvia Sapora and Chris Lu contributed to

the implementation, provided feedback, and assisted with writing and editing. Stefan Zohren, Jakob Foerster, and Anisoara Calinescu offered guidance and supported project design and manuscript revision.

- **Section 5.2**

  Peer Nagy, Sascha Frey, Silvia Sapora, Kang Li, Anisoara Calinescu, Stefan Zohren, and Jakob Foerster [2023b]. "Generative AI for End-to-End Limit Order Book Modelling: A Token-Level Autoregressive Generative Model of Message Flow Using a Deep State Space Network". In: *Proceedings of the Fourth ACM International Conference on AI in Finance*

  Peer Nagy conceptualised the project and led its implementation and manuscript preparation, with input from all authors. Sascha Frey, Silvia Sapora, and Kang Li contributed to implementation, provided feedback, and assisted with writing and editing. Anisoara Calinescu, Stefan Zohren, and Jakob Foerster offered guidance and supported project design and manuscript revision.

- **Section 6.2**

  Peer Nagy, Sascha Frey, Kang Li, Bidipta Sarkar, Svitlana Vyetrenko, Stefan Zohren, Anisoara Calinescu, and Jakob Foerster [2025]. "LOB-Bench: Benchmarking Generative AI for Finance - an Application to Limit Order Book Data". In: *Proceedings of the 42nd International Conference on Machine Learning*. Vol. 267. PMLR. Vancouver, Canada.

  Peer Nagy and Sascha Frey contributed equally to this work. Both worked on the conceptualisation and implementation with input from all authors. Kang Li and Bidipta Sarkar contributed to parts of the project implementation and manuscript preparation. Svitlana Vyetrenko, Stefan Zohren, Anisoara Calinescu, and Jakob Foerster offered guidance and supported project design and manuscript revision.

Chapter 2

Background and Literature Review

An adaptation of the following section has been published as *Machine learning for microstructure data-driven execution algorithms* in *Handbook of Artificial Intelligence and Big Data Applications in Investments*, 2023 [Nagy et al. 2023c].

2.1 Machine Learning for Microstructure Data-Driven Execution Algorithms

2.1.1 Introduction to Order Execution Problems

A central task in implementing any trading strategy is executing the trades necessary to achieve a target portfolio position. For funds with larger assets under management, this often requires executing larger trades – and the larger a trade, the more impact it tends to have on market prices. This impact is typically measured as slippage – the difference between a reference price observed before execution begins and the actual execution prices. To minimise this slippage, which can significantly degrade performance over time, machine learning (ML) methods can be employed to enhance execution algorithms in various ways.

An execution problem is typically framed as follows. The execution algorithm receives a block of shares to buy or sell within a specified time frame – usually ranging from seconds to hours. To minimise the adverse impact on market prices, the large order is divided into smaller slices that are executed over the available time horizon. The algorithm's task is to determine an execution schedule that minimises the resulting slippage cost. If, instead, the entire order were submitted as a market order at the time the instruction is received – without temporal distribution – a large buy order could drive prices upward, a sell order could push them downward, or available market liquidity might be insufficient to fill the order fully, resulting in worse slippage or incomplete execution.

Every execution problem involves a trade-off between market impact and the risk of adverse price movements over the execution horizon. In a risk-neutral setting, with no information about future price direction or trading activity, the optimal strategy is to divide the total order into equally sized slices executed at regular intervals throughout the time window. This approach, known as TWAP (time-weighted average price) execution, serves as a standard benchmark that more sophisticated algorithms aim to outperform. Under these assumptions, TWAP can be formally shown to be optimal using methods from stochastic optimal control, making it a theoretically justified baseline [Almgren and Chriss 2001].

However, it is intuitively clear that execution performance can be improved with accurate short-term price forecasts. Another widely used execution strategy leverages information about trading volume or market turnover, as higher volume permits larger trades with reduced price impact. This approach aims to execute a block of shares at the volume-weighted average price (VWAP) by allocating trade sizes over time in proportion to the expected volume profile. Although expected volume is a forward-looking quantity, it exhibits strong intraday patterns and can be predicted with reasonable accuracy using machine learning models. The mathematical finance literature further demonstrates that VWAP is the optimal execution strategy under risk neutrality and permanent market impact, when volume information is available [Kato 2015; Cartea and Jaimungal 2016a]. In practice,

execution performance can be improved relative to TWAP by scaling trade sizes in line with expected volume. Estimating volume can range from simple heuristics based on historical intraday profiles to complex deep learning models incorporating a wide array of market features.

One role of machine learning (ML) in execution is to generate forecasts of short-term price movements and expected trading volume, which the execution algorithm can use to front- or back-load the execution schedule – effectively accelerating or decelerating trading activity in response to market conditions. For instance, if a model predicts that prices will decline over the next few seconds, the algorithm may choose to execute a sell trade immediately rather than delay and incur a worse price. This task constitutes a supervised learning problem – specifically, a regression task – and has been tackled using approaches ranging from traditional statistical models to deep learning techniques. The viability of deep learning in this context is largely due to the availability of high-frequency data, which is crucial for training such models effectively.

Predicting short-term price movements is just one of many tasks to which machine learning (ML) can be applied in the context of execution. ML can also be used to forecast other key variables that inform execution decisions, such as bid-ask spreads, future market liquidity, and return volatility. Beyond forecasting, reinforcement learning algorithms can be employed to directly select execution actions – including the sizing, timing, and pricing of individual order slices – effectively learning execution strategies that optimise performance through interaction with a simulated or real environment.

2.1.2 Microstructure Data and Analysis

Most electronic exchanges involved in the trading of cash equities, futures, or options use limit order books (LOBs) to match buy and sell orders using a price–time priority matching mechanism. In this prioritisation scheme, orders are first matched by their price levels, with orders at the same price on a first-come, first-served ba-

sis. Every limit order to buy with a price lower than any sell order in the LOB is added to the bid side of the LOB, where levels are ordered from best (highest price and earliest arrival) to worst (lowest price and latest arrival). Similarly, if an order to sell is posted at a price higher than any order to buy in the LOB, it is added to the ask side of the LOB. Together, the bid and ask levels of the LOB constitute the visible supply and demand for any instrument and any moment in time (see Figure 2.1). The bid level with the highest price is called the best bid, and the ask level with the lowest price is called the best ask. The difference in price between the best ask and best bid is called the spread. If a market order is placed to buy (sell), it is first executed against the best price level of the ask (bid) side, then executed in order of arrival time of the corresponding limit order, and finally executed against the next level if the order is larger than the number of shares available at the best price. Rather than market orders, practitioners often use limit orders that are targeted against the best bid or ask. A refinement of this tactic is to use immediate-or-cancel (IOC) orders, which are automatically cancelled if they cannot be filled at the limit price.

Given this order matching mechanism, LOBs are often described as double-sided continuous auctions, since a continuous order flow changes the book dynamically over time. Limit orders that cannot be matched immediately at the time of arrival, because either the bid is too low for a buy order or the ask is too high for a sell order, enter the book and provide liquidity to the market. Market orders, IOCs, or executable limit orders – buy (sell) limit orders with prices higher (lower) than the best ask (bid) – however, take away liquidity from the market, do not enter the LOB, and are executed immediately. In the case of executable limit orders, one can also encounter a situation where part of the limit order is executed, which thus removes liquidity, while the remainder stays on the book and provides liquidity.

To train ML models for execution tasks, LOB data are required in some form. These data can be represented at different levels of granularity. For most tasks in this space, a univariate price time series is insufficient, and at the least, a dataset with best bid and ask prices and volumes over time is required to learn useful

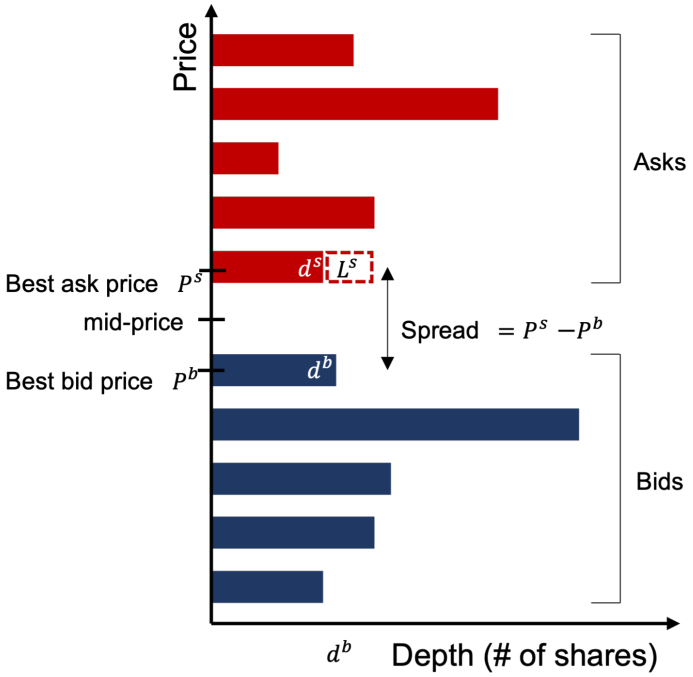


Figure 2.1: **Illustration of an LOB with Five Bid and Ask Levels.** The arrival of a new limit sell order of size L^s at the best ask is added to the book by increasing the depth at that level.

relationships. Some ML models even train on data including multiple or all levels of the LOB to forecast variables of interest. Such datasets can be represented either as a stream of individual orders – so-called market-by-order (MBO) data – or as a sequence of snapshots of the state of the LOB over time. MBO data contain different order types (limit, market, cancel, and modify), which allow a dynamic reconstruction of the LOB in full fidelity. For modelling purposes, however, time series of features of the LOB state are usually more amenable to be used as model inputs.

As mentioned in the previous section, supervised ML techniques can be fruitfully applied to several prediction targets in the execution domain. Models that can predict future values of these variables over even part of the execution horizon can help reduce slippage. One such target is the market spread, because lower spreads imply more favourable execution prices for marketable orders crossing the spread. Similarly, high trading volume makes it more likely that a passively placed limit

order will be executed at each time step. Conversely, forecasting return volatility can be useful for gauging the risk of large price moves over the execution horizon. In periods of high volatility, for example, it might pay to front-load the execution schedule to limit exposure to the downside risk of adverse price shocks.

Over the short term, one of the best indicators of immediate price moves in the LOB is the order flow imbalance [Rama Cont et al. 2014]. The definition of the order flow imbalance (OFI) is the order flow on the buy side: incoming limit buy orders at the best bid, L^b, net of order cancellations, C^b, and market orders, M^s, minus the opposing sell-side flow, within a period of time:

$$OFI = (L^b - C^b - M^s) - (L^s - C^s - M^b) \tag{2.1}$$

This measure captures an imbalance between demand and supply in the market, which is an essential determinant of the market price. A high positive order flow imbalance indicates excessive buy pressure at current prices, thereby making it more likely that prices will rise imminently. Rama Cont et al. [2014] describe an empirical linear relationship between OFI and price changes.

A related LOB-derived measure of supply–demand imbalance, which can be used as a predictive signal of short-term price changes, is the order book imbalance (OBI):

$$OBI = \frac{d^b - d^s}{d^b + d^s} \tag{2.2}$$

Here, d^b and d^s are the depths of the best bid and best ask, respectively (see Figure 2.1). The OBI calculates the normalised difference between the depth (number of available shares) on the buy side at the best bid, d^b, and the number of shares posted on the sell side at the best ask, d^s. This measure is limited between -1 and 1, ranging from a strong downward price pressure to a strong upward price pressure.

The order book imbalance is also often used by practitioners to calculate a *micro-price*, p^{micro}, which more closely reflects the microstructure effects than the mid-

price. The micro-price simply weighs the bid and ask prices, p^b and p^s, respectively, by the imbalance, I:

$$p^{micro} = I\ p^s + (1 - I)\ p^b, \tag{2.3}$$

where

$$I = \frac{d^b}{d^b + d^s} = \frac{OBI + 1}{2}. \tag{2.4}$$

2.1.3 Predictive Machine Learning Signals for Execution Algorithms

A classical approach in statistical modelling is to start out with simple, perhaps linear, models and a small set of variables, or features, that are likely to have some predictive power for the quantity of interest. Over the modelling process, model complexity is gradually increased, and features are further engineered and refined to extract more information from the raw data. Driven by the domination of the ML literature by deep learning and artificial neural network (ANN) models, most recent approaches, however, have moved away from handcrafted feature engineering and instead approached prediction problems using raw data directly. This trend has also taken hold in financial ML and quantitative trading.

A recent exception to this rule is the deep order flow imbalance model [Kolm et al. 2021], which uses order flow imbalance, as described in the previous section, to predict a vector of returns over multiple short-term horizons. The authors show that extracting order flow features from the top 10 levels of the order book is sufficient to achieve state-of-the-art predictive performance with relatively simple conventional neural network models, such as LSTM (long short-term memory) networks [Hochreiter and Schmidhuber 1997], or combinations of LSTMs with multilayer perceptrons (MLPs) [Haykin 1994]. This implies that practitioners might be able to get away with simpler models in some cases by performing an input data transformation from raw data to order flows. These results contrast with those of the same simple neural network models that instead use raw order book features, which cannot achieve any predictive power on the same task, implying

that the data transformation is essential. However, another result of this study is that linear models and simple MLP feed-forward networks alone are not useful for forecasting such short-term alpha signals.

Notwithstanding deep order flow imbalance models, the trend in the current ML literature on forecasting short-term price signals points in the direction of using raw order book states directly, using more advanced ANN architectures to automatically extract feature representations amenable to forward return regression or classification tasks. One such model is DeepLOB [Zihao Zhang et al. 2019a], which uses a deep neural network architecture with convolutional layers and an inception module [Szegedy et al. 2015] also based on convolutional layers. Convolutional neural networks (CNNs) were originally developed for visual classification tasks, such as handwritten digit recognition or classifying images based on their content. The start of the current popularity of CNNs came with AlexNet [Krizhevsky et al. 2012] and its superior performance on the ImageNet challenge, a classification problem of a large database of images with more than 1,000 prediction classes. Convolutional layers act as local filters on an image, aggregating local information in every spatial region. During learning, the weights of many such filters are updated as the overall system learns to recognise distinct features in the data, such as horizontal or vertical lines, corners, and regions of similar contrast. LOBs can be likened to images because they also contain local information; for example, price and volume information at each level of the book can be combined with adjacent levels to automatically extract new features using convolutional layers. Similarly, CNNs can learn local temporal information by convolving order book states over the time dimension.

Building on this analogy between images and LOBs, the state-of-the-art deep learning model DeepLOB [Zihao Zhang et al. 2019a], which uses raw order book states directly as inputs to the network, constitutes a new tool in the execution toolbox. This type of model currently provides the most accurate short-term price signals, which can be used to improve execution trajectories. To improve the robustness of forecasts, DeepLOB can also be extended to perform quantile

regression on the forward return distribution [Zihao Zhang et al. 2019c]. To do this, the final LSTM layer of the network is split into multiple separate parallel parts for as many quantiles as should be forecast, and each network branch uses a corresponding quantile loss function. Quantile predictions can then be combined to compute more robust point estimates and add a measure of uncertainty and risk for the practitioner. Training separate models for the bid and ask sides has the additional advantage of producing estimates of the market spread, which can help in deciding the prices at which the order should be placed and whether the spread should be crossed.

To improve the planning of an execution schedule, a point-in-time price forecast can be useful; ultimately, however, a price path over the execution horizon would be more beneficial for concrete planning. Using ideas from natural language processing and machine translation, a sequence-to-sequence model with an attention [Vaswani et al. 2017] mechanism can be used to achieve such multi-horizon return forecasts using LOB data [Zihao Zhang and Zohren 2021]. To translate written text from one language to another, the idea of the sequence-to-sequence model [Sutskever et al. 2014] is to use an LSTM encoder to learn a representation of a sentence as a fixed-length vector and then use a separate LSTM-based decoder to again translate this vector representation into the target language. Adapting this idea to predict return trajectories, the model in Zihao Zhang and Zohren [2021] uses the DeepLOB network [Zihao Zhang et al. 2019a] as an encoder, while an attention mechanism [Vaswani et al. 2017] allows using selected hidden states of the encoder layers in the decoding step, producing the forecast time series. This way, the model drastically improves forecasting performance over longer time horizons as forecasts sequentially build on each other in an autoregressive fashion.

Because deep learning models benefit from large amounts of data, in the extreme case, models could even be trained on market-by-order messages directly. Given that deep order flow imbalance models already showed the advantage of this more stationary data source, using all the raw data in a deep learning framework is a logical next step.

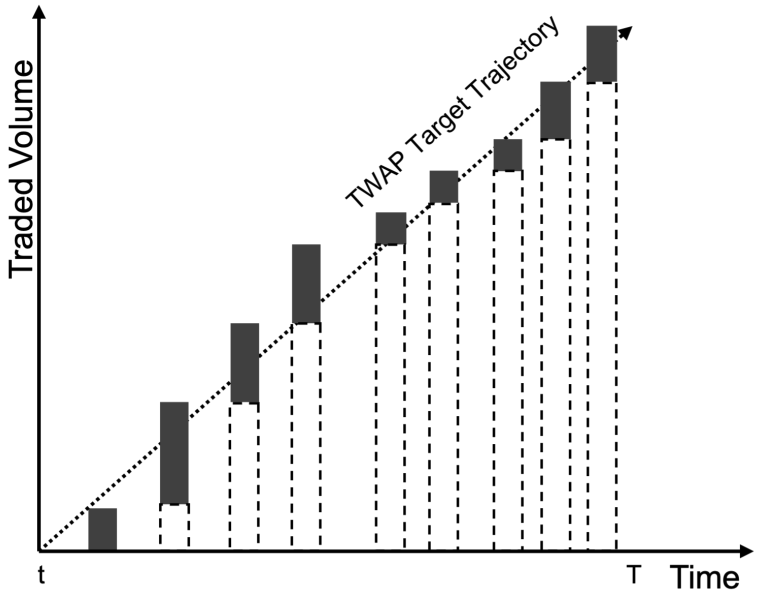


Figure 2.2: **Simple TWAP Strategy.** This exhibit shows an illustration of an execution schedule with a linear TWAP target trajectory and executed discrete slices as blocks over time around the target trajectory.

2.1.4 From Integrating Predictive Signals into Execution Algorithms to Automatically Learning Policies Using Reinforcement Learning

An execution strategy in an LOB market can be viewed as essentially two dimensional. One dimension is how very large order volumes are sliced up over time, while the other dimension describes the prices at which slices are placed in the order book. A simple TWAP strategy might, for instance, split a large order uniformly over time and always place marketable limit orders by crossing the spread (see Figure 2.2). This would imply, however, that half the spread is lost with every trade relative to the mid-price, incurring slippage costs even if the mid-price stays constant over the entire execution time. In contrast, placing trades passively by using limit orders at the near touch (i.e., without crossing the spread) has the opposite effect of earning half a spread whenever a marketable order is placed in the opposing direction. The downside of passive orders is that execution is uncertain and depends on other market participants' order flow and future price moves.

Thus, if one had a directional signal of future prices, it would be sensible to deviate from the simple TWAP strategy. For example, if the execution task is to sell a block of shares, a favourable price signal might indicate that the mid-price is expected to rise; hence, a slice of the order could be placed more passively deeper in the book (at a higher price), anticipating that a price swing could complete the trade. If price forecasts point downward, however, a market order could still make use of the higher price by executing a slice before the move happens.

Using a combination of ML models, one can thus engineer a complete execution strategy. An example algorithm might work as follows. Volatility forecasts, obtained using any method from historical averaging over *generalized autoregressive conditional heteroskedasticity* (GARCH) [Bollerslev 1986] models to deep learning, can be used to schedule either the time allowed for an execution ticket or the amount of front-loading over a fixed time horizon by controlling how execution slice sizes decrease over time. Using ML forecasts of trade volume, one can further modulate execution schedules by proportionately changing future slice sizes with expected volume forecasts. Predicted price paths can then be used to fine-tune the strategy by varying placement levels (prices) dynamically. Should one expect a favourable price move, one would place a passive limit order in the book at the first level – or even deeper into the book if the expected price move is sufficiently large. Such a strategy could be refined in various ways – for example, by using a probabilistic level placement to increase the probability of passive placements with more favourable price moves and wider spread forecasts. A probabilistic strategy also has the benefit of being less predictable by other market participants, who might want to exploit the effect that large orders have in moving the market.

Another ML approach encompassing the execution problem uses reinforcement learning (RL) algorithms to plan the execution trajectory. RL – and especially deep RL using deep neural networks – has been tremendously successful in solving complex problems, from learning to play games [V. Mnih et al. 2013] to controlling the plasma in nuclear fusion reactors [Degrave et al. 2022]. In the execution context, Ning et al. [2021] describe a good example using deep RL for optimal

execution problems using double Q-learning [Hasselt 2010], employing a modern algorithm previously used to train autonomous agents to play Atari games at a superhuman level from raw pixel input.

On a theoretical level, the execution problem can be framed as a partially observable Markov decision process (POMDP) [Spaan 2012], which can be amenable to being solved using RL algorithms. The RL learning paradigm works analogously to biological learning processes in animals and humans. The agent, our execution algorithm, interacts with a market environment in state s, which describes all information necessary to characterise the current state of the market, by performing an action, a, thereby transitioning the state to $s' = T(s, a)$. The environment state is further assumed to satisfy the Markov property, which means that past states do not add any further relevant information for the future. The agent, however, perceives not the entire state of the world but only an observation, $o' = obs(s')$, and hence does usually not know the underlying state exactly. In addition to the new observation o' at each step, the learner also receives a reward signal, r. Based on the reward signal, the RL algorithm learns over time which sequence of actions leads to the highest expected cumulative rewards.

Observational features for the RL algorithm might include various data from the LOB, including such handcrafted features as order flow imbalance and order book imbalance, or even specific model predictions, such as future price paths, spreads, or volatility. In the extreme case, using deep RL can help one even learn policies directly from raw order book data. Rewards in this scenario are usually based on incurred slippage during training.

A difficulty in real-world RL applications is that training the algorithm necessarily must be done in a simulated environment. The most basic kind of "simulator" simply uses historical market prices. This approach limits the action space to timing market orders, because past prices alone cannot determine whether a limit order would have been executed or not. Another shortcoming of relying solely on historical prices for simulation is that trades do not generate any market impact,

because neither do they take away liquidity in the book nor can they cause any other market participant to react to the trade. To alleviate the latter problem, simulation environments are sometimes enhanced with stochastic models of price impact to represent more realistic costs of aggressive trading. Another approach to help with the former problem of handling limit orders is to model the environment as a complete market replay using market-by-order data. This way, new market orders or even limit orders can be injected into the historical order flow. This approach accurately handles how the order would have been executed at the time.

However, this alone does not solve the problem of counterfactual behaviour by other agents. For example, if one of our orders is executed, it might imply that someone else's order was not executed. They then might have placed another order at a different price; however, this is not represented in the historical data. One approach to handle these counterfactual scenarios is agent-based modelling, which represents individual traders explicitly in a simulation. These simulated agents follow their own trading strategies and can react to changes in the market caused by our execution algorithm, as well as to actions and reactions of other agents. Capturing realistic trading behaviour remains a challenging task, and building realistic LOB models is the subject of active research.

2.1.5 Conclusions on Machine Learning for Execution Algorithms

The optimal execution of large trades depends critically on the specifics of the market's microstructure. This thesis has focused on price–time priority limit order book (LOB) markets, as they represent the predominant design across major exchanges trading cash equities, futures, and options. LOBs offer rich high-frequency data, which can be leveraged to develop data-driven execution algorithms using machine learning (ML) methods. This data enables the construction of informative features for predicting a variety of execution-relevant variables, including spreads, trade volume, volatility, and short-term price movements (so-called *fast alpha sig-*

nals). Individual ML models, each targeting a specific variable, can be integrated into a unified execution algorithm that dynamically plans both the timing and price levels of order placement across the execution horizon.

The broader trend in machine learning research favours leveraging increasing computational resources to automate feature extraction from raw data [R. Sutton 2019], reducing reliance on handcrafted features. This shift is evident in finance as well, particularly in trade execution, as exemplified by the DeepLOB model introduced by Zihao Zhang et al. [2019a] and its successors. With access to extensive LOB datasets and modern computational infrastructure, deep learning models are already outperforming simpler methods in tasks such as generating fast alpha signals. Pushing trade automation further, reinforcement learning (RL) provides a promising framework for minimising slippage by enabling execution algorithms to learn optimal actions directly. These actions can operate at varying levels of abstraction – from dynamically adjusting parameters within existing algorithms to placing trades outright. The research outlook for ML in execution thus points toward increasingly end-to-end systems, combining advanced deep learning and (deep) RL architectures to enhance predictive accuracy while addressing key concerns such as model robustness.

End of Published Paper

2.2 Model Architectures

2.2.1 State-Space Models and Transformers

Much of the recent progress in large language models (LLMs) is driven by the transformer architecture [Vaswani et al. 2017], which relies on a self-attention mechanism to model long-range dependencies. In its standard form, self-attention computes pairwise attention scores between all tokens in a sequence, producing a weighted aggregation of token representations. While highly expressive, this mechanism incurs a computational cost that scales quadratically with sequence length $\mathcal{O}(L^2)$ during training, since attention scores must be computed for each token with respect to every other token in the sequence. During inference, memory usage grows linearly $\mathcal{O}(L)$ due to the need to store a key-value cache that allows the model to reuse past computations for efficient autoregressive generation. These limitations pose significant scalability challenges, especially for applications requiring long-context modelling such as financial time series or high-frequency LOB data.

In practice, large-scale transformer training relies heavily on implementation optimisations, including efficient GPU kernels, memory-aware attention algorithms such as FlashAttention [Dao et al. 2022; Dao 2023], and highly optimised codebases for parallel and distributed training [Dao and A. Gu 2024]. Despite these engineering advances, the inherent scaling limitations of standard attention have motivated the development of numerous efficient transformer variants [Tay et al. 2022], which aim to reduce the computational and memory complexity associated with long sequences. While many of these architectures achieve substantial improvements in specific benchmarks, most continue to face empirical challenges when applied to very long contexts – particularly in domains like financial time series, where long-range dependencies are common and critical to model fidelity.

One prominent class of models proposed as an alternative to transformers are state-space models (SSMs). The appeal of SSMs lies in their favourable computa-

tional properties: linear time complexity $\mathcal{O}(L)$ and constant memory requirements $\mathcal{O}(1)$, making them particularly well-suited for long-sequence modelling. SSMs are closely related to recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which share the same asymptotic complexity class.

Recent advances have introduced *structured* state-space models into deep learning architectures, including S4 [A. Gu et al. 2021a], S5 [J. T. Smith et al. 2022], Mamba [A. Gu and Dao 2023], and Mamba 2 [Dao and A. Gu 2024]. These models impose specific structures on the state transition matrices to maintain computational tractability, such as diagonal plus low-rank structures in S4 or purely diagonal matrices in S5. In all these cases, the underlying models are linear state-space systems, where the latent state evolves as a linear function of the previous state and current input. Non-linearities are introduced only through activation functions applied after each SSM layer. Despite their architectural simplicity, these models retain strong representational power and have been shown to be universal approximators [Shida Wang and Xue 2023], making them a compelling alternative to attention-based models for sequence modelling tasks.

S4 and S5 models are not only linear but also *time-invariant* (LTI), meaning their parameters remain fixed over time and do not depend on the input data. This structural property allows for highly efficient computation, either through a parallel scan of the recurrence relation or by transforming the model into an equivalent global convolution using a precomputed kernel. These approaches enable fast and memory-efficient processing, particularly advantageous for long sequence modelling.

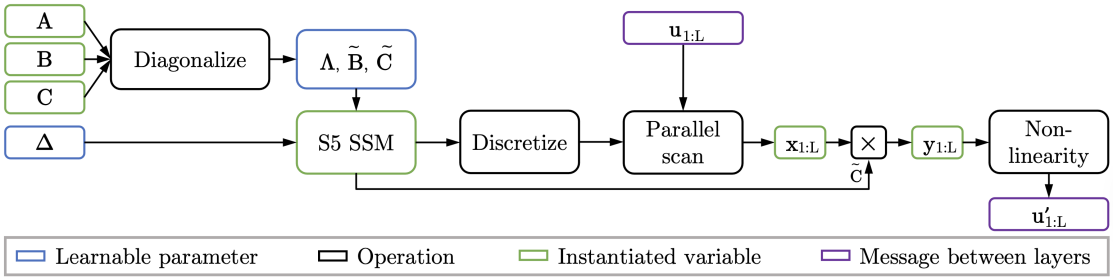
In contrast, Mamba and Mamba 2 are *time-varying* state-space models, where certain parameters are dynamically modulated by the input data. As a result, they lack an efficient convolutional kernel representation and cannot leverage parallel scan techniques. Instead, outputs must be computed sequentially, analogously to classical recurrent models. Despite this limitation, Mamba 2 achieves notable efficiency gains of $\sim 2-8\times$ speed-up by exploiting a duality between time-varying

linear SSMs and a subclass of transformers. This duality enables the reuse of certain architectural and computational tricks developed in the transformer literature, narrowing the performance gap between time-varying and time-invariant sequence models [Dao and A. Gu 2024].

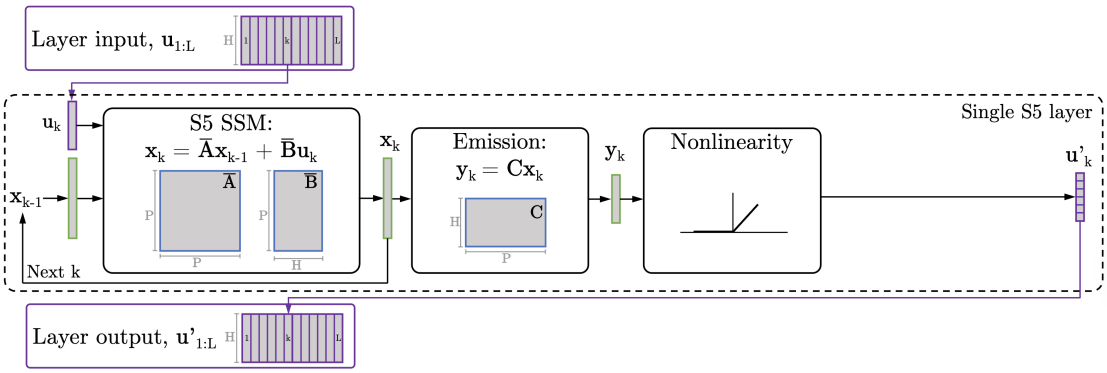
SSMs have achieved competitive, and in some cases even superior, performance compared to transformers on language modelling tasks involving small to medium-sized models [A. Gu and Dao 2023; Dao and A. Gu 2024]. Whether SSMs can match or surpass transformers in the large-model regime remains an open question. However, it is increasingly likely that the most effective architectures will combine the strengths of both paradigms. Hybrid models that incorporate a small fraction of attention layers of around 10% into otherwise SSM-based networks have already demonstrated improved performance and stability [Dao and A. Gu 2024]. For comprehensive overviews of different SSM architectures and their integration into deep learning, see Patro and Agneeswaran [2024] and X. Wang et al. [2024].

The *simplified* state-space layers (S5) [J. T. Smith et al. 2022] represent a refinement of the earlier S4 architecture[A. Gu et al. 2021a], designed to streamline computation and reduce architectural complexity. Unlike S4, which employs a bank of parallel single-input-single-output (SISO) systems followed by a mixing layer, S5 adopts a multi-input-multi-output (MIMO) formulation that enables a more compact latent representation. This design allows for efficient training using an associative scan based on the linear recurrence relation, eliminating the need for a convolutional representation. As a result, all computations in S5 are performed directly in the time domain, in contrast to S4's reliance on spectral methods using the fast Fourier transform (FFT).

The simplification extends to the parameterisation of the state matrix. Whereas S4 uses a diagonal plus low-rank (DPLR) structure, S5 employs a purely diagonal state matrix. This choice is supported by recent findings showing that diagonal state-space models can match the empirical performance of more complex structured variants [Gupta et al. 2022]. Two schematic illustrations of the S5 layer



(a) Schematic of the computational components of an S5 layer. Taken from Figure 1 in J. T. Smith et al. [2022].



(b) Schematic of a single S5 layer. Taken from Figure 2 in J. T. Smith et al. [2022].

Figure 2.3: **S5 schematics.** (a) The SSM is first diagonalised and then discretised. The latent state $\mathbf{x_{1:L}}$ is then computed using a parallel scan over the layer input $\mathbf{u_{1:L}}$, which in turn is used to compute the layer pre-output $\mathbf{y_{1:L}}$. A nonlinear activation function then produces the final layer output $\mathbf{u'_{1:L}}$. (b) The layer computations are illustrated for a single input element $\mathbf{u_k} \in \mathbf{u_{1:L}}$ in matrix form.

architecture are shown in Figure 2.3.

2.2.2 Discretisation of State-Space Models

The state-space models discussed in this chapter are fundamentally based on linear time-invariant (LTI) ordinary differential equations (ODEs) to model the latent state evolution of a single network layer. The system is often most conveniently expressed in continuous time, which requires an extra step to discretise the system to perform parameter updates when using it with discrete data. This approach enables the derivation of stable initial network parameters from the stability results of the continuous system, which can then be converted to the equivalent discrete system parameters, as explained in this section.

Given a continuous input signal $\mathbf{u}(t) \in \mathbb{R}^U$, and a latent state variable $\mathbf{x}(t) \in \mathbb{R}^P$, we want to compute an output signal $\mathbf{y}_t \in \mathbb{R}^M$. An LTI state-space model (SSM) is defined by a differential equation defining the latent state evolution, and a linear output equation. The system is parametrised by a state matrix $\mathbf{A} \in \mathbb{R}^{P\times P}$, an input matrix $\mathbf{B} \in \mathbb{R}^{P\times U}$, an output matrix $\mathbf{C} \in \mathbb{R}^{M\times P}$, and a feedthrough matrix $\mathbf{D} \in \mathbb{R}^{M\times U}$.

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)\ , \qquad \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{2.5}$$

To reduce the computational cost of the state matrix multiplication during the recurrent update computation from $\mathcal{O}(P^3)$ for a dense matrix-matrix multiplication, we can diagonalise the system to achieve $\mathcal{O}(P)$ complexity using the eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}$:

$$\frac{d\mathbf{V^{-1}}\mathbf{x}(t)}{dt} = \mathbf{\Lambda}\mathbf{V^{-1}}\mathbf{x}(t) + \mathbf{V^{-1}}\mathbf{B}\mathbf{u}(t), \tag{2.6}$$

where $\mathbf{V} \in \mathbb{R}^{P\times P}$ contains the eigenvectors and $\mathbf{\Lambda} \in \mathbb{R}^{P\times P}$ the eigenvalues of A as diagonal entries. This way, we get the new system in $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$ with parameters $\mathbf{\Lambda},\ \tilde{\mathbf{B}} = \mathbf{V^{-1}}\mathbf{B},\ \tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$.

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = \mathbf{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t)\ , \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{2.7}$$

As a next step, we can discretise the diagonalised system by using the zero-order hold (ZOH) assumption that inputs $\mathbf{u}(t)$ are constant between steps of constant size Δ: $\mathbf{u}(t) = \mathbf{u}_k\ \forall t \in [k\Delta, (k+1)\Delta)$. The general solution to this ODE over a single sampling step is:

$$\tilde{\mathbf{x}}\left((k+1)\Delta\right) = \exp(\mathbf{\Lambda}\Delta)\tilde{\mathbf{x}}(k\Delta) + \left(\int_0^{\Delta} \exp(\mathbf{\Lambda}\tau)d\tau\right)\tilde{\mathbf{B}}\mathbf{u}_k. \tag{2.8}$$

Since $\mathbf{\Lambda}$ is diagonal, and hence invertible, the matrix integral can then be simplified

as $\int_0^\Delta \exp(\mathbf{\Lambda}\tau)d\tau = \mathbf{\Lambda}^{-1}[\exp(\mathbf{\Lambda}\Delta) - \mathbf{I}]$. Defining $\tilde{\mathbf{x}}_k = \tilde{\mathbf{x}}(k\Delta)$, we can write

$$\tilde{\mathbf{x}}_{k+1} = \exp(\mathbf{\Lambda}\Delta)\tilde{\mathbf{x}}_k + \mathbf{\Lambda}^{-1}(\exp(\mathbf{\Lambda}\Delta) - \mathbf{I})\mathbf{u}_k. \tag{2.9}$$

Simplifying notation by denoting $\bar{\mathbf{\Lambda}} = \exp(\mathbf{\Lambda}\Delta)$, we have thus arrived at a discrete form of our continuous system:

$$\tilde{\mathbf{x}}_{k+1} = \bar{\mathbf{\Lambda}}\tilde{\mathbf{x}}_k + \mathbf{\Lambda}^{-1}(\bar{\mathbf{\Lambda}} - \mathbf{I})\mathbf{u}_k. \tag{2.10}$$

Were we to use different assumptions on the continuity of $\mathbf{u}(t)$, we would end up with a slightly different form of the discrete system's parameters.

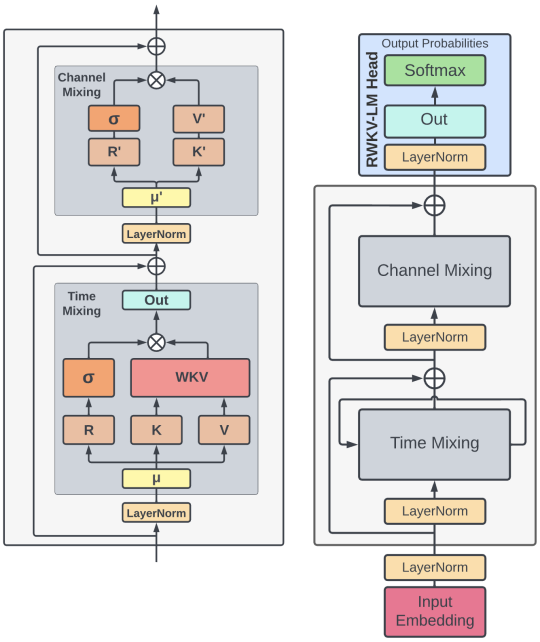
Structured state-space models, such as S5 [J. T. Smith et al. 2022] use this diagonalisation approach to provide structure to the state matrix, which is required for effective use in deep neural networks.

2.2.3 Receptance Weighted Key Value (RWKV) Models

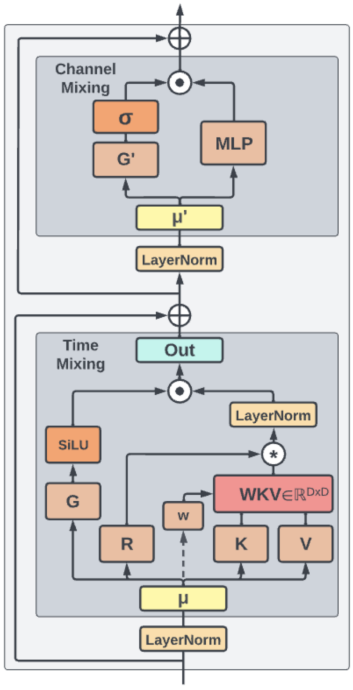
This chapter introduces a new class of generative LOB models based on the Receptance Weighted Key Value (RWKV) architecture [Peng et al. 2023; Peng et al. 2024]. Like recent structured state-space models, including S4 [A. Gu et al. 2021a], S5 [J. T. Smith et al. 2022], Mamba [A. Gu and Dao 2023], and Mamba-2 [Dao and A. Gu 2024], RWKV aims to overcome the quadratic time complexity of the original transformer architecture [Vaswani et al. 2017], making large-scale model training more efficient and accessible without significantly compromising expressivity.

RWKV is inspired by the linear attention mechanism [Zhai et al. 2021], but introduces further simplifications to form a channel-wise, time-decaying attention structure. This design enables efficient *time-parallel* training, analogous to transformers, while also supporting a recurrent mode for autoregressive generation. In the recurrent setting, RWKV achieves constant-time and constant-memory complexity per token, $\mathcal{O}(1)$ in both computation and memory, making it well-suited

for long-context sequence modelling and low-latency inference in high-frequency domains such as LOB data.



(a) RWKV 4 block (left) and model architecture for language modelling tasks (right).



(b) RWKV 5 (Eagle) and 6 (Finch) block

Figure 2.4: **The RWKV block** is split between a time-mixing and a channel-mixing sub-module. The time-mixing component uses a decaying linear attention mechanism, resulting in a time complexity of $\mathcal{O}(BLd^2)$ per layer, for batch size B, sequence length L, and feature dimension d. μ describes the token shift mechanism, producing a linear combination of the current and previous step input. The Figures are taken from Figure 2 in Peng et al. [2023] and Figure 1 in Peng et al. [2024].

The RWKV block schematic is shown in Figure 2.4. In this architecture, the *receptance* R integrates information from previous time steps, while the *weight* W serves as a learnable, channel-wise decay parameter that modulates the influence of past tokens based on their relative positions. The *key* K and *value* V retain analogous roles to those in the standard transformer architecture, encoding content and contextual relevance, respectively.

A key innovation in RWKV lies in the formulation of W as a position-sensitive vector applied per channel, introducing temporal decay into the attention computation in a learnable and efficient way. Rather than using conventional transformer

components, RWKV replaces the attention and feedforward (MLP) sub-blocks with two new modules: *time-mixing* and *channel-mixing*. The time-mixing block employs a time-dependent softmax and layer normalisation, which help stabilise training by mitigating vanishing and exploding gradients. Both blocks also include a *token shift* operation, denoted μ, which computes a linear combination of the current and previous input tokens, enabling local temporal context integration without full-sequence attention.

The classic self-attention mechanism [Vaswani et al. 2017] is expressed in matrix form as $\text{Attn}(Q, K, V) = \text{softmax}(QK^T)V$ for query, key, and value matrices Q, K, V. In element-wise notation, this can be written as

$$\text{Attn}(Q, K, V) = \frac{\sum_{i=1}^{T} \exp\{q_t^T k_i\} \odot v_i}{\sum_{i=1}^{T} \exp\{q_t^T k_i\}}. \tag{2.11}$$

This formulation highlights the quadratic cost of attention, as each token attends to all others through a dense similarity computation. Instead of using standard dot-product attention, RWKV defines its attention weights through a time-decaying formulation:

$$wkv_t = \frac{\sum_{i=1}^{t-1} \exp\{-(t-1-i)w + k_i\} \odot v_i + \exp\{u + k_t\} \odot v_t}{\sum_{i=1}^{t-1} \exp\{-(t-1-i)w + k_i\} + \exp\{u + k_t\}}. \tag{2.12}$$

Here, w is a learnable, channel-specific decay parameter that governs the exponential discounting of past tokens' influence over time. The inner product of keys and values is replaced with a sum over exponentially weighted key-value pairs, which decays as a function of temporal distance. To ensure that the current token is explicitly attended to, regardless of the value of w, an additional term involving u is added. This prevents degeneracies where large values of w could suppress contributions from the current token and ensures that every position maintains its own signal in the output.

The output vector after the WKV operator in the time-mixing submodule is com-

puted as

$$o_t = W_0(\sigma(r_t) \odot wkv_t), \tag{2.13}$$

where σ is the sigmoid activation, r_t is the receptance vector at time t, and W_0 is a learned output projection. This defines an explicit output-gating mechanism that modulates the contribution of wkv_t based on input-dependent control signals.

Similarly, in the channel-mixing submodule, the output is computed as

$$o'_t = \sigma(r'_t) \odot (W'_v \max(k'_t, 0)^2) \tag{2.14}$$

where $\max(k'_t, 0)^2)$ corresponds to a squared ReLU activation [So et al. 2021], enhancing sparsity and selectivity in the activation pattern.

During autoregressive data generation, the time-mixing module can be expressed in a recurrent form, referred to as *time-sequential mode*, which enables efficient token-by-token inference with constant memory and computation. The recurrent formulation for RWKV 4 [Peng et al. 2023] (and similarly RWKV 5 and 6) is given by:

$$\begin{aligned} a_0 &= 0, \quad b_0 = 0 \\ wkv_t &= \frac{a_{t-1} + \exp\{u + k_t\} \odot v_t}{b_{t-1} + \exp\{u + k_t\}} \\ a_t &= \exp\{-w\} \odot a_{t-1} + \exp\{k_t\} \odot v_t \\ b_t &= \exp\{-w\} \odot b_{t-1} + \exp\{k_t\}. \end{aligned} \tag{2.15}$$

This recurrence maintains two running sums, a_t and b_t, that accumulate exponentially decayed key-value and key terms, respectively. These allow the model to compute wkv_t in $\mathcal{O}(1$ time per step, enabling low-latency autoregressive generation over arbitrarily long sequences.

The exposition so far has focused on the RWKV 4 model [Peng et al. 2023], which introduced the core architectural innovations behind the Receptance Weighted Key Value framework. RWKV5 (Eagle) and RWKV6 (Finch) [Peng et al. 2024] extend this foundation to improve empirical performance, while preserving the model's

favourable training and inference characteristics – namely, linear time complexity during training and constant-time autoregressive generation.

RWKV 5 introduces a key architectural upgrade by replacing the original vector-valued state with a more expressive matrix-valued, multi-headed state. This enhances the model's capacity to represent and process richer temporal dependencies across channels. Additional refinements include the application of layer normalisation over attention heads, the removal of the sigmoid activation following the receptance (thereby reducing potential gating saturation effects), and the adoption of the SiLU (Sigmoid Linear Unit) activation function as the primary gating mechanism, which has shown superior empirical performance across deep learning tasks.

RWKV 6 further extends the model by introducing input-dependent modulation of time-mixing and token-shift components. This is achieved using the low-rank adaptation (LoRA) technique [Hu et al. 2022], which allows the model to learn an augmented, data-dependent decay vector with minimal additional parameter overhead. This extension increases the model's ability to adapt temporal weighting dynamically based on input content, improving flexibility and expressiveness without compromising efficiency.

While the RWKV family of models offers substantial efficiency gains in both training and inference, it inherits a key limitation from its linear attention mechanism: a reduced ability to explicitly copy or reference information across long temporal spans. Unlike full self-attention, which allows unrestricted pairwise interactions between all tokens, linear attention decays past information exponentially and lacks a mechanism for direct long-range token-to-token referencing. As a result, RWKV models may struggle with tasks that require precise memorisation or retrieval of distant contextual information, such as repeating exact sequences or maintaining structured dependencies over extended horizons. This limitation suggests a potential trade-off between computational efficiency and long-range expressiveness that future research may aim to address, possibly through hybrid architectures or

auxiliary memory mechanisms.

Chapter 3

Limit Order Books and Reinforcement Learning

3.1 Preface and Contextual Overview

Reinforcement learning (RL) offers a compelling framework: through trial and error, an agent learns to maximise its objective by interacting with an environment and receiving rewards aligned with its performance. Financial markets, and especially limit order books (LOBs), seem naturally suited to RL, given their dynamic nature and the need for strategies that adapt to evolving inventory and market conditions. However, in practice, several significant challenges must be addressed before RL policies can be safely and effectively deployed in live trading environments.

As a first step in this direction, we explore an application where RL is theoretically well-suited. Quantitative trading operations are typically divided into "alpha research", execution, and portfolio optimisation. To bridge the gap between individual trading signals and realised portfolio decisions, we apply RL to translate signals into fine-grained long or short positions within predefined limits. This study serves as proof of concept, demonstrating RL's potential to perform effective market timing using a noisy signal while managing inventory. Although we

provide a forward-looking signal, the task explicitly extends beyond the traditional objective of executing a fixed number of shares at minimal cost, which is the usual focus of execution research (see Section 2.1).

A key challenge in applying RL to LOB environments lies in modelling realistic simulation dynamics. The simplest data-driven approach uses only mid-price data and assumes a fixed bid-ask spread. Under this simplification, buy orders are executed at the mid-price plus half the spread, and sell orders at the mid-price minus half the spread, with arbitrary volume assumed to be available. However, this method does not utilise LOB data, failing to account for time-varying bid-ask spreads and the actual available volumes at each level of the book.

In contrast to simplistic approaches that ignore LOB data, we leverage the LOBSTER dataset [R. Huang and Polak 2011] to simulate the full LOB by injecting new orders into the historical order stream. This yields more realistic execution probabilities and ensures that traded volume never exceeds available volume. In the limit of infinitesimally small orders, which are assumed not to affect prices or market dynamics, this method is exact. To approximate this regime, we restrict each potential action to a single share per message. To further increase realism, we go beyond the no-impact assumption by modelling the mechanistic *direct* market impact: executed orders or cancellations remove volume from the book, while passive limit orders add volume.

The following paper has been published in *Frontiers in Artificial Intelligence 6*, 2023 [Nagy et al. 2023a].

3.2 Asynchronous Deep Double Duelling Q-Learning for Trading-Signal Execution in Limit Order Book Markets

Peer Nagy[1], Jan-Peter Calliess[1], and Stefan Zohren[1]

[1]Oxford-Man Institute of Quantitative Finance, Department of Engineering Science, University of Oxford

Abstract

We employ deep reinforcement learning (RL) to train an agent to successfully translate a high-frequency trading signal into a trading strategy that places individual limit orders. Based on the ABIDES limit order book simulator, we build a reinforcement learning OpenAI gym environment and utilise it to simulate a realistic trading environment for NASDAQ equities based on historic order book messages. To train a trading agent that learns to maximise its trading return in this environment, we use Deep Duelling Double Q-learning with the APEX (asynchronous prioritised experience replay) architecture. The agent observes the current limit order book state, its recent history, and a short-term directional forecast. To investigate the performance of RL for adaptive trading independently from a concrete forecasting algorithm, we study the performance of our approach utilising synthetic alpha signals obtained by perturbing forward-looking returns with varying levels of noise. Here, we find that the RL agent learns an effective trading strategy for inventory management and order placing that outperforms a heuristic benchmark trading strategy having access to the same signal.

3.2.1 Introduction

Successful quantitative trading strategies often work by generating trading signals, which exhibit a statistically significant correlation with future prices. These signals are then turned into actions, aiming to assume positions in order to gain from future price changes. The higher the signal frequency and strategy turnover, the more critical is the execution component of the strategy, which translates the signal into concrete orders that can be submitted to a market. Such markets are oftentimes organised as an order ledger represented by a *limit order book (LOB)*.

Limit order book prices have been shown to be predictable over short time periods, predicting a few successive ticks into the future with some accuracy. This has been done by either utilising the recent history of order book states [Zihao Zhang et al. 2019a; Zihao Zhang and Zohren 2021], order-flow data [Kolm et al. 2021], or market-by-order (MBO) data directly as features [Zihao Zhang et al. 2021b]. However, given the short time horizons over which these predictions are formed, and correspondingly small price movements, predictability does not directly translate into trading profits. Transaction costs, strategy implementation details, and time delays add up to the challenging problem of translating high-frequency forecasts into a trading strategy that determines when and which orders to send to the exchange. In addition, different predictive signals have to be traded differently to achieve optimal results, depending on the forecast horizon, signal stability, and predictive power.

In this paper, we use asynchronous off-policy reinforcement learning (RL), specifically *Deep Duelling Double Q-learning* with the *APEX* architecture [Horgan et al. 2018], to learn an optimal trading strategy, given a noisy directional signal of short-term forward mid-quote returns. For this purpose, we developed an OpenAI gym [Brockman et al. 2016] limit order book environment based on the ABIDES [Byrd et al. 2020] market simulator, similar to [Amrouni et al. 2021]. We use this simulator to replay NASDAQ price-time priority limit order book markets using message data from the LOBSTER data set [R. Huang and Polak 2011].

If a financial trader wants to transact a number of shares of a financial security, such as shares of cash equities, for example Apple stock, they need to send an order to an exchange. Most stock exchanges today accept orders electronically and in 2022 in US equity markets approximately 60 - 73% of orders are submitted by algorithms, not human traders [Mordor Intelligence 2022]. A common market mechanism to efficiently transact shares between buyers and sellers is the *limit order book (LOB)*. The LOB contains all open limit orders for a given security and is endowed with a set of rules to clear marketable orders. A more detailed introduction to LOBs can be found in section 3.2.3.

High-frequency trading (HFT) is an industry with high barriers to entry and a regulatory landscape varying by geographical region. Generally, trading on both sides of the LOB simultaneously by submitting both limit buy and limit sell orders is the purview of *market makers*. While their principal role is to provide liquidity to the market, they also frequently take directional bets over short time periods. Transaction costs for such trading strategies consist of two main components, *explicit costs*, such as trading fees, and *implicit costs*, such as market impact [L. Harris 2003]. Trading fees vary by institution but can be negligible for institutional market makers, with some fee structures even resulting in zero additional costs if positions are closed out at the end of the day. The main consideration of transaction costs should thus be given to market impact. Our simulation environment models direct market impact by injecting new orders into historic order flow, thereby adding to or consuming liquidity in the market. One limitation of this approach is that indirect market impact and, generally, the reactions of other market participants are not modelled.

We study the case of an artificial or synthetic signal, taking the future price as known and adding varying levels of noise, allowing us to investigate learning performance and to quantify the benefit of an RL-derived trading policy compared to a baseline strategy using the same noisy signal. This is not an unrealistic setup when choosing the correct level of noise. Practitioners often have dedicated teams researching and deriving alpha signals, often over many years, while other teams

might work on translating those signals into profitable strategies. Our aim is to focus on the latter problem, which becomes increasingly more difficult as signals become faster. It is thus interesting to see how an RL framework can be used to solve this problem. In particular, we show that the RL agent learns policies superior to the baselines, both in terms of strategy return and Sharpe ratio. Machine learning methods, such as RL, have become increasingly important to automate trade execution in the financial industry in recent years [Nagy et al. 2023c], underlining the practical use of research in this area.

We make a number of contributions to the existing literature. By defining a novel action and state space in a LOB trading environment, we allow for the placement of limit orders at different prices. This allows the agent to learn a concrete high-frequency trading strategy for a given signal, trading either aggressively by crossing the spread, or conservatively, implicitly trading off execution probability and cost. In addition to the timing and level placement of limit orders, our RL agent also learns to use limit orders of single units of stock to manage its inventory as it holds variably sized long or short positions over time. More broadly, we demonstrate the practical use case of RL to translate predictive signals into limit order trading strategies, which is still usually a hand-crafted component of a trading system. We thus show that simulating limit order book markets and using RL to further automate the investment process is a promising direction for further research. To the best of our knowledge, this is also the first study applying the APEX [Horgan et al. 2018] algorithm to limit order book environments.

The remaining paper is structured as follows: Section 3.2.2 surveys related literature, section 3.2.3 explains the mechanics of limit order book markets and the APEX algorithm, section 3.2.4 details the construction of the artificial price signal, section 3.2.5 showcases our empirical results, and section 3.2.6 concludes our findings.

3.2.2 Related Literature

Reinforcement learning has been applied to learn different tasks in limit order book market environments, such as optimal trade execution [Nevmyvaka et al. 2006a; Ning et al. 2021; Dabérius et al. 2019; Karpe et al. 2020; Schnaubelt 2022], market making [Abernethy and Kale 2013; Kumar 2020], portfolio optimisation [P. Yu et al. 2019] or trading [Kearns and Nevmyvaka 2013; Wei et al. 2019; Briola et al. 2021]. The objective of optimal trade execution is to minimise the cost of trading a predetermined amount of shares over a given time frame. Trading direction and the number of shares is already pre-defined in the execution problem. Market makers, on the other hand, place limit orders on both sides of the order book and set out to maximise profits from capturing the spread, while minimising the risk of inventory accumulation and adverse selection. We summarise using the term "RL for trading" such tasks which maximise profit from taking directional bets in the market. This is a hard problem for RL to solve as the space of potential trading strategies is large, leading to potentially many local optima in the loss landscape, and actionable directional market forecasts are notoriously difficult due to arbitrage in the market.

The work of [Kearns and Nevmyvaka 2013] is an early study of RL for market microstructure tasks, including trade execution and predicting price movements. While the authors achieve some predictive power of directional price moves, forecasts are determined to be too erroneous for profitable trading. The most similar work to ours is [Briola et al. 2021] that provides the first end-to-end DRL framework for high-frequency trading, using PPO [Schulman et al. 2017] to trade Intel stock. To model price impact, [Briola et al. 2021] use an approximation, moving prices proportionately to the square-root of traded volume. The action space is essentially limited to market orders, so there is no decision made on limit prices. The trained policy is capable of producing a profitable trading strategy on the evaluated 20 test days. However, this is not compared to baseline strategies and the resulting performance is not statistically tested for significance. In contrast, we consider a larger action space, allowing for the placement of limit orders at

different prices, thereby potentially lowering transaction costs of the learned HFT strategy. For a broader survey of deep RL (DRL) for trading, including portfolio optimisation, model-based and hierarchical RL approaches the reader is referred to [Kumar 2020].

One strand of the literature formulates trading strategies in order-driven markets, such as LOBs, as optimal stochastic control problems, which can often even be solved analytically. A seminal work in this tradition is Almgren and Chriss [2001], which solves a simple optimal execution problem. More recently, Cartea et al. [2018] use order book imbalance, i.e. the relative difference in volume between buy and sell limit orders, to forecast the direction of subsequent market orders and price moves. They find that utilising this metric can thereby improve the performance of trading strategies. In addition to market orders, LOB imbalance has also been found to be predictive of limit order arrivals, and illegal manipulation of the LOB imbalance by *spoofing* can be a profitable strategy [Cartea et al. 2020]. Stochastic models, assuming temporary and permanent price impact functions have also found that using order flow information can reduce trading costs when trading multiple assets [Cartea et al. 2019; Cartea and Jaimungal 2016b]. In contrast to the stochastic modelling literature, we employ a purely data-driven approach using simulation. This allows us to make fewer assumptions about market dynamics, such as specific functional forms and model parameters. The stochasticity of the market is captured in large samples of concrete data realisations.

3.2.3 Background

Limit Order Book Data

Limit order books (LOBs) are one of the most popular financial market mechanisms used by exchanges around the world [Gould et al. 2013]. Market participants submit limit buy or sell orders, specifying a maximum (minimum) price at which they are willing to buy (sell), and the size of the order. The exchange's LOB then keeps track of unfilled limit orders on the buy side (bids) and the sell side (asks). If

an incoming order is marketable, i.e. there are open orders on the opposing side of the order book at acceptable prices, the order is matched immediately, thereby removing liquidity from the book. The most popular matching prioritisation scheme is price-time priority. Here, limit orders are matched first based on price, starting with the most favourable price for the incoming order, and then based on arrival time, starting with the oldest resting limit order in the book, at each price level.

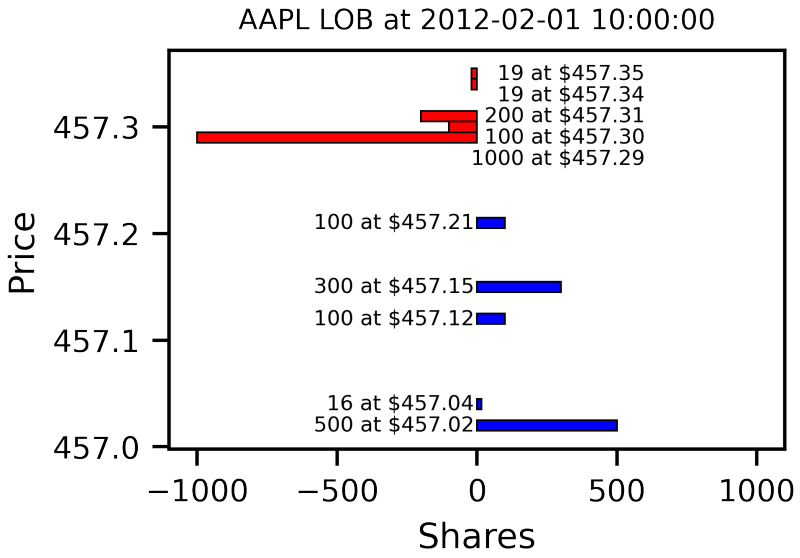


Figure 3.1: **Example Limit Order Book Snapshot** of Apple Stock (AAPL) on 01 February 2012 at 10 am. Displayed are the 5 best bid (blue) and ask levels (red). Ask sizes are shown as negative values to indicate limit *sell* orders. AAPL is an example of a small-tick stock since the minimum tick size of ¢1 is small compared to the stock price. For a given number of orders, this results in sparser books with more empty price levels. (NASDAQ data from LOBSTER [R. Huang and Polak 2011])

Figure 3.1 shows an example snapshot of an Apple LOB, traded at the NASDAQ exchange, on Wednesday 01 February 2012 at 10 am. Shown are the best 5 price levels on the bid and ask side and the aggregated available volume at each price. In this example, the best bid lies at $457.21 for 100 shares, which is the maximum instantaneous price a potential seller would be able to trade at, for example using a market order. Conversely, a potential buyer could receive up to 1000 shares at a price of $457.29. Trading larger quantities on either side would use up all volume at the best price and consume additional liquidity at deeper levels. Submitting a buy

limit order at a price below the best ask would not be marketable immediately and instead enter the LOB as new volume on the bid side. For a more comprehensive review of limit order book dynamics and pertaining models, we refer the reader to [Gould et al. 2013].

In this paper, we consider equity limit order book data from the NASDAQ exchange [R. Huang and Polak 2011], which also uses a price-time priority prioritisation. Our market simulator keeps track of the state of the LOB by replaying historical message data, consisting of new incoming limit orders, order cancellations, or modifications. The RL agent can then inject new messages into the order flow and thereby, change the LOB state from its observed historical state.

Our simulator reconstructs LOB dynamics from message data, so every marketable order takes liquidity from the book and thus has a direct price impact. Beyond that, we make no further assumptions on permanent market impact or reactions of other agents in the market, which we leave to future work.

Deep Reinforcement Learning

We model the trader's problem as a Markov Decision Process (MDP) [Bellman 1957; Puterman 1990], described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$. $\mathcal{S}$ denotes a state space, $\mathcal{A}$ an action space, $\mathcal{T}$ a stochastic transition function, r a reward function, and γ a discount factor. Observing the current environment state $s_t \in \mathcal{S}$ at time t, the trader takes action $a_t \in \mathcal{A}$, which causes the environment to transition state according to the stochastic transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$. After transitioning from s_t to s_{t+1}, the agent receives a reward $r_{t+1} = r(s_t, a_t, s_{t+1})$.

Solving the MDP amounts to finding an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$, which maximises the discounted expected sum of future rewards $\sum_{i=t+1}^{T} \gamma^{i-t} \mathbb{E}_t r_i$ between current time t and terminal time T, given a discount factor $\gamma \in (0, 1]$. As the transition kernel $\mathcal{T}$ is unknown, we use reinforcement learning (RL) to learn an optimal policy from observed trajectories of state-action transitions. RL algorithms fall broadly within two categories: value-based methods, which learn representations

of value functions, and policy-based methods, which learn explicit representations of the agent's policy[1]. In this paper, we are using a value-based RL algorithm, based on *Q-learning*, which explicitly approximates the action-value function Q^* for the optimal policy π^*. The action-value function for a given policy π is defined recursively as $Q^\pi(s,a) = \mathbb{E}_t r_{t+1} + \gamma \max_{a'} Q^\pi(s_{t+1}, a')$. One benefit of this class of algorithms is that they are *off-policy* learners, which means that they approximate the optimal value function Q^* using transitions sampled from the environment using a, potentially sub-optimal, exploratory policy π. This allows for computational efficiency due to asynchronous sampling and learning steps, as described in the next section. Modern Deep RL algorithms, such as DQN [Volodymyr Mnih et al. 2013], use neural networks as function approximators, in this case, of the Q-function. This way algorithms make use of the generalisation abilities provided by deep learning, necessitated by large or continuous state spaces. Network parameters are updated using *temporal difference learning* with gradient-based optimisers, such as stochastic gradient descent or the popular Adam optimizer [D. P. Kingma and Welling 2014]. For a comprehensive treatment of RL, we refer the interested reader to R. S. Sutton and Barto [2018], and for a survey of recent progress in RL for financial applications to Hambly et al. [2023].

Double DQN with Distributed Experience Replay

We use Deep Double Q-learning [Van Hasselt et al. 2016] with a duelling network architecture [Z. Wang et al. 2016] to approximate the optimal Q-function $Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | a_t = a, s_t = s]$. To speed up the learning process we employ the *APEX* training architecture [Horgan et al. 2018], which combines asynchronous experience sampling using parallel environments with off-policy learning from experience replay buffers. Every episode i results in an experience trajectory $\tau_i = \{s_t, a_t, r_{t+1}\}_{t=1}^T$, many of which are sampled from parallel environment instances and are then stored in the replay buffer. The environment sampling is done asynchronously using parallel processes running on CPUs. Ex-

[1] Actor-critic algorithms fall between the two as they keep explicit representations of both policy (actor) and value functions (critic).

perience data from the buffer is then sampled randomly and batched to perform a policy improvement step of the Q-network on the GPU. Prioritised sampling from the experience buffer has proven to degrade performance in our noisy problem setting, hence we are sampling uniformly from the buffer.[2] After a sufficient number of training steps, the new policy is then copied to every CPU worker to update the behavioural policy.

Double Q-learning [Hasselt 2010; Van Hasselt et al. 2016] stabilises the learning process by keeping separate Q-network weights for action selection (main network) and action validation (target network). The target network weights are then updated gradually in the direction of the main network's weight every few iterations. Classical Q-learning without a separate target network can be unstable due to a positive bias introduced by the max operator in the definition of the Q-function, leading to exploding Q-values during training. The duelling network architecture [Z. Wang et al. 2016] additionally uses two separate network branches (for both main and target Q-networks). One branch estimates the value function $V(s) = \max_a Q(s, a)$, while the other estimates the advantage function $A(s, a) = Q(s, a) - V(s)$. The benefit of this architecture choice lies therein that the advantage of individual actions in some states might be irrelevant, and the state value, which can be learnt more easily, suffices for an action-value approximation. This leads to faster policy convergence during training.

3.2.4 Framework

Artificial Price Signal

The artificial directional price signal $d_t \in \Delta^2 = \{x \in \mathbb{R}^3 : x_1 + x_2 + x_3 = 1,\ x_i \geq 0 \text{ for } i = 1, 2, 3\}$ the agent receives is modelled as a discrete probability distribution over 3 classes, corresponding to the averaged mid-quote price decreasing,

[2] In many application domains prioritised sampling, whereby we resample instances more frequently where the model initially performs poorly tends to aide learning. However, in our low signal-to-noise application domain, we noted poor performance. Investigating the matter, we found that prioritised sampling caused more frequent resampling of highly noisy instances where learning was particularly difficult, thus degrading performance.

remaining stable, or increasing over a fixed future time horizon of $h \in \mathbb{N}_+$ seconds. To achieve realistic levels of temporal stability of the signal process, d_t is an exponentially weighted average, with persistence coefficient $\phi \in (0, 1)$, of Dirichlet random variables ϵ_t. The Dirichlet parameters α depend on the realised smoothed future return $\overline{r_{t+h}}$, specifically on whether the return lies within a neighbourhood of size k around zero, or above or below. Thus we have:

$$\begin{aligned} d_t &= \phi d_{t-1} + (1-\phi)\epsilon_t \\ \epsilon_t &= \text{Dirichlet}\left(\alpha(\overline{r_{t+h}})\right) \\ \overline{r_{t+h}} = \frac{\overline{p_{t+h}} - p_t}{p_t} \quad &\text{where} \quad \overline{p_{t+h}} = \frac{1}{h}\sum_{i=1}^{h} p_{t+i} \end{aligned} \tag{3.1}$$

and prices p_t refer to the mid-quote price at time t. The Dirichlet distribution is parametrised, so that, in expectation, the signal d_t updates in the direction of future returns, where a^H and a^L determine the variance of the signal. The Dirichlet parameter vector is thus:

$$\alpha(r_{t+h}) = \begin{cases} (a^H, a^L, a^L) & \text{if } \overline{r_{t+h}} < -k \\ (a^L, a^H, a^L) & \text{if } -k \leq \overline{r_{t+h}} < k \\ (a^L, a^L, a^H) & \text{if } k \leq \overline{r_{t+h}}. \end{cases} \tag{3.2}$$

RL Problem Specification

At each time step t, the agent receives a new state observation s_t. s_t consists of the time left in the current episode $T-t$ given the episode's duration of T, the agent's cash balance C_t, stock inventory X_t, the directional signal $d_t \in \Delta^2$, encoded as probabilities of prices decreasing, remaining approximately constant, or increasing; and price and volume quantities for the best bid and ask (level 1), including the agent's own volume posted at bid and ask: $o_{b,t}$ and $o_{a,t}$ respectively. In addition to the most current observable variables at time t, the agent also observes a history of the previous l values, which are updated whenever there is an observed change

in the LOB. Putting all this together, we obtain the following state observation:

$$s_t = \begin{pmatrix} T-u \\ C_u \\ X_u \\ (d_u^1, d_u^2, d_u^3)' \\ (p_{a,u}, v_{a,u}, o_{a,u}, p_{b,u}, v_{b,u}, o_{b,u})' \end{pmatrix}_{u=\{t-l,\dots,t\}} .$$

After receiving the state observation, the agent then chooses an action a_t. It can place a buy or sell limit order of a single share at bid, mid-quote, or ask price; or do nothing and advance to the next time step. Actions, which would immediately result in positions outside the allowed inventory constraints $[pos_{min}, pos_{max}]$ are disallowed and do not trigger an order. Whenever the execution of a resting limit order takes the inventory outside the allowed constraints, a market order in the opposing direction is triggered to reduce the position back to pos_{min} for short positions or pos_{max} for long positions. Hence, we define

$$a_t \in \mathcal{A} = (\{-1, 1\} \times \{-1, 0, 1\}) \cup \{skip\}$$

so that in total there are 7 discrete actions available, three levels for both buy and sell orders, and a skip action. For the six actions besides the "skip" action, the first dimension encodes the trading direction (sell or buy) and the second dimension the price level (bid, mid-price, or ask). For example, $a = (1, 0)$ describes the action to place a buy order at the mid price, and $a = (-1, 1)$ a sell order at best ask. Rewards R_{t+1} consist of a convex combination of a profit-and-loss-based reward R_{t+1}^{pnl} and a directional reward R_{t+1}^{dir}. R_{t+1}^{pnl} is the log return of the agent's mark-to-market portfolio value M_t, encompassing cash and the current inventory value, marked at the mid-price. The benefit of log-returns is that they are additive over time, rather than multiplicative like gross returns, so that, without discounting ($\gamma = 1$) the total profit-and-loss return $\sum_{s=t+1}^{T} R_s^{pnl} = M_T - M_t$. The directional reward term R_{t+1}^{dir} incentivizes the agent to hold inventory in the direction of the

signal and penalises the agent for inventory positions opposing the signal direction. The size of the directional reward can be scaled by the parameter $\kappa > 0$. R^{dir}_{t+1} is positive if the positive prediction has a higher score than the negative ($d_{t,3} > d_{t,1}$) and the current inventory is positive; or if $d_{t,3} < d_{t,1}$ and $X_t < 0$. Further, if the signal $[-1, 0, 1] \cdot d_t$ has an opposite sign than inventory X_t, R^{dir}_{t+1} is negative. This can be summarised as follows:

$$\begin{aligned}
\text{Mark-to-Market Value} \quad & M_t = C_t + X_t p^m_t \\
& \Delta M_t = \Delta C_t + X_{t-1}\Delta p^m_t + \Delta X_t p^m_t \\
\text{PnL Reward} \quad & R^{pnl}_{t+1} = \ln(M_t) - \ln(M_{t-1}) \\
\text{Directional Reward} \quad & R^{dir}_{t+1} = \kappa[-1, 0, 1] \cdot d_t X_t \\
\text{Total Reward} \quad & r_{t+1} = w^{dir} R^{dir}_{t+1} + (1 - w^{dir}) R^{pnl}_{t+1}
\end{aligned} \tag{3.3}$$

The weight on the directional reward $w^{dir} \in [0, 1)$ is reduced every learning step by a factor $\psi \in (0, 1)$,

$$w^{dir} \leftarrow \psi w^{dir}$$

so that initially the agent quickly learns to trade in the signal direction. Over the course of the learning process, R^{pnl}_t becomes dominant and the agent maximises its mark-to-market profits.

3.2.5 Experimental Results

We train all RL policies using the problem setup discussed in section 3.2.4 on 4.5 months of Apple (AAPL) limit order book data (2012-01-01 to 2012-05-16) and evaluate performance on 1.5 months of out-of-sample data (2012-05-17 to 2012-06-31). We only use the first hour of every trading day (09:30 to 10:30) as the opening hour exhibits higher-than-average trading volume and price moves. Each hour of the data corresponds to a single RL episode. After analysing the results, we also performed a robustness check by repeating the training and analysis pipeline

on more recent AAPL data from 2022. Results are reported in section 3.2.5 and confirm the main conclusions based on earlier data.

Our neural network architecture consists of 3 feed-forward layers, followed by an LSTM layer, for both the value- and advantage stream of the duelling architecture. The LSTM layer allows the agent to efficiently learn a memory-based policy with observations including 100 LOB states.

We compare the resulting learned RL policies to a baseline trading algorithm, which receives the same artificially perturbed high-frequency signal of future mid-prices. The baseline policy trades aggressively by crossing the spread whenever the signal indicates a directional price move up or down until the inventory constraint is reached. The signal direction in the baseline algorithm is determined as the prediction class with the highest score (down, neutral, or up). When the signal changes from up or down to neutral, indicating no immediate expected price move, the baseline strategy places a passive order to slowly reduce position size until the inventory is cleared. This heuristic utilises the same action space as the RL agent and yielded better performance than trading using only passive orders (placed at the near touch), or only aggressive orders (at the far touch).

Figure 3.2 plots a 17 second simulation window from the test period, comparing the simulated baseline strategy with the RL strategy. It can be seen that prices in the LOB are affected by the trading activity as both strategies inject new order flow into the market, in addition to the historical orders, thereby consuming or adding liquidity at the best bid and ask. During the plotted period, the baseline strategy incurs small losses due to the signal switching between predicting decreasing and increasing future prices. This causes the baseline strategy to trade aggressively, paying the spread with every trade. The RL strategy, on the other hand, navigates this difficult period better by trading more passively out of its long position, and again when building up a new position. Especially in the second half of the depicted time period, the RL strategy adds a large number of passive buy orders (green circles in the second panel of figure 3.2). This is shown by the green straight lines,

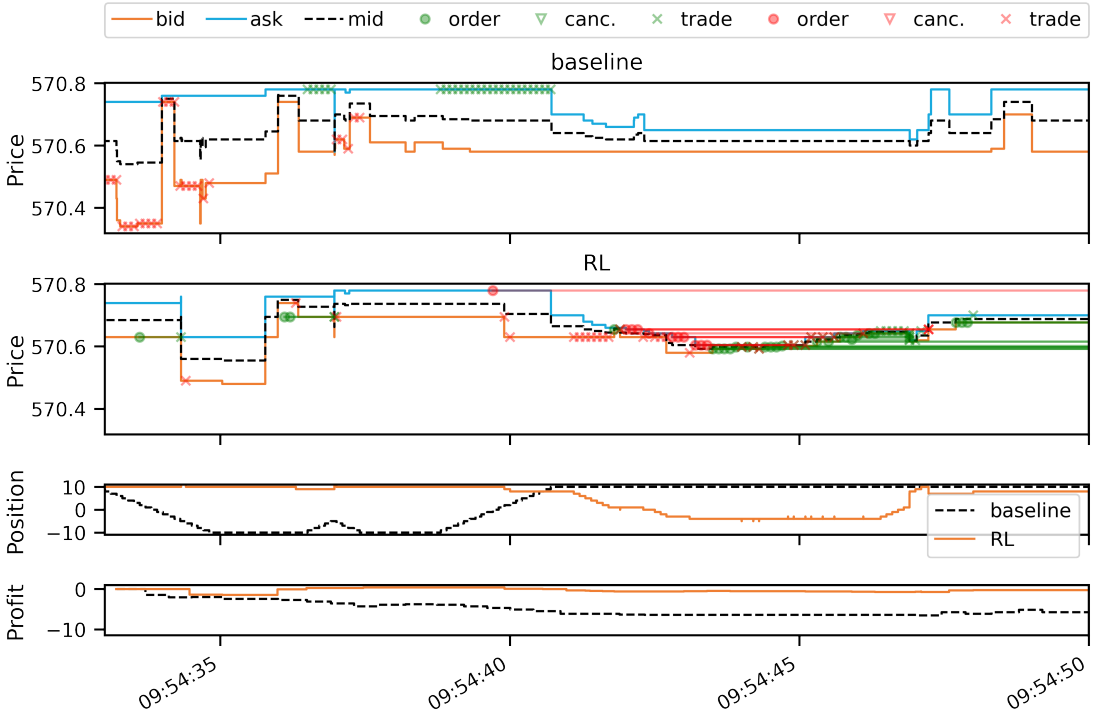


Figure 3.2: **A short snapshot of simulation results** (AAPL on 2012-06-14), comparing the RL policy (second panel) with the baseline (first panel). The first two panels plot the best bid, ask, and mid-price, overlaying trading events of buy orders (green) and sell orders (red). Circles mark new unmarketable limit orders entering the book. Crosses mark order executions (trades) and triangles order cancellations. Open orders are connected by lines to either cancellations or trades. Since we are simulating the entire LOB, trading activity can be seen to affect bid and ask prices. The third panel plots the evolution of the inventory position of both strategies, and the last panel the trading profits over the period in USD.

which connect the orders to their execution or cancellation, some of which occur after the depicted period.

Oracle Signal

The RL agent receives a noisy oracle signal of the mean return $h = 10$ seconds into the future (see equation 3.2.4). It chooses an action every 0.1s, allowing a sufficiently quick build-up of long or short positions using repeated limit orders of single stocks. The algorithm is constrained to keep the stock inventory within bounds of $[pos_{min}, pos_{max}] = [-10, 10]$. To change the amount of noise in the signal, we vary the a^H parameter of the Dirichlet distribution, keeping $a^L = 1$

constant in all cases. To keep the notation simple, we hence drop the H superscript and refer to the variable Dirichlet parameter a^H simply as a. We consider three different noise levels, parametrising the Dirichlet distribution with $a = 1.6$ (low noise), $a = 1.3$ (mid noise), and $a = 1.1$ (high noise). A fixed return classification threshold $k = 4 \cdot 10^{-5}$ was chosen to achieve good performance of the baseline algorithm, placing around 85% of observations in the up or down category. The signal process persistence parameter is set to $\phi = 0.9$.

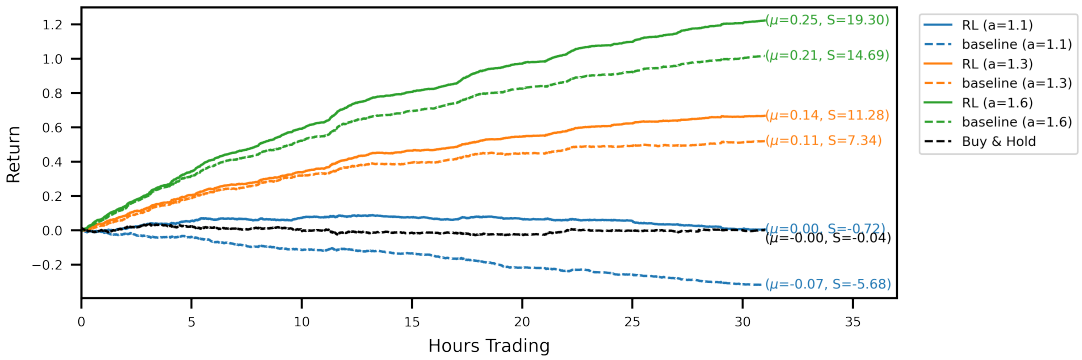
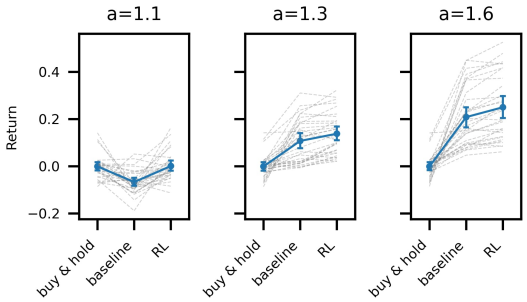
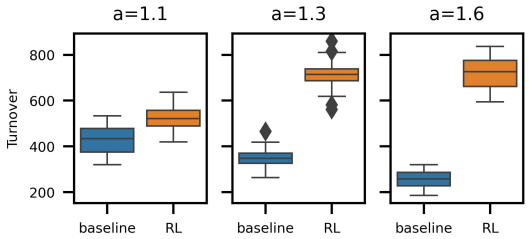


Figure 3.3: **Account curves**, trading the noisy oracle signal in the test set, comparing the learned RL policies (solid lines) with the baseline trading strategy (dashed). The black line shows the performance of the buy & hold strategy over the same period. Different colours correspond to different signal noise levels. The RL policy is able to improve the trading performance across all signal noise levels.

Out-of-sample trading performance is visualised by the account curves in figure 3.3. The curves show the evolution of the portfolio value for a chronological evaluation of all test episodes. Every account curve shows the mean episodic log-return μ and corresponding Sharpe ratio S next to it. We show that all RL-derived policies are able to outperform their respective baseline strategies for the three noise levels investigated. Over the 31 test episodes, the cumulative RL algorithm out-performance over the baseline strategy ranges between 14.8 ($a = 1.3$) and 32.2 ($a = 1.1$) percentage points (and 20.7 for $a = 1.6$). In the case of the signal with the lowest signal-to-noise ratio (a=1.1), for which the baseline strategy incurs a loss for the test period, the RL agent has learned a trading strategy with an approximately zero mean return. Temporarily, the strategy even produces positive gains. Overall, it produces a sufficiently strong performance to not lose money while still trading



(a) **Episodic mean strategy return** of buy & hold, baseline, and RL strategies for high ($a = 1.1$), mid ($a = 1.3$), and low noise ($a = 1.6$) in 31 evaluation episodes. The grey dashed lines connect mean log returns across strategies for all individual episodes. The blue line connects the mean of all episodes with 95% bootstrapped confidence intervals.



(b) **Turnover per episode:** comparison between baseline and RL strategy. Lower noise results in a more persistent signal, decreasing baseline turnover, but a higher quality signal, resulting in the RL policy to increase trading activity and turnover.

Figure 3.4: **Mean return and turnover** of the baseline and RL trading strategies.

actively and incurring transaction costs. Compared to a buy-and-hold strategy over the same time period, the noisy RL strategy similarly produces temporary out-performance, with both account curves ending up flat with a return around zero. Inspecting Sharpe ratios, we find that using RL to optimise the trading strategy is able to increase Sharpe ratios significantly. The increase in returns of the RL strategies is hence not simply explained by taking on more market risk.

Figure 3.4a compares the mean return between the buy & hold, baseline, and RL policies for all out-of-sample episodes across the three noise levels. A single dashed grey line connects the return for a single test episode across the three trading strategies: buy & hold, baseline, and the RL policy. The solid blue lines representing the mean return across all episodes. Error bars represent the 95% bootstrapped confidence intervals for the means. Testing for the significance of the differences between RL and baseline returns across all episodes (t-test) is statistically significant ($p \ll 0.01$) for all noise levels. Differences in Sharpe ratios are similarly significant. We can thus conclude that the high frequency trading strategies learned by RL outperform our baseline strategy for all levels of noise we

have considered.

It is also informative to compare the amount of trading activity between the baseline and RL strategies (see figure 3.4b). The baseline turnover decreases with an increasing signal-to-noise ratio (higher a), as the signal remains more stable over time, resulting in fewer trades. In contrast, the turnover of the RL trading agent increases with a higher signal-to-noise ratio, suggesting that the agent learns to trust the signal more and reflecting that higher transaction costs, resulting from the higher trading activity, can be sustained, given a higher quality signal. In the high noise case ($a = 1.1$), the RL agent learns to reduce trading activity relative to the other RL strategies, thereby essentially filtering the signal. The turnover is high in all cases due to the high frequency of the signal and the fact that we are only trading a small inventory. Nonetheless, performance is calculated net of spread-based transaction costs as our simulator adequately accounts for the execution of individual orders.

	a=1.1	a=1.3	a=1.6
action skipped [%]	24.5	43.8	7.8
sell levels (bid, mid, ask) [%]	(95.4, 3.1, 1.5)	(94.6, 2.8, 2.65)	(97.2, 1.9, 0.9)
buy levels (bid, mid, ask) [%]	(1.1, 1.3, 97.5)	(1.6, 52.9, 45.5)	(1.7, 13.0, 85.3)

Table 3.1: **Actions taken by RL policy** for the three different noise levels: the first row shows how often the policy chooses the "skip" action. Not choosing this action does however not necessarily result in an order being placed, e.g. if inventory constraints are binding. The last two rows show the relative proportion of limit order placement levels for sell orders, and buy orders, respectively.

Table 3.1 lists action statistics for all RL policies, including how often actions are skipped, and the price levels at which limit orders are placed, grouped by buy and sell orders. With the least informative signal, the strategy almost exclusively uses marketable limit orders, with buy orders being placed at the bid and sell orders at the ask price. With better signals being available ($a = 1.3$ and $a = 1.6$), buy orders are more often placed at the mid-quote price, thereby trading less aggressively and saving on transaction costs. Overall, the strategies trained on different signals all place the majority of sell orders at the best bid price, with the amount of skipped

actions varying considerably across the signals.

Robustness Evaluation on Recent Data

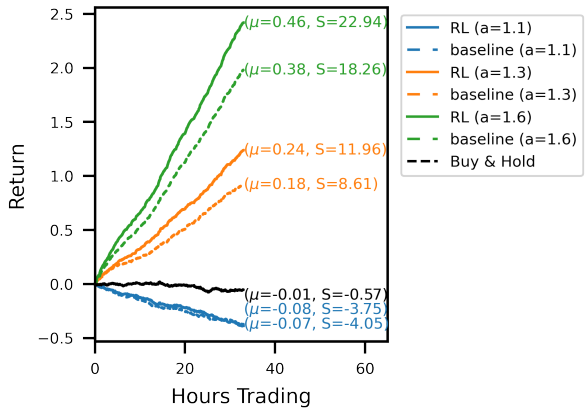


Figure 3.5: **Robustness check:** account curves for second evaluation period 2022-11-14 to 2022-12-31, comparing RL performance to baselines for three noise levels. Across all noise levels, the RL strategies produce a similar level of out-performance during this period than during the original period in 2012. For a high noise signal ($a = 1.1$), neither the baseline nor the RL policies are profitable. In this case, our RL strategy performs similarly to the baseline and does not learn to stop trading completely. This supports the hypothesis that similar micro-structure effects can be utilised in more recent periods, during a different macroeconomic landscape.

To evaluate our results on more recent LOB data, we also trained RL policies using 4.5 months of AAPL LOB data from the second half of 2022 (2022-07-01 to 2022-11-13) and evaluated on 1.5 months of held-out test data (2022-11-14 to 2022-12-31). Figure 3.5 shows account curves over the evaluation period for all 3 noise levels. For low ($a = 1.6$) and medium ($a = 1.3$) noise levels, the RL policies, again, beat the baselines by a significant margin and increase profits significantly. For a low-quality signal with high noise levels ($a = 1.1$), the RL policy performs similarly to the losing baseline strategy and also doesn't make a profit. Interestingly, the strategy also does not learn to stop trading altogether, which would represent a superior policy in this case. This could be due to a common problem with RL: effective exploration. The local optimum of not trading could not be found using an epsilon-greedy exploration policy in this case. Overall, even though macroeconomic and financial market conditions in 2022 differed markedly

from 2012, our results support the conclusion that the RL policies can utilise similar market micro-structure effects in both periods to improve the execution of a trading strategy based on a price forecast signal.

3.2.6 Conclusions

Using Deep Double Duelling Q-learning with asynchronous experience replay, a state-of-the-art off-policy reinforcement learning algorithm, we train a limit order trading strategy in an environment using historic market-by-order (MBO) exchange message data. For this purpose we develop an RL environment based on the ABIDES [Byrd et al. 2020] market simulator, which reconstructs order book states dynamically from MBO data. Observing an artificial high-frequency signal of the mean return over the following 10 seconds, the RL policy successfully transforms a directional signal into a limit order trading strategy. The policies acquired by RL outperform our baseline trading algorithm, which places marketable limit orders to trade into positions and passive limit orders to exit positions, both in terms of mean return and Sharpe ratio. We investigate the effect of different levels of noise in the alpha signal on the RL performance. Unsurprisingly, more accurate signals lead to higher trading returns but we also find that RL provides a similar added benefit to trading performance across all noise levels investigated.

The task of converting high-frequency forecasts into tradeable and profitable strategies is difficult to solve as transaction costs, due to high portfolio turnover, can have a prohibitively large impact on the bottom line profits. We suggest that RL can be a useful tool to perform this translational role and learn optimal strategies for a specific signal and market combination. We have shown that tailoring strategies in this way can significantly improve performance, and eliminates the need for manually fine-tuning execution strategies for different markets and signals. For practical applications, multiple different signals could even be combined into a single observation space. That way the problem of integrating different forecasts into a single coherent trading strategy could be directly integrated into the RL problem.

A difficulty for all data-driven simulations of trading strategies relying on market micro-structure is accurately estimating market impact. We address this partially by injecting new orders into historical order streams, thereby removing liquidity from the LOB. If liquidity at the best price is used up, this would automatically increase transaction costs by consuming deeper levels of the book. This mechanism accurately models temporary *direct* market impact, but cannot take account of the *indirect* or permanent component due to other market participants' reactions to our orders. By only allowing to trade a single stock each time step, we posed the problem in a way to minimise the potential effect indirect market impact would have on the performance of the RL strategy. The strategy trades small quantities on both sides of the book without accumulating large inventories. Such trading strategies are *capacity* constrained by the volume available in the book at any time and belong to a different class than *impact* constrained strategies, which build up large inventories by successively submitting orders in only one direction. Furthermore, we measure trading performance relative to a baseline strategy, which makes the same assumptions on market impact. However, accurately modelling the full market impact of high-frequency trading in LOB markets in a data-driven approach is an interesting direction for future research and would allow evaluating strategies with larger order sizes. Recent attempts in this vein have used agent-based models [Byrd et al. 2020] or generative models [Coletta et al. 2021; Coletta et al. 2022; Coletta et al. 2023].

We chose to focus our investigations in this paper on AAPL stock as a challenging test case of a small-tick stock, i.e. one where the minimum tick size is small relative to the stock price, with a high trading volume. Showing that we can train an RL agent to improve the profitability of an alpha signal in this example, indicates that similar performance improvements could be possible in larger-tick stocks with less trading activity. Although results are limited to a single company due to computational constraints, functional relationships in the micro-structure of the market have been found to be stable over time and across companies in prior work. In a large-scale study of order flow in US equity LOBs J. Sirignano and Rama Cont

[2019] found a *universal* and *stationary* relationship between order flow and price changes, driven by robust underlying supply and demand dynamics. Similarly, supervised training of deep neural networks to predict the mid-price direction a few ticks into the future has been shown to work for a wide range of stocks [Zihao Zhang et al. 2019a]. In contrast to lower-frequency trading strategies, whose performance often varies with market conditions, such as the presence of price trends or macroeconomic conditions, high-frequency strategies do not suffer the same degree of variability. Nonetheless, a systematic investigation of potential changes in LOB dynamics due to crisis periods or rare events could be an interesting avenue for future research.

While we here show an interesting use case of RL in limit order book markets, we also want to motivate the need for further research in this area. There are many years of high-frequency market data available, which ought to be utilised to make further progress in LOB-based tasks and improve RL in noisy environments. This, together with the newest type of neural network architectures, such as attention-based transformers [Vaswani et al. 2017; Child et al. 2019], enables learning tasks in LOB environments directly from raw data with even better performance. For the task we have considered in this paper, future research could enlarge the action space, allowing for the placement of limit orders deeper into the book and larger order sizes. Allowing for larger sizes however would require a realistic model of market impact, considering the reaction of other market participants.

End of Published Paper

3.3 Discussion and Integration

One limitation of our historical-data-based simulation is that *indirect* market impact, i.e. the reaction of other market participants, is not modelled, as the simulator replays a static historical message stream. Orders therefore do not induce any permanent market impact. Chapter 5 addresses this limitation by introducing generative models to simulate market impact, enabling the generation of alternative future rollouts and supporting a frequentist distributional simulation approach. This research could ultimately support the use of generative models as environment simulators.

To build on existing work, we began by developing a simulation environment based on the ABIDES codebase [Byrd et al. 2020]. At the time, ABIDES-gym [Amrouni et al. 2022] had not yet been released, so our study uses a custom implementation. Although implementation details differ, both approaches serve the same purpose: they wrap the ABIDES simulator in an OpenAI Gym [Brockman et al. 2016] interface, enabling interaction with RL algorithms.

Beyond contributing to the quantitative finance literature by demonstrating the potential of RL to automate parts of the LOB trading pipeline, this project also yielded key insights for subsequent research. We found that the ABIDES-based simulator lacked the speed and parallelism required for more compute-intensive RL applications. Its fragmented, object-oriented codebase further complicates extensions, particularly for integrating generative models or enabling multi-agent RL. These limitations motivated the development of an alternative simulation framework, JAX-LOB, introduced in Chapter 4.

To address the relative sample inefficiency of RL algorithms, we employed APEX [Horgan et al. 2018], which uses asynchronous CPU-based parallelisation via multithreading of independent environment instances. Each environment generates trajectories – sequences of observations, actions, and rewards – that are stored in a central buffer for sampling training batches. This setup is constrained by the number of CPU threads that can be effectively parallelised and by the over-

head of transferring batched trajectory data from system RAM to GPU memory, where policy inference and updates are typically performed using neural networks. Recognising these bottlenecks motivated the design of a GPU-native implementation with JAX-LOB.

As in much applied RL research, we found that achieving satisfactory performance on our LOB task is highly dependent on hyperparameter choices. While this does not preclude practical use of learned policies in real markets, it significantly increases the computational cost of training across the hyperparameter search space. Future work on developing more robust RL algorithms for long-horizon stochastic tasks is therefore warranted and could yield benefits beyond financial applications.

One limitation discussed in the paper is the reliance on a synthetic signal. While this choice supports the goal of demonstrating RL's role within a specific part of the trading pipeline, an end-to-end deep learning approach could aim to span the entire process from raw data to trading decisions. However, we found that APEX [Horgan et al. 2018] and PPO [Schulman et al. 2017] were unable to extract profitable trading signals from historical LOB data independently. Likewise, early experiments using the DeepLOB [Zihao Zhang et al. 2019b] architecture as a value-function network failed to yield successful trading strategies. Future research could address this by exploring supervised pre-training or representation learning prior to applying RL.

3.4 Author Contributions

In the previous paper "Asynchronous Deep Double Dueling Q-learning for Trading-Signal Execution in Limit Order Book Markets," Peer Nagy performed the experiments and wrote the manuscript with input from all authors. All authors devised the project, the conceptual ideas, and the design of the experiments. All authors contributed to the article and approved the submitted version.

Chapter 4

Simulating Limit Order Books

4.1 Preface and Contextual Overview

Following the work on RL trading in limit order books (LOBs) presented in Chapter 3 [Nagy et al. 2023a], it became evident that the limitations of existing simulation tools constrained research progress. In particular, the *Agent-Based Interactive Discrete Event Simulation* (ABIDES) platform [Byrd et al. 2020; Amrouni et al. 2021] used in prior work is a general-purpose discrete event simulator, with LOB simulation as a special case. While flexible, its fragmented Python-native implementation poses challenges in terms of extensibility and runtime efficiency. To overcome these issues, we developed a new LOB-specific simulator designed for fast, parallelisable machine learning applications. The resulting platform, *JAX-LOB* [S. Y. Frey et al. 2023], is natively implemented in the JAX framework [Bradbury et al. 2018], leveraging just-in-time (JIT) compilation and hardware acceleration, particularly on GPUs.

At a high level, JAX can be viewed as an extension of the widely used `NumPy` library [C. R. Harris et al. 2020], augmenting it with just-in-time (JIT) compilation for CPU, GPU, and TPU, as well as automatic differentiation capabilities. JAX JIT operates by tracing a native Python function on its first call and converting it into an intermediate representation known as *jaxpr*. This representation is strictly

typed and purely functional, imposing constraints on the permissible Python control flow. In particular, functions must be free of side effects and adhere to a functional programming paradigm. Additionally, array shapes must be static, precluding dynamic resizing within JIT-compiled code and requiring recompilation when invoked with different data types or array shapes. Standard Python control flow constructs, such as conditionals and loops, cannot be dynamically traced. To address this, JAX provides dedicated functional control flow primitives that enable these patterns within the JIT compilation framework[1].

The result of addressing these limitations is a LOB simulator that runs natively on the GPU, thereby reducing communication overhead between system and GPU memory. It supports parallel execution across hundreds or even thousands of LOB instances and integrates seamlessly into the broader JAX ecosystem. This ecosystem includes libraries for deep learning (Flax) [Heek et al. 2024], reinforcement learning (Gymnax, PureJaxRL) [Lange 2022; C. Lu et al. 2022], multi-agent reinforcement learning (JAX-MARL) [Rutherford et al. 2024], and probabilistic programming (NumPyro) [Phan et al. 2019], enabling efficient development of complex, high-performance machine learning workflows.

JAX-LOB natively supports LOBSTER data [R. Huang and Polak 2011][2], one of the most widely used LOB datasets for NASDAQ equities. LOBSTER provides level-3 (L3) market-by-order (MBO) data, which includes new limit orders, partial and full order cancellations, and order executions, for a configurable number of n price levels. While the dataset is comprehensive and pre-cleaned, it exhibits certain limitations when used for LOB simulation.

Specifically, when the volume of the "invisible levels", beyond the observed n levels, moves into the "visible range", the data set does not contain dedicated message events. This can introduce inconsistencies when reconstructing the book state from a long sequence of message events, compared to the true book state at that time.

[1] Further details are available in the official JAX documentation: https://docs.jax.dev/en/latest/jit-compilation.html.

[2] https://lobsterdata.com/

Furthermore, execution messages do not specify whether the initiating order was an *immediate-or-cancel* (IOC), *market*, or *limit* order. This lack of information necessitates assumptions in counterfactual simulations – for instance, how to interpret an execution message in an alternative LOB state where the relevant price level does not exist. We address this by converting execution messages into limit orders at the original execution price within the simulator. Lastly, as is typical for MBO data, LOBSTER does not provide any agent or agent-type identifiers, limiting its applicability for multi-agent learning tasks.

The following paper has been published in the *Proceedings of the Fourth ACM International Conference on AI in Finance*, 2023 [S. Y. Frey et al. 2023] and has received the *Best Paper Award* at the conference.

4.2 JAX-LOB: A GPU-Accelerated Limit Order Book Simulator to Unlock Large Scale Reinforcement Learning for Trading

Sascha Frey[*1], Kang Li[*2], Peer Nagy[*3,4], Silvia Sapora[4], Chris Lu[4], Stefan Zohren[3,5], Jakob Foerster[4], and Anisoara Calinescu[1,5]

[1]Department of Computer Science, University of Oxford

[2]Department of Statistics, University of Oxford

[3]Oxford-Man Institute of Quantitative Finance, University of Oxford

[4]Foerster Lab for AI Research, University of Oxford, University of Oxford

[5]The Alan Turing Institute

Abstract

Financial exchanges across the world use limit order books (LOBs) to process orders and match trades. For research purposes it is important to have large scale efficient simulators of LOB dynamics. LOB simulators have previously been implemented in the context of agent-based models (ABMs), reinforcement learning (RL) environments, and generative models, processing order flows from historical data sets and hand-crafted agents alike. For many applications, there is a requirement for processing multiple books, either for the calibration of ABMs or for the training of RL agents. We showcase the first GPU-enabled LOB simulator designed to process thousands of books in parallel, whether for identical or different securities, with an up to 75x faster per-message processing time. The implementation of our simulator – JAX-LOB – is based on design choices that aim to best exploit the powers of JAX without compromising on the realism of LOB-related mechanisms. We integrate JAX-LOB with other JAX packages, to provide

[*]These authors contributed equally to this work.

an example of how one may address an optimal execution problem with reinforcement learning, and to share some preliminary results from end-to-end RL training on GPUs. The project code is available on GitHub [1].

4.2.1 Introduction

Markets, i.e. matching buyers and sellers while determining a price, are a crucial component of modern economies. In many instances these markets rely on *auction mechanisms* for their operation. At the most basic level, for a single good sold at a specific time, these mechanisms allow all buyers to state how much they are willing to pay for this good, and the good then goes to the highest bidder.

In contrast, in financial markets the price finding process usually needs to happen *continuously* and for *any number* of shares during the operating hours of the market to ensure liquidity. To accomplish this, the limit order book (LOB) [Gould et al. 2013] mechanism is used in most modern electronic exchanges as a price finding tool.

At a high level, LOBs implement an *any-time* auction mechanism that allows all market participants to submit *bid* and *ask* orders specifying a *quantity* of stocks and a *price* at which they are willing to trade. As soon as there are compatible bid and ask orders, i.e. where the buying price of one order matches or exceeds the selling price of another order, a trade happens.

Due to their central role in the financial system, the ability to accurately and efficiently model LOB dynamics is extremely valuable. For example, it might allow a financial company to offer better services or may enable the government to predict the impact of financial regulation on the stability of the financial system.

Practically, there are a number of possible scientific approaches for taking advantage of such an LOB simulator, including agent-based models, reinforcement learning (RL), and generative models [Nagy et al. 2023b].

[1] https://github.com/KangOxford/jax-lob

However, due to the low signal-to-noise ratio of financial data a common downside of these approaches is the need for large scale simulations that take advantage of modern compute hardware and parallelism. Addressing this, **a GPU accelerated LOB simulator (JAX-LOB) is the core contribution** of our paper. One specific task of interest is the execution of orders by brokers or funds, where the goal is to sell/buy a specific quantity of stocks over a given time frame at the best possible price. This *optimal execution problem* is both a motivation and an example use-case for the work presented in this paper.

In Section 4.2.3 we provide a brief overview of the LOB and the trade execution problem, and in Section 4.2.2 we consider related works both in building LOB simulators and addressing the execution problem with reinforcement learning (RL). In Section 4.2.4 we present our contribution, the first GPU-accelerated LOB simulator using JAX [Bradbury et al. 2018], and discuss the detailed workings of the LOB, both general and specific to our implementation. In Section 4.2.5 we lay the groundwork for the use of our simulator for RL, an application domain which benefits significantly from the parallelism enabled by our implementation.

In Section 4.2.5 we contribute to solving the optimal execution problem with RL by wrapping the JAX-LOB simulator in a JAX-native *gymnax* [Lange 2022] execution environment. In our environment, both the experience rollout (i.e. agents interacting with the world to collect data) and the learning updates (i.e. agents training on the data) are conducted on the same GPU which avoids the GPU-CPU communication bottleneck. Finally, in Section 4.2.6 we integrate the environment with *PureJaxRL* [C. Lu et al. 2022] and use recurrent proximal policy optimization (PPO) to train an execution agent as an example application. Lastly, we compare the computational speed with equivalent CPU based implementations and show that the maximum speedup occurs when applying JAX-LOB to an RL task, taking full advantage of the parallelism.

Our main contributions are as follows:

1) The first GPU-accelerated LOB simulator, JAX-LOB,

2) At least 75x speedup in training an RL agent, compared to equivalent CPU-based LOB simulators: 5555 versus 74 steps per second during training, on the same hardware (Section 4.2.6).

We have open-sourced JAX-LOB incl. the RL environments and strongly believe it will unlock a number of new research avenues for the community.

4.2.2 Related Literature

There are several implementations that reconstruct the mechanics of the LOB [Byrd et al. 2020; Belcak et al. 2022; Jerome et al. 2022a; Paulin 2019; Casgrain 2020]. The implementations may be further subdivided by the application they are used for: agent-based models (ABMs) and market replay. Frameworks built for ABMs, such as the agent-based interactive discrete event simulation (ABIDES) framework [Byrd et al. 2020] or the multi-agent exchange environment (MAXE) [Belcak et al. 2022] implement a heterogeneous set of agents to submit fictional orders to the order book. ABIDES is implemented in standard Python and has an extension, ABIDES-gym [Amrouni et al. 2022], which creates a gym-like interface for reinforcement learning. MAXE is implemented in C++ and benefits from increased speed due to compilation [Belcak et al. 2022]. Similarly, the Economic Simulation Library (ESL) [M. Scholl 2022] is implemented in C++ and reports very fast per-message processing speeds for serial tasks. Our simulator is, like ABIDES [Byrd et al. 2020] and MAXE [Belcak et al. 2022], based on discrete events in the sense that individual messages are processed sequentially. However, the processing of these messages is done in vector form across multiple instances in parallel. In market replay models, such as the reinforcement learning for market making (RL4MM) framework [Jerome et al. 2022a], historical orders based on message data, such as the "limit order book system - the efficient reconstructor" (LOBSTER) [R. Huang and Polak 2011] data set are submitted. The primary limitation of ABMs is the difficulty in creating sufficiently realistic agent behaviors. Despite significant recent efforts [Vyetrenko et al. 2020], this remains a limitation, especially when compared to replayed market data, which directly replicates the

data distribution. However, the strength of ABMs is also a key limitation of market replay [Jerome et al. 2022b]. The implementation of strategic agents results in a dynamically reactive order flow based on outside intervention (e.g. a RL agent), whereas replayed historical order flow will remain unchanged. JAX-LOB, while originally designed to process data from LOBSTER [R. Huang and Polak 2011], may process messages from any source, including strategic agents. In contrast to the discussed implementations, JAX-LOB runs on the GPU, allowing for large-scale parallelization.

Ning et al. [2021] use double deep Q Networks (DDQN) [Van Hasselt et al. 2016] to solve the optimal execution problem using only market orders. These are, however, generally sub-optimal as trades may "walk the book", i.e. consume the entire available volume on one or more levels of the LOB, and eliminate potential gains from the spread by using passive limit orders. Furthermore, market impact is stylized by using a penalty function for larger orders. In this setting, they outperform the time-weighted average price (TWAP) benchmark in 7 out of 9 evaluated stocks. Dabérius et al. [2019] compares the use of DDQN [Van Hasselt et al. 2016] and PPO [Schulman et al. 2017] for optimal execution using only market orders in a stochastic model of price dynamics without price impact. In building a LOB simulator, we provide the tools to move away from having to stylize the market impact and create an environment that allows for the submission of limit orders at various prices, thus avoiding the risk of inadvertently consuming volume on multiple levels of the book. Nevertheless, the exclusive use of historical LOB data means that only direct market impact is accounted for. Such historical messages have no strategic behavior that may react to an RL agent's actions.

Addressing this drawback, Karpe et al. [2020] uses DDQN in an agent-based simulation environment based on the ABIDES simulator [Byrd et al. 2020], which replays real LOB message data while adding additional heuristic agents, which react to changes in the LOB state caused by the RL agent. Their simulation setup allows addressing optimal execution and optimal level placement jointly. They find that their agent converges to a TWAP strategy. While additional agents al-

low for more realistic market impact in principle, the ABIDES heuristics use a somewhat simplistic momentum strategy based on aggregate summary statistics, instead of a learned rational strategy. Similarly, Fang et al. [2022] use PPO to address the combined optimal execution and placement problem using the approach of replaying historical data in the ABIDES [Byrd et al. 2020] simulator. Their agent learns placement of up to three levels of the LOB concurrently. They use a large number of engineered features, including technical indicators, an attention-based network architecture, dubbed Dual-Window Denoise PPO, and a reward formulation based on execution cost and an imitation term, steering the policy towards a TWAP policy. Results indicate a potential improvement over TWAP but suffer from high standard errors. The environment we set up aims to move away from such engineered feature spaces and reward functions, and focuses on using recurrent neural networks exclusively to automatically extract features and account for the longer-term memory of the agent.

4.2.3 Background

The LOB, the underlying data structure and associated mechanisms which power modern electronic exchanges, has been studied extensively [Gould et al. 2013; Bouchaud et al. 2018]. The LOB is the collection of all orders to buy or sell a security that have yet to be matched. Two types of orders may be submitted to the LOB at any time: *limit orders* which require the specification of a price and a quantity, and *market orders*, which require only a quantity. Traders are free to submit orders at any time, at which point they are either matched with a compatible opposite order in the book, or added to the book – in the case of a limit order. Further details of the LOB are discussed in Section 4.2.4, together with details specific to our implementation. LOB simulators have a number of application domains, including generative models [Nagy et al. 2023b]; in this paper we focus on their use to train trading agents by using RL for trading tasks such as market making [Beysolow II and Beysolow II 2019; Ganesh et al. 2019] or trade execution [Nevmyvaka et al. 2006b].

JAX [Bradbury et al. 2018] is an accelerator agnostic framework that enables just-in-time (JIT) compilation using accelerated linear algebra (XLA) [Google 2023], automatic differentiation, and automatic vectorization which can easily be executed on the GPU. The framework is designed for high-performance machine learning research and forms the basis of both the *gymnax* [Lange 2022] and *PureJaxRL* [C. Lu et al. 2022] frameworks which we leverage, using optimal trade execution as an example problem. JAX has also been used for other applications where massive parallelism is advantageous, such as fluid dynamics [Kochkov et al. 2021] or molecular dynamics [Schoenholz and Cubuk 2020]. Whilst the focus of this paper is on the JIT compilation and GPU vectorization, the differentiation of LOB models for calibration [Quera-Bofarull et al. 2023] is a topic that is thus far underexplored in the literature.

Optimal trade execution is a well-studied problem in finance, with traditional approaches using methods from stochastic optimal control and analytic market-impact functions [Bertsimas and Lo 1998; Almgren and Chriss 2001; Kyle 1985; Obizhaeva and J. Wang 2013] to derive optimal policies. The aim of optimal trade execution is to buy or sell a specified number of stocks in a specified time frame, obtaining the best possible average price at low risk. This usually requires minimizing the cost due to market impact, especially when the trades are large compared to the liquidity of the market [Almgren and Chriss 2001].

4.2.4 The JAX-LOB Simulator

Some of the key challenges in applying deep RL to trade execution and other high-frequency tasks are the low signal-to-noise ratio, the risk of overfitting to specific training days, and building a simulator with realistic market impact. A straightforward way to compensate for the first two issues is to increase the number of state-action transitions available for training. To speed up the generation of observations of faithful LOB representations using high-frequency data we use JAX [Bradbury et al. 2018]. Further, JAX-LOB simulates direct market impact by reproducing the LOB, but does reproduce the strategic reactions by other market

participants to changes in the LOB state. Reproducing this indirect market impact is a more difficult problem which is out of the scope of this paper, but generative models for order-flow [Nagy et al. 2023b] may be an applicable tool.

We make several design decisions to address some of the constraints of JIT-compiled JAX code:

- Pure functions without side effects (e.g. global variables)
- Fixed array size and type
- Control-flow that can be compiled and parallelized efficiently (See Section 4.2.4).

Most CPU implementations of LOBs are based on hash-maps, queues, doubly-linked lists and sorted dictionaries [Belcak et al. 2022; Byrd et al. 2020; Jerome et al. 2022a] which enable quick access to data and maintain sorting of the book throughout. Given the JAX requirement of compile-time fixed-size arrays, implementing a similar structure means memory must be pre-allocated for all price-levels and orders. The use of arrays means that re-ordering upon removal of entries is far more costly than with linked lists. We, therefore, opt for an architecture that does not use a tree-like structure, and also do not keep orders sorted. Instead, we define two arrays $\mathcal{A}$ and $\mathcal{B}$ to represent the sides of the order book containing all active ask and bid orders, respectively.

$$\mathcal{A} = [\boldsymbol{a_1}, ..., \boldsymbol{a_N}] \quad \mathcal{B} = [\boldsymbol{b_1}, ..., \boldsymbol{b_N}] \tag{4.1}$$

$$\boldsymbol{o_i} = [P_i, Q_i, OID_i, TID_i, Ts_i, Tns_i] \in \mathcal{A} \cup \mathcal{B}, \quad i \in [1, N] \tag{4.2}$$

Each side of the book has a fixed capacity for N orders, where each order $\boldsymbol{o}$ has six features (4.2): Price P, Quantity Q, Order ID OID, Trader ID TID, Time (seconds) Ts, and Time (nanoseconds) Tns. Empty positions in $\mathcal{A}$ or $\mathbf{B}$ are indicated by setting all features to -1. The size N must be chosen such that the book is not saturated during a given experiment.

Basic Operations

There are three operations that can be applied to either side of the order book; the *addition* of a new order, the *cancellation* of an existing order, and the *matching* of an existing order with an incoming order on the other side of the book followed by its removal from the book.

1. The addition of an order requires the identification of an empty position ($\boldsymbol{o_i} = \boldsymbol{-1}$) in the array, and the insertion of the order-specific data into the correct fields.

2. Cancellation requires locating the order ID (OID) of the order to be canceled and removing the corresponding quantity from the book.

3. For an order to match with another, it needs to be "aggressive", meaning the bid (ask) price is higher (lower) than the best ask (bid) price in the book, such orders are said to be *crossing the spread*. During the matching operation, an incoming aggressive order, denoted $\boldsymbol{o_a}$, is matched against an existing order on the other side of the book, the standing order $\boldsymbol{o_s}$. The matching logic aims to find the remaining quantities for both the aggressive ($Q_a \in \boldsymbol{o_a}$) and standing ($Q_s \in \boldsymbol{o_s}$) orders using the following operations

$$Q'_s = max(0, Q_s - Q_a), \quad Q'_a = Q_a - Q_s.$$

When two orders are matched, a trade $\boldsymbol{t_j}$ is logged (4.3).

$$\boldsymbol{t_j} = [P_j, Q_j, OID_{a,j}, OID_{s,j}, Ts_j, Tns_j] \tag{4.3}$$

Whereby:

$$
\begin{aligned}
P_j &= P_s \\
Q_j &= Q_s - Q'_s \\
OID_{a,j} &= OID_a \in \boldsymbol{o_a} \\
OID_{s,j} &= OID_s \in \boldsymbol{o_s} \\
Ts_j, Tns_j &= Ts_a, Tns_a
\end{aligned}
$$

Up to N trades are logged in a fixed-size array $\mathcal{T}$ due to constraints of the XLA compiler (4.4).

$$\mathcal{T} = [\boldsymbol{t_1}, ..., \boldsymbol{t_N}]^T \tag{4.4}$$

Again, $\boldsymbol{t_j} = \boldsymbol{-1}$ indicates an empty slot.

For all operations, upon completion, both *Asks* and *Bids* are checked for orders $\boldsymbol{o_i}$ where $Q_i \leq 0$, in which case we set $\boldsymbol{o_i} = \boldsymbol{-1}$.

A single incoming order may match with more than just one standing order. The matching logic contains a while loop that repeatedly attempts to match the incoming order against the next-best standing order on the opposite side of the book. The best standing order $Best(\boldsymbol{o_s})$ is defined by the price-time order priority algorithm, the most commonly used LOB matching algorithm [Gould et al. 2013]. The best order is the one with the ask price or bid price from (4.5). If multiple orders share this price, the one with the earliest arrival time is considered.

$$P_{ask} = \underset{o \in \mathcal{A}}{min}(P_i) \quad P_{bid} = \underset{o \in \mathcal{B}}{max}(P_i) \tag{4.5}$$

The while-loop continues as long as the book is non-empty, $Q_a > 0$, and the prices overlap as follows

$$
\begin{aligned}
P_a \leq P_s \qquad & \text{if marketable ask order,} \\
P_a \geq P_s \qquad & \text{if marketable bid order.}
\end{aligned}
$$

Based on the above descriptions it is intuitive to expect varying computational complexity across these basic operations. To validate this intuition, we time the basic operations in order books of different maximum capacities N (Table 4.1). The operations take longer with increasing capacity in the book N, and as expected, the slowest operation (*Match*) takes more than twice as long to execute as the fastest (*Cancel*).

Table 4.1: **Average run-time** for the three basic operations that may be applied to the LOB for different sizes N. The tests are conducted on an Nvidia 2080 Ti and averaged across 1000 serial trials and 10 experiments. The book is initially filled to one-third capacity with random orders. The standard error over 10 experiments is given.

Capacity N	Add (ms)	Cancel (ms)	Match (ms)
10	0.116 ± 0.003	0.087 ±0.002	0.178±0.004
100	0.138 ± 0.002	0.123 ±0.002	0.171±0.004
1000	0.172 ± 0.04	0.143 ±0.0005	0.167±0.001

The relative size of the incoming order $Q_a \in \boldsymbol{o_a}$ has a significant impact on the computation time for the matching operation. All else being equal, a larger quantity requires the consideration of more standing orders to match the arriving order, thus computing more iterations of the matching logic. A simple way to illustrate this is to consider an order book with capacity $N = 100$, filled to one-third capacity, and to submit market orders of varying sizes (Table 4.2). This increase in time is not drastic until particularly large orders are submitted, but shows the importance of constraining the action space to reasonable quantities to ensure quick execution.

Message Types

Which of the three basic operations (Section 4.2.4) is called depends on the type T of message $\boldsymbol{m}$ (4.6) passed to the order book.

$$\boldsymbol{m} = [T, S, Q, P, OID, TID, Ts, Tns] \tag{4.6}$$

Table 4.2: **Analysis of the effect of incoming order size on the time required to complete the matching operation.** As the size increases, more standing orders need to be considered to fully match the incoming order. The capacity for the book is set to $N = 100$. Testing is done on an Nvidia 2080 Ti GPU. The mean and standard error across 10 experiments is given.

Market Q_a	Time to match (ms)
0	0.207±0.003
10	0.338±0.002
500	0.334±0.003
1,000	0.328±0.006
10,000	0.321±0.002

The structure of data contained in messages $\boldsymbol{m}$ is:

1. T - The type of message: limit orders ($T = 1$) and cancel orders ($T = 2$).
2. S - The side of the order: bid ($S = 1$) or ask ($S = -1$).
3. P - The price at which the order should be submitted or the price of the order to be cancelled/deleted. This field is disregarded in the case of market orders.
4. Q - The size of the order, i.e. the quantity to buy/sell or the to remove from an existing order for cancel/delete orders.
5. OID - The unique order ID.
6. TID - The trader ID, an identification marking the source of the order. Used to identify orders from individual agents. This is not typically available in the data, but it enables agents to identify their own orders and is a useful tool for experiments from a regulatory point of view.
7. Ts, Tns - The time of receipt of the order, split into two fields representing the time as full seconds and fractional digits as nanoseconds represented by 32-bit integers.

While the reader familiar with the LOBSTER [R. Huang and Polak 2011] data set will recognize significant similarities to the message structure in this data set, they are not identical. We take the following steps to process orders of different

types:

1. Limit orders are first matched against the opposite side of the book, the remaining quantity is then added as a single new order to the corresponding side.

2. Cancel orders call a *cancellation* (Section 4.2.4).

Considering each type and side as eight separate cases allows defining explicit functions for each case, enabling the use of a single conditional statement upon receipt of the message, rather than multiple branches within the matching logic, which we find improves performance under `vmap` (Section 4.2.4). The compute time (Table 4.3) to process each of the three message types varies and is related to the basic operations required. When compared to the results in Table 4.1 the time is only marginally increased by the branching statement, and there is still a notable difference across different order types.

Table 4.3: **Time to process different message types.** The last column differs from the second column in that the price crosses the spread, requiring a matching operation. Tests are done on an Nvidia 2080 Ti GPU, the book capacity is $N = 100$. The mean and standard error across 10 experiments is given.

Capacity N	Limit Order (ms)	Cancel Order (ms)	Limit Order across book (ms)
10	0.175±0.001	0.149±0.0004	0.251±0.001
100	0.230±0.002	0.166±0.001	0.329±0.006
1000	0.254±0.011	0.174±0.001	0.372±0.008

Vectorising Map - `vmap`

The benefit of implementing the LOB in JAX is derived from parallelizing computation on the GPU. However, the processing of messages sent to the order book in a continuous double-auction is intrinsically a serial process: both the treatment of messages and the internal matching logic must be strictly ordered. This is in opposition to a classical auction with, say, a central clearing process [M. P. Scholl et al. 2021], or even stochastic models of the limit order book [Rama Cont et al.

2010a]. Therefore, to achieve parallelism we process multiple books in parallel using the `vmap` operator.

One particularity of the `vmap` operator is that it transforms a number of control-flow statements into `select` statements which execute all of the conditional branches at run time. In the case of limit order books, this means that each of the possible four cases is computed and the run-time is therefore limited by the slowest branch. This can be observed when looking at the timed results (Table 4.4) when processing the same message across $N_book = 1000$ identical order books in parallel. The time no longer varies across different types of messages but only depends on the capacity. The reason we avoid continuous sorting is that when removing orders in an array-based view, the entire array needs to be re-sorted. This order sorting is comparatively costly, which would not be problematic if called rarely. However, the aforementioned behavior of the `vmap` operator means that it would be called for every message, even if no order needs to be removed. Careful consideration of this behavior has enabled significant performance improvements during the development of JAX-LOB, but additional optimisation may still be possible.

Table 4.4: **Time to process different message types with `vmap`.** The difference to table 4.3 is that messages are processed by $N_Book = 1000$ books in parallel. Times measure the processing of a message in all books. The mean and standard error across 10 experiments is given.

Capacity	Limit Order (ms)	Cancel Order (ms)	Limit Order across book (ms)
10	1.15±0.021	1.12±0.001	1.14±0.027
100	1.66±0.013	1.64±0.008	1.61±0.015
1000	10.6±0.056	10.4±0.076	10.4±0.054

The resulting times per message type are compared to two CPU implementations (Table 4.5): one implementation of the LOB using RB trees and linked lists as well as a similar implementation named RL4MM [Jerome et al. 2022a] using numpy. The CPU environments are described in more detail in sections 4.2.5 and 4.2.6. When parallelizing to more than 1000 books, our JAX-LOB has a faster per-message processing time.

Table 4.5: **Processing times** for messages on the CPU order book implementation based on *linked list* and *RB trees*, and the implementation used in RL4MM [Jerome et al. 2022a] based on *numpy* arrays, and for the JAX-LOB implementation based on *jax arrays* with capacity $N = 100$ and $N_Books = 1000$ books in parallel. A key difference between the CPU implementation and that in RL4MM is that data is pre-loaded, as opposed to being accessed from a database. Both CPU implementations are tested on a single core of an Apple M1 chip and the JAX-LOB on an Nvidia 2080 Ti.

Order type	RL4MM Time (μs)	CPU Time (μs)	JAX-LOB Time (μs)
Limit	4.9	5.3	1.6
Cancel	16.1	3.6	1.6
Limit Cross	37.9	7	1.6

4.2.5 `Gymnax` Environments for the LOB

Our JAX-LOB is well suited for the *gymnax* reinforcement learning environment framework designed around JAX to produce parallel environment roll-outs on the GPU. In the following sections we briefly describe the construction of a general base environment that can be used to derive environments for various tasks, such as market-making, trade execution, or other intraday tasks. We further describe our example order execution environment, which we use in Section 4.2.6 to begin training a recurrent PPO-based RL agent.

Base Environment

The primary roles of the base environment are to load LOBSTER [R. Huang and Polak 2011] data for replay, provide an interface to the JAX order book, and set up a skeleton trading environment based on JAX-LOB.

Data Loading

We opt to pre-load message data in fixed-size, non-overlapping time windows (figure 4.1) to save time during execution. The number of messages per step is constant, with a variable time duration per step. Conversely, the total duration of all steps of a single time window is fixed, implying a variable maximum number of steps [Amrouni et al. 2022; Jerome et al. 2022a]. This allows the execution of

trades with a strict time constraint. To satisfy the fixed-size array constraints, additional steps (indicated by the blue borders in Figure 4.1) with all values set to zero are added. These steps are never processed as the episode termination condition is satisfied beforehand.

Further, the order book state (Level 2) of the first ten levels of the book at the beginning of each time window is used to initialize the book. Unlike the message data, this does not provide insight into whether price levels contain one or multiple orders, nor does it provide any order IDs. We work around this by assuming that there is exactly one order per price level containing the sum of the listed volumes. We use $OID_i \in [-inf, -9000]$ starting at -9000 and descending for these initial orders. We allow for the cancellation of these orders if a cancellation message satisfies $P_i = P_{cancel}$ and $OID_i < -9000$. Whilst this is a limiting assumption, it is due to the nature of the LOBSTER data set. The alternative is to only use the Level 2 state for the beginning of the day, and simulate the arrival of messages to obtain an order book state for the beginning of the remaining windows. However, due to the data set only including messages pertaining to the first 10 levels this would result in significant drift between the simulated and actual (albeit less granular) Level 2 order book state.

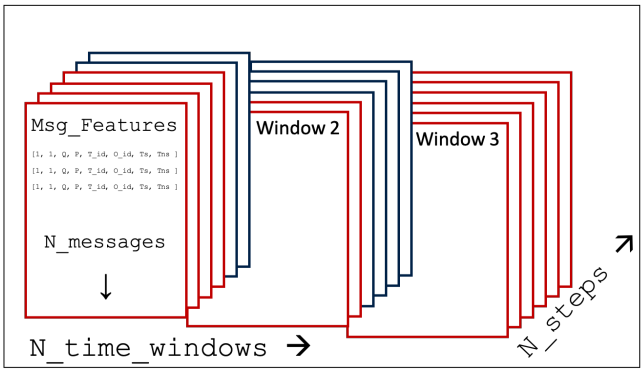


Figure 4.1: **Structure of data** loaded into the *gymnax* environments. In our experiments, each step processes `N_messages` = 100 messages. The number of `N_steps` is variable with a fixed time of 30 minutes but the data is padded with zeros (blue boxes). There are `N_time_windows` per trading day of data.

Core State-Space Requirements

The base environment defines the core features required in the state space, additional elements may be added for derived environments:

- Order book:
  - Ask side: Array($N_{orders} \times N_{features}$)
  - Bid side: Array($N_{orders} \times N_{features}$)
- Trades: Array($N_{trades} \times N_{features}$)
- Initial time: $[T_init_s, T_init_{ns}]$
- Current time: $[T_cur_s, T_cur_{ns}]$
- ID counter
- Window Index
- Step counter

The ID counter is used to generate orders submitted by the agent(s), and is incremented accordingly during an episode.

Functionality

We keep the description of the base environment brief, leaving consideration of the action and observation spaces, and the reward function to the execution environment in Section 4.2.5. The core functionality of the environment during a step is as follows:

1. The received actions (environment dependent) are transformed into messages to be processed by the JAX-LOB.
2. The data messages are obtained from the loaded data in the environment parameters using the window index and step counter variables (Section 4.2.5) for indexing.
3. The current time is updated based on the last data message.
4. All messages are processed by the JAX-LOB.

The episode terminates when the elapsed time is greater than the episode time.

Execution Environment

The execution environment extends the state space and defines the observation and action space, reward function, and a new termination condition.

Augmented State Space

The state space in 4.2.5 is augmented with intra-step data and agent-specific information:

- Level 1 data for every processed message from data since the last step:
  - Best ask price: Array($N_{messages}$)
  - Best ask volume: Array($N_{messages}$)
  - Best bid price: Array($N_{messages}$)
  - Best bid volume: Array($N_{messages}$)
- The initial mid-price $\frac{P_{ask}+P_{bid}}{2}$ at the beginning of the episode
- The size of the execution task
- The quantity executed thus far
- The total revenue due to execution thus-far

Observation Space

The state space is internal to the environment. The observation space is accessible by the RL agent and is designed to contain information that is accessible in real-life. The structure is a single one-dimensional array of size $N_{messages} \times 6 + 2 \times 2 + 4$ containing the following metrics:

- Best bid prices between steps: Array($N_{messages}$)
- Best ask prices between steps: Array($N_{messages}$)
- Mid-prices between steps: Array($N_{messages}$)

- The prices n ticks deep on the side of the book of the task between steps

$$P_{passive} = \left\{ \begin{array}{ll} P_{ask} + n \times ticksize, & \text{if } task \text{ is } sell \\ P_{bid} - n \times ticksize, & \text{if } task \text{ is } buy \end{array} \right\} \tag{4.7}$$

- The spreads between steps: $spread = P_{ask} - P_{bid}$: Array($N_{messages}$)
- The current time: $[T_s, T_{ns}]$
- The elapsed time in the episode: $[T_s, T_{ns}]$
- The initial mid price at the episode start: P_{init}
- The price drift: $P_{mid} - P_{init}$
- The size of the execution task: Q_{task}
- The quantity executed thus far: Q_{exec}
- The Level 1 imbalances between steps: $Q_{ask} - Q_{bid}$: Array($N_{messages}$)

Action Space

To have a rich set of actions whilst maintaining a reasonably simple learning task, we opt for a continuous action space of four dimensions. Similar to [Fang et al. 2022], at every step, the agent chooses the size of the order to place at four pre-defined prices:

1. The "far touch" price: the best price on the opposite side of the book

$$P_{far_touch} = \left\{ \begin{array}{ll} P_{bid} & \text{if } task \text{ is } sell \\ P_{ask} & \text{if } task \text{ is } buy \end{array} \right\}. \tag{4.8}$$

2. The Mid-price (Section 4.2.5).
3. The "near touch" price: best price on the side of the book of the task.
4. The "passive" price, as defined in Section 4.2.5.

Reward Function

We propose a reward function that combines the agent's ability to outperform the baseline volume-weighted average price (VWAP) [Cartea and Jaimungal 2016a] strategy and the ability to use predicted price trends. It is composed of two parts, the advantage and the drift. The former represents the advantage gained over the average price of execution, weighted by volume, during the step. The latter represents the effect of price movements of the average traded price with respect to the mid-price at the beginning of the episode. The parameter λ can be adjusted to weigh the effect of this drift. In equations (4.9) and (4.10), the indices i and j respectively refer to the set of executed trades during a step, and the set of executed trades of the RL agent.

$$R = \sum_j Q_j(P_j - P_{VWAP}) + \lambda \sum_j Q_j(P_{VWAP} - P_{init}) \tag{4.9}$$

$$P_{VWAP} = \frac{\sum_i Q_i P_i}{\sum_i Q_i}, \quad \sum_j Q_j < \text{ task size} \tag{4.10}$$

Termination Condition and Market Order Submission

In comparison to the base environment, the execution environment allows for an additional termination condition: the completion of the task, i.e. the execution of the desired quantity. In practice, the step function is modified such that this becomes the only termination condition, by submitting a market order for the remaining quantity one minute before the episode time ends.

Rollout Cost Comparison

Before considering the RL problem in Section 4.2.6, we consider the run-time improvements of the execution environment based on the JAX-LOB over comparable gym environments running on the CPU.

We compare our execution environment with the RL4MM [Jerome et al. 2022a] environment designed for market making, but with comparable core logic. All

orderbooks are run with data from January 2nd 2015 for ten levels for the stock TSLA. In the CPU and JAX-LOB environments, 100 messages are processed per step. RL4MM measures step duration in historical time, but we find that 100 messages are processed roughly every 2 minutes.

In the case of the JAX-LOB environment, we run different numbers of environments in parallel on an Nvidia 2080 Ti GPU, whilst the RL4MM and the CPU environments are run in series on an Apple M1 CPU. The improvement (Table 4.6) is roughly a factor 10 over the RL4MM environment and a factor 5 over our CPU implementation for 1000 parallel environments. Increasing the number of environments further is a feasible possibility, especially on GPUs with larger memory, but such economies of scale are eventually limited due to memory.

Table 4.6: **Average time per step** comparison between the RL4MM environment, our CPU-based *gym* environment, and the JAX-LOB based *gymnax* environment for a varying number of environments in parallel. The CPU environments are run on an Apple M1 processor and the *gymnax* environment is run on an Nvidia 2080 Ti.

-	RL4MM	CPU	*gymnax*		
N_Envs	-	-	10	1000	10000
Total time (s)	29.6	33.8	19.9	59.4	431
N_steps	5k	9k	5k	500k	5000k
Time/step (ms)	5.9	3.5	3.98	0.12	0.08

4.2.6 Using the `gymnax` Environment for Reinforcement Learning

Training Loop

We re-implement Recurrent PPO, as described in [Fang et al. 2022], to allow for a continuous action space in a partially observable setting. The full architecture of the actor and critic networks is given in Figure 4.2. For every network update step, 10 steps are collected across 1,000 environments to create a batch size of 10,000 transitions. For each of the four epochs per update step, this batch is subdivided into four mini-batches which are used to calculate the loss functions, take the

gradients and update the network through gradient descent using the Adam [D. Kingma and Ba 2015] optimizer implemented in optax [Babuschkin et al. 2020].

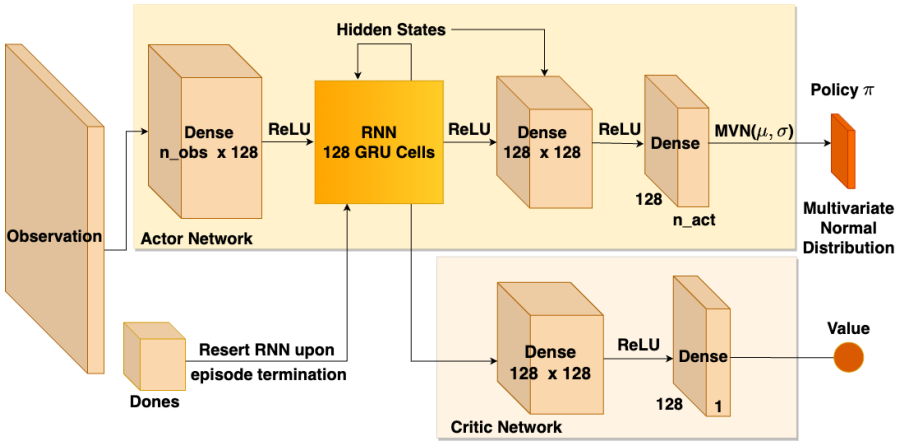


Figure 4.2: **The architecture of the RNN-PPO actor and critic networks.** The hidden states of the RNN layer add the necessary recurrence for the network to have memory of previous states. The Boolean termination values ("Dones") are required to reset the memory of the RNN at episode termination. The policy π is a multivariate normal distribution $mvn(\boldsymbol{\mu}, \boldsymbol{\sigma})$ whereby $\boldsymbol{\sigma}$ is a trained network parameter and $\boldsymbol{\mu}$ is the output vector from the final dense network layer.

JAX-Enabled Training Speedup

Whilst we do not have benchmarks for the exact same problem definition in other approaches, we nevertheless compare with the training speeds for another environment running on the CPU. This is in addition to the data we reported in Section 4.2.5 as the transfer of data from the CPU to GPU for the network update adds significant overhead to the run-time. With *gymnax* running on the GPU, this is not necessary and we observe further improvements, see Figure 4.3. The RL4MM package, which we use for comparison in Sections 4.2.4 and 4.2.5, is omitted in this comparison as we could not replicate fair experimental conditions due to issues arising with the use of the GPU. The CPU-based order book is based on a LOB simulator using the traditional tree-like structure where the orders and price-levels are implemented as queues. Further, the Stable-Baselines3 implementation of Recurrent PPO is used to train an execution task. Only the environment is run on the CPU, the updates to the network are, like with JAX-LOB, run on the GPU. There are a number of key variables which are common across both experiments

which we aim to control as far as possible to make the comparison equitable:

- *Messages processed per step* - 100 messages
- *Levels of LOBSTER data considered* - 10 levels
- *Episode length* - 30 minutes
- *Hardware* - GPU: Nvidia A40, CPU: 32 core AMD EPYC 7513, both experiments run network updates on the GPU.
- *Parallelization* - 1000 environments in parallel for JAX-LOB, 1000 vectorized environments for CPU but collected serially.
- *Batch size* - 10,000, 10 steps across 1000 environments
- *Mini-batch size* - 2,500, 4 per batch
- *Number of Epochs* - 4 epochs
- *Data* - April 2021 LOBSTER high frequency data for AMZN

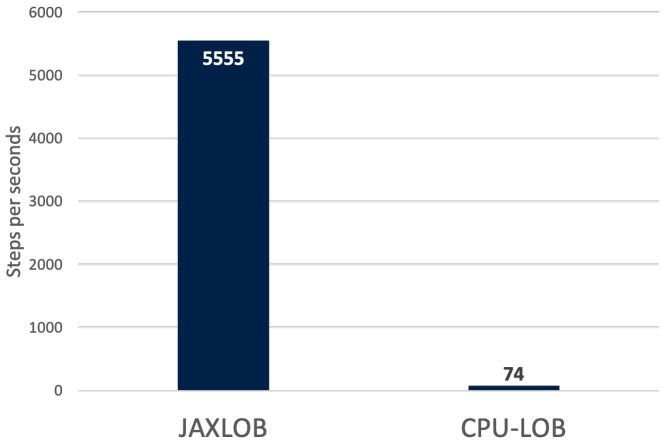


Figure 4.3: **Training steps per second** for gymnax environment based on JAX-LOB and our CPU gym environment implementation. The former is more than 75 times faster.

We observe a 75x speedup of the training loop based on the JAX-LOB simulator compared to that using the CPU implementation of the LOB. This is larger than the reported 30x speedup in Table 4.6. We attribute this to the lack of data transfer across processing units. This does not yet consider hyper-parameter tuning which would further benefit from parallelization, and is trivial to achieve with our fully JAX-based implementation.

4.2.7 Conclusions

We present the first implementation of a limit order book simulator for GPUs, called JAX-LOB. Using this as part of a *gymnax* environment for reinforcement learning we obtain at least a 30x speedup compared to a comparable CPU implementation. When using the environment to train a RL agent with *PureJaxRL* we even see a 75x speed-up with respect to our comparable CPU implementation. This speedup due to parallelization is expected to contribute to research in applying RL to high-frequency trading and execution problems that require a reactive LOB simulator.

The immediate next step is to use the simulator and training environment to effectively train an RL execution agent which exhibits converging, optimal, rewards in training as well as good out-of sample performance. Whilst we have been able to obtain some preliminary training runs, we do not report them as we are not confident in their convergence and performance at the time of writing and require additional hyper-parameter tuning and out-of-sample testing, where the parallel nature of JAX-LOB is expected to provide additional speedup, especially if running on multiple GPUs. Detailed experiments with respect to the action-space, feature-space, reward function, and network architecture will be required to train a robust and cost-effective RL execution agent. Further, the execution problem is a single application of high-frequency RL trading, with market making being another notable example which we plan to explore next.

End of Published Paper

4.3 Discussion and Integration

Section 4.1 outlined general limitations associated with JIT-compilable JAX code. In the case of *JAX-LOB*, these constraints motivated several innovative design choices to enable efficient parallelisation without incurring unnecessary overhead. The challenge of reconciling static array shapes with a dynamic number of orders in the book is addressed by padding both sides of the LOB to a fixed maximum capacity. Since GPU-based execution does not support traditional dynamic data structures such as queues, linked lists, or trees efficiently, the LOB is represented using unordered arrays, which are explicitly sorted only when necessary – e.g. during price–time priority matching. Furthermore, Python control flow is almost entirely replaced with JAX-native primitives, allowing *JAX-LOB* to be end-to-end JIT-compilable. Most loops are expressed using efficient fixed-length *scan* operations. However, the order-matching algorithm requires a `jax.lax.while_loop`, which precludes *reverse-mode* automatic differentiation through the simulator. Nonetheless, *forward-mode* differentiation, via Jacobian–vector products, remains fully supported. This could enable future applications such as efficient local sensitivity analysis for parametrised simulated agents.

The aforementioned limitations are particularly challenging in the context of LOB simulation, which typically involves extensive branching logic and dynamic data structures. This further highlights the practical significance of the speed-ups achieved in JAX-LOB. In the extant literature, there have been similar efforts to migrate simulation workloads to JAX to leverage hardware acceleration. Most of these focus on physics-based environments, such as *Brax* [Freeman et al. 2021], *Kinetix* with its custom `Jax2D` engine [Matthews et al. 2024], or the *MuJoCo playground* [Zakka et al. 2025]. One could argue that physics simulators are more naturally suited to differentiable and JIT-compatible implementations in JAX, as they generally involve smooth dynamics and relatively little branching logic. In contrast, the successful implementation of a performant LOB simulator – despite its inherent discrete and conditional structure – constitutes a meaningful and tech-

nically non-trivial contribution.

Section 4.2.7 discusses the training of reinforcement learning (RL) agents on a trade execution task using *JAX-LOB*. Applying RL to this problem introduces several challenges. Despite its theoretical appeal, deep RL has yet to convincingly and consistently outperform simple heuristic strategies, such as time-weighted average price (TWAP) execution, in a statistically significant and robust, data-driven manner [Fang et al. 2022; Ning et al. 2021]. While some promising results exist, RL-based execution policies often lack generalisation across different stocks, are typically evaluated only in-sample, and rely heavily on supervised pre-training of the network.

There are multiple reasons for these limitations. First, the expected performance gain over TWAP is generally small in magnitude, requiring large sample sizes to achieve statistical significance at conventional confidence levels (e.g. 95% or 99%). Second, our experiments using proximal policy optimization (PPO) [Schulman et al. 2017] with a deep S5-based policy network [J. T. Smith et al. 2022] indicate that RL agents struggle to learn effective feature representations from raw input data without manual feature engineering. Even when evaluating purely in-sample, where, in principle, an optimal execution schedule could be memorised, RL agents often fail to reach optimal performance, particularly over long episode horizons. In such cases, the policies regularly become stuck in local optima and do not trade at globally optimal prices.

Even more importantly, the theoretical benefit of reinforcement learning (RL) lies in its capacity to learn dynamic policies, thereby internalising the effects of trades on future market states. In contrast, simulated environments relying solely on historical data replays cannot achieve this, as actions within these simulations exert only a mechanistic, direct price impact. They disregard the crucial indirect impact stemming from other agents' responses. Addressing this challenge of dynamic price impact necessitates the use of generative models, which we discuss in Chapter 5.

4.4 Author Contributions

In the previous paper "JAX-LOB: A GPU-Accelerated Limit Order Book Simulator to Unlock Large Scale Reinforcement Learning for Trading," Sascha Frey, Kang Li, and Peer Nagy contributed equally to this work. All three contributed to the conception, implementation, and writing of the paper with input from all authors. Silvia Sapora and Chris Lu contributed to the implementation, provided feedback, and assisted with writing and editing. Stefan Zohren, Jakob Foerster, and Anisoara Calinescu offered guidance and supported project design and manuscript revision.

Chapter 5

Generating Limit Order Book Data

5.1 Preface and Contextual Overview

Financial markets are often modelled as complex systems, where intricate aggregate dynamics emerge from interactions among numerous participants with diverse information, incentives, and constraints [Hommes 2001; Marschinski and Matassini 2001]. Even the most granular historical data, such as messages sent to exchanges, serve as a ground-truth record of observed behaviour. However, the dynamic and interactive nature of these markets makes counterfactual analysis particularly challenging. Market actions, such as submitting limit orders, are visible to other participants, who may respond strategically. This visibility leads to *market impact*, whereby prices tend to move adversely – rising after buy orders and falling after sell orders – in response to the anticipated behaviour of other market actors.

Market impact can be decomposed into *direct* and *indirect* components. *Direct* market impact refers to the immediate, mechanistic change in price resulting from the market's clearing rules, such as price-time priority in a limit order book (LOB). When an order removes liquidity at the current best price level, the next occupied price level becomes the new best price. The difference between these two levels

reflects the direct impact, which occurs independently of other participants' behaviour. *Indirect* market impact, by contrast, arises from the responses of other agents to the original order, for example, by cancelling resting orders or submitting new ones.

As counterfactual behaviour is not directly observable, a generative machine learning approach to modelling market impact can involve learning conditional distributions of the data, ideally with a "causal" – that is, time-ordered and history-dependent – structure. This enables the modelling of aggregate market behaviour while retaining as much data granularity as desired. An accurate generative model of a limit order book (LOB) can serve as a dynamic scenario simulator. Such a model may be conditioned on past data, real or hypothetical, to establish a desired context, as well as on potential actions within that context. This enables the simulation of possible future market states, supporting a broad range of applications.

Realistic, fully distributional models of LOBs enable a wide range of downstream applications involving dynamic decision-making, with relevance to both researchers and financial practitioners. These applications include the development, evaluation, and testing of trading, execution, and market-making strategies, as well as, for example, the analysis of collusive behaviours and market stability. Unlike traditional static back-testing on historical data or reliance on ad-hoc market impact heuristics, generative models allow for back-testing within realistic, simulated market environments, eliminating the need to risk real capital, or at least improving robustness when employing the learned policy in the market. Moreover, such rich, dynamic simulations can serve as training environments for reinforcement learning algorithms applied to these tasks.

The following paper has been published in the *Proceedings of the Fourth ACM International Conference on AI in Finance*, 2023 [Nagy et al. 2023b].

5.2 Generative AI for End-to-End Limit Order Book Modelling

A Token-Level Autoregressive Generative Model of Message Flow Using a Deep State Space Network

Peer Nagy[1,2], Sascha Frey[3], Silvia Sapora[2], Kang Li[4], Anisoara Calinescu[3,5], Stefan Zohren[1,5], and Jakob Foerster[2]

[1]Oxford-Man Institute of Quantitative Finance, University of Oxford
[2]Foerster Lab for AI Research, University of Oxford
[3]Department of Computer Science, University of Oxford
[4]Department of Statistics, University of Oxford
[5]The Alan Turing Institute

Abstract

Developing a generative model of realistic order flow in financial markets is a challenging open problem, with numerous applications for market participants. Addressing this, we propose the first end-to-end autoregressive generative model that generates tokenized limit order book (LOB) messages. These messages are interpreted by the *JAX-LOB* simulator, which updates the LOB state. To handle long sequences efficiently, the model employs *simplified structured state-space layers* to process sequences of order book states and tokenized messages. Using LOBSTER data of NASDAQ equity LOBs, we develop a custom tokenizer for message data, converting groups of successive digits to tokens, similar to tokenization in large language models. Out-of-sample results show promising performance in approximating the data distribution, as evidenced by low model *perplexity*. Furthermore, the mid-price returns calculated from the generated order flow exhibit a significant correlation with the data, indicating impressive conditional forecast performance. Due to the granularity of generated data, and the accuracy of

the model, it offers new application areas for future work beyond forecasting, e.g. acting as a world model in high-frequency financial reinforcement learning applications. Overall, our results invite the use and extension of the model in the direction of autoregressive large financial models for the generation of high-frequency financial data[1].

5.2.1 Introduction

Ever since OpenAI opened access to ChatGPT, generative large language models (LLMs) have skyrocketed in popularity. Part of their appeal is their impressive few-shot learning and in-context learning abilities [Brown et al. 2020]. Besides LLMs, generative diffusion models have had a similar rise in popularity for image generation [Croitoru et al. 2023]. Financial applications similarly benefit from generative models for various applications, from data augmentation [Naritomi and Adachi 2020; C. Liu et al. 2022], over anomaly detection [Henry-Labordere 2019; Geiger et al. 2020], to forecasting [Zhou et al. 2018]. Most current financial machine learning approaches employ training paradigms based on generative adversarial networks (GANs) [Goodfellow et al. 2014; Eckerli and Osterrieder 2021], which usually generate series of price returns directly. Just a few years ago, Takahashi et al. [2019] stated that "Building auto-regressive models of financial time-series meets insurmountable difficulties." Only very recently has there been work on bottom-up generators of limit order book (LOB) market micro-structure data [Hultin et al. 2023]. Our paper proposes to take the next step towards more powerful autoregressive financial micro-structure generative models, which do not suffer from problems commonly encountered with GANs, such as mode collapse [Bau et al. 2019; Zhaoyu Zhang et al. 2018]. Autoregressive models enable generating sequences of variable length and allow for improved interpretability, as the model learns conditional distributions of the next sequence token by design. The trained model, in conjunction with the *JAX-LOB* replay simulator [S. Y. Frey et al. 2023], which processes generated messages, is used to generate a realistic order flow. Such

[1]The code is open-sourced on GitHub (`https://github.com/peernagy/LOBS5`) to facilitate future research.

a conditional simulation model constitutes an interactive world model, which, in principle, can be used in downstream tasks, such as trading, order execution, or market making, potentially using a reinforcement learning algorithm [Coletta et al. 2023]. We leave these applications to future work.

We propose the use of a *deep state space model* composed of *simplified structured state space* layers for sequence modeling (S5) [J. T. Smith et al. 2022], which are computationally efficient and excel at learning long-range dependencies. Recognizing the similarity between sequences of order book messages and natural language, our model learns the conditional data distribution of *tokenized* message sequences in a cross-entropy minimization task, accurately predicting the next token in the sequence. A token in a natural language task corresponds to a part of a word or a sequence of successive digits, whereas here it is the latter. The autoregressive setup has numerous virtues, compared to GANs, such as model scalability – as evidenced by LLMs – but also wins in model interpretability, as the model defines a distribution over all message sequences of arbitrary length. Challenges include modeling extremely long sequences (>10,000 tokens), as we need to represent *sequences of sequences*, since each message is itself a sequence of tokens; and long-range data referentiality.

To our knowledge, this is the first paper to propose an autoregressive end-to-end generative model for micro-structure messages. Furthermore, we develop a tokenizer, converting LOB messages to a finite vocabulary of tokens. Model performance is quantified by calculating *perplexity* scores and by evaluating conditional distributions of generated data in an inference loop (see section 5.2.6) with a custom error correction mechanism (see section 5.2.5). We utilize the Jax framework [Bradbury et al. 2018] for hardware acceleration of both the model and simulator, accelerating training and model inference.

5.2.2 Related Literature

The generation of synthetic financial data and LOBs is an active topic of research, with a variety of approaches employed, including deep learning methods [Assefa et al. 2020; Sezer et al. 2020]. Traditional methods focus on computational statistical methods to generate probabilities of LOB events but are often limited, as they make strong assumptions [Rama Cont et al. 2010b] and are not accurate enough to be used in many practical applications. Another approach is agent-based modeling (ABM). In ABMs, simulations are used to understand how the interaction of autonomous agents leads to aggregate statistics and emergent behavior in a system [Korczak and Hemes 2017; Byrd et al. 2020]. Recently, generative neural networks have been used as black-box models of the dynamics of LOBs [Coletta et al. 2021; Coletta et al. 2022; Hultin et al. 2023]. Other work has focused on the use of LSTMs [Rumelhart and McClelland 1987; Hultin et al. 2023] and generative adversarial networks (GANs) [Goodfellow et al. 2014], such as Coletta et al. [2021] and Coletta et al. [2022]. Many variations of GANs have been developed, including autoregressive implicit quantile networks [Ostrovski et al. 2018], ExGAN [Bhatia et al. 2021], TimeGAN [Yoon et al. 2019] and CGAN [Mirza and Osindero 2014], making GANs more easily applicable to financial data.

Examples of GAN's applications to probabilistic forecasting of financial time series include Tail-GAN [Rama Cont et al. 2022] and FinGAN [Vuletić et al. 2023]. Both use custom economics-driven loss functions, making them better suited for financial applications. Vuletić et al. [2023] adapt FOR-GAN [Koochali et al. 2019], a combination of CGANs and RNNs used in probabilistic forecasting. Rama Cont et al. [2022] simulate multivariate prices with the goal of producing accurate tail risk statistics for a set of benchmark strategies. To do this, they combine GANs with Principal Component Analysis (PCA), allowing the architecture to scale effectively to a large number of assets. Their loss function is a bi-level optimization equivalent to a max-min game, where the discriminator's goal is to predict Value-at-Risk and Expected Shortfall.

An attempt at generating LOB messages directly on a granular level is Hultin et al. [2023]. They model the LOB as a continuous-time Markov chain with volumes across price levels defining the state, extending the model of Rama Cont et al. [2010b]. For each state transition, every feature (event type, order size, and price) is modeled separately, using a dedicated RNN per feature. The joint probability is then obtained from the product of the individual conditional probabilities. Instead of tokenizing messages, large numerical values – such as order sizes or timestamps – are binned, leading to a loss of precision. In contrast, our approach models the full level-3 representation, referencing individual orders, rather than the level-2 aggregation.

5.2.3 Background

Limit Order Books (LOBs)

The LOB contains a set of all unmatched limit orders submitted to an exchange and a mechanism by which incoming orders are matched [Gould et al. 2013]. In a price-time priority book, limit orders are ordered, first by their price, and second by their arrival time. For buy orders (bids), the orders with the highest price are prioritized, and for sell orders (asks) the lowest prices are. When an incoming order "crosses the spread", i.e. accepts a lower (higher) price than the best bid (ask), the incoming order is matched with the existing orders according to price-time priority. Upon matching two orders, a trade occurs, and the ownership of the underlying security is transferred. The evolution of the state of the LOB can be wholly reconstructed given an initial book state and the total set of arriving order messages in a time interval. LOB messages types are described in Section 5.2.4.

State-Space Models

Recently, transformers [Vaswani et al. 2017] have been the most successful and widely used approach to long-range sequence modeling [Rombach et al. 2022; Jumper et al. 2021; Chowdhery et al. 2022], despite their quadratic complexity $O(L^2)$ in sequence length L. The S4 architecture [A. Gu et al. 2021b] (and sub-

sequent variants S4D [A. Gu et al. 2022] and S5 [J. T. Smith et al. 2022]) achieve state-of-the-art performance in the *Long Range Arena* task [Tay et al. 2020], which is designed to measure long-range reasoning abilities, while maintaining linear complexity in L at inference time. The S4 architecture employs the state-space model (SSM), commonly used in control theory. The SSM is defined as:

$$\begin{aligned} \mathbf{x}'(t) &= A\mathbf{x}(t) + B\mathbf{u}(t), \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t). \end{aligned} \tag{5.1}$$

The state vector $x(t)$ denotes the current state of the system and $u(t)$ the input vector, which is the set of variables affecting $x(t)$. Combined with deep learning and the HiPPO framework [A. Gu et al. 2020], matrices A, B, C and D can be learned through standard gradient descent to achieve high performance. While the SSM equations are defined in continuous-time, they can be easily discretized using either the bilinear method [Tustin 1947], or zero-order hold (ZOH), using a fixed time step. S4's features allow it to be extremely computationally efficient: it is easily parallelizable during training by "unrolling" it as a convolution using fast Fourier transform (FFT), and has $O(H^2)$ complexity per step at inference time (where H is the hidden dimension). The S5 model further improves on its predecessor by employing a parallel scan operation to compute the hidden state each step and using matrix multiplication instead of applying a convolutional kernel. This is facilitated by S5's use of one multi-input, multi-output SSM instead of the many single-input, single-output SSMs in S4, and the approximate diagonalization of the HiPPO matrix. This allows the model to maintain linear complexity over sequence length while also being able to model time sequences with varying sampling time steps [J. T. Smith et al. 2022].

Autoregressive Models

An autoregressive generative model, for example an LLM, defines a (conditional) probability distribution over entire sequences of tokens. Model quality can thereby

be measured by evaluating the probability the model assigns to the test data. One such statistic that is often applied in NLP is perplexity (PPL) [S. F. Chen et al. 1998]. PPL measures the expected per-token surprisal by the data. It is the exponential of the per-token cross-entropy loss:

$$\begin{aligned} PPL &= \exp\left\{\frac{1}{N}\sum_{\{\mathbf{m},\mathbf{b},y\}} -\log p(y|\mathbf{m},\mathbf{b})\right\} \\ &= \exp\left\{\frac{1}{N}\sum_{\{\mathbf{m},\mathbf{b},y\}} -\log \hat{y}_{\{i=y\}}\right\}, \end{aligned} \tag{5.2}$$

where $\hat{y}_i = p(y_i|\mathbf{m},\mathbf{b})$ denotes the conditional probability of token i under the model. Perplexity for individual sequences is calculated as the exponential of the mean loss over autoregressive one-step-ahead predictions of all tokens of that sequence. This is because log probabilities of sequences are conditionally separable:

$$\begin{aligned} \log\ p(y_1 y_2 \dots y_n) = log\ p(y_1) + log\ p(y_2|y_1) + log\ p(y_3|y2, y1) \\ + \dots + log\ p(y_n|y_{n-1}, \dots, y_1). \end{aligned}$$

5.2.4 Data

We use the LOBSTER data of LOBs of NASDAQ cash equities [R. Huang and Polak 2011]. In particular, we train and evaluate our model separately on data from Alphabet (GOOG) and Intel (INTC). For each stock, we use 102 days of training data (1 July 2022 to 11 November 2022), and 12 days of validation (28 November to 13 December 2022) and test data (14 December to 30 December 2022). These two stocks represent examples of small-tick (GOOG) and large-tick LOBs, which exhibit different dynamics [Zoltan Eisler et al. 2012]. Large-tick stocks, those where the tick size of \$0.01 is large relative to the stock price, tend to have less sparse LOBs and a more constant spread over time. We only use data during regular NASDAQ trading times, on working days between 09:30 and 16:00 US East Coast time. We utilize level-3 messages pertaining to the best l price levels recorded at any given moment in time. In contrast to level-2 data, where

only aggregate open volumes are recorded per price level, level-3 messages allow for a full-fidelity reconstruction of LOB dynamics within certain constraints. For each new message, the data set also contains a snapshot of prices and volumes at the best l price levels, for both buy orders (bids) and sell orders (asks). We use $l = 10$ levels of LOB data, which is a common threshold for full LOB investigations [Rama Cont et al. 2021].

LOBSTER data contains 7 message types: new limit orders, partial order cancellations, full order deletions, visible order executions, hidden order executions, auction trades, and trading halts. However, we only use the first 4 types, ignoring hidden orders, which do not affect the visible dynamics of the LOB, and trading halts. Auction trades only occur outside regular daily trading times, and are therefore not contained in our data. Each message has 6 constituting elements: a timestamp in nanoseconds after midnight, the event type (1-4), the order ID, the order size, the price, and the trading direction (buy or sell). An example order is shown in Figure 5.1.

Pre-processing

The data is pre-processed to make it better suited for a deep learning task, such as generative modeling. However, our choice of data representation is general enough for other machine learning tasks. Our proposed model architecture uses both order flow information, in the form of messages, and sequences of level-2 order book states. While some models use LOB data in a *price-volume* representation, e.g. DeepLOB [Zihao Zhang et al. 2019b], transforming the data into a more stationary representation can considerably improve performance [Kolm et al. 2021]. We propose and use a sparse representation of liquidity in the book as P separate volume features around the mid-price, coupled with one feature representing mid-price changes from the previous observation. This results in a $P + 1$ dimensional vector at each point in time, which is sparse when the LOB's price levels are sparsely populated. This representation fixes a dollar-range *volume image*, which also preserves price levels in the price range without any volume. Empty levels can carry

important information, such as the size of the bid-ask spread, or the shape of book volume at deeper levels, which are usually squashed in other data representations. Furthermore, our representation is especially useful for a generative model, since the model can place orders at a new price level without changing the absolute price a specific book feature refers to. Since we do not generate order book states directly, but message sequences, there is no need to tokenize the book data. Instead, these can be input directly into the model as continuous features.

Before messages are encoded as token sequences, we convert them to a more stationary data representation and a finite token vocabulary. After filtering the data to the relevant order types, prices are converted from dollar values to ticks (cents) from the previous mid-price. In case the mid-price lies between valid ticks, it is rounded down to the next tick. Modified prices can thus be represented as integers and become more stationary over time. We then truncate modified prices below -999 and above 999 ticks around the mid-price to enable encoding the data with a finite and fixed vocabulary. Similarly, extremely rare individual large orders are truncated to an order size of 9999. In practice, these thresholds affect less than 0.1% of the data.

Besides new limit orders, all other message types are referential, as they subtract liquidity from an existing open order in the book. This might be obvious for partial order cancellations or full deletions, but it is modeled analogously for order executions. Execution messages must therefore refer to the order ID of the highest priority bid or ask order in the book. A challenge arises in encoding this referentiality in the message data, since numeric order IDs are a poor choice, due to their arbitrary nature and non-stationarity. As an alternative, we append information about the referenced original message as additional features to all messages. These fields are the original modified price, size, and timestamp, which we use to identify the referenced limit order.[2] Since arrival times are monotonically increasing, and thus a difficult distribution to generate from, we also add inter-arrival times be-

[2]In the rare case of exactly identical orders, recorded at the same nanosecond, we choose the last order in the list.

tween messages as additional features. Finally, we reorder the message features so that longer fields, corresponding to higher-entropy token distributions, occur later in the message and can thus condition on previous fields in our target task of left-to-right "causal" prediction. The order of features is: event type, direction, price, size, inter-arrival time Δt, arrival time; followed by the reference fields: price, size, and time of the original message. In the case of a new limit order, the reference fields receive an NA value.

Encoding

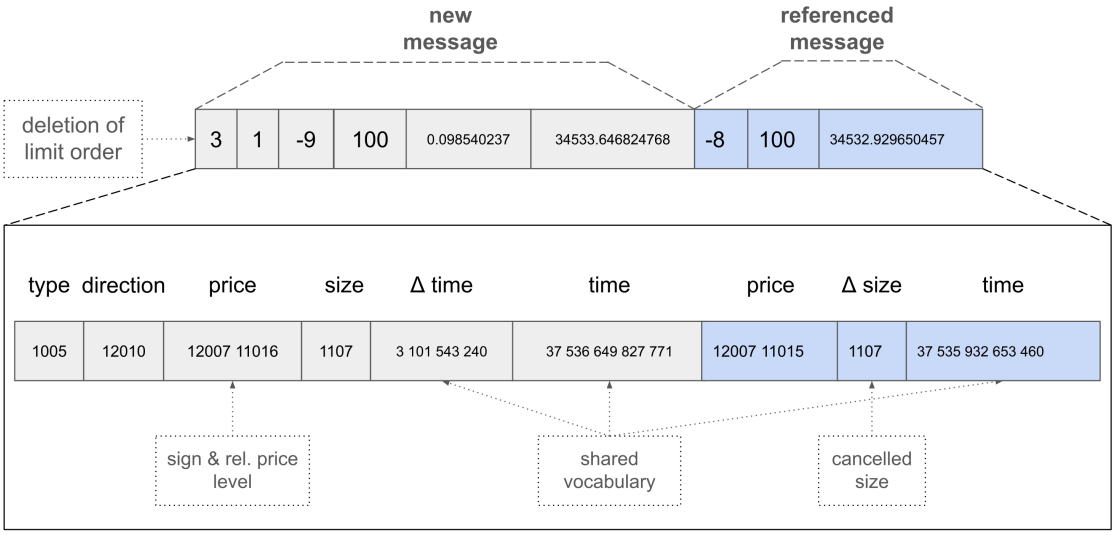


Figure 5.1: **Schematic of message tokenization** of a limit order deletion message. Each field of the pre-processed message at the top corresponds to a sequence of tokens in the encoded message at the bottom. The grey message part corresponds to the new message arriving at the exchange, while the blue part encodes the referenced message to be deleted.

Analogously to a generative language model, we propose a *token-based encoding* scheme for LOB messages. This allows the model to be trained using cross-entropy loss on flattened token sequences so that it learns the conditional distribution over a target token. Figure 5.1 illustrates the encoding mechanism of a pre-processed order deletion message. Here, each field of the pre-processed message corresponds to a sequence of tokens in the encoded message. The message vocabulary contains 12011 distinct tokens, which are represented as integer values. Event type, direction, and size fields are each encoded with a single token, while prices are

split into two, the first corresponding to the sign (above or below the mid-price) and the second to the tick distance from the mid-price. A difference to language encoders is that we use non-overlapping token ranges for some fields. For example, even though event type and price level might have the same raw numerical value, they are encoded using different tokens to make effective use of our a priori knowledge of their semantic difference. Due to their similarity, arrival times and inter-arrival times (Δt) share the same vocabulary and are tokenized in groups of 3 digits. A timestamp with nanosecond precision (15 digits) thus corresponds to a 5-token sequence and the 9 fields of the pre-processed message are converted to 22 tokens. While we conduct all data pre-processing for our entire data set before model training and inference, message encoding and decoding are done dynamically when loading the data. Dynamically decoding messages is required for generated messages to be submitted to the simulator during model inference. This is done computationally efficiently by relying on just-in-time (JIT) compilation on hardware accelerators (such as GPUs) using the Jax framework [Bradbury et al. 2018].

5.2.5 Model

Model Architecture

The model architecture uses a deep network of *simplified structured state-space layers* (S5) [J. T. Smith et al. 2022] (see section 5.2.3). Model inputs are flattened sequences of n tokenized LOB messages $\mathbf{m} \in \mathcal{V}^{22n}$ and of n *volume images* of the level-2 LOB states $\mathbf{b} \in \mathcal{B}^n$, where $\mathcal{V} \subset \mathbb{N}$ is the token vocabulary and $\mathcal{B}$ the space of transformed book snapshots containing P volumes $\in \mathbb{R}_+$ and the previous mid-price change $\in \mathbb{Z}$. This way, message m_i at sequence location $i \in [0, n-1]$ acts upon order book state b_i and transitions it to b_{i+1}. So, for each book state, the model receives the corresponding successor message.

Similarly to masked language modeling [Kenton and Toutanova 2019], during training, we mask a random token in the last message, using a MSK token, and replace

all tokens to the right of this with HID tokens, so the model does not condition on those fields. Time tokens of new messages are not predicted but instead calculated from generated inter-arrival times Δt. Therefore, these are not selected to be masked during training. Time tokens do however remain part of input sequences and time tokens of referenced messages *are* prediction targets as these are essential in identifying original messages. These are easier targets, as they are already part of the message sequence when the referenced limit order had been submitted, which is often contained in the input sequence.

We define the model $f_{\boldsymbol{\theta}} : (\mathbf{m}, \mathbf{b}) \mapsto \hat{\mathbf{y}}$, parametrised by $\boldsymbol{\theta}$, as mapping a message sequence $\mathbf{m}$ and a book sequence $\mathbf{b}$ to a vector of logits $\hat{\mathbf{y}} \in \mathbb{R}^v$ where $v = |\mathcal{V}|$ is the size of the token vocabulary. By requiring that $\sum_{i=1}^{v} \exp(\hat{y}_i) = 1$ the model defines a distribution over $\mathcal{V}$, which learns the distribution of the masked token, conditional on the input sequences. To achieve this, the model parameters $\boldsymbol{\theta}$ are trained by minimizing cross-entropy loss

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{\{\mathbf{m},\mathbf{b},y\}} L(y, \hat{\mathbf{y}}) = \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{\{\mathbf{m},\mathbf{b},y\}} -\log \hat{y}_{\{i=y\}}, \\ &\text{where} \quad \hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{m}, \mathbf{b}), \end{aligned} \tag{5.3}$$

over N tuples of training data $\{\mathbf{m}, \mathbf{b}, y\}$ using gradient descent with the Adam [D. P. Kingma and Welling 2014] optimizer. The scalar $y \in \mathcal{V}$ denotes the target token from the data. In practice, tokens are one-hot encoded, so that the model learns an input embedding.

The model architecture is shown in Figure 5.2. The network has two separate input branches, one receiving the masked message sequences, and the other the book state sequences. Both are processed separately for a few layers, combining S5 with dense layers, before projecting both sequences to a common sequence length L. The concatenated sequences are then further processed by a stacked block of S5 layers, before projecting the output to v output neurons. The message branch starts with a linear embedding layer, projecting each one-hot token vector to the model's hidden dimension H, followed by S5 layers. The book branch, on the other

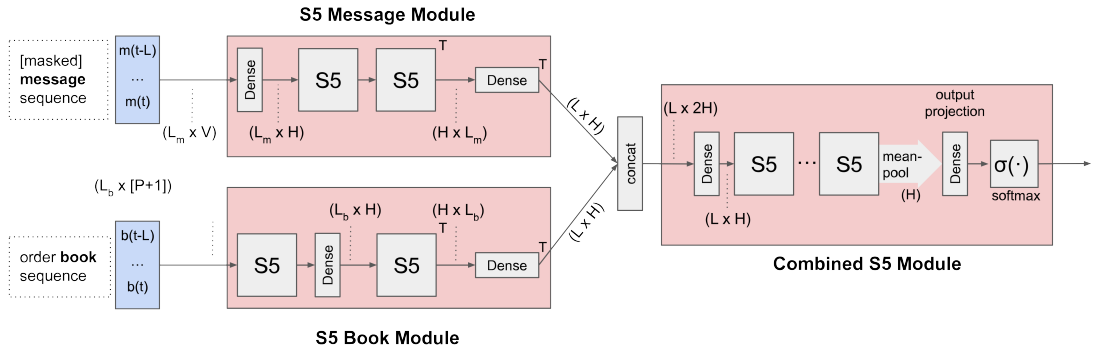


Figure 5.2: **Schematic of model architecture.** Book sequences and flattened tokenized message sequences are initially processed separately before being processed by a combined S5 module (using 6 S5 layers, resulting in $\sim 6.3x10^6$ parameters). Sequence outputs from the last S5 layer are averaged along the sequence dimension to produce token logits.

hand, first passes each $(P+1)$-dimensional observation through an S5 layer before also projecting it to the embedding dimension H.

We trained the model on input sequences of $n = 500$ messages and LOB states, corresponding to encoded sequence lengths of 11,000 tokens for the messages (22 tokens per message), and 500 observations for the book states as these are not tokenized. To vary the exact input sequences and MSK token location, in every training epoch, a random number of observations – between 0 and $n-1$ – are skipped from the start of each day.

Inference: Combining Model and Simulator

We learned that training and validation loss drastically improves when using book data in addition to message sequences. To reconstruct the state of the LOB solely from order flow, extremely long message sequences need to be combined non-linearly, which is why both have been found to contain orthogonal information in prediction tasks [Zihao Zhang et al. 2021a]. While it is easy to combine both data sources during training, we require a mechanism to generate new book states during multi-step autoregressive inference.

Our solution is a LOB simulator, which takes the most recent LOB state and applies the generated message according to LOB matching rules. To ensure proper

processing by the simulator, the generated message is initially decoded and subsequently passed through an error correction mechanism. The simulator is therefore an essential component of the inference pipeline. We utilize the novel LOB simulator, introduced in S. Y. Frey et al. [2023], implemented in Jax [Bradbury et al. 2018], and end-to-end just-in-time (JIT) compiled for GPU.

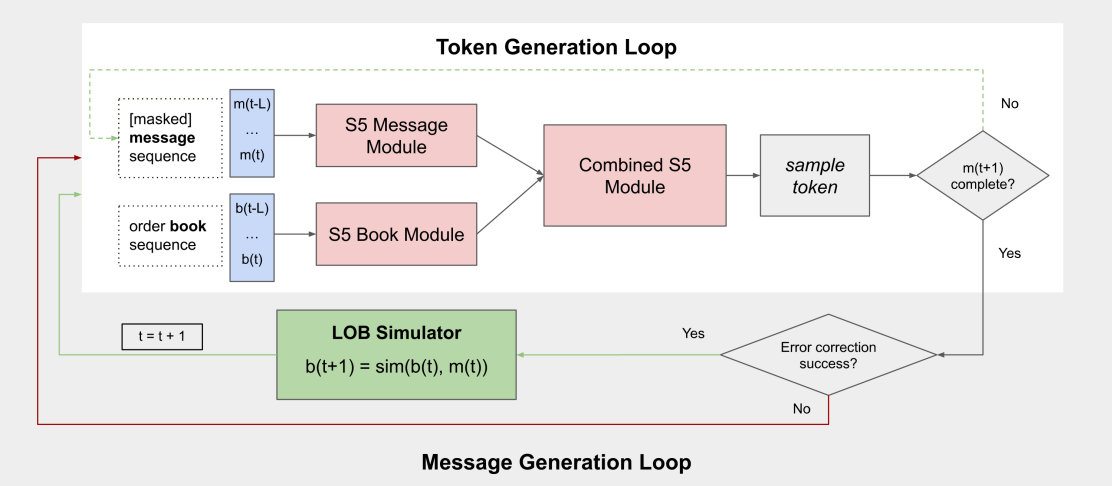


Figure 5.3: **Inference Loop.** Tokens are generated autoregressively from model logits until a message is complete (Token Generation Loop). Completed messages are error-corrected and input to the LOB simulator. The updated book state and the generated message are then added to the input sequences to start generation at $t + 1$. (Message Generation Loop)

The inference pipeline is illustrated in Figure 5.3. It starts with the model receiving a tokenized message sequence, where the first token of the last message is set to MSK, and the remainder to HID. Tokens are then sampled autoregressively left-to-right using the *softmax* over output logits. Once all tokens constituting Δt have been sampled, the message's arrival time t is calculated by adding Δt to the previous message's time stamp. The new arrival time is then tokenized and inserted into the message sequence. During sampling, we restrict the distribution to syntactically valid tokens for the currently masked field but otherwise sample proportionately to the predicted token scores without truncation or beam-search.

After initializing the simulator with a LOB state, the message sequence is then replayed to advance the state to the current time step. This is necessary as the book states are represented as a level-2 image, which means that we only have the

aggregate volume of each individual order at a price level. This initial volume is represented by the simulator as a single order. By replaying the message sequence, we can thereby recreate a partial level-3 representation, which is required to replay referential orders.

The role of the error correction procedure is to correct the occasionally hallucinated message reference components, which do not exist in the data and the simulator. As new limit orders do not require a reference component, this is only done for cancellations, deletions, and order executions. Past limit orders in the sequence, which are still in the LOB, are first searched for an order direction, price, size, and time matching the generated reference. If this matching should fail, the search is repeated excluding the time field as this field exhibits the highest error rates. Should there still be no satisfactory match, which would happen if a message references order flow before the start of the input sequence, cancellations are applied to the initial volume, if the simulator has any left at the correct price level. As executions of multiple order blocks at the same time are modeled as separate events, correct order executions are easier to guarantee. There is only a single referenced candidate order on each side of the book, namely the limit order at the best price with the earliest arrival time.

5.2.6 Results

The primary objective of a generative model is to produce data that closely approximates the target distribution. We propose to evaluate model performance in three distinct ways.

1. We compare various unconditional marginal distributions produced by the model with the corresponding data distributions.
2. Next, to evaluate the model's capacity to match conditional distributions, correlations between generated mid-price returns and realized returns are calculated. Results indicate that the model's significant forecasting horizon is competitive with deep learning models, trained explicitly to forecast mid-prices [Zihao Zhang

et al. 2019b].

3. Finally, as a succinct measure of model performance across the entire distribution, we calculate *perplexity* scores, which are common in the evaluation of large language models (LLMs) [S. F. Chen et al. 1998].

For evaluation purposes, we sample 1000 random test sequences, each comprising 600 messages and corresponding LOB states, which temporally follow the training and validation period. From each sequence, we extract the first $n = 500$ observations between time steps $t - n + 1$ and t as the model input. The model generates the succeeding 100 messages from time steps $t + 1$ to $t + 100$. We then compare the generated data against the actual 100 realized messages from the data.

The mid-price p_t at time t is the mean of the best bid $p_s^{b(1)}$ and the best ask price $p_s^{a(1)}$, at the most recent time s when there are orders on both sides of the book:

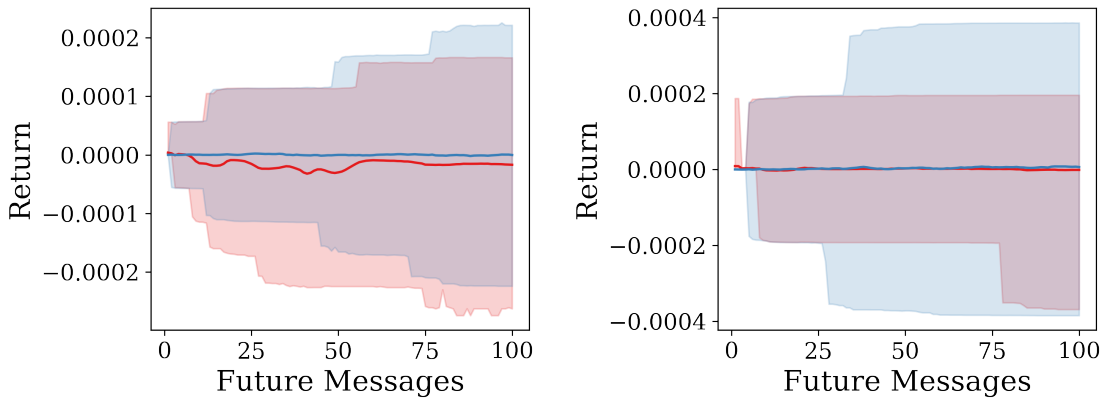
$$\begin{aligned} p_t &= \frac{p_s^{b(1)} + p_s^{a(1)}}{2} \\ \text{where} \quad s &= \max_{u \leq t} u \quad \text{s.t. } p_u^{b(1)} \text{ and } p_u^{a(1)} \text{ exist.} \end{aligned} \tag{5.4}$$

The mid-price returns s messages into the future r_{t+s} between time t and $t + s$ is then defined as

$$r_{t+s} = \frac{r_{t+s} - r_t}{r_t} \tag{5.5}$$

Figure 5.4 compares the generated and realized return distributions for 100 future messages. This shows that the model reproduces mid-price return distributions without this being an explicit part of the training loss. The mean returns do not exhibit any obvious drift or trend, and the shaded intervals, covering 95% of the distribution, overlap approximately.

Another desirable distribution to match is the relative frequency of message types (see Figure 5.5). Partial cancellations and full deletions are aggregated as the simulator only tracks individual orders if they are contained in the input sequence. As the removal of liquidity from limit orders submitted before the start of the sequence affects the aggregate initialization volume, a partial cancellation cannot



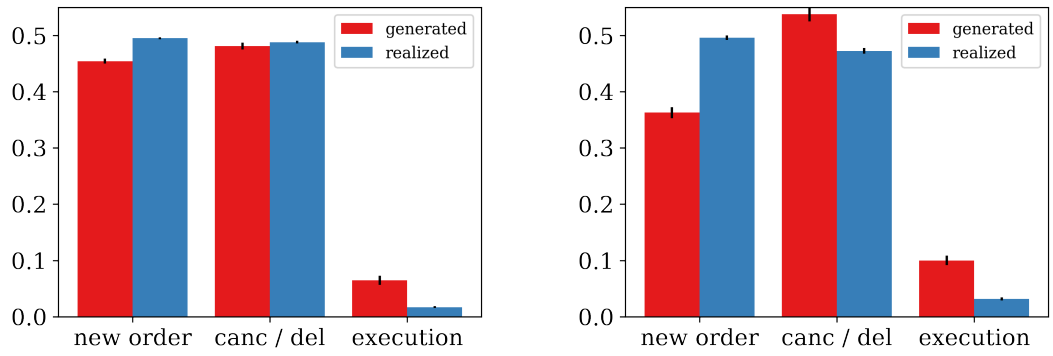
(a) GOOG (b) INTC

Figure 5.4: **Generated (red) and realized (blue) mid-price return distributions** show how well the model matches the unconditional distribution's mean (solid lines) and 95%ile intervals (shaded regions). Returns are calculated between the mid-price s messages after, and the mid-price just before the start of generation. (n=1000 test sequences)

be differentiated from a full deletion. While both models approximately match event frequencies, execution events are over-represented in the generated data. Similar effects can be observed for rare events in LLMs, and are potentially due to the mode-covering property of cross-entropy loss [Bishop and Nasrabadi 2006; C. T. Li and Farnia 2023].

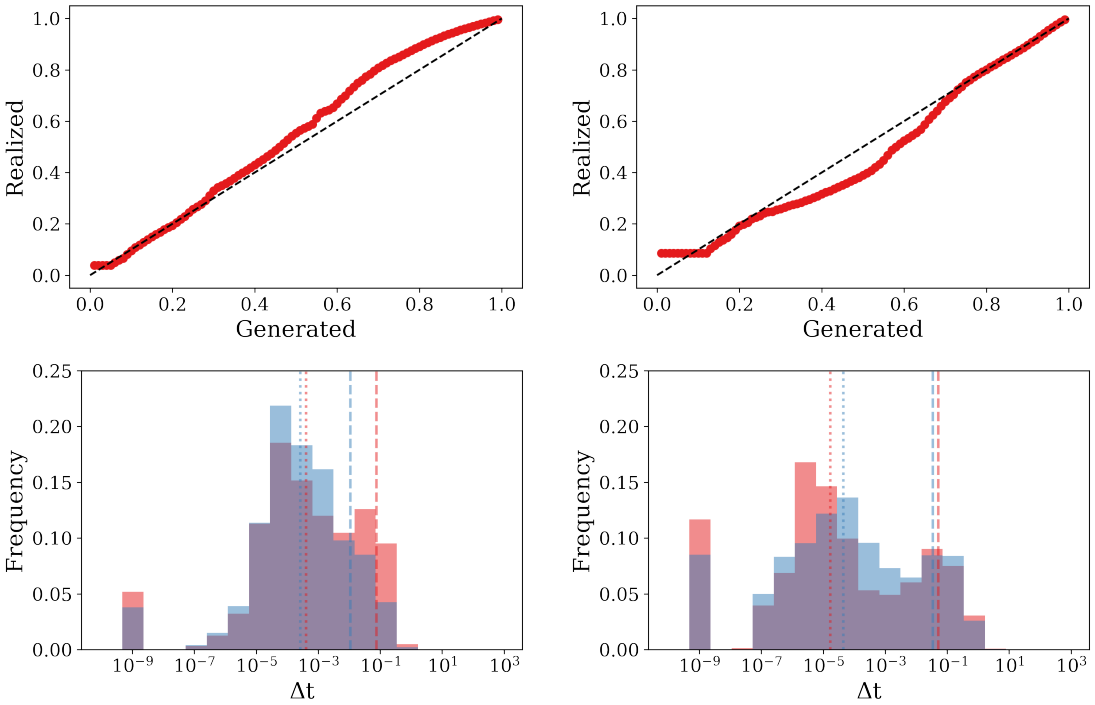
As described in section 5.2.4, for every message the model generates 4 inter-arrival time tokens Δt, which are decoded and added to the previous message's time stamp. Figure 5.6 compares the generated and realized message inter-arrival time distributions. The top two panels show probability plots (P-P plots), tracing the points $(F_g(\Delta t), F_r(\Delta t))$ across the support of Δt, where $F_g(\cdot)$ is the empirical cumulative distribution function (ECDF) of the generative distribution of Δt and $F_r(\cdot)$ the realized ECDF. An equivalent interpretation is that $F_r(F_g^{-1}(x))$ is shown on the y-axis for $x \in (0, 1)$, where deviations from the diagonal indicate distributional deviations. The lower panels depict corresponding histograms of Δt on a log-x-axis, as most observations are close to 0 with a long tail. Overall the model does a good job of matching the time features.

Sampling from conditional distributions enables the model to make forecasts. To



(a) GOOG (b) INTC

Figure 5.5: **Comparison of order type frequency** between generated (red) and realized data. While relative frequencies produced by the model roughly match magnitudes in the data, there is still a significant mismatch, with both models overestimating the probability of execution messages *in test data*. n=(1000 sequences)×(100 events) from the test set.



(a) GOOG (b) INTC

Figure 5.6: **Generated decoded 4-token sequences of Δt** match message data counterparts well. First row: probability plots comparing generated and realized inter-arrival times. Second row: histograms of generated (red) and realized (blue) inter-arrival times on a log-x scale. The dashed vertical lines plot the mean and the dotted lines the median of the distributions.

evaluate the performance of the model prediction, we calculate the Pearson correlation coefficient between the generated returns r^g_{t+s} and the realized returns r^r_{t+s} for $s \in [1, \ldots, 100]$ messages into the future. Figure 5.8 shows a sustained positive correlation for both GOOG ($\rho \approx 0.1$) and INTC ($\rho \approx 0.2$). For INTC, the stock with less liquidity, correlations remain statistically significant at least 100 messages into the future.

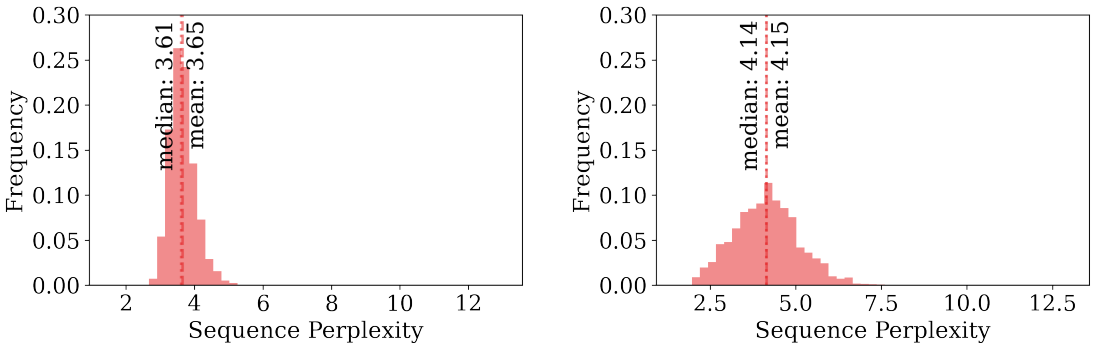
Figure 5.7 plots histograms of 100-message test sequence perplexity scores for both stocks. The model trained on GOOG data gets an overall lower perplexity than INTC with fewer extreme values of low probability sequences. This could be owed to more GOOG data being available during training as the stock has a higher trading volume and more messages in the same time period. Calculating overall per-token perplexity results in a score of **3.63** (*std.err.* 0.0047) for GOOG and **4.04** (*std.err.* 0.0043) for INTC. Table 5.1 calculates the perplexity for each of the token positions generated in a message. We observe that later digits of inter-arrival times Δt become harder to predict as these are higher entropy distributions. In contrast, reference time tokens are easier to predict as these usually refer to earlier time tokens in the sequence. Generally, for this reason, reference tokens have lower perplexity than their counterpart fields in the new message. Comparing GOOG with INTC we observe, that the GOOG model has higher perplexity in generating an order's price level, while INTC has higher uncertainty of order sizes. The difference in price level predictability agrees with the fact that INTC, as a small-tick stock, exhibits a denser order book on average [Zoltan Eisler et al. 2012], while GOOG order sizes have lower entropy than INTC.

5.2.7 Conclusions

We develop an end-to-end autoregressive generative model for electronic exchange message flow, trained on tokenized message sequences and LOB states, using a custom tokenizer. By embedding the model in an inference loop with a market replay simulator, we are able to generate full-fidelity granular trajectories of entire LOBs. A challenge of predictive machine learning models in finance, particularly

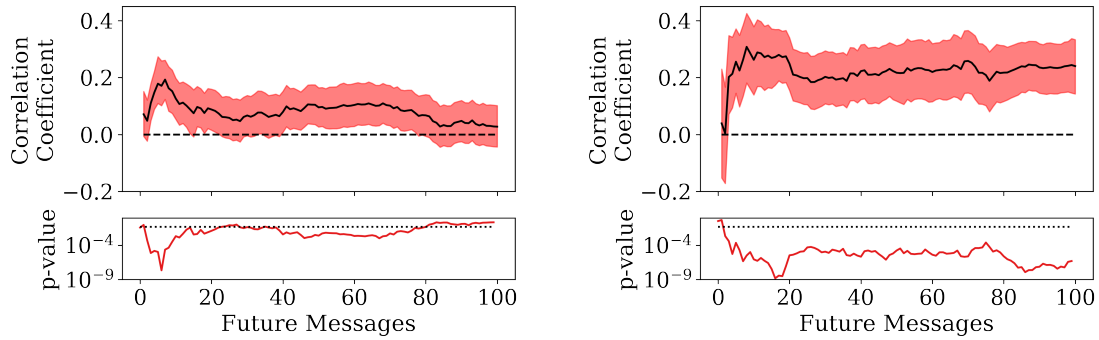
Table 5.1: **Token Perplexity Scores by Message Position**

Feature	GOOG	INTC
Type	2.15	1.92
Direction	1.71	1.55
Price	1.04	1.02
	5.55	2.41
Size	3.18	13.51
Δt	1.00	1.01
	6.21	5.45
	405.49	122.93
	770.01	524.66
$\text{Price}_{\text{ref}}$	1.04	1.03
	1.45	1.21
Size_{ref}	1.07	1.22
Time_{ref}	1.01	1.02
	1.63	1.96
	2.41	3.76
	2.09	6.58
	1.93	7.94



(a) GOOG (b) INTC

Figure 5.7: **Per-token sequence perplexity:** the GOOG model exhibits better data fit on average and fewer extreme values. Both distributions appear benign without much mass in the right tail. An observation corresponds to the PPL of a single realized 100-message sequence from the test set. Dotted lines indicate medians and dashed lines the means.



(a) GOOG (b) INTC

Figure 5.8: ***Top:*** **Pearson correlation coefficient** ρ between generated and realized returns, indicating directional forecasting performance. The dashed line indicates a 0 correlation. ***Bottom:*** **corresponding p-values** of t-tests testing $H_1 : \rho > 0$ against $H_0 : \rho = 0$. The dotted line plots the 5% significance threshold. The GOOG model stays below or around 5% significance for up to 80 messages into the future, while correlation for INTC remains significant for at least 100 messages.

in high-frequency market micro-structure, is market impact, i.e. the fact that market actions affect dynamics. Modeling realistic market impact is thus an important open problem, for which generative models of this kind provide a promising solution.

For tasks in natural language processing, autoregressive LLMs have proven superior to GAN-based models in cases where sufficient training data is available. Arguably, market micro-structure provides a similar domain with 2500 different stocks trading on the NASDAQ exchange alone, each with up to millions of daily messages. Our results show that such models are reasonable, can provide good performance, and could potentially be scaled up to something like autoregressive large financial environment models, which could be used for reinforcement learning. Similar approaches using GANs have not yet been very successful due to agents learning to exploit model errors [Coletta et al. 2023].

One challenge to generative micro-structure models is posed by message-arrival times. To our knowledge, we propose the first generative micro-structure data model, which treats times the same as other data features and learns a generative distribution, compared to e.g. binning time intervals as done in [Hultin et al.

2023].

Despite the fact that the model is trained on tokenized micro-structure messages, we are able to match data distributions, such as the distribution of mid-price returns. This radical bottom-up approach, in combination with large neural networks, has been driving recent successes in LLMs and could similarly usher in the next generation of generative financial models.

Given these promising results, our work provides many interesting directions for future research. Current model limitations include errors when generating referential orders and computationally intensive message error correction. These issues could be abated by increasing the model and data set size. Besides larger models, future work could similarly investigate training on longer sequences. Different architectural choices, either improvements to the S5 layer [M. Zhang et al. 2022], or efficient transformer-based networks [Child et al. 2019] could be explored as alternatives.

End of Published Paper

5.3 Discussion and Integration

The *LOBS5* project adapts successful machine learning approaches, particularly those from Large Language Models (LLMs), to generate LOB data. Similar to natural language, LOB messages possess a specific grammar and causal, time-ordered structure. However, LOB messages exhibit a much stricter grammar defining valid messages and a limited vocabulary. This increased structure offers distinct advantages. When sampling messages from the model, we mask the generated logits to permit only syntactically correct tokens for the current position; for example, price tokens for price fields or time tokens for inter-arrival times. In this regard, autoregressive generation for LOB data is less complex than for natural language in LLMs.

Conversely, LLMs typically struggle with arithmetic and numerical manipulation [Dziri et al. 2023; Guo et al. 2024]. For instance, when calculating the product of two numbers, LLMs often rely on matching patterns observed in training data, leading to increasing inaccuracies as numbers grow larger [Y. Deng et al. 2024]. In the context of LOBs, such computations are crucial as models should learn changes in relative price levels and track the size of resting orders. To enhance the accuracy of the LOBS5 model, we designed the pre-processing pipeline to minimise potentially difficult operations. For example, we normalise prices relative to the mean price and encode the mid-price as a sequence of one-step changes. This approach reduces the need for complex additions of absolute prices and relative levels. Similarly, expressing times as message inter-arrival times makes the distribution stationary and eliminates the need for the model to learn timestamp addition.

In contrast to natural language, LOB messages often contain numerical content with inherent ordinal or cardinal structure. However, standard tokenisation and model training using cross-entropy loss do not exploit these structural properties. As a result, generation quality, particularly for rarely seen tokens, can suffer. A promising direction for future research is to incorporate distance-aware loss

functions, such as the Wasserstein distance, when training on cardinal features (e.g., prices or timestamps). This approach would require defining a suitable distance metric in token space, which is natural for cardinal features and could be extended to categorical tokens by assigning fixed distances between categories. Such a modification could help models learn more effectively from the semantic relationships among tokens, potentially improving both generalisation and sample efficiency.

Notwithstanding these steps, the generative model continued to produce erroneous messages in up to 20% of cases, necessitating message regeneration. To mitigate this issue, we implemented the heuristic error correction module described in the paper. One of the most frequent error types involved the generation of invalid cancellation messages that referenced non-existent limit orders. As outlined in Section 5.2.4, resting limit orders are referenced using their timestamp and relative price level at the time of arrival. This approach was adopted to eliminate arbitrary order identifiers, which could introduce unnecessary complexity for the generative model.

Motivated by the observation that larger language models typically yield improved generation quality, we hypothesised that increasing the model size and training on more data could reduce the error rate. This hypothesis is supported by the scaled-up model version used in Chapter 6, which achieves sufficiently high generation quality to render the heuristic error correction module unnecessary.

A key difference from natural language generation is the duality inherent in LOB data, which consists of both message events and corresponding book states. The LOBSTER dataset provides a level-2 (L2) snapshot of the limit order book following each message, summarising the aggregate volume at the top n price levels on both the bid and ask sides. However, this representation omits information about individual orders within the queues. Consequently, any generative model must internally maintain order-specific information required for constructing future valid messages.

This dual structure introduces a challenge for naïve LLM-style training on raw LOB message sequences. If only the message data is used, the model must implicitly learn and update a latent representation of the evolving book state after each message. Alternatively, including both messages and book states in the sequence increases the input length substantially, leading to slower training and inference, and compounding the difficulty of generating coherent long-range sequences.

Our solution to this challenge in the *LOBS5* model is to generate only the message sequence while using preprocessed level-2 (L2) book state data as additional input features at every step. This design significantly improves performance but introduces added engineering complexity. Since the model expects book state information at each time step, yet does not generate it, the JAX-LOB simulator is employed during inference to maintain and update the book state in parallel with the message generation process.

An alternative approach, more aligned with the pure autoregressive LLM paradigm, would be to interleave book states after every x messages and tokenise the combined data stream. This method would eliminate the need for an external simulation engine but would likely require a larger model to handle the increased sequence length and capture the latent book dynamics effectively.

We also experimented with an alternative LLM-inspired approach: masked language modelling (MLM). In this setup, instead of training the model strictly in an autoregressive left-to-right fashion, we pre-trained it to predict randomly masked tokens from the final message in the input sequence, leveraging bidirectional S5 layers. In natural language processing, such pre-training is often employed to improve contextual understanding and downstream task performance. However, in our experiments, this pre-training strategy did not yield any measurable improvement in the final performance of the model on the autoregressive generation task.

Another key challenge in generating LOB data are the long-range temporal dependencies inherent in market microstructure. For instance, a substantial number

of messages may separate the arrival of a limit order from its eventual cancellation or execution. More generally, reconstructing the current LOB state often requires access to a relatively large window of recent message history. Capturing such dependencies is a particular strength of state-space models, such as the S5 layer [J. T. Smith et al. 2022] employed in this work. However, long-range dependency also introduces difficulty when generating long sequences. Applications such as back-testing trading strategies or generating synthetic datasets typically require the generation of full trading days of coherent microstructure data. As demonstrated in Chapter 6, this remains a significant challenge not only for the *LOBS5* model but also for existing generative LOB modelling approaches more broadly.

Although still an emerging area of research, realistic generative models of LOB data hold considerable potential for a wide range of impactful applications. For instance, they can enable more accurate evaluation of high-frequency trading strategies compared to traditional back-testing on static historical data. Generative models also offer a principled way to compare trade execution strategies and assess their market impact. Furthermore, synthetic LOB data can support downstream tasks, such as enhancing the training of machine learning models by augmenting scarce or imbalanced datasets. One of the most promising applications lies in dynamic simulation settings, such as RL environments, where generative models can be used to create interactive and realistic market scenarios for policy learning. Looking ahead, generative models could be embedded as background processes in multi-agent simulations. In such setups, orders from explicitly modelled agents could be interleaved with those generated by the model, bringing simulations one step closer to fully agent-based, high-fidelity LOB environments.

5.4 Author Contributions

In the previous paper "Generative AI for End-to-End Limit Order Book Modelling," Peer Nagy conceptualised the project and led its implementation and

manuscript preparation, with input from all authors. Sascha Frey, Silvia Sapora, and Kang Li contributed to implementation, provided feedback, and assisted with writing and editing. Anisoara Calinescu, Stefan Zohren, and Jakob Foerster offered guidance and supported project design and manuscript revision.

Chapter 6

Evaluating Generated Limit Order Book Data

6.1 Preface and Contextual Overview

While deep learning approaches to generative LOB models have existed for several years [Coletta et al. 2023; Rama Cont et al. 2023b; Lim and Gorse 2021; Shi et al. 2021; Junyi Li et al. 2020], there is still no consensus on how to evaluate them. Existing studies typically rely on ad-hoc metrics and qualitative comparisons of stylised facts. The lack of consistent, comprehensive evaluation tools hinders progress – making it difficult to synthesise findings across the literature or identify the methods that drive strong model performance.

Benchmarks and standardised datasets have underpinned much of the progress in machine learning – particularly in deep learning. The current wave of breakthroughs can be traced back to the release of the ImageNet dataset [J. Deng et al. 2009], which catalysed advances in image classification. Similarly, the development of novel RL algorithms has relied on extensive benchmark suites and consistent environment implementations [Bellemare et al. 2013; Duan et al. 2016; Brockman et al. 2016]. More recently, the rapid progress in LLM research has been driven not only by large training datasets, but also by benchmarks that enable systematic

evaluation and model comparison [Mao et al. 2024].

The essential question is how to construct a benchmark that addresses the specific challenges of LOB data. This data is high-dimensional and structured, with both categorical and continuous features, and supports a range of downstream applications. Cross-entropy could serve as a general statistical evaluation metric without introducing domain-specific biases, as is common in language modelling. However, computing cross-entropy requires each model to support efficient evaluation of the data distribution, typically by tokenising the input and learning the next-token distribution autoregressively. Many model classes, such as agent-based models or GANs, do not represent this distribution explicitly. As a result, estimating cross-entropy would require sampling extensively for each token, rendering the approach computationally infeasible.

Another limitation of cross-entropy is its reliance on teacher forcing, which does not reflect the accumulation of autoregressive errors during generation. In teacher forcing, one-step-ahead errors are calculated based on the true past data – a setting that diverges from actual generation, where past inputs are sampled, not observed. Even small errors in predicted token distributions can compound over time, gradually pushing the model off-distribution. This shift is self-reinforcing, as conditioning on previously misgenerated tokens increases the likelihood of future errors. This phenomenon is commonly referred to as "snowballing errors."

Combining the challenge of accumulating generative errors with the need to evaluate LOB data across diverse downstream tasks motivates a quantifiable, distributional evaluation approach. The proposed LOB-Bench framework enables this by supporting the evaluation of both unconditional and conditional data distributions, while also measuring the extent of error accumulation during generation. It is model-agnostic and requires only samples of generated and real data in the LOBSTER [R. Huang and Polak 2011] format.

In addition to distributional evaluation, we consider an alternative paradigm based on downstream data utility. Here, the quality of model-generated data is assessed

by its usefulness for specific tasks. We focus on mid-price return forecasting, training models on a mix of real and synthetic data, and evaluating performance on held-out real data. This setup reflects a typical synthetic data use case, where generated data supplements limited real data. However, this sim-to-real evaluation paradigm has notable drawbacks: it is complex, time-consuming, application-specific, and only implicitly evaluates a subset of relevant data features. Most importantly, we argue that performance in such tasks is difficult to trace back to specific model failures, limiting its value for guiding future model development.

With LOB-Bench, we also address a key question in autoregressive LOB generation: how far the LLM-style approach can be pushed without data pre-processing. To explore this, we train a pre-trained LLM based on the RWKV state-space architecture [Peng et al. 2023; Peng et al. 2024] to predict raw LOBSTER messages, using a byte-pair tokeniser trained on the same data. This setup uses only message data, omitting corresponding book states. Results reveal the limitations of this approach and highlight that the more tailored LOB-S5 model achieves significantly better performance.

The following paper has been published in the *Proceedings of the 42nd International Conference on Machine Learning*, 2025 [Nagy et al. 2025].

6.2 LOB-Bench: Benchmarking Generative AI for Finance – an Application to Limit Order Book Data

Peer Nagy[*1,2], Sascha Frey[*3], Kang Li[4], Bidipta Sarkar[2], Svitlana Vyetrenko[5], Stefan Zohren[1], Anisoara Calinescu[3], and Jakob Foerster[2]

[1]Oxford-Man Institute of Quantitative Finance, University of Oxford
[2]Foerster Lab for AI Research, University of Oxford
[3]Department of Computer Science, University of Oxford
[4]Department of Statistics, University of Oxford
[5]J.P. Morgan AI Research

Abstract

While financial data presents one of the most challenging and interesting sequence modelling tasks due to high noise, heavy tails, and strategic interactions, progress in this area has been hindered by the lack of consensus on quantitative evaluation paradigms. To address this, we present **LOB-Bench**, a benchmark, implemented in python, designed to evaluate the quality and realism of generative message-by-order data for limit order books (LOB) in the LOBSTER format. Our framework measures distributional differences in conditional and unconditional statistics between generated and real LOB data, supporting flexible multivariate statistical evaluation. The benchmark also includes commonly used LOB statistics such as spread, order book volumes, order imbalance, and message inter-arrival times, along with scores from a trained discriminator network. Lastly, LOB-Bench contains "market impact metrics", i.e. the cross-correlations and price response functions for specific events in the data. We benchmark generative autoregressive state-space models, a (C)GAN, as well as a

[*]These authors contributed equally to this work.

parametric LOB model, and find that the autoregressive GenAI approach beats traditional model classes. Code and generated data are available at: https://lobbench.github.io/.

6.2.1 Introduction

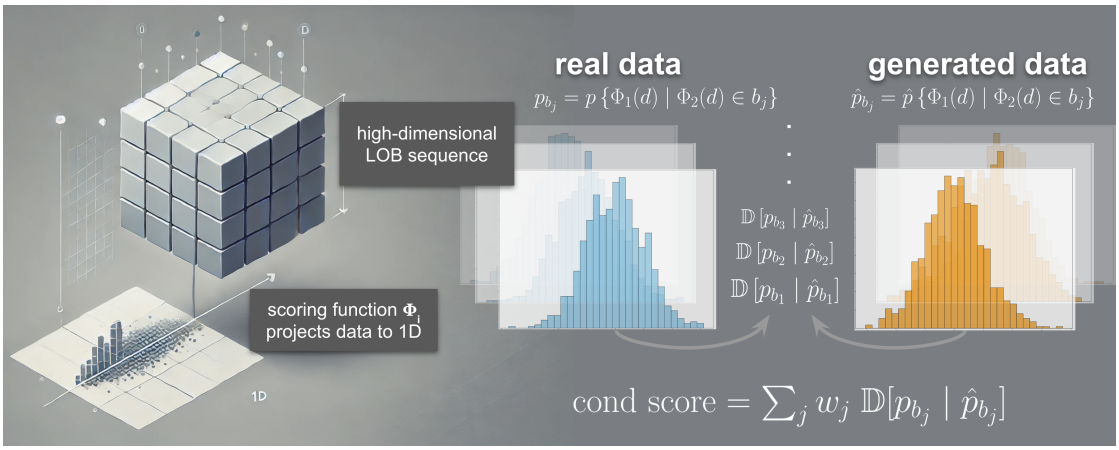


Figure 6.1: **Schematic of LOB-Bench methodology** for conditional distributional evaluation

Practitioners have long been interested in high-quality synthetic financial data, which is especially difficult in the high-frequency domain due to high noise, heavy tails, and multi-agent interactions. Generative AI (GenAI) is currently revolutionizing different fields, ranging from natural language processing to image generation and real world applications. Perhaps surprisingly, the backbone of these methods is simply self-supervised pre-training on large datasets using a next-token prediction loss on autoregressive sequence models [Nie et al. 2024; Dubey and et. al. 2024; Y. Liu et al. 2024].

Recently, Nagy et al. [2023b] applied this paradigm to *limit order books* (LOB), the mechanism through which stock markets keep track of buy and sell orders to determine any-time prices. Specifically, in contrast to prior works, which model only high level features [Rama Cont et al. 2010b; Coletta et al. 2022; Byrd et al. 2020], this approach learns a *token-level* distribution over messages in the LOBSTER dataset [R. Huang and Polak 2011].

An *accurate*, *low level* generative model of the financial system is extremely valuable from a societal and commercial point of view. For example, it could unlock better mechanism design, stability analysis, or learned-algorithms (e.g. order execution [S. Y. Frey et al. 2023]) by providing counterfactuals.

A key question then is how to determine the realism and trustworthiness of GenAI, and of other generative financial models. On the one hand, for high-level approaches and "old school" agent-based modelling [Byrd et al. 2020; Chiarella and Iori 2002; Paulin 2019; Llacay and Peffer 2018] the evaluation is usually based on a qualitative analysis of whether the model reproduces known high-level patterns – "stylized facts" from the literature, "impact" or the famous "square-root law" [Tóth et al. 2016; Brokmann et al. 2015; Almgren et al. 2005]. However, most of these metrics are unquantifiable and may be disconnected from ground-truth data.

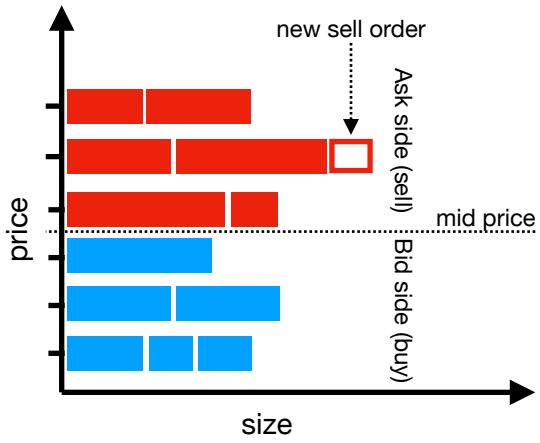


Figure 6.2: **Schematic of the LOB.** Not immediately executable orders are placed in first-in, first-out (FIFO) queues at the specified limit price. Sell limit orders are placed on the Ask side (red) and Buy limit orders on the Bid side (blue).

On the other hand, for GenAI the standard evaluation for pre-training is simply *cross-entropy*, i.e. how closely the model is able to predict the next token on held-out data. Unfortunately, this does not capture how the model performs under autoregressive sampling, when *generating* sequences of data one token at a time, where error accumulation can cause distribution shifts. In many applications of GenAI this is not a problem, since the pre-trained models are merely used as *starting points* for task specific finetuning (e.g. RLHF), rather than in their "bare"

form. In contrast, we want to evaluate the pre-trained models in the *sampling* regime to unlock the mentioned use-cases.

To address this, we propose a general framework for evaluating the similarity between the distribution induced by generative LOB models and the ground-truth data. At a high level, our *unconditional* evaluation consists of three steps. We first introduce a set of *aggregator functions*, $\mathbf{\Phi}$, which map from high-dimensional time series LOB data into a set of 1d subspaces. Secondly, we compute histograms to estimate distributions for the ground-truth and generated data in these subspaces and, finally, use a distance metric, e.g. L_1, to compare differences in these estimates. Some of the aggregator functions chosen are closely inspired by metrics used in literature [Vyetrenko et al. 2021; Paulin 2019; Chiarella and Iori 2002; R. Cont 2001]. They also directly relate to *generative adversarial networks* [Goodfellow et al. 2014], where the discriminator network is equivalent to a *worst-case* aggregator function for a given generator.

For *conditional* distributional evaluation, we first apply an aggregator function and group these results into "buckets" based on the conditioning variable. We then score each of the resulting conditional distributions using the process described earlier. This approach enables, for example, assessing whether the distribution of bid-ask spreads, conditioned on the time of day, aligns with the corresponding conditional distribution in real data. To derive a single metric, we compute the average loss across the conditioning buckets, weighted by the probability of each bucket. Furthermore, we can also use this to evaluate model-drift by aggregating on the *sampling step* and comparing to the unconditional data, which is a good proxy for model-derailment in open-loop sampling. See Figure 6.1 for a process schematic.

We test our evaluation framework on five different generative models: four modern GenAI models [Coletta et al. 2022; Nagy et al. 2023b; Peng et al. 2023; Peng et al. 2024] and a widely-used classic model as a baseline [Rama Cont et al. 2010b]. All models are tested on data of Alphabet Inc (GOOG) and Intel Corporation (INTC)

stock. We don't present detailed results for the *Coletta* model trained on INTC, because the architecture was developed only for small-tick stocks and therefore fails on INTC data [Coletta et al. 2022]. We find evidence of "model derailment," since the distance scores increase for longer unrolls (Figure 6.5). We also find that the *LOBS5* model is best able to reproduce the standard *price-impact curves* that are well-known in the economics and finance literature [Zoltan Eisler et al. 2012].

Our contributions are summarized as follows:

A novel LOB benchmark for distributional evaluation: the first LOB benchmark focused on full distributional quantification of model performance. This addresses limitations of prior work, which relied on qualitative comparisons of stylized facts, making rigorous model comparisons infeasible and hindering research progress.

Interpretable scoring functions for targeted improvements: using intuitive scoring functions enables targeted model development and refinement.

Difficult challenge of discriminator scores: discriminator-based scoring sets a high bar for future generative models, even when most other statistics are closely aligned.

Identification of a common failure mode: divergence metrics, computed as distributional errors as a function of unroll step, highlight a prevalent failure mode to guide research.

Ease of use and accessibility: open-source, straightforward to apply benchmark which only requires data in the LOBSTER format.

Extensibility to additional scoring functions.

Transferability to other domains: the theoretical framework is adaptable to other high-dimensional generative time series tasks beyond LOB data.

We hope our benchmark will provide a much-needed starting point for evaluating GenAI models in finance and allow more machine learning scientists to develop

new sequence models for this important and challenging domain. Our code and additional resources are available at: https://lobbench.github.io/.

6.2.2 Related Literature

The LOB plays a crucial role in modern financial markets. With the FI-2010 dataset, Ntakaris et al. [2018] released the first publicly available high-frequency LOB dataset for benchmarking mid-price prediction models. This pre-processed dataset containst orders for five stocks on the Nasdaq Nordic market for ten consecutive days. Although useful and effective for preliminary tests and comparisons of LOB algorithms, FI-2010 does not allow a comprehensive evaluation of robustness and generalisation ability [Zihao Zhang et al. 2019b]. A similar benchmark for average price and volume prediction in Chinese stock markets is provided by C. Huang et al. [2021]. As with other currently available benchmarks, this work falls short of evaluating GenAI models with a fully distributional lens. D. Cao et al. [2022] propose a benchmark dataset, which plays a complementary role to LOB-Bench. With DSLOB, they provide a synthetic LOB dataset, generated by a multi-agent simulation with shocks, which generates labelled in- and out-of-distributions samples. In contrast, LOB-Bench does not require training on a specific dataset, and instead focuses on general-purpose model evaluation and comparison.

To evaluate the performance of generative models in the LOB environment, several studies have proposed relevant metrics. Coletta et al. [2023] investigated the interpretability, challenges, and robustness of conditional generative models. They grouped LOB states based on certain attributes and statistics and then performed conditional generation on these groups. Vyetrenko et al. [2021] proposed several statistics to assess the realism of LOB simulators, such as order arrival rate, order distance distribution, and price volatility, whilst Paulin [2019] further considers lagged autocorrelations, and liquidity of trades.

In summary, although some studies have addressed the evaluation of generative

models for LOBs, a unified benchmarking framework is still lacking. Existing research often uses *qualitative* methods to compare statistical regularities of generated data with real data, lacking quantitative evaluation metrics. Therefore, establishing a comprehensive benchmarking framework for evaluating LOB generative models is essential for advancing the field.

6.2.3 Background

Limit Order Book (LOB)

Later sections of this paper rely on the reader's understanding of the mechanisms of electronic markets, so we briefly review them here. Public exchanges such as NASDAQ and NYSE facilitate the buying and selling of assets by accepting and satisfying buy and sell orders from multiple market participants. The exchange maintains an order book data structure for each asset traded. The LOB represents a snapshot of the supply and demand for the asset at a given time. It is an electronic record of all the outstanding buy and sell limit orders organized by price levels. A matching engine, commonly using a *price-time* priority mechanism, is used to pair incoming buy and sell order interest as mentioned in Bouchaud et al. [2018]. Order types are further distinguished between limit orders and market orders. A limit order (Figure 6.2) specifies a price that should not be exceeded in the case of a buy order (bid), or should not be gone below in the case of a sell order (ask). A limit order queues a resting order in the LOB at the corresponding side of the book. A market order indicates that the trader is willing to accept the best price available immediately.

In real-time trading, injecting orders into the market induces other market participant activity that typically drives prices away from the agent. This activity is known as market impact [Almgren and Chriss 2001; Almgren et al. 2005]. Presence of market impact in real time implies that a realistic trading strategy simulation should include deviation from historical data. Therefore, realistic market impact emulation is an important consideration in limit order book modelling.

LOB Models

LOB simulation is an important technique for evaluating trading strategies and testing counterfactual market scenarios. The extent to which results from such simulations can be trusted depends on how accurately they emulate real world environments. Traditionally, it is common to use historical market data to train and backtest a trading strategy, thereby making the assumption of negligible market impact. This is based on the premise of small agent orders and a sufficient time between consecutive trades [Spooner et al. 2018]. However, the "no market impact" assumption is not valid for larger order sizes or a high frequency of orders. Agent-based methods naturally allow to study such phenomena, which emerge as a consequence of multiple participant interactions, which are difficult to model otherwise. However, they are notoriously challenging to calibrate [Vyetrenko et al. 2021; Paulin 2019]. To circumvent calibration, conditional generative adversarial networks were used to learn simulators from historical LOB data, that are both realistic and responsive [Coletta et al. 2023]. Most recently, an end-to-end autoregressive generative model that produces tokenized LOB messages in the spirit of generative AI was shown to achieve a high degree of realism [Nagy et al. 2023b].

Autoregressive LOB models

In machine learning, autoregressive modelling is a key component of language models like GPT. By learning the probability distribution of the next token given the previous tokens, autoregressive language models can generate coherent text [Radford et al. 2019]. Cross-entropy is a loss function commonly used to train classification models in deep learning. It measures the dissimilarity between the predicted class probabilities and the true class labels [Goodfellow et al. 2016]. Cross-entropy loss is the negative log likelihood of the true class labels under the predicted distribution. Minimizing the cross-entropy is equivalent to maximizing the likelihood of the data [Murphy 2012]. The cross-entropy loss over a sample of size N, with V classes can be expressed as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{v=1}^{V} y_i^{(v)} \log \hat{y}_i^{(v)},$$

where $y_i^{(v)}$ equals 1 if the true class is v and 0 otherwise, and $\hat{y}_i^{(v)}$ is the predicted probability for class v. Cross-entropy loss heavily penalizes confident misclassifications and incentivizes the model to output calibrated probabilities that match the empirical distribution of the classes. Although it is different from the KL divergence, cross-entropy can be expressed as the sum of the entropy of the true distribution and the KL divergence between the true and predicted distributions [Cover and Thomas 1999].

6.2.4 Evaluation Framework

As the success of LLMs has shown, generative models can already achieve impressive performance by autoregressive training, or "next-token prediction" alone. However, not all model classes are auto-regressive or allow the explicit computation of conditional "next-token probabilities," prohibiting cross-entropy based evaluation or calculating model perplexity [S. F. Chen et al. 1998]. However, there is still a need to evaluate such model classes, where we can merely sample data. Another reason why single-token cross-entropy loss is insufficient is the so-called "autoregressive trap" [Ziyang Zhang et al. 2024]. Even small errors in a next-token prediction task can accumulate over long sequences, moving away from the training distribution. Out-of-distribution forecasts then become increasingly worse until the generating distribution completely derails or collapses. This emphasizes the need to evaluate statistics over entire sequences, rather than focusing solely on cross-entropy. A benchmark framework should therefore also measure how fast such errors accumulate by evaluating distributions conditional on the forecasting horizon.

Evaluating generative models in any domain is fundamentally a matter of comparing distributions. Our benchmark performs exactly this task. It reduces a high-dimensional distribution of sequences of order book states $\mathbf{b} \in \mathcal{B}$ and mes-

sage events $\mathbf{m} \in \mathcal{M}$ to scalars by using scoring functions $\Phi_i : (\mathcal{M} \times \mathcal{B}) \mapsto \mathbb{R}$, $i \in \mathbb{N}$. One-dimensional score distributions can then be compared between real and model-generated data using various norms or divergences $\mathbb{D}$. By estimating the difference between the unconditional real data distribution $p\{\Phi(d)\}$ and the data distribution under the model $\hat{p}\{\Phi(d)\}$, i.e. $\mathbb{D}\left[p\left\{\Phi(d)\right\} \mid\mid \hat{p}\left\{\Phi(d)\right\}\right]$, different generative models can be ranked on their ability to match features of the data.

To evaluate the magnitude of the "autoregressive trap" the benchmark evaluates error divergence of distributions, conditional on the interval of the forecasting step $t \in \mathbb{N}$, for interval limits $a, b \in \mathbb{N}$: $\mathbb{D}\left[p\left\{\Phi(d)\right\} \mid\mid \hat{p}\left\{\Phi(d_{t\in[a,b)})\right\}\right]$. This allows quantifying distribution shift during inference.

Our framework uses both the L_1 norm and the Wasserstein-1 distance as loss metrics. To estimate the L_1 norm, we first bin the data. As a robust binning algorithm, we use the Freedman-Diaconis rule [Freedman and Diaconis 1981], which computes the bin width as $2\frac{IQR}{\sqrt[3]{n}}$, where n is the combined sample size and IQR the inter-quartile range of the real and generated data. The $[0, 1]$-scaled L_1 norm, also called the *total variation distance*, can then be estimated as:

$$\frac{1}{2}\left\|p - \hat{p}\right\|_1 = \sum_{b\in bins} \frac{1}{2}\left|p\left(\frac{b_{count}}{b_{width}}\right) - \hat{p}\left(\frac{b_{count}}{b_{width}}\right)\right|. \tag{6.1}$$

While the L_1 measure has the benefit of being bounded in the interval $[0, 1]$, the Wasserstein-1 distance, or earth mover's distance, as proposed by Rubner et al. [2000], has the advantage of being sensitive to the distance between the scores. To make losses between different scoring functions comparable, we mean-variance normalize the data before calculating the Wasserstein-1 distance.

For equal sample sizes we can compute the Wasserstein-1 distance as follows. Let $\Phi(d_{real})_{(i)}$ be the i-th order statistic of a score computed from a real data sample drawn from p and $\Phi(d_{gen})_{(i)}$ the i-th order statistic using generated data drawn

from $\hat{p}$. Then we have:

$$W_1(p, \hat{p}) = \sum_{i=1}^{n} \left\| \Phi(d_{real})_{(i)} - \Phi(d_{gen})_{(i)} \right\|_1 . \tag{6.2}$$

To evaluate a generative model's ability to adapt to different contexts, we also estimate differences between conditional score distributions

$$\mathbb{D}\left[p\left\{\Phi_1(d) \mid \Phi_2(d)\right\} \,||\, \hat{p}\left\{\Phi_1(d) \mid \Phi_2(d)\right\}\right]. \tag{6.3}$$

In this case, $\Phi_2(d)$ is binned into 10 data deciles b_j of the pooled real and generated data. Distance estimates of these 10 conditional distributions are then weighted according to the mean of the estimated density of both distributions. Letting $X = \Phi_1(d)$ and $Y = \Phi_2(d)$, a conditional metric can be evaluated as

$$\sum_{b_j} \mathbb{D}\left[p(X \mid Y \in b_j) \,\|\, \hat{p}(X \mid Y \in b_j)\right] \times \frac{p(Y \in b_j) + \hat{p}(Y \in b_j)}{2} \tag{6.4}$$

This approach addresses a specific type of distribution shift: the variation of scores, Φ_1, across the distribution of another score, Φ_2. For instance, if the conditioning function Φ_2 represents the mean time of messages within a data sequence, this framework allows us to analyze how distribution shifts affect any score of interest, Φ_1, and to assess the generative model's ability to replicate this dynamic behavior accurately.

Our methodology formalizes and naturally extends common evaluation practices for synthetic one-dimensional time series, such as financial returns, which typically emphasize distributional similarity. Our framework enables a quantitative assessment of distributional properties in *structured high-dimensional* time series. By adapting the scoring functions, our approach could also be applied to financial transactions, payment data, streamed price quotes in forex markets, multi-asset limit order books, or decentralized crypto market protocols.

Impact Response Functions

A primary difficulty with historical LOB data is that counterfactual scenarios are impossible to evaluate, as the data do not respond to additional injected orders. Generative models are a unique opportunity to generate a response to counterfactual scenarios as they address this limitation.

It is therefore crucial that such models be evaluated on their ability to provide a realistic response to different events. As an underlying methodology, the seminal work by Zoltan Eisler et al. [2012] is used as a basis to compare the impact of different event types. This methodology focuses only on the impact of events, which change the price or quantity of the best bid and ask orders (also known as touch orders), which also constitutes a key limitation of the method.

All events which affect the best prices are classified into one of six order types $\pi \in \Pi$: market orders (MO), limit orders (LO) and cancellations (CA), which are further subdivided into those which affect the mid-price, indicated with subscript 1, and those who do not, with subscript 0: $\Pi = \{MO_0, MO_1, LO_0, LO_1, CA_0, CA_1\}$.

Using the convention in *LOBSTER* data, we define the direction (dir) as 1 for events on the bid side and -1 on the ask side. The events are given an ϵ value based on the expected direction of impact on the mid-price they will provoke. In this context, executions are considered market orders.

$$\epsilon = \begin{cases} dir & \text{if event type is MO or LO;} \\ -dir & \text{if event type is CA.} \end{cases} \tag{6.5}$$

This allows calculation of the response function – equation (6.6). This is calculated empirically using the time average ($\langle \quad . \quad \rangle_T$) of the change in the sign-adjusted mid-price $p_t = \frac{a_t + b_t}{2}$ following a given event, for different lag-times l. The event lag times are chosen to be distributed uniformly on a logarithmic scale between 1 and 200 ticks. The prices are normalized by tick size to enable a comparison

between various stocks.

$$R_\pi(l) = \langle (p_{t+l} - p_t)\epsilon_t | \pi_t = \pi \rangle_T \tag{6.6}$$

Zoltan Eisler et al. [2012] identify averaged response functions for 14 random stocks over a period of 53 trading days. Whilst such analysis gives a good baseline to which we can compare our results, for model evaluation we instead directly compare the functions between model-generated and real sequences (matched based on the seeded starting point or the time of day) for individual stocks. Once the response functions are calculated, we create a measure of comparison to obtain a score of dissimilarity:

$$\Delta R_\pi = \frac{1}{L} \sum_{l=1}^{L} |R_\pi^{real}(l) - R_\pi^{gen}(l)|, \tag{6.7}$$

which is aggregated across all event types by taking the mean $\Delta R = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \Delta R_\pi$.

Adversarial Measurement

The concept of adversarial measurement involved developing a pre-trained discriminator capable of effectively distinguishing between true and generated trajectories. This discriminator is a binary classifier, generating a probability estimate of a trajectory being real. We only use the orderbook states as input, ignoring the message sequences. The discriminator is trained using two batches of data, each of dimension $(S \times T \times D)$. S denotes the number of sequence samples within the batch, T the length of the orderbook sequences, and D is the dimension of the orderbook state representation.

The discriminator aims to find the "worst-case" function Φ^* that maximally separates the real and generated distributions by choosing Φ^* such that it maximizes the divergence between them, i.e., $\Phi^* = \arg\max_\Phi D[p(\Phi(d)), \hat{p}(\Phi(d)))]$. This Φ^*, which can be interpreted as a dimensionality reduction operation on the sequence of order book states $\mathbf{b} \in \mathcal{B}$ to a scalar s, $\Phi^* : (\mathcal{M} \times \mathcal{B}) \mapsto \mathbb{R}$, can be consid-

ered to be an adversarial scoring function. The discriminator attempts to identify the most glaring flaws and differences between the real and generated samples, distilling these into a single dimension.

Given the sparsity of changes between successive orderbook states, we devised an encoding scheme to optimize the discriminator's performance. An orderbook state comprises the price and quantity from the top n price levels on both the bid and ask sides. In our experiments, $n = 10$ resulting in dimension $D = 40$. Changes in the orderbook state are typically triggered by events that affect a single price-quantity pair. To achieve a more concise, yet informative, representation of the discriminator network, we chose to represent the orderbook based on these changes. Thus the book states $\mathbf{b} \in \mathcal{B}$ and message events $\mathbf{m} \in \mathcal{M}$ map to three-dimensional vectors through $i \in \mathbb{N}$ functions $\Psi_i : (\mathcal{M} \times \mathcal{B}) \mapsto \mathbb{R}^3$. These changes encompass each change in the mid-price, the relative price level where the change occurs, and the corresponding change in quantity. Our discriminator utilizes a 1D convolutional neural network (Conv1D) [Lecun and Bengio 1995; Kiranyaz et al. 2019] as a feature extractor, followed by an attention mechanism [Vaswani et al. 2017] to capture long-term dependencies across the time steps. Empirical results show that this model, trained and tested on GOOG data from 2023, achieves a Receiver Operating Characteristic (ROC) score of 0.83, indicating that the generated data can be discriminated reasonably accurately. However, the baseline model's performance for GOOG and INTC was poor, with a discriminator ROC score of around 1, indicating significant room for future model improvement. High discriminability may result from model errors, as indicated by imperfect model scores (see Section 6.2.6 and Figures B.5ff). A distributional mismatch in a single scoring function can be sufficient to make fake data identifiable. To mitigate this issue, future research could evaluate adversarial performance by training a discriminator on perturbed data and reporting scores conditioned on the noise level, particularly as models improve on this benchmark.

Mid-price Prediction

To evaluate the potential value added from additional generated synthetic data, we measure the impact of the additional training data on prediction quality for a simple downstream task. Concretely, we adapt the LOBCAST [Prata et al. 2024] implementation to train an MLP with one million parameters to classify the mid-price movement for different prediction horizons. The MLP is trained to predict three classes: *up*, *down*, and *stationary*. Following Prata et al. [2024], classes are defined based on the movement of the mid-price over a threshold value. For each generative model evaluated, two separate MLPs are trained, one on only real data, and another on real & generated data. F1-scores calculated on held-out real data are then evaluated as a measure of the impact of using the generated data in training.

6.2.5 LOB-Bench Package

Based on the evaluation framework outlined in section 6.2.4, we develop a Python benchmark package, allowing for a convenient and comprehensive evaluation of generated LOB data. The benchmark is highly customizable, as scoring functions Φ can easily be added, removed, or modified, and provides a standardized model comparison using the default scoring functions provided. Table 6.1 describes the main scoring functions computed as part of the standard benchmark. The benchmark reports aggregate model scores by computing the mean, median, and inter-quartile mean (IQM[1]) across all conditional and unconditional scoring functions, along with bootstrapped confidence intervals.

The benchmark performs both unconditional and conditional evaluation of generated data, by computing distributions of statistics of interest conditionally on the value of another statistic. To evaluate the magnitude of the effect of error divergence or "snowballing errors," distributions are also evaluated conditional on the prediction horizon. Distributional accuracy is measured by computing the

[1] mean of all values between the 25. and 75. percentile

Statistic	Description
Bid-Ask Spread	Difference between the highest price a buyer is willing to pay (the bid) and the lowest price a seller is willing to accept (the ask)
Order Book Imbalance	Imbalance for the best prices is computed as (bid size − ask size)/ (bid size + ask size)
Message Inter-Arrival Time	Time between successive order book events (on a log-scale due to a long right tail)
Time-to-Cancel	Time between submission and first (partial) cancellation for cancelled limit orders, measured on a log-scale.
Bid/Ask Volume	The volume of all orders on the bid, respectively ask, side of the LOB. We also evaluate the volume only at the best price levels.
Limit & Cancellation Depths	Absolute distance of new limit orders or cancellations from the mid-price
Limit & Cancellation Levels	The price levels at which events occur $\in \mathbb{N}$
Volume per Minute	Traded volume in one-second intervals, scaled to a minute.
Order Flow Imbalance (OFI)	Metric from [Rama Cont et al. 2014] considering the imbalance in submitted orders for a rolling window of messages.
OFI (Up/Stay/Down)	OFI (see above), conditional on the subsequent message's mid-price move: Up/Static/Down

Table 6.1: Scoring functions computed as part of the standardised LOB-Bench suite.

L_1-norm and Wasserstein-1 distance between the real and generated distributions. Specific supported examples of more complex conditional distributions are the response functions, describing the distribution of events conditional on other events having occurred at a certain prior lag. As these distributions usually have high variance, and to be consistent with the extant literature, we instead measure mean absolute differences in their means for a range of lags to evaluate market impact curves.

We include multiple conditional scoring functions from the finance literature, for example, ask volume conditional on the spread, the spread conditional on the hour of the day, and the spread conditional on the volatility of 10ms returns.

The benchmark also evaluates model response functions (6.6) in aggregate. Individual L_1 distances ΔR_π are calculated for each lag time and averaged to produce aggregate impact scores.

6.2.6 Results

As a first test case for our benchmark, we adapt the autoregressive state-space model using S5 layers [A. Gu et al. 2021b] from Nagy et al. [2023b] (*LOBS5*). Particularly, we scale up the model size to 35 million parameters and more than double the training period to the entire year of 2022.

We also evaluate data generated by the models from Rama Cont et al. [2010b] (*baseline*), Coletta et al. [2022] (*Coletta*) and models based on Peng et al. [2023] and Peng et al. [2024] (*RWKV 4* and *6*). The baseline model, which employs parametric arrival processes, is adapted to generalize across both small and large tick limit order book (LOB) dynamics by utilizing estimated empirical arrival rates directly, rather than fitting a power law. Additionally, we infer data features present in *LOBSTER*, such as individual message IDs, which are not generated by Rama Cont et al. [2010b]. This inference is particularly important for capturing order cancellations, as we uniformly sample target limit orders from the available orders at the specified price level. For the *Coletta* model, we implement a *LOBSTER* data interface to facilitate the conversion of data formats. For the *RWKV* models, we apply autoregressive next-token prediction, but on a larger model (170 million parameters), without any data pre-processing and using an off-the-shelf byte-pair tokenizer [Sennrich et al. 2016], as is used for LLMs. These models are trained solely on message data, without *any* order books, which do not require propagation of a calculated orderbook state, unlike in Nagy et al. [2023b]. The S5, RWKV, and baseline results presented here are computed, following Nagy et al. [2023b], for Alphabet (GOOG) and Intel (INTC) stock on a sub-sample of the test data from January 2023. The *Coletta* model is trained on three days from January 2019 and tested on three subsequent days, following the procedure in Coletta et al. [2022], which is necessary due to the high computational cost for training and inference with *Coletta*. Comparing all models, we observe that the *LOBS5* model provides state-of-the-art performance on the benchmark task.

Figure 6.3 presents a key benchmark feature to compare multiple models across

multiple score dimensions, allowing an examination of individual strengths and weaknesses. To provide summary scores per model, Figure 6.4 reports the mean, median, and interquartile mean for the L1 and Wasserstein-1 metrics for all models [2]. Error bars demarcate the 99% bootstrapped confidence intervals. Metrics for individual scoring functions are shown in Figure B.4 in appendix B.5.

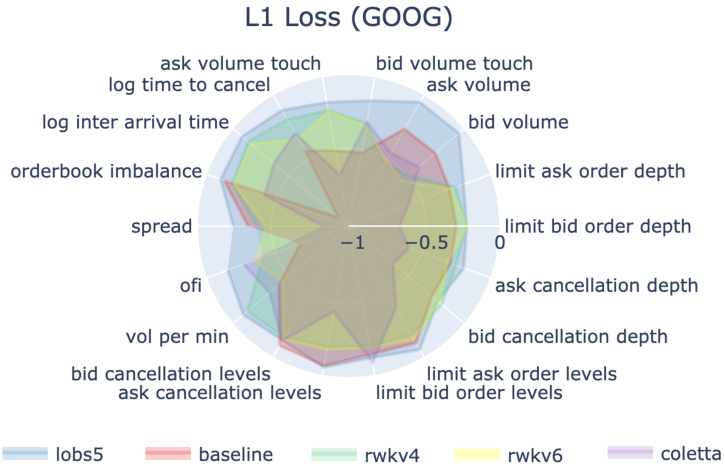


Figure 6.3: **Model comparison spider plot:** the *LOBS5* model beats the *baseline* and *coletta* model on almost all scores. Note: the radial axis is inverted by plotting the negative loss (larger is better).

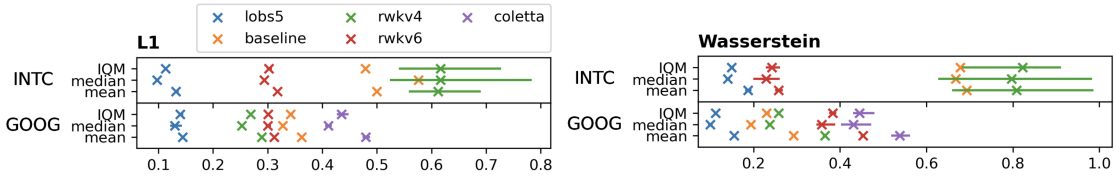


Figure 6.4: **Model score summaries** (lower is better). The *LOBS5* model achieves the lowest overall scores. *Coletta* beats the *baseline* on the Wasserstein metric, but not for L_1. Error bars are bootstrapped 99% CIs.

The benchmark also measures error divergence by comparing distributions of scoring functions, conditional on the inference time step. These demonstrate the rate at which distributions diverge from real data. Results show increasing errors across all models with the fastest divergence exhibited by the RWKV models across most scores. For the S5 model in particular, scoring functions with a dependence on features of the book states, which only gradually change, such as book volume,

[2]The *Coletta* model [Coletta et al. 2022] was trained on both GOOG and INTC data, but failed to produce reasonable results for INTC, which is expected as the model was designed for small-tick stocks, whereas INTC is not.

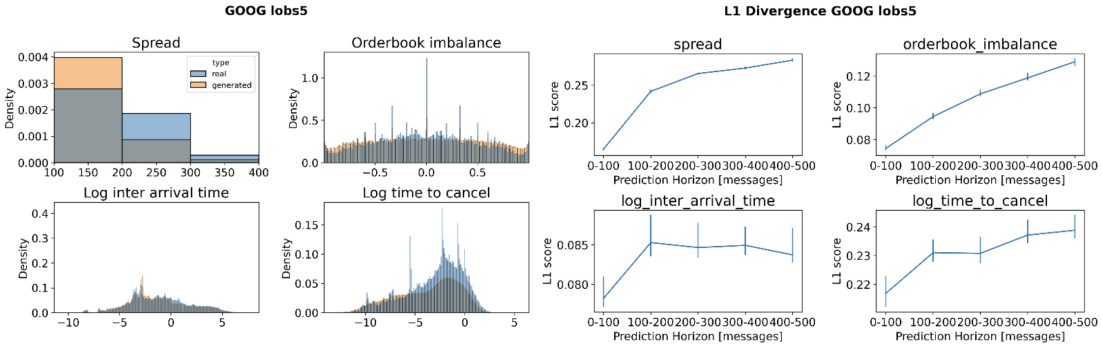


Figure 6.5: ***LOBS5* results** – (left): histogram matching of unconditional score distributions for real and generated data. (right): error accumulation – the further out the prediction horizon, the worse is the model performance – an important model characteristic to measure.

are expected to diverge, as the initial real data seed decays. However, the rate of decay can still be compared between models. See Figure B.10 in Appendix B.5 for L1 divergence curves and Appendix B.4 for a discussion of and an ablation experiment on the effect of bin sizes on divergence scores.

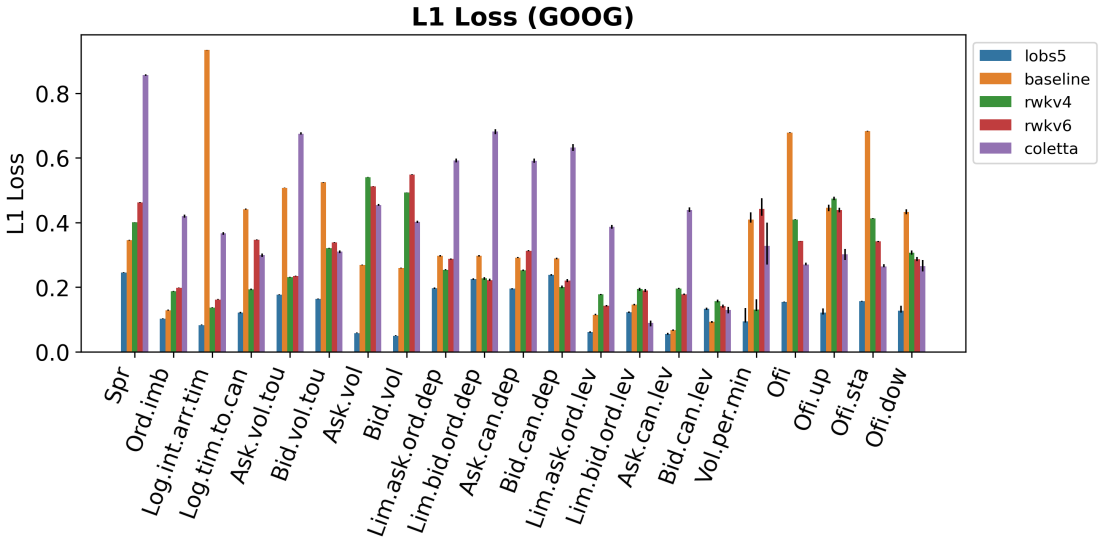


Figure 6.6: **L_1 distance** between real and generated data histograms (incl. 99% CIs). *baseline* performs well on LOB depth and level-related scores, and much worse on time and volume metrics. *LOBS5* dominates L1 loss for GOOG.

The response functions for Alphabet (GOOG) are shown in Figure 6.7 for the *Baseline* and *LOBS5* models. The *LOBS5* model generally reproduces curves similar to real data but does so better for small-tick stock GOOG. In contrast, the *baseline*

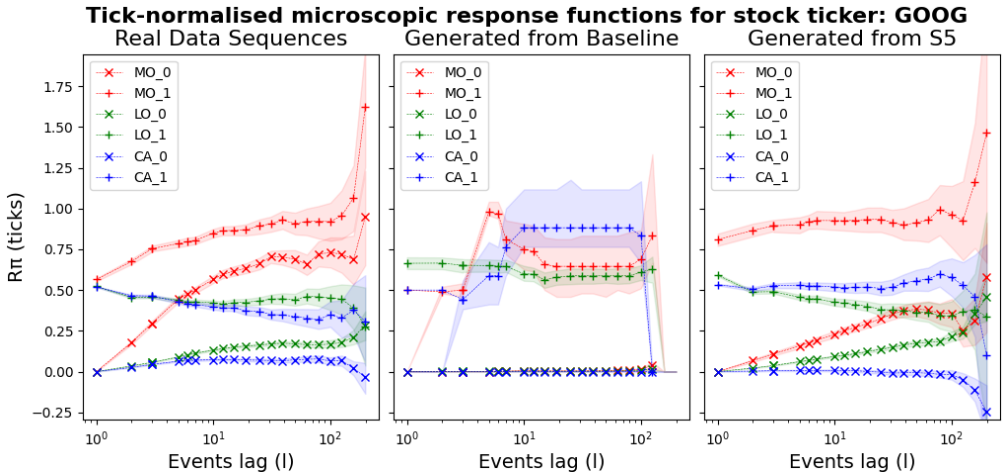


Figure 6.7: Comparison of tick-normalized **mid-price impact response functions** for different event types between real and generated data-sets. Shaded regions are 99% confidence intervals. Compared are the *LOBS5* and stochastic *baseline* models. In contrast to the *baseline*, *LOBS5* reproduces most features of the expected impact response functions.

model [Rama Cont et al. 2010b] cannot faithfully reproduce impact curves. We do not post the impact curves for the *Coletta* and *RWKV* models, as they quickly diverge due to error accumulation in inference. The average L_1 distance between real and generated impact curves (see equation (6.7)) is $\Delta R = \mathbf{2.45}$ for *LOBS5*, and $\Delta R = \mathbf{126}$ for *RWKV-6*. For the *LOBS5* model, we observe differences mainly in the *MO* orders at short lags. This is due to the JAX-LOB simulator [S. Y. Frey et al. 2023], which is used by the *LOBS5* model during inference. JAX-LOB splits limit orders, which can only be partially filled, into execution messages and additional resting limit orders, thereby merging a multilevel midprice change into a single order book update.

F1-scores for mid-price trend forecasting using a simple MLP, following the LOBCAST implementation [Prata et al. 2024], are illustrated for Alphabet (GOOG) in Figure 6.8. We observe that, for the Coletta model in particular, including generated synthetic data markedly reduces the resulting prediction precision and recall. This is in line with model rankings based on the distributional distances shown in Figure 6.5. Generally, the same pattern is also present across the other models, particularly for short prediction horizons, although to a less extreme ex-

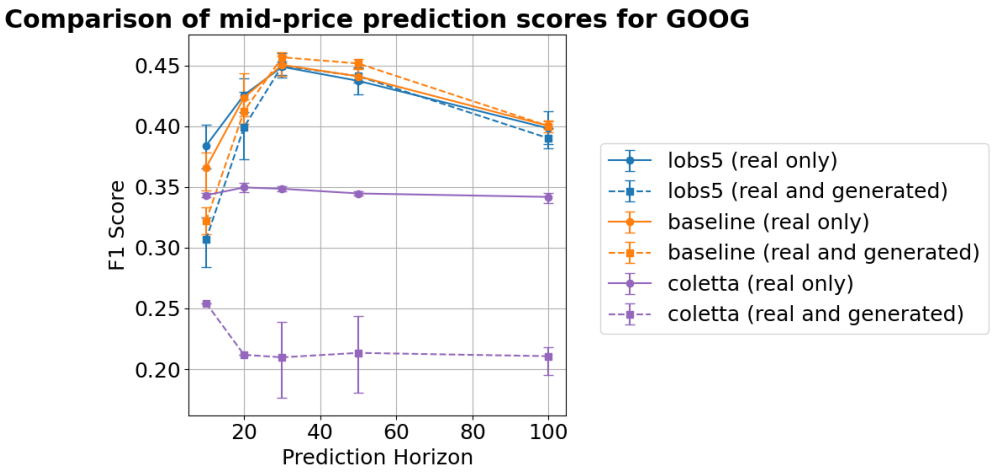


Figure 6.8: Comparison of **F1-scores for the MLP classification models** trained on historical (real) data only (solid lines), and trained on both real data and data generated (dashed lines) for a subset of the generated models. The error bars depict the 95% confidence intervals bootstrapped from five different training seeds.

tent. Mixing generated data from either the baseline or LOB-S5 models into the training set has no significant effect for longer prediction horizons. Although it is desirable for models to generate data which increase prediction performance, not acting detrimentally is a minimum expectation. The results show that current generative models are not yet able to generate data aiding this mid-price trend prediction task. Beyond this conclusion, further quality differentiation between the models based on this task is limited. In contrast, the distributional LOB-Bench evaluation results provide a clearer picture of model strengths and potential for future improvement.

6.2.7 Conclusions

We introduce LOB-Bench, an evaluation framework for generative AI models for order-book modelling. Crucially, our framework contains analysis tools that make it easy for users across the machine learning and finance domains to benchmark their message-level order-flow models.

We believe that LOB-Bench will greatly facilitate core ML research working on sequence modelling to apply their innovations to this challenging and relevant

real-world problem while also making it easier for finance practitioners to use best-practice tools.

One of the interesting aspects of generative AI models for microstructure data is the ability to model counterfactuals, which is closely related to the notion of price impact in financial modelling. Factoring in the reactions of other market participants to one's actions with conventional approaches is very challenging, but our benchmark suite for generative LOB models provides extensive tests to evaluate whether generated data reproduces the expected response functions at a larger scale. Future research could involve measuring the extent to which generative models match the market impact laws in the literature, such as the "square root law" (SRL) [Tóth et al. 2016]. We hope that LOB-Bench opens the door to many new studies, including the training of reinforcement learning algorithms and multi-agent models for tasks such as trade execution, with the ability to model realistic reactions of different market participants.

End of Published Paper

6.3 Discussion and Integration

To ensure compatibility with LOB-Bench, all evaluated models were adapted to produce output in the LOBSTER format. The baseline model from Rama Cont et al. [2010b] was reimplemented in JAX to enable efficient parameter estimation and data generation. To improve adaptability across stocks, we replaced the original power-law assumption with empirically estimated arrival rates per price level, which better capture stock-specific characteristics. The model was also extended with a post-processing step to generate valid numeric order identifiers, which are required for cancellation messages in the LOBSTER format. While the original model cancels arbitrary volume at a given price, LOBSTER specifies individual orders. We address this by uniformly sampling from eligible resting limit orders with sufficient volume at the specified prices. The complete implementation has been open-sourced alongside LOB-Bench, providing users with a readily available baseline that can be fitted to custom datasets and compared against new models.

Thanks to J.P. Morgan AI Research, we were granted access to the unpublished Coletta [Coletta et al. 2022] model code, allowing us to retrain it on INTC and GOOG data – the same datasets used for other models. While the model performs reasonably well on some benchmark dimensions, it fails almost entirely on INTC. We attribute this to modelling assumptions tailored to small tick stocks, which break down for large tick stocks. For GOOG, a small tick stock, a single tick represents a minor percentage of the price. In contrast, INTC's lower absolute price means a one-tick move corresponds to a relatively larger price shift. Large tick stocks therefore exhibit more discrete price changes and distinct distributional characteristics [Zoltán Eisler et al. 2012], which the Coletta model does not capture effectively.

As detailed in the paper, we substantially increased the LOB-S5 model size from 6 to 35 million parameters between the initial publication [Nagy et al. 2023b] and the release of LOB-Bench [Nagy et al. 2025]. While still small compared to billion- or trillion-parameter LLMs, this scale-up led to marked performance gains. Message

error rates, for example due to references to non-existent limit orders, dropped significantly, allowing us to remove the heuristic error correction module from the generation loop. This not only improved generation speed but also enabled more efficient data parallelism.

For the RWKV 4 [Peng et al. 2023] and RWKV 6 [Peng et al. 2024] models, we used 170-million-parameter variants pre-trained on natural language. To adapt them to the LOB-based vocabulary, constructed using a byte-pair tokeniser, we replaced both the embedding and output layers with randomly initialised counterparts. Although these layers had to be trained from scratch, overall training time was significantly reduced compared to training from a fully random initialisation. This aligns with findings in the LLM literature, where pre-training on text followed by task-specific fine-tuning is standard practice [B. Zhang et al. 2024; Parthasarathy et al. 2024].

With LOB-Bench, we provide a rigorous evaluation pipeline for generative LOB models, which is easily extendable to more complex scoring functions as models improve and approach the currently measured distributions up to sampling noise. We advocate moving beyond siloed model development and ad-hoc evaluation toward a more ambitious objective: models should generate order flow that is statistically indistinguishable from real LOB message data. We adopt the LOBSTER data format, a widely used standard in academic research due to its accessibility and broad coverage of stocks and ETFs over multi-year horizons.

The benchmark also provides the first comparative ranking of state-of-the-art models, spanning statistical, GAN-based, and autoregressive deep learning approaches. Results demonstrate the superior performance of the LOB-S5 model, introduced in Chapter 5, which consistently outperforms all other models. We see substantial potential for future work on autoregressive LOB generation. The underperformance of the RWKV models highlights the importance of effective data pre-processing and leveraging both message and order book state information. Nonetheless, future research may aim to simplify the model architecture to better align with the

LLM-style approach of RWKV.

Another open challenge is improving long sequence generation – both in terms of computational efficiency and data quality. Adopting standard LLM architectures enables efficient data parallelism and multi-GPU inference, addressing the former. To reduce error accumulation and improve long-range distributional accuracy, training with sequence-level losses rather than next-token losses offers a promising direction. Methods such as *SeqGAN* [L. Yu et al. 2017] or *time series generation by contrastive imitation* [Jarrett et al. 2021] address the challenge that sequence-level losses are non-differentiable due to intermediate sampling steps and thus incompatible with standard gradient descent, by employing alternative optimisation techniques.

6.4 Author Contributions

In the previous paper "LOB-Bench: Benchmarking Generative AI for Finance," Peer Nagy and Sascha Frey contributed equally to this work. Both worked on the conceptualisation and implementation with input from all authors. Kang Li and Bidipta Sarkar contributed to parts of the project implementation and manuscript preparation. Svitlana Vyetrenko, Stefan Zohren, Anisoara Calinescu, and Jakob Foerster offered guidance and supported project design and manuscript revision.

Chapter 7

Conclusion

7.1 Summary of Contributions

This thesis addresses the challenge of modelling and interacting with high-frequency financial markets by developing machine learning tools tailored to limit order book (LOB) environments. The core contributions span reinforcement learning (RL), scalable simulation, and generative modelling, each forming a crucial building block for data-driven financial agents that learn from and act within complex microstructure dynamics.

The thesis presents a novel application of deep RL to trading-signal execution in LOBs. By leveraging synthetic directional signals, we demonstrate that RL agents trained in realistic LOB environments can learn fine-grained trading strategies that outperform heuristic baselines in return and risk-adjusted performance. These results validate the potential of RL beyond traditional execution problems, enabling dynamic position management based on noisy short-term forecasts.

To enable scalable and efficient experimentation, we introduce JAX-LOB [S. Y. Frey et al. 2023], a GPU-accelerated LOB simulator built on the JAX ecosystem. Overcoming the limitations of static array shapes, and conditional control flow, and by using end-to-end just-in-time compilation (JIT), JAX-LOB achieves up

to 75x speed improvements over conventional simulators. It supports large-scale policy training, forward simulations, and integration with JAX toolchains, such as RL and MARL frameworks, addressing a core bottleneck in applied market microstructure research.

We introduce LOB-S5 [Nagy et al. 2023b], a state-of-the-art autoregressive generative model based on structured state-space layers [J. T. Smith et al. 2022]. Trained on tokenised LOBSTER [R. Huang and Polak 2011] data, the model captures both short- and long-range dependencies in order flow and produces realistic LOB message sequences. Unlike black-box GANs or point-process models, LOB-S5 matches real data distributions much more accurately and even exhibits forecasting ability, evidenced by a significant positive correlation between generated and real mid-prices returns.

To systematically assess the quality of generative models, we introduce LOB-Bench, a comprehensive benchmarking framework that evaluates both unconditional and conditional statistics of generated LOB data. The package also includes divergence metrics and price response functions, enabling reproducible and quantitative comparison across modelling approaches. LOB-Bench is released as an open-source package, promoting transparency and reproducibility in this emerging field.

To summarise, this thesis demonstrates the successful application of RL to trading and inventory management in a LOB environment using noisy alpha signals. To better support future advances using RL in LOB environments, we develop JAX-LOB, a fast, GPU-accelerated, and parallelisable LOB simulator, which can form the computational backbone for large-scale RL training. Beyond single-agent learning, JAX-LOB enables multi-agent simulations and MARL applications, such as learning strategic policies for execution and market making. Building on this foundation, we introduce LOB-S5, the first end-to-end autoregressive generative model for LOB message sequences. Integrated with dynamic book state updates via JAX-LOB, LOB-S5 achieves state-of-the-art performance in our novel eval-

uation framework, LOB-Bench, outperforming CGAN-based and queue-adaptive point-process baselines in like-for-like comparisons. We further evaluate it against a new autoregressive model based on the RWKV architectures [Peng et al. 2023; Peng et al. 2024], trained on raw LOBSTER message data, which underperforms, highlighting the value of domain-specific data pre-processing in this setting.

Taken together, these contributions lay the foundation for a new generation of intelligent, data-driven trading systems. They provide key building blocks for scalable and realistic LOB simulation, including generative world models that capture agent interactions and support reinforcement learning for sequential decision-making. The methodological advances and practical tools developed in this thesis address important gaps in financial machine learning and contribute to broader research on learning and planning in complex, dynamic, and noisy environments. Beyond their academic relevance, these contributions also hold practical value for financial practitioners, offering new tools for strategy development, simulation, and evaluation that can inform real-world trading, execution, or market-making systems.

7.2 On the Design of RL Algorithms in Finance

Reinforcement learning offers a compelling framework for financial market modelling, as it naturally captures the sequential, decision-making nature of trading and portfolio management. Nevertheless, its application in this domain comes with considerable limitations and design challenges that practitioners must navigate carefully.

Financial markets present a particularly hostile environment for RL agents. Market dynamics are non-stationary, partially observable, and subject to structural breaks driven by macroeconomic shifts, regulatory changes, and evolving participant behaviour. Unlike controlled simulation environments, financial markets offer no reset mechanism, and the data-generating process itself may drift over the course of training or deployment. Sample inefficiency is a persistent concern:

historical data is finite, and unlike in physical simulations, the environment cannot be queried arbitrarily. Reward signals are often delayed and confounded by market noise, making credit assignment difficult. Furthermore, the presence of other learning agents in the market introduces a non-stationarity that is endogenous and fundamentally distinct from the i.i.d. assumptions underlying much of statistical learning theory. Overfitting to historical regimes – a well-known hazard in any data-driven financial model – is exacerbated in RL because the policy optimisation process has many more degrees of freedom than a typical supervised model.

The design of the observation space should follow principles analogous to those in supervised deep learning, while accounting for additional challenges specific to RL. Raw price series are non-stationary and trending, and should be replaced wherever possible with stationary transformations such as returns, spreads, or z-scored features with rolling normalisation. Feature engineering and pre-processing of complex states remain necessary: the promise of end-to-end learning from raw inputs, which has proven fruitful in vision and language domains, does not transfer straightforwardly to financial time series, where informative signals are low-dimensional relative to noise. Crucially, environment trajectories in RL are typically not stationary even when individual observations are, because the agent's evolving policy induces distributional shifts in the data it encounters. Two practical strategies follow from this. If sufficient offline data is available, supervised representation learning can be used to pre-train informative state encoders before the RL optimisation loop; otherwise, manual feature engineering guided by domain knowledge remains the more reliable path to a well-conditioned observation space.

The action space should be kept as small as is consistent with expressive policy representations. Continuous action spaces over portfolio weights or order sizes are seductive but substantially increase the variance of policy gradient estimators and the difficulty of exploration. Discretising actions, for instance, into a small number of directional or sizing tiers often yields more stable learning without meaningfully sacrificing policy quality for most practical tasks. Practitioners should resist the

temptation to design large, flexible action spaces on the grounds of generality; complexity in this dimension typically compounds the already substantial instability of RL training in financial settings.

Where a well-defined financial objective exists, such as risk-adjusted return, a Sharpe ratio target, or a mark-to-market P&L, this should serve as the primary reward signal. Using the natural task reward preserves alignment between optimisation and deployment objectives and avoids the subtle but consequential misspecifications that can arise from proxy rewards. When training with the natural reward proves unstable or slow to converge, as frequently occurs when rewards are sparse or delayed, reward shaping should be considered as a pragmatic intermediate step. Shaped rewards might, for example, provide intermediate feedback on inventory risk or transaction cost control. It is essential, however, to verify that the shaping does not inadvertently redirect the agent toward a policy that achieves high shaped reward at the expense of the true objective; this can be partially assessed by monitoring both reward components throughout training.

Validation of learned RL policies is considerably more nuanced than in supervised learning, and the appropriate strategy depends critically on the intended use case. If the practitioner's goal is to deploy a single high-performing policy, it may be sufficient to train across a range of random seeds and hyperparameter configurations and select the best-performing run, provided that this policy is then subjected to rigorous out-of-sample evaluation — across different market regimes, time periods, and stress scenarios. However, this selection procedure has an important implication: it precludes a reliable continuous online learning setup, because the same algorithm that happened to converge well once cannot be assumed to do so reliably on subsequent retraining. For applications requiring adaptive re-learning, a stronger criterion is needed. Ideally, one would evaluate the full distribution of policy performance across a sufficiently large number of seeds, assessing not merely the best-case outcome but the algorithm's consistency in producing good policies. This distributional perspective is standard in rigorous RL benchmarking but is often impractical in financial settings due to computational constraints or

the inherent brittleness of deep RL architectures under distribution shift.

Beyond quantitative performance metrics, interpretability of the learned policy warrants serious attention. Techniques such as attention visualisation, saliency maps over the observation space, and behavioural analysis, characterising how the agent's actions vary systematically with interpretable state variables, can provide both reassurance and diagnostic insight. A policy that achieves strong backtest performance but whose behaviour cannot be understood in terms of plausible market mechanisms should be treated with scepticism, as it may be exploiting spurious regularities in the training data. Taken together, these considerations suggest that RL in finance is best approached as an iterative research process rather than a black-box optimisation procedure, with careful attention to design choices at every stage of the pipeline.

7.3 “Causal” Sequence Modelling

A terminological distinction warrants clarification, as the word "causal" carries substantially different meanings across the communities whose methods intersect in machine learning for finance.

In the sequence modelling literature, and particularly in the context of autoregressive models, “causal” refers to the structural property that each output depends only on prior inputs. Generation proceeds strictly left-to-right, with no access to future observations. A causal model in this sense respects the arrow of time: the prediction at each step is conditioned solely on the past, and no look-ahead bias is introduced. This is a fundamental requirement for any model deployed in a live trading or forecasting context, where future data is by definition unavailable, and its violation constitutes one of the most consequential and insidious forms of data leakage in financial machine learning.

This usage is categorically distinct from “causal inference” as understood in the statistics and econometrics literature, where causality refers to the identification

of intervention effects: the question of what would happen to an outcome if a variable were externally manipulated, as opposed to merely observed to co-vary with it. Causal inference in this sense is concerned with the estimation of structural relationships that remain invariant under distribution shift, and draws on frameworks such as *potential outcomes*, *structural causal models*, and *do-calculus*. The distinction is not merely semantic: a model can be causal in the autoregressive sense while remaining entirely correlational in the inferential sense, and vice versa.

The relationship between predictive modelling and causal inference is a substantive one in the context of ML for finance, and not simply a matter of terminology. Predictive models, including the vast majority of supervised and self-supervised approaches applied to financial time series, optimise for statistical association between inputs and targets. They can achieve strong in-sample and out-of-sample predictive performance while exploiting relationships that are spurious, regime-specific, or subject to structural breaks. Causal structure, by contrast, encodes relationships that are in principle more robust to such shifts because they reflect underlying mechanisms rather than surface-level correlations. It is worth noting, however, that a purely correlational model is not precluded from recovering causal structure: a model trained across a sufficiently long and varied history of market regimes may, in practice, learn associations that are stable precisely because they reflect genuine causal mechanisms. The diversity of training conditions effectively pressures the model toward relationships that generalise. The boundary between predictive and causal is therefore not always sharp in practice, even if it remains conceptually important.

This distinction becomes practically significant in at least two respects. First, a predictive model trained on limited historical data may fail to generalise precisely because it has learned to exploit non-causal associations that do not persist across market regimes. Second, many questions of practical interest in finance, concerning the impact of a trading strategy on market prices, or the effect of a macroeconomic policy on asset returns, are fundamentally causal rather than predictive, and cannot be answered by predictive models alone without strong

additional assumptions.

For practitioners, the key implication is that predictive performance on historical data is a necessary but not sufficient condition for robust deployment, and that the assumptions under which a learned association can be given a causal or structural interpretation deserve explicit scrutiny. Where the goal is forecasting under the assumption of a stationary data-generating process, high-quality predictive modelling may suffice. Where the goal is decision-making under potential distribution shift, or understanding the consequences of interventions, the tools of causal inference become relevant, and the limitations of purely predictive approaches should be honestly acknowledged.

7.4 Societal Impact of Large Financial Models

The integration of machine learning and artificial intelligence into financial markets raises questions that extend well beyond technical performance, touching on market structure, fairness, and the broader social function of financial systems.

Financial markets have never offered a level playing field. Informational asymmetries have always conferred advantage: access to superior data sources, earlier receipt of material news, proximity to exchange infrastructure, and – in more troubling instances – insider knowledge have historically determined who profits and who does not. The technologisation of trading from the 1990s onward extended this dynamic into a new dimension. As algorithmic and subsequently high-frequency trading came to dominate equity and derivatives markets, competitive advantage increasingly resided in engineering capability: the ability to design, implement, and iterate on automated strategies faster than rivals, and to operate profitably at timescales inaccessible to human traders. The relevant horizon compressed from days to milliseconds, and the barriers to entry shifted from financial capital alone toward a compound of financial, computational, and human capital that few institutions could assemble.

The emergence of large-scale machine learning models represents a further evolution of this dynamic, and one whose implications deserve careful prospective scrutiny. One can now plausibly envisage large financial models, analogous in architecture and training philosophy to large language models, trained on vast corpora of market data, news, filings, macroeconomic indicators, and alternative data sources. The development and deployment of such models would require substantial investments in data acquisition, computational infrastructure, and energy, concentrating this capability still further among a small number of well-resourced institutions. If history is any guide, this concentration will translate into systematic informational and strategic advantages, with distributional consequences for other market participants and, potentially, for market stability itself.

These considerations motivate a serious discussion about the role of public and academic investment in this space. Open-weight financial foundation models, developed and released through publicly funded research programmes, could serve several important functions. They would lower the barrier to rigorous independent study of how such models behave when deployed at scale, including their effects on price discovery, liquidity, volatility, and systemic risk. They would also enable researchers in market design and financial regulation to develop empirically grounded frameworks for oversight, rather than reasoning about hypothetical systems they cannot directly examine. Perhaps most importantly, open models would facilitate the study of collective dynamics: when many participants deploy similar or correlated learned strategies, the potential for unintended coordination, feedback loops, or flash-crash-type instabilities increases in ways that are difficult to anticipate from the behaviour of any single model in isolation. The risk of tacit algorithmic collusion, where independently trained agents converge on strategies that are individually rational but collectively anticompetitive, is a concrete regulatory concern that requires access to the models themselves to investigate meaningfully.

The ethical stakes are real. A financial system in which the most consequential participants are opaque learned systems, accessible only to institutions with the resources to develop them, poses challenges for fairness, accountability, and demo-

cratic oversight that extend beyond the usual concerns of quantitative finance. Ensuring that the academic and regulatory communities are not structurally excluded from understanding the systems that increasingly govern capital allocation is not merely a matter of scientific curiosity, it is a prerequisite for informed market governance. Investing in open research infrastructure in this domain is therefore not in tension with commercial innovation, but complementary to the broader goal of markets that are efficient, stable, and fair.

7.5 Future Work

The autoregressive generative LOB paradigm has demonstrated strong performance, but its full potential remains to be realised. Future research could extend this line of work in several directions. One key challenge is improving long sequence generation by mitigating error accumulation over time. A promising avenue is to fine-tune models using sequence-level objectives rather than relying solely on token-level losses. This could involve integrating GAN-style loss functions to encourage the generation of realistic sequences, or applying reinforcement learning to optimise models through non-differentiable intermediate sampling steps. Models could also be trained to explicitly match higher-order distributional statistics, such as those evaluated in LOB-Bench, or to optimise performance on downstream tasks using RL. This would mirror reinforcement learning from human feedback (RLHF) approaches used in the fine-tuning of large language models.

Another promising direction is the exploration of alternative loss functions that better reflect the numeric nature of LOB data. Unlike natural language, LOB messages often encode ordinal or continuous quantities, making them amenable to distance-aware objectives such as the Wasserstein loss. Replacing or augmenting the standard cross-entropy loss with a Wasserstein-based objective could improve both the stability and quality of generation, particularly for rarely observed or numerically structured tokens.

Compared to state-of-the-art large language models, the models presented in this

thesis remain relatively small in terms of parameter count. Scaling up model capacity and training on broader LOB datasets across multiple assets holds significant promise for improving performance. Empirical evidence from recent work, such as Junjie Li et al. [2024], suggests that scaling laws also apply in the LOB domain, indicating that larger models trained on more data can yield systematically better generative performance and generalisation.

Another important research direction concerns computational efficiency. Enhancing model generation speed, through alternative network architectures, optimised GPU kernels, or model quantisation, could significantly broaden the practical applicability of generative LOB models. Reducing latency and computational cost would make these models more viable for real-time or latency-sensitive applications, with the long-term goal of achieving faster-than-real-time simulation and inference.

Another promising research direction lies in advancing multi-agent LOB simulation. Extending JAX-LOB to include multiple interacting agents, trainable via online multi-agent reinforcement learning (MARL) [Rutherford et al. 2024] could enable the development of more robust and generalisable trading policies, while narrowing the sim-to-real performance gap. For instance, training an execution agent alongside an adaptive market-making agent, with a generative model acting as a stochastic background process, would create a more dynamic and adversarial training environment. Such settings could lead to policies that are more resilient to changing market conditions and better aligned with the complexities of real-world trading.

Another promising approach is to apply inverse reinforcement learning (IRL) to infer the underlying reward functions that drive observed market behaviour. A key challenge in this setting is the lack of agent-labelled LOB data, which could be resolved by combining IRL with unsupervised clustering to identify distinct agent types. One potential strategy is to leverage the internal representations learned by generative models: if the model captures agent-specific structure to improve

generation quality, then its embeddings may implicitly encode latent agent types. These embeddings could serve as the basis for clustering and, in turn, inform the reward functions used in downstream IRL tasks.

The ultimate objective for future LOB models is the development of a causal, multi-asset simulator capable of ingesting time-ordered LOB data alongside relevant contextual information – such as news or macroeconomic events – and inferring the agent type responsible for each order based on the current market context. Such a model would integrate structured market microstructure data with unstructured external signals, while capturing the strategic heterogeneity of market participants. Although this vision may still appear distant, the pace of progress in machine learning suggests that such developments could arrive sooner than anticipated. Building toward this goal will require advances in causal inference, multi-modal learning, and agent identification from unlabelled data.

References

Frédéric Abergel, Marouane Anane, Anirban Chakraborti, Aymen Jedidi, and Ioane Muni Toke (2016). *Limit order books*. Cambridge University Press.

Jacob D Abernethy and Satyen Kale (2013). "Adaptive market making via online learning". In: *Advances in Neural Information Processing Systems*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. (2023). "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774*.

Robert Almgren and Neil Chriss (Jan. 2001). "Optimal execution of portfolio transactions". In: *The Journal of Risk*. Publisher: Infopro Digital Services Limited.

Robert Almgren, Chee Thum, Emmanuel Hauptmann, and Hong Li (2005). "Direct Estimation of Equity Market Impact". en. In: *Risk*.

Selim Amrouni, Aymeric Moulin, Jared Vann, Svitlana Vyetrenko, Tucker Balch, and Manuela Veloso (Oct. 2021). "ABIDES-Gym: Gym Environments for Multi-Agent Discrete Event Simulation and Application to Financial Markets". In: *arXiv:2110.14771 [cs, q-fin]*. arXiv: 2110.14771.

Selim Amrouni, Aymeric Moulin, Jared Vann, Svitlana Vyetrenko, Tucker Balch, and Manuela Veloso (May 2022). "ABIDES-gym: gym environments for multi-agent discrete event simulation and application to financial markets". In: *Proceedings of the Second ACM International Conference on AI in Finance*. ICAIF '21. New York, NY, USA: Association for Computing Machinery.

Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso (2020). "Generating synthetic data in

finance: opportunities, challenges and pitfalls". In: *Proceedings of the First ACM International Conference on AI in Finance*.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola (2020). *The DeepMind JAX Ecosystem*.

Alfred Backhouse, Kang Li, Jakob Foerster, Anisoara Calinescu, and Stefan Zohren (2025). "Painting the market: generative diffusion models for financial limit order book simulation and forecasting". In: *arXiv preprint arXiv:2509.05107*.

Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy (2015). "Hawkes processes in finance". In: *Market Microstructure and Liquidity*.

David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba (2019). "Seeing what a gan cannot generate". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Peter Belcak, Jan-Peter Calliess, and Stefan Zohren (2022). "Fast Agent-Based Simulation Framework with Applications to Reinforcement Learning and the Study of Trading Latency Effects". en. In: *Multi-Agent-Based Simulation XXII*. Ed. by Koen H. Van Dam and Nicolas Verstaevel. Lecture Notes in Computer Science. Cham: Springer International Publishing.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The arcade learning environment: An evaluation platform for general agents". In: *Journal of artificial intelligence research*.

Richard Bellman (1957). "A Markovian decision process". In: *Journal of mathematics and mechanics*.

Dimitris Bertsimas and Andrew W. Lo (Apr. 1998). "Optimal control of execution costs". en. In: *Journal of Financial Markets*.

Taweh Beysolow II and Taweh Beysolow II (2019). "Market making via reinforcement learning". In: *Applied Reinforcement Learning with Python: With OpenAI Gym, Tensorflow, and Keras*.

Siddharth Bhatia, Arjit Jain, and Bryan Hooi (2021). "Exgan: Adversarial generation of extreme samples". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Christopher M Bishop and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Springer.

Tim Bollerslev (1986). "Generalized autoregressive conditional heteroskedasticity". In: *Journal of econometrics*.

Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould (2018). *Trades, quotes and prices: financial markets under the microscope*. Cambridge: Cambridge University Press.

Jean-Philippe Bouchaud, Marc Mézard, and Marc Potters (2002). "Statistical properties of stock order books: empirical results and models". In: *Quantitative finance*.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*.

Antonio Briola, Jeremy Turiel, and Tomaso Aste (2020). "Deep learning modeling of limit order book: A comparative perspective". In: *arXiv preprint arXiv:2007.07319*.

Antonio Briola, Jeremy Turiel, Riccardo Marcaccioli, and Tomaso Aste (2021). "Deep Reinforcement Learning for Active High Frequency Trading". In: *arXiv preprint arXiv:2101.07107*.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "Openai gym". In: *arXiv preprint arXiv:1606.01540*.

X. Brokmann, E. Sérié, J. Kockelkoren, and J.-P. Bouchaud (Dec. 2015). "Slow Decay of Impact in Equity Markets". In: *Market Microstructure and Liquidity*. Publisher: World Scientific Publishing Co.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). "Language models are few-shot learners". In: *Advances in neural information processing systems*.

David Byrd, Maria Hybinette, and Tucker Hybinette Balch (June 2020). "ABIDES: Towards High-Fidelity Multi-Agent Market Simulation". In: *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS '20. New York, NY, USA: Association for Computing Machinery.

Defu Cao, Yousef El-Laham, Loc Trinh, Svitlana Vyetrenko, and Yan Liu (2022). "DSLOB: a synthetic limit order book dataset for benchmarking forecasting algorithms under distributional shift". In: *arXiv preprint arXiv:2211.11513*.

Álvaro Cartea, Patrick Chang, and Rob Graumans (2025). "Anonymity, Signaling, and Collusion in Limit Order Books". In: *Available at SSRN*.

Álvaro Cartea, Patrick Chang, and José Penalva (2022). "Algorithmic collusion in electronic markets: The impact of tick size". In: *Available at SSRN 4105954*.

Álvaro Cartea, Ryan Donnelly, and Sebastian Jaimungal (2018). "Enhancing trading strategies with order book signals". In: *Applied Mathematical Finance*.

Álvaro Cartea, Luhui Gan, and Sebastian Jaimungal (2019). "Trading co-integrated assets with price impact". In: *Mathematical Finance*.

Álvaro Cartea and Sebastian Jaimungal (Nov. 2016a). "A Closed-Form Execution Strategy to Target Volume Weighted Average Price". en. In: *SIAM Journal on Financial Mathematics*. Publisher: Society for Industrial and Applied Mathematics.

Álvaro Cartea and Sebastian Jaimungal (2016b). "Incorporating order-flow into optimal execution". In: *Mathematics and Financial Economics*.

Álvaro Cartea, Sebastian Jaimungal, and Yixuan Wang (2020). "Spoofing and price manipulation in order-driven markets". In: *Applied Mathematical Finance*.

Philippe Casgrain (2020). *LimitOrderBook.jl*. Version 0.0.1.

Anirban Chakraborti, Ioane Muni Toke, Marco Patriarca, and Frédéric Abergel (July 2011). "Econophysics review: II. Agent-based models". In: *Quantitative Finance*. Publisher: Routledge _eprint: https://doi.org/10.1080/14697688.2010.539249.

Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld (1998). *Evaluation metrics for language models*. Tech. rep. Carnegie Mellon University.

Carl Chiarella and Giulia Iori (Oct. 2002). "A simulation analysis of the microstructure of double auction markets*". In: *Quantitative Finance*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever (2019). "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. (2022). "Palm: Scaling language modeling with pathways". In: *arXiv preprint arXiv:2204.02311*.

Andrea Coletta, Joseph Jerome, Rahul Savani, and Svitlana Vyetrenko (2023). "Conditional Generators for Limit Order Book Environments: Explainability, Challenges, and Robustness". In: *arXiv preprint arXiv:2306.12806*.

Andrea Coletta, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch (2022). "Learning to simulate realistic limit order book markets from data as a World Agent". In: *Proceedings of the Third ACM International Conference on AI in Finance*.

Andrea Coletta, Matteo Prata, Michele Conti, Emanuele Mercanti, Novella Bartolini, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch (2021). "Towards realistic market simulations: a generative adversarial networks approach". In: *Proceedings of the Second ACM International Conference on AI in Finance*.

R. Cont (Feb. 2001). "Empirical properties of asset returns: stylized facts and statistical issues". en. In: *Quantitative Finance*.

Rama Cont (2011). "Statistical modeling of high-frequency financial data". In: *IEEE Signal Processing Magazine*.

Rama Cont, Mihai Cucuringu, Vacslav Glukhov, and Felix Prenzel (2023a). "Analysis and modeling of client order flow in limit order markets". In: *Quantitative Finance*.

Rama Cont, Mihai Cucuringu, Jonathan Kochems, and Felix Prenzel (2023b). "Limit order book simulation with generative adversarial networks". In: *Available at SSRN 4512356*.

Rama Cont, Mihai Cucuringu, Renyuan Xu, and Chao Zhang (2022). "Tail-GAN: Learning to Simulate Tail Risk Scenarios". In: *Available at SSRN 3812973*.

Rama Cont, Mihai Cucuringu, and Chao Zhang (2021). "Cross Impact of Order Flow Imbalances: Contemporaneous and Predictive". In: *arXiv preprint arXiv:2112.13213*.

Rama Cont, Arseniy Kukanov, and Sasha Stoikov (Jan. 2014). "The Price Impact of Order Book Events". In: *Journal of Financial Econometrics*.

Rama Cont, Sasha Stoikov, and Rishi Talreja (June 2010a). "A Stochastic Model for Order Book Dynamics". en. In: *Operations Research*.

Rama Cont, Sasha Stoikov, and Rishi Talreja (2010b). "A stochastic model for order book dynamics". In: *Operations research*.

Thomas M Cover and Joy A Thomas (1999). *Elements of information theory*. John Wiley & Sons.

Matthieu Cristelli (2013). *Complexity in financial markets: modeling psychological behavior in agent-based models and order book models*. Springer Science & Business Media.

Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah (2023). "Diffusion models in vision: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kevin Dabérius, Elvin Granat, and Patrik Karlsson (2019). "Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks". In: *Available at SSRN 3374766*.

Tri Dao (2023). "Flashattention-2: Faster attention with better parallelism and work partitioning". In: *arXiv preprint arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré (2022). "Flashattention: Fast and memory-efficient exact attention with io-awareness". In: *Advances in neural information processing systems*.

Tri Dao and Albert Gu (2024). "Transformers are ssms: Generalized models and efficient algorithms through structured state space duality". In: *arXiv preprint arXiv:2405.21060*.

Aaron Defazio and Konstantin Mishchenko (2023). "Learning-Rate-Free Learning by D-Adaptation". In: *The 40th International Conference on Machine Learning (ICML 2023)*.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, and Diego de Las Casas (2022). "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee.

Yuntian Deng, Yejin Choi, and Stuart Shieber (2024). "From explicit cot to implicit cot: Learning to internalize cot step by step". In: *arXiv preprint arXiv:2405.14838*.

Laurent Deville (2008). "Exchange traded funds: History, trading, and research". In: *Handbook of financial engineering*. Springer.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (20–22 Jun 2016). "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Proceedings of Machine Learning Research. New York, New York, USA: PMLR.

Abhimanyu Dubey and et. al. (July 2024). *The Llama 3 Herd of Models*. en.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. (2023). "Faith and fate: Limits of transformers on compositionality". In: *Advances in Neural Information Processing Systems*.

Florian Eckerli and Joerg Osterrieder (2021). "Generative Adversarial Networks in finance: an overview". In: *arXiv preprint arXiv:2106.06364*.

Zoltan Eisler, Jean-Philippe Bouchaud, and Julien Kockelkoren (2012). "The price impact of order book events: market orders, limit orders and cancellations". In: *Quantitative Finance*.

Zoltán Eisler, Jean-Philippe Bouchaud, and Julien Kockelkoren (Sept. 2012). "The price impact of order book events: market orders, limit orders and cancellations". In: *Quantitative Finance*. Publisher: Routledge _eprint: https://doi.org/10.1080/14697688.2010.528444.

Jin Fang, Jiacheng Weng, Yi Xiang, and Xinwen Zhang (2022). "Imitate then Transcend: Multi-Agent Optimal Execution with Dual-Window Denoise PPO". In: *arXiv preprint arXiv:2206.10736*.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, and Grzegorz Swirszcz (2022). "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature*.

David Freedman and Persi Diaconis (1981). "On the histogram as a density estimator: L 2 theory". In: *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem (2021). "Brax–a differentiable physics engine for large scale rigid body simulation". In: *arXiv preprint arXiv:2106.13281*.

Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu (2023). "JAX-LOB: A GPU-Accelerated limit order book simulator to unlock large scale reinforcement learning for trading". In: *Proceedings of the Fourth ACM International Conference on AI in Finance*.

Roman Frigg and Stephan Hartmann (2025). "Models in Science". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Summer 2025. Metaphysics Research Lab, Stanford University.

Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso (2019). "Reinforcement learning for market making in a multi-agent dealer market". In: *arXiv preprint arXiv:1911.05892*.

Xuefeng Gao and SJ Deng (2018). "Hydrodynamic limit of order-book dynamics". In: *Probability in the Engineering and Informational Sciences*.

Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni (2020). "Tadgan: Time series anomaly detection using generative adversarial networks". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*.

Google (Mar. 2023). *XLA: Optimizing Compiler for machine learning : tensorflow*.

Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison (2013). "Limit order books". In: *Quantitative Finance*.

Albert Gu and Tri Dao (2023). "Mamba: Linear-time sequence modeling with selective state spaces". In: *arXiv preprint arXiv:2312.00752*.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré (2020). "Hippo: Recurrent memory with optimal polynomial projections". In: *Advances in neural information processing systems*.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré (2022). "On the parameterization and initialization of diagonal state space models". In: *Advances in Neural Information Processing Systems*.

Albert Gu, Karan Goel, and Christopher Re (2021a). "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations*.

Albert Gu, Karan Goel, and Christopher Ré (2021b). "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations*.

Gao-Feng Gu, Wei Chen, and Wei-Xing Zhou (2008). "Empirical regularities of order placement in the Chinese stock market". In: *Physica A: Statistical Mechanics and its Applications*.

Siyuan Guo, Aniket Didolkar, Nan Rosemary Ke, Anirudh Goyal, Ferenc Huszár, and Bernhard Schölkopf (2024). "Learning beyond pattern matching? assaying mathematical understanding in llms". In: *arXiv preprint arXiv:2405.15485*.

Ankit Gupta, Albert Gu, and Jonathan Berant (2022). "Diagonal state spaces are as effective as structured state spaces". In: *Advances in Neural Information Processing Systems*.

Ben Hambly, Renyuan Xu, and Huining Yang (2023). "Recent advances in reinforcement learning in finance". In: *Mathematical Finance*.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant (Sept. 2020). "Array programming with NumPy". In: *Nature*.

Larry Harris (2003). *Trading and exchanges: Market microstructure for practitioners*. OUP USA.

Hado Hasselt (2010). "Double Q-learning". In: *Advances in Neural Information Processing Systems*.

Simon Haykin (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee (2024). *Flax: A neural network library and ecosystem for JAX*. Version 0.10.6.

Pierre Henry-Labordere (2019). "Generative models for financial data". In: *Available at SSRN 3408007*.

Sepp Hochreiter and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Computation*. Conference Name: Neural Computation.

Cars H Hommes (2001). "Financial markets as nonlinear adaptive evolutionary systems". In: *Quantitative Finance*.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver (2018). "Distributed Prioritized Experience Replay". In: *arXiv preprint arXiv:1803.00933*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. (2022). "Lora: Low-rank adaptation of large language models." In: *ICLR*.

Charles Huang, Weifeng Ge, Hongsong Chou, and Xin Du (2021). "Benchmark dataset for short-term market prediction of limit order book in china markets". In: *The Journal of Financial Data Science*.

Ruihong Huang and Tomas Polak (2011). "Lobster: Limit order book reconstruction system". In: *Available at SSRN 1977207*.

Weibing Huang, Charles-Albert Lehalle, and Mathieu Rosenbaum (2015). "Simulating and analyzing order book data: The queue-reactive model". In: *Journal of the American Statistical Association*.

Henrik Hult and Jonas Kiessling (2010). "Algorithmic trading with Markov chains". PhD thesis on webpage https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A355746&dswid=9147.

Hanna Hultin, Henrik Hult, Alexandre Proutiere, Samuel Samama, and Ala Tarighati (2023). "A generative model of a limit order book using recurrent neural networks". In: *Quantitative Finance*.

Konark Jain, Nick Firoozye, Jonathan Kochems, and Philip Treleaven (2024). "Limit order book simulations: A review". In: *arXiv preprint arXiv:2402.17359*.

Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar (2021). "Time-series generation by contrastive imitation". In: *Advances in neural information processing systems*.

Joseph Jerome, Gregory Palmer, and Rahul Savani (Nov. 2022a). "Market Making with Scaled Beta Policies". en. In: *Proceedings of the Third ACM International Conference on AI in Finance*. New York NY USA: ACM.

Joseph Jerome, Leandro Sanchez-Betancourt, Rahul Savani, and Martin Herdegen (Sept. 2022b). *Model-based gym environments for limit order book trading*. arXiv:2209.07823 [cs, q-fin].

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. (2021). "Highly accurate protein structure prediction with AlphaFold". In: *Nature*.

Michäel Karpe, Jin Fang, Zhongyao Ma, and Chen Wang (2020). "Multi-agent reinforcement learning in a realistic limit order book market simulation". In: *Proceedings of the First ACM International Conference on AI in Finance*. ACM.

Takashi Kato (2015). "VWAP execution as an optimal strategy". In: *JSIAM Letters*.

Michael Kearns and Yuriy Nevmyvaka (2013). "Machine learning for market microstructure and high frequency trading". In: *High Frequency Trading: New Realities for Traders, Markets, and Regulators*.

Frank Kelly and Elena Yudovina (2018). “A Markov model of a limit order book: thresholds, recurrence, and trading strategies”. In: *Mathematics of Operations Research*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova (2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of NAACL-HLT*.

Diederik Kingma and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA.

Diederik P. Kingma and Max Welling (2014). “Auto-Encoding Variational Bayes”. In: *Proc. ICLR*.

Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman (May 2019). *1D Convolutional Neural Networks and Applications: A Survey*. arXiv:1905.03554.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer (May 2021). “Machine learning–accelerated computational fluid dynamics”. In: *Proceedings of the National Academy of Sciences*. Publisher: Proceedings of the National Academy of Sciences.

Petter N Kolm, Jeremy Turiel, and Nicholas Westray (2021). “Deep order flow imbalance: Extracting alpha at multiple horizons from the limit order book”. In: *Available at SSRN 3900141*.

Alireza Koochali, Peter Schichtel, Andreas Dengel, and Sheraz Ahmed (2019). “Probabilistic Forecasting of Sensory Data With Generative Adversarial Networks – ForGAN”. In: *IEEE Access*.

Jerzy Korczak and Marcin Hemes (2017). “Deep learning for financial time series forecasting in a-trader system”. In: *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE.

A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NeurIPS*.

Pankaj Kumar (2020). “Deep reinforcement learning for market making”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*.

Albert S. Kyle (1985). "Continuous Auctions and Insider Trading". In: *Econometrica*. Publisher: [Wiley, Econometric Society].

Robert Tjarko Lange (2022). *gymnax: A JAX-based Reinforcement Learning Environment Library*. Version 0.0.4.

Robert Tjarko Lange (2023). "evosax: Jax-based evolution strategies". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*.

Yann Lecun and Yoshua Bengio (1995). "Convolutional networks for images, speech, and time-series". In: *The handbook of brain theory and neural networks*. MIT Press.

Cheuk Ting Li and Farzan Farnia (2023). "Mode-Seeking Divergences: Theory and Applications to GANs". In: *International Conference on Artificial Intelligence and Statistics*. PMLR.

Junjie Li, Yang Liu, Weiqing Liu, Shikai Fang, Lewen Wang, Chang Xu, and Jiang Bian (2024). "MarS: a Financial Market Simulation Engine Powered by Generative Foundation Model". In: *arXiv preprint arXiv:2409.07486*.

Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael Wellman (2020). "Generating realistic stock market order streams". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica (2018). "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. PMLR.

Ye-Sheen Lim and Denise Gorse (2021). "Intra-Day Price Simulation with Generative Adversarial Modelling of the Order Flow". In: *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE.

Chunli Liu, Carmine Ventre, and Maria Polukarov (2022). "Synthetic Data Augmentation for Deep Reinforcement Learning in Financial Trading". In: *Proceedings of the Third ACM International Conference on AI in Finance*.

Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long (Feb. 2024). *AutoTimes: Autoregressive Time Series Forecasters via Large Language Models*. en.

Bàrbara Llacay and Gilbert Peffer (Sept. 2018). "Using realistic trading strategies in an agent-based stock market model". en. In: *Computational and Mathematical Organization Theory*.

Edward N Lorenz (1996). "Predictability: A problem partly solved". In: *Proc. Seminar on predictability*. Reading.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster (2022). "Discovered policy optimisation". In: *Advances in Neural Information Processing Systems*.

Xiaofei Lu and Frédéric Abergel (2018). "High-dimensional Hawkes processes for limit order books: modelling, empirical analysis and numerical calibration". In: *Quantitative Finance*.

Hugh Luckock (2001). "A statistical model of a limit order market". In: *Sidney University preprint (September 2001)*.

Hugh Luckock (2003). "Asteady-state model of the continuous double auction". In: *Quantitative Finance*.

Rui Mao, Guanyi Chen, Xulang Zhang, Frank Guerin, and Erik Cambria (2024). "GPTEval: A Survey on Assessments of ChatGPT and GPT-4". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*.

Robert Marschinski and Lorenzo Matassini (2001). *Financial markets as a complex system: A short time scale perspective*. Tech. rep. Frankfurt a. M.: Deutsche Bank Research.

Henry Martin and Scott Otterson (2018). "German intraday electricity market analysis and modeling based on the limit order book". In: *2018 15th international conference on the European energy market (EEM)*. IEEE.

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster (2024). "Kinetix: Investigating the Training of General Agents through Open-Ended Physics-Based Control Tasks". In: *arXiv preprint arXiv:2410.23208*.

Mehdi Mirza and Simon Osindero (2014). "Conditional Generative Adversarial Nets". In: *CoRR*. arXiv: 1411.1784.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *CoRR*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602*.

Mordor Intelligence (2022). "Algorithmic Trading Market Share, Size & Trading Statistics". In: https://www.mordorintelligence.com/industry-reports/algorithmic-trading-market Accessed: 2023-08-02.

Kevin P Murphy (2012). *Machine learning: a probabilistic perspective*. MIT press.

Peer Nagy, Jan-Peter Calliess, and Stefan Zohren (2023a). "Asynchronous Deep Double Dueling Q-learning for trading-signal execution in limit order book markets". In: *Frontiers in Artificial Intelligence*.

Peer Nagy, Sascha Frey, Kang Li, Bidipta Sarkar, Svitlana Vyetrenko, Stefan Zohren, Anisoara Calinescu, and Jakob Foerster (2025). "LOB-Bench: Benchmarking Generative AI for Finance - an Application to Limit Order Book Data". In: *Proceedings of the 42nd International Conference on Machine Learning*. PMLR. Vancouver, Canada.

Peer Nagy, Sascha Frey, Silvia Sapora, Kang Li, Anisoara Calinescu, Stefan Zohren, and Jakob Foerster (2023b). "Generative AI for End-to-End Limit Order Book Modelling: A Token-Level Autoregressive Generative Model of Message Flow Using a Deep State Space Network". In: *Proceedings of the Fourth ACM International Conference on AI in Finance*.

Peer Nagy, James Powrie, and Stefan Zohren (2023c). "Machine learning for microstructure data-driven execution algorithms". In: *Handbook of Artificial Intelligence and Big Data Applications in Investments*.

Yusuke Naritomi and Takanori Adachi (2020). "Data augmentation of high frequency financial data using generative adversarial network". In: *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. IEEE.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns (2006a). "Reinforcement learning for optimized trade execution". en. In: *Proceedings of the 23rd international conference on Machine learning - ICML '06*. Pittsburgh, Pennsylvania: ACM Press.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns (2006b). "Reinforcement learning for optimized trade execution". In: *Proceedings of the 23rd international conference on Machine learning*.

Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M. Mulvey, H. Vincent Poor, Qingsong Wen, and Stefan Zohren (June 2024). *A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges*. en.

Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal (2021). "Double deep q-learning for optimal execution". In: *Applied Mathematical Finance*.

Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2018). "Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods". In: *Journal of Forecasting*.

Anna A. Obizhaeva and Jiang Wang (Feb. 2013). "Optimal trading strategy and supply/demand dynamics". en. In: *Journal of Financial Markets*.

Georg Ostrovski, Will Dabney, and Rémi Munos (2018). "Autoregressive quantile networks for generative modeling". In: *International Conference on Machine Learning*. PMLR.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid (2024). "The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities". In: *arXiv preprint arXiv:2408.13296*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio (2013). *On the difficulty of training Recurrent Neural Networks*. arXiv: 1211.5063 [cs.LG].

Badri Narayana Patro and Vijay Srinivas Agneeswaran (2024). "Mamba-360: Survey of state space models as transformer alternative for long sequence modelling: Methods, applications, and challenges". In: *arXiv preprint arXiv:2404.16112*.

James Paulin (Jan. 2019). *Understanding flash crash contagion and systemic risk: a calibrated agent-based approach*. en.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. (2023). "Rwkv: Reinventing rnns for the transformer era". In: *arXiv preprint arXiv:2305.13048*.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, et al. (2024). "Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence". In: *arXiv preprint arXiv:2404.05892*.

Du Phan, Neeraj Pradhan, and Martin Jankowiak (2019). "Composable effects for flexible and accelerated probabilistic programming in NumPyro". In: *arXiv preprint arXiv:1912.11554*.

Matteo Prata, Giuseppe Masi, Leonardo Berti, Viviana Arrigoni, Andrea Coletta, Irene Cannistraci, Svitlana Vyetrenko, Paola Velardi, and Novella Bartolini (2024). "Lob-based deep learning models for stock price trend prediction: a benchmark study". In: *Artificial Intelligence Review*.

Martin L Puterman (1990). "Markov decision processes". In: *Handbooks in operations research and management science*.

Arnau Quera-Bofarull, Joel Dyer, Anisoara Calinescu, and Michael Wooldridge (July 2023). *Some challenges of calibrating differentiable agent-based models*. arXiv:2307.01085 [cs, q-fin, stat].

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners". In: *OpenAI blog*.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer (June 2022). "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ioanid Roşu (2009). "A dynamic model of the limit order book". In: *The Review of Financial Studies*.

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas (2000). "The Earth Mover's Distance as a Metric for Image Retrieval". In: *Int. J. Comput. Vis.*

David E Rumelhart and James L McClelland (1987). "Learning Internal Representations by Error Propagation". In.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson Juto, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, et al. (2024). "Jaxmarl: Multi-agent rl environments and algorithms in jax". In: *Advances in Neural Information Processing Systems*.

Matthias Schnaubelt (2022). "Deep reinforcement learning for the optimal placement of cryptocurrency limit orders". In: *European Journal of Operational Research*.

Matthias Schnaubelt, Jonas Rende, and Christopher Krauss (2019). "Testing stylized facts of bitcoin limit order books". In: *Journal of Risk and Financial Management*.

Samuel S. Schoenholz and Ekin D. Cubuk (2020). "JAX M.D. A Framework for Differentiable Physics". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Maarten Scholl (2022). *Economic Simulation Library*. https://github.com/INET-Complexity/ESL.

Maarten P. Scholl, Anisoara Calinescu, and J. Doyne Farmer (June 2021). "How market ecology explains market malfunction". In: *Proceedings of the National Academy of Sciences*. Publisher: Proceedings of the National Academy of Sciences.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347*.

Rico Sennrich, Barry Haddow, and Alexandra Birch (Aug. 2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu (2020). "Financial time series forecasting with deep learning : A systematic literature review: 2005–2019". In: *Applied Soft Computing*.

Zijian Shi, Yu Chen, and John Cartlidge (2021). "The lob recreation model: Predicting the limit order book from taq history using an ordinary differential equation recurrent neural network". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton (2017). "Mastering the game of Go without human knowledge". In: *nature*.

Justin Sirignano and Rama Cont (2019). "Universal features of price formation in financial markets: perspectives from deep learning". In: *Quantitative Finance*.

Justin A Sirignano (2019). "Deep learning for limit order books". In: *Quantitative Finance*.

Eric Smith, J Doyne Farmer, László Gillemot, and Supriya Krishnamurthy (2003). "Statistical theory of the continuous double auction". In: *Quantitative finance*.

Jimmy TH Smith, Andrew Warrington, and Scott Linderman (2022). "Simplified State Space Layers for Sequence Modeling". In: *The Eleventh International Conference on Learning Representations*.

David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le (2021). "Searching for efficient transformers for language modeling". In: *Advances in neural information processing systems*.

Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon (2018). "Multi-agent generative adversarial imitation learning". In: *Advances in neural information processing systems*.

Matthijs TJ Spaan (2012). "Partially observable Markov decision processes". In: *Reinforcement learning: State-of-the-art*. Springer.

T. Spooner, J. Fearnley, R. Savani, and A. Koukorinis (2018). "Market Making via Reinforcement Learning". In: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. Stockholm, Sweden.

I. Sutskever, O. Vinyals, and Q. Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*.

Richard Sutton (2019). *The Bitter Lesson*. http://www.incompleteideas.net/IncIdeas/BitterLesson.html. Accessed: 2025-06-02.

Richard S Sutton and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going Deeper with Convolutions". In: *Proc. CVPR*.

Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii (2019). "Modeling financial time-series with generative adversarial networks". In: *Physica A: Statistical Mechanics and its Applications*.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente (2017). "Multiagent cooperation and competition with deep reinforcement learning". In: *PloS one*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler (2020). "Long Range Arena: A Benchmark for Efficient Transformers". In: *International Conference on Learning Representations*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler (2022). "Efficient transformers: A survey". In: *ACM Computing Surveys*.

Bence Tóth, Zoltán Eisler, and J-P Bouchaud (2016). "The Square-Root Impact Law Also Holds for Option Markets". In: *Wilmott*.

Alan M Turing (2009). *Computing machinery and intelligence*. Springer.

Arnold Tustin (1947). "A method of analysing the behaviour of linear systems in terms of time series". In: *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*.

Hado Van Hasselt, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double Q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, and Petko Georgiev (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature*.

Milena Vuletić, Felix Prenzel, and Mihai Cucuringu (2023). "Fin-gan: Forecasting and classifying financial time series via generative adversarial networks". In: *Available at SSRN 4328302*.

Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Balch (Oct. 2020). "Get real: realism metrics for robust limit order book market simulations". In: *Proceedings of the First ACM International Conference on AI in Finance*. ICAIF '20. New York, NY, USA: Association for Computing Machinery.

Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Balch (2021). "Get real: realism metrics for robust limit order book market simulations". In: *Proceedings of the First ACM International Conference on AI in Finance*. ICAIF '20. New York, New York: Association for Computing Machinery.

Shida Wang and Beichen Xue (2023). "State-space models with layer-wise nonlinearity are universal approximators with exponential decaying memory". In: *Advances in Neural Information Processing Systems*.

Xiao Wang, Shiao Wang, Yuhe Ding, Yuehang Li, Wentao Wu, Yao Rong, Weizhe Kong, Ju Huang, Shihao Li, Haoxiang Yang, et al. (2024). "State space model for new-generation network alternative to transformers: A survey". In: *arXiv preprint arXiv:2404.09516*.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas (2016). "Dueling network architectures for deep reinforcement learning". In: *International Conference on Machine Learning*. PMLR.

Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker (2019). "Model-based reinforcement learning for predictions and control for limit order books". In: *arXiv preprint arXiv:1910.03743*.

Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar (2019). "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc.

Lantao Yu, Jiaming Song, and Stefano Ermon (2019). "Multi-agent adversarial inverse reinforcement learning". In: *International Conference on Machine Learning*. PMLR.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu (2017). "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Proceedings of the AAAI conference on artificial intelligence*.

Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta (2019). "Model-based deep reinforcement learning for dynamic portfolio optimization". In: *arXiv preprint arXiv:1901.08740*.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. (2025). "MuJoCo Playground". In: *arXiv preprint arXiv:2502.08844*.

Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind (2021). "An attention free transformer". In: *arXiv preprint arXiv:2105.14103*.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat (2024). "When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method". In: *The Twelfth International Conference on Learning Representations*.

Michael Zhang, Khaled Kamal Saab, Michael Poli, Tri Dao, Karan Goel, and Christopher Re (2022). "Effectively Modeling Time Series with Simple Discrete State Spaces". In: *The Eleventh International Conference on Learning Representations*.

Zhaoyu Zhang, Mengyan Li, and Jun Yu (2018). "On the convergence and mode collapse of GAN". In: *SIGGRAPH Asia 2018 Technical Briefs*.

Zihao Zhang, Bryan Lim, and Stefan Zohren (Jan. 2021a). "Deep Learning for Market by Order Data". In: *Applied Mathematical Finance*. Publisher: Routledge _eprint: https://doi.org/10.1080/1350486X.2021.1967767.

Zihao Zhang, Bryan Lim, and Stefan Zohren (2021b). "Deep learning for market by order data". In: *Applied Mathematical Finance*.

Zihao Zhang and Stefan Zohren (2021). "Multi-Horizon Forecasting for Limit Order Books: Novel Deep Learning Approaches and Hardware Acceleration using Intelligent Processing Units". In: *arXiv preprint arXiv:2105.10430*.

Zihao Zhang, Stefan Zohren, and Stephen Roberts (June 2019a). "DeepLOB: Deep Convolutional Neural Networks for Limit Order Books". In: *IEEE Transactions on Signal Processing*. Conference Name: IEEE Transactions on Signal Processing.

Zihao Zhang, Stefan Zohren, and Stephen Roberts (2019b). "Deeplob: Deep convolutional neural networks for limit order books". In: *IEEE Transactions on Signal Processing*.

Zihao Zhang, Stefan Zohren, and Stephen Roberts (2019c). "Extending deep learning models for limit order books to quantile regression". In: *arXiv preprint arXiv:1906.04404*.

Ziyang Zhang, Qizhen Zhang, and Jakob Foerster (2024). "PARDEN, Can You Repeat That? Defending against Jailbreaks via Repetition". In: *arXiv preprint arXiv:2405.07932*.

Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao (2018). "Stock market prediction on high-frequency data using generative adversarial nets." In: *Mathematical Problems in Engineering*.

Appendix A

Trading in Limit Order Book Markets using RL

We use the RLlib library [Liang et al. 2018] for a reference implementation of the APEX algorithm. Table A.1 shows a selection of relevant parameters we used for RL training.

Paramter	Value
timesteps_total	300e6
framework	torch
num_gpus	1
num_workers	42
batch_mode	truncate_episode
gamma	.99
lr_schedule	[[0,2e-5], [1e6, 5e-6]]
buffer_size	2e6
learning_starts	5000
train_batch_size	50
rollout_fragment_length	50
target_network_update_freq	5000
n_step	3
prioritized_replay	False

Table A.1: **Selected RL parameters** for APEX algorithm using RLlib [Liang et al. 2018] library for training.

Figure A.1 shows confusion matrices interpreting the oracle signal scores as prob-

abilities over the three classes: down, stationary, and up. The predicted class is thus the one with the highest score.

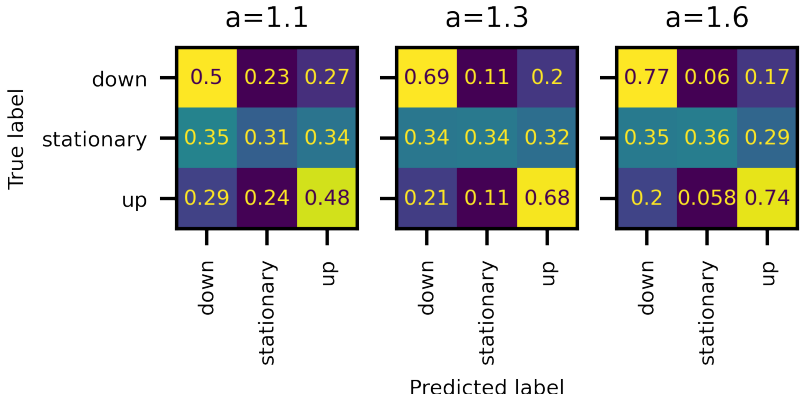


Figure A.1: **Confusion matrices** of the artificial oracle signal for three noise levels, from low to high noise.

Appendix B

LOB-Bench

B.1 Benchmark Code

The benchmark code can be found on GitHub at https://github.com/peernagy/lob_bench, additional resources can be found on the project website https://lobbench.github.io/.

The benchmark suite provides a convenient API functionality to evaluate model data for a range of scoring functions and metrics. A specification of such functions and loss metrics can be defined in a configuration dictionary, which can then be passed to a function performing the unconditional and conditional model evaluation. Similarly, the benchmark provides functions to compute the market impact curves, along with a mean L1 score. A default configuration dictionary, specifying the scoring functions reported here, evaluated using L1 and Wasserstein-1 loss, is similarly provided for easy reproducibility.

To run the benchmark, real and generated data sequences must be stored in LOBSTER format[1] as `csv` files. Files must be separated by real data, generated data, and (real) data used to condition the generation. A more detailed description can be found on GitHub.

[1] https://lobsterdata.com/info/DataStructure.php

B.2 LOBS5 Training Details



Figure B.1: **Test set loss curves (2023 data)** for the LOBS5 model, measuring the mean *per-message* negative log-likelihood for INTC (green) and GOOG (red) throughout 100 training epochs. Message cross-entropy after 100 epochs is 19.14 for INTC and 17.80 for GOOG.

Starting from the model introduced by Nagy et al. [2023b], we have scaled up the model size by adding additional S5 layers, with a resulting parameter count of approximately 35M (compared to originally 6.3M). The training consisted of 100 epochs of shuffled data sequences from the entire year of 2022, training with a total training budget of 30.4 L40 days (3.8 days across 8 GPUs). Adam [D. P. Kingma and Welling 2014] was used as an optimizer with a cosine learning rate schedule. Losses on the test set over the course of training are displayed in Figure B.1.

With this larger model, we also successfully removed the explicit error correction mechanism, which originally rejected semantically incorrectly generated messages, as error rates could be sufficiently reduced by scaling the model.

B.3 RWKV Training Details

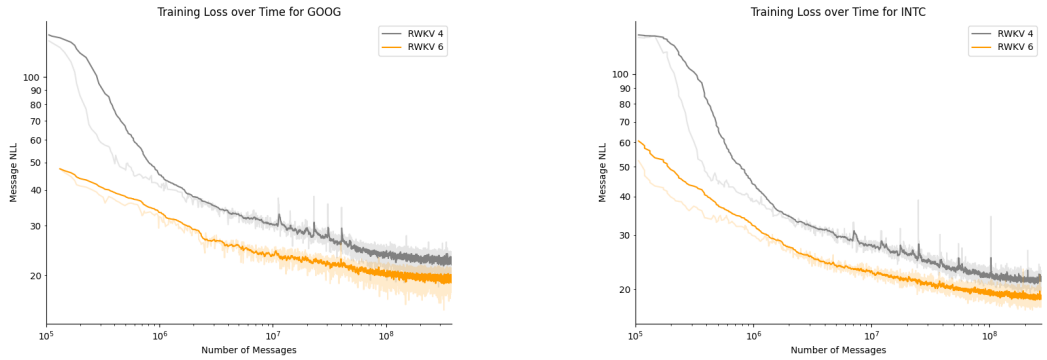


Figure B.2: **Training loss curves for RWKV model training.** The y-axis represents the average negative log likelihood of the messages being trained on, calculated as the sum of negative log likelihoods of each token in the message. The bold lines represent the exponential moving average of the true curve (presented with lower opacity) with an α of 0.1 to better highlight the trend.

We trained all RWKV models using an autoregressive training scheme with only message data (i.e., without any orderbook state information), and apply the same message filtering protocol as Nagy et al. [2023b]. We first tokenized each LOBSTER dataset using a byte pair encoder [Sennrich et al. 2016] trained on GOOG 2017 messages, resulting in datasets of 5.5 billion tokens for INTC 2022 (corresponding to 276 million messages) and 7.5 billion tokens for GOOG 2022 (corresponding to 380 million messages). We divide each dataset into chunks of 16384 tokens, and randomly shuffle these chunks for training.

For training, we initialize the parameters from the open source base pretrained RWKV models, each consisting of 170 million parameters, and train them on 8 chunks per optimization step, using the DAdapt-AdamW optimizer [Defazio and Mishchenko 2023] in Optax [Babuschkin et al. 2020], without scaling the learning rate or using any learning rate schedulers. For stability, we clipped the maximum global gradient norm to 1.0 [Pascanu et al. 2013]. In total, training all 4 of our RWKV models (2 model architectures, each over 2 datasets) took 10 L40S days. We present loss curves in Figure B.2.

B.4 Sensitivity of Divergence Metrics to the Bin Size

We adopt a dynamic bin size determined by the Freedman–Diaconis (FD) rule, which is specifically designed to adapt to the underlying data distribution. As a result, we do not anticipate significant sensitivity to the choice of bin size. This choice is further supported by a theoretical convergence property: the FD rule minimizes the integrated mean squared error (IMSE) between the histogram and the true data distribution [Freedman and Diaconis 1981].

To empirically evaluate the impact of different bin sizes on the reported divergence scores, we conduct the following experiments. The same divergence metric computations are repeated with half the regular bin size, as well as with double the regular bin size. To illustrate low bin size sensitivity, we perform this experiment for the LOBS5 model across all evaluated divergence metrics. We note that smaller bin sizes tend to increase both L1 and Wasserstein-1 errors, as data points, which would have been grouped into a single bin, could then be part of two separate bins. On the contrary, larger bin sizes tend to decrease errors, as data points close to each other are more likely to be grouped together.

To empirically assess the robustness of our results to bin size variation, we repeat the divergence metric computations using both half and double the default FD bin size. This analysis is conducted using the LOBS5 model across all evaluated divergence scores. We observe that reducing the bin size generally increases the L_1 errors. This is likely because data points that would have previously fallen within the same bin may now fall into separate bins, amplifying divergence estimates. Conversely, increasing the bin size tends to reduce these errors, as nearby data points are more likely to be aggregated into the same bin, leading to smoother approximations.

Figure B.3 shows the mean divergence scores for the LOBS5 model, along with 99% confidence intervals. Doubling or halving the bin size results in a consistent

decrease or increase in error scores, respectively. Notably, the magnitude of these deviations is comparable to the width of the confidence intervals under the default FD bin size. Given that these changes are relatively small and systematic, we conclude that while the choice of a theoretically grounded binning rule is important, model ranking remains stable as long as the same binning strategy is applied uniformly across all evaluated models.

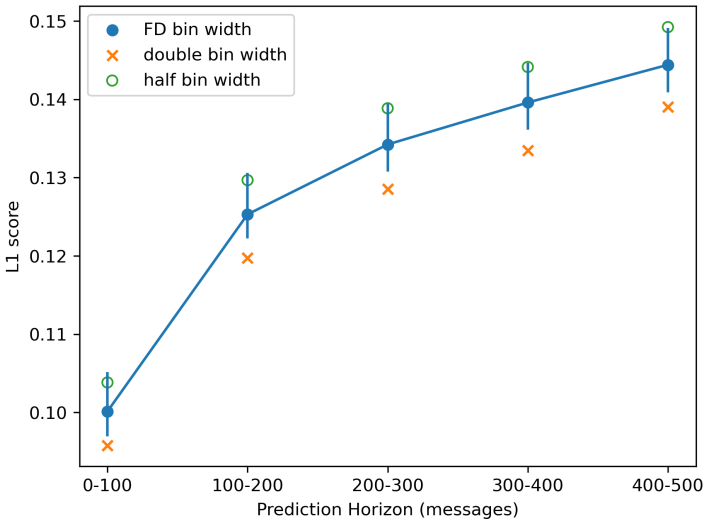
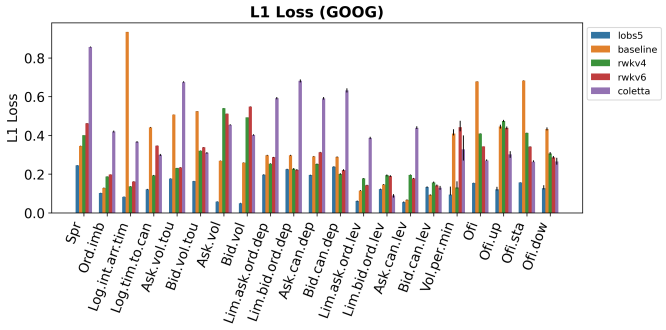
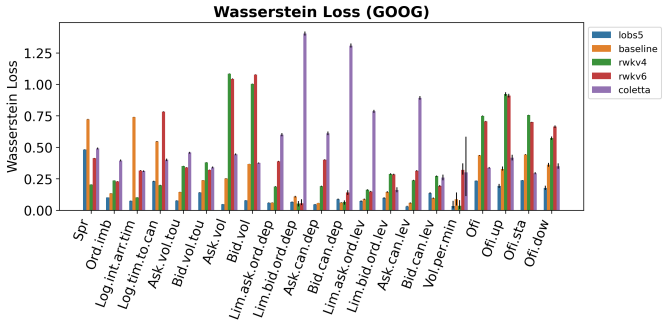


Figure B.3: **L1 Divergence Scores with half and double regular bin size.** Large bin size deviations (halfing or doubling) systematically affect the level of scores but not their ranking.

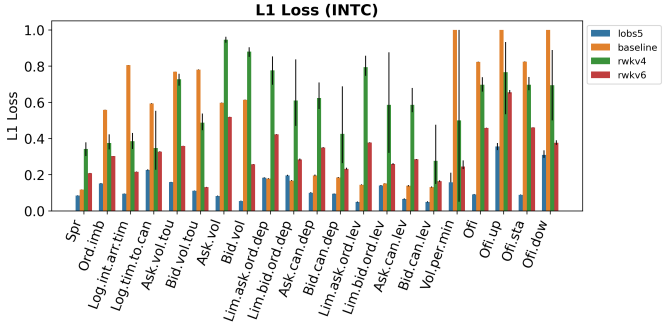
B.5 Additional Figures



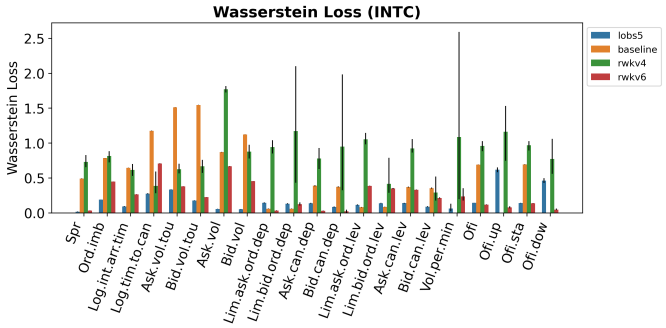
(a) GOOG - L1



(b) GOOG - Wasserstein

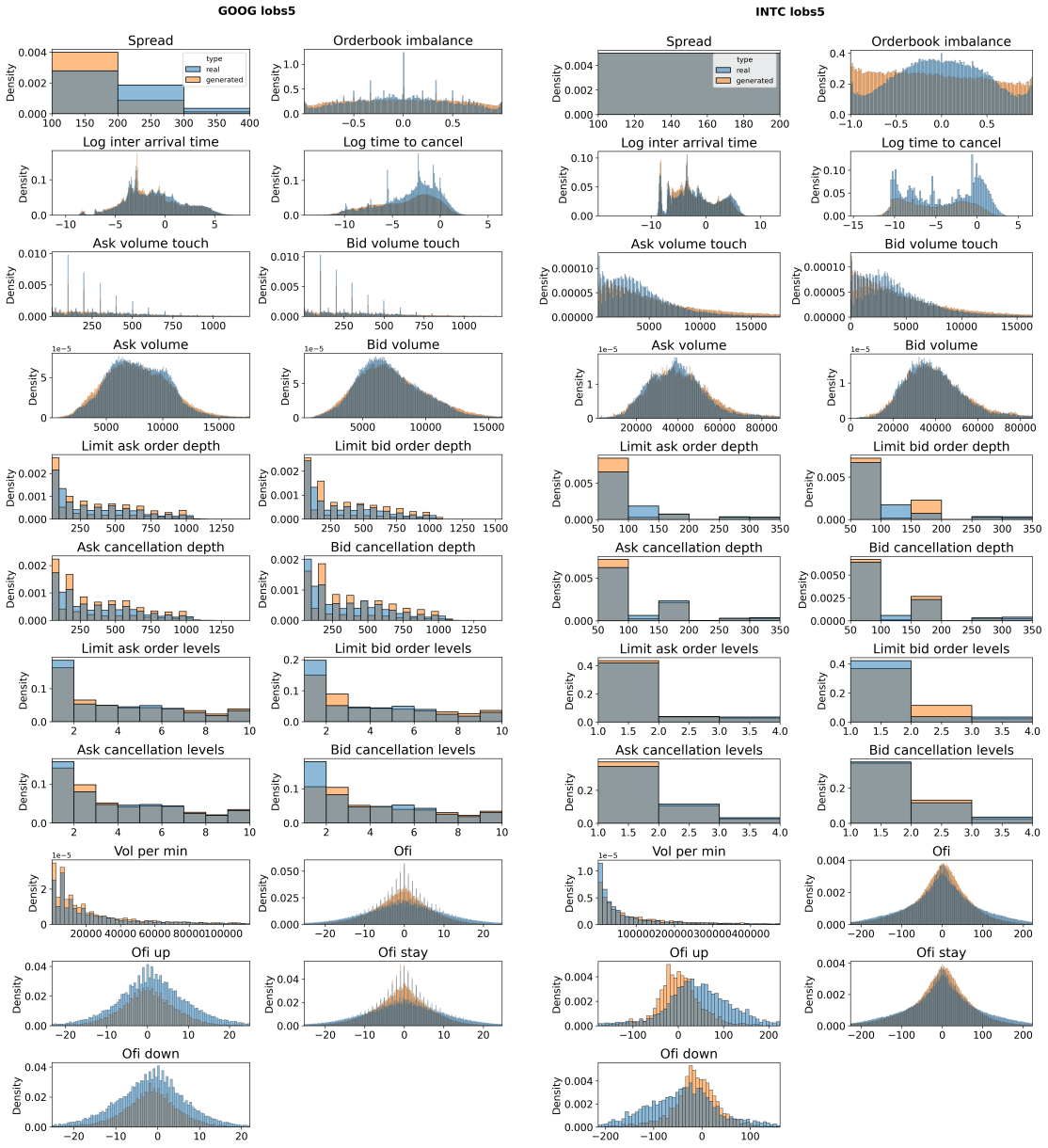


(c) INTC - L1



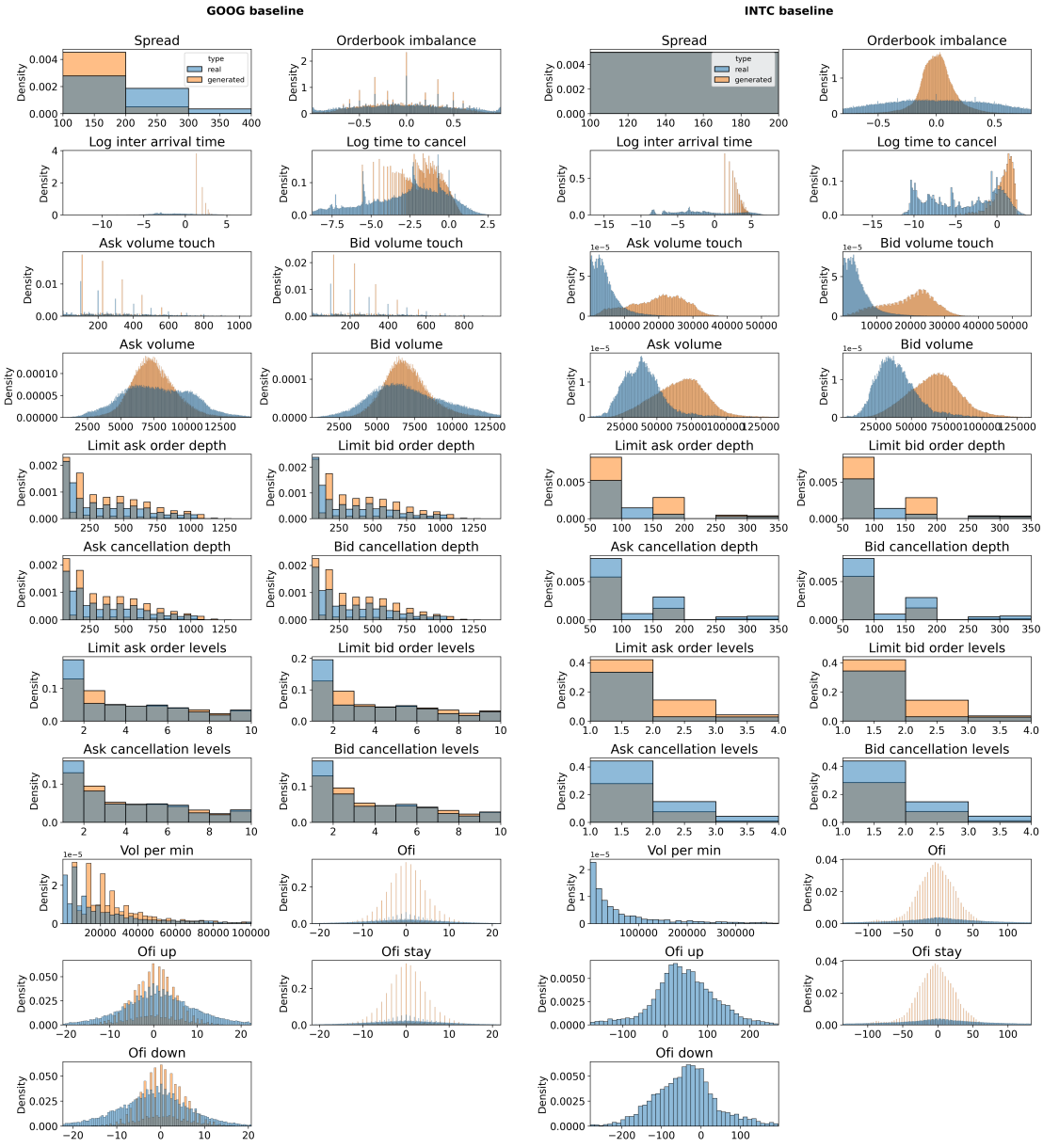
(d) INTC - Wasserstein

Figure B.4: **L1 and Wasserstein-1 errors** of generated unconditional distributions for easy comparison between Alphabet (GOOG) and Intel (INTC). Error bars show 99% bootstrapped confidence intervals.



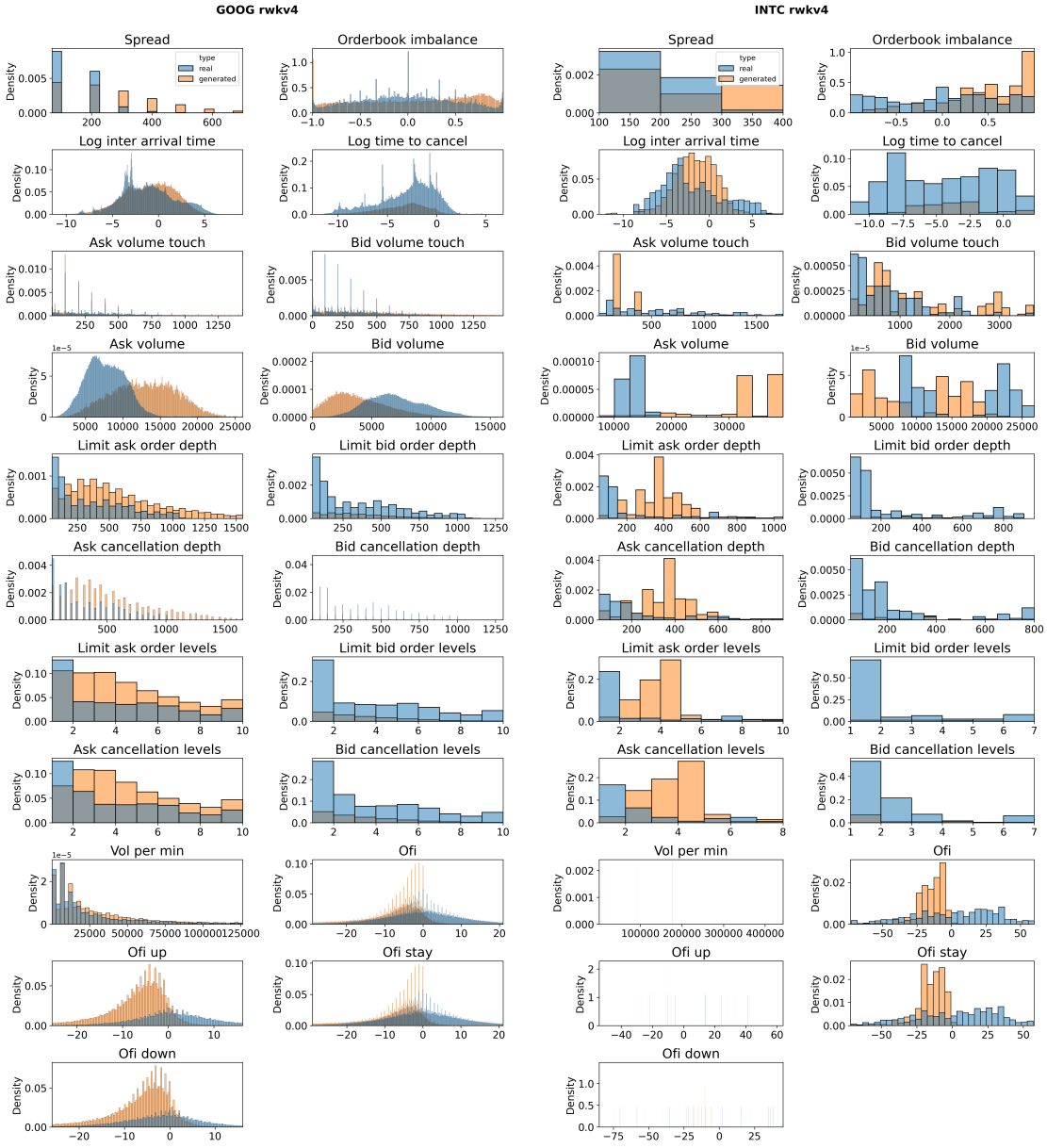
(a) GOOG (b) INTC

Figure B.5: ***LOBS5* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. Overall, the generative *LOBS5* model evaluated here, adapted from Nagy et al. [2023b], does a good job in matching data along various dimensions. Bigger errors in matching distributions are visible in e.g. spread (GOOG), orderbook imbalance (INTC) and time to cancel (GOOG and INTC).



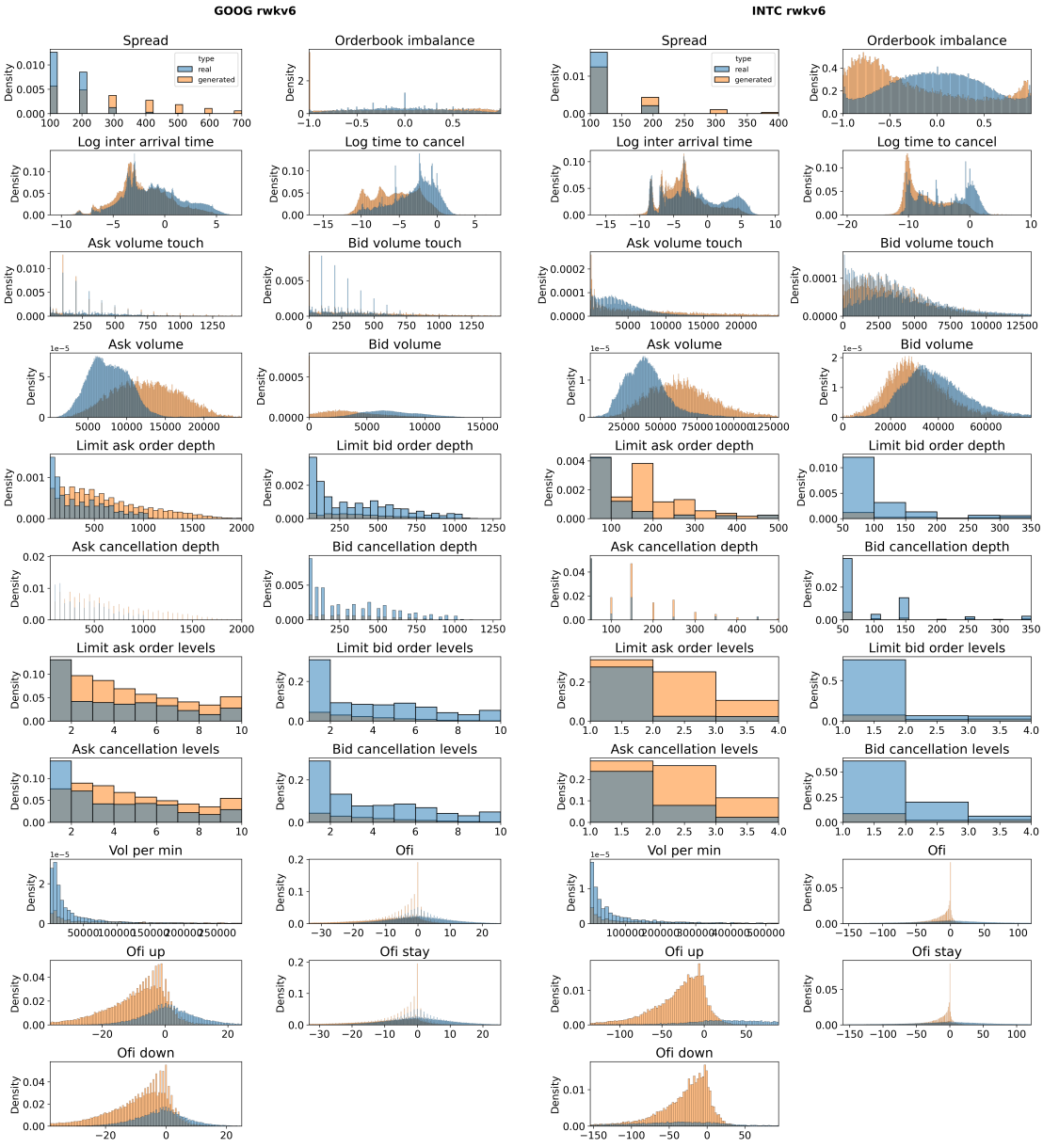
(a) GOOG (b) INTC

Figure B.6: ***baseline* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The [Rama Cont et al. 2010b] model does a decent job matching some of the scores, particularly discrete ones, such as depths and levels. Clear shortcomings are visible in scores such as orderbook imbalance or volumes.



(a) GOOG (b) INTC

Figure B.7: ***rwkv4* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The model produces volatile data with larger spreads, missing correct order levels, leading to difficulty matching book volumes.



(a) GOOG (b) INTC

Figure B.8: ***rwkv6* - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks. The model has similar shortcomings to RWKV 4 (wrong price levels, mismatched book volumes etc.) due to tokenization of raw data and missing order book information.

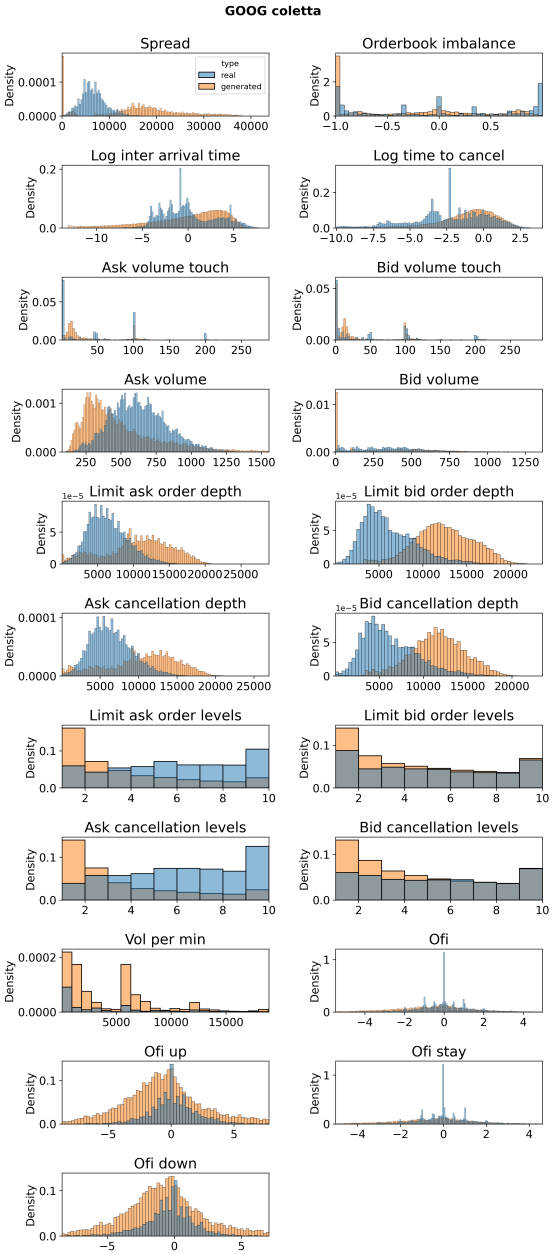
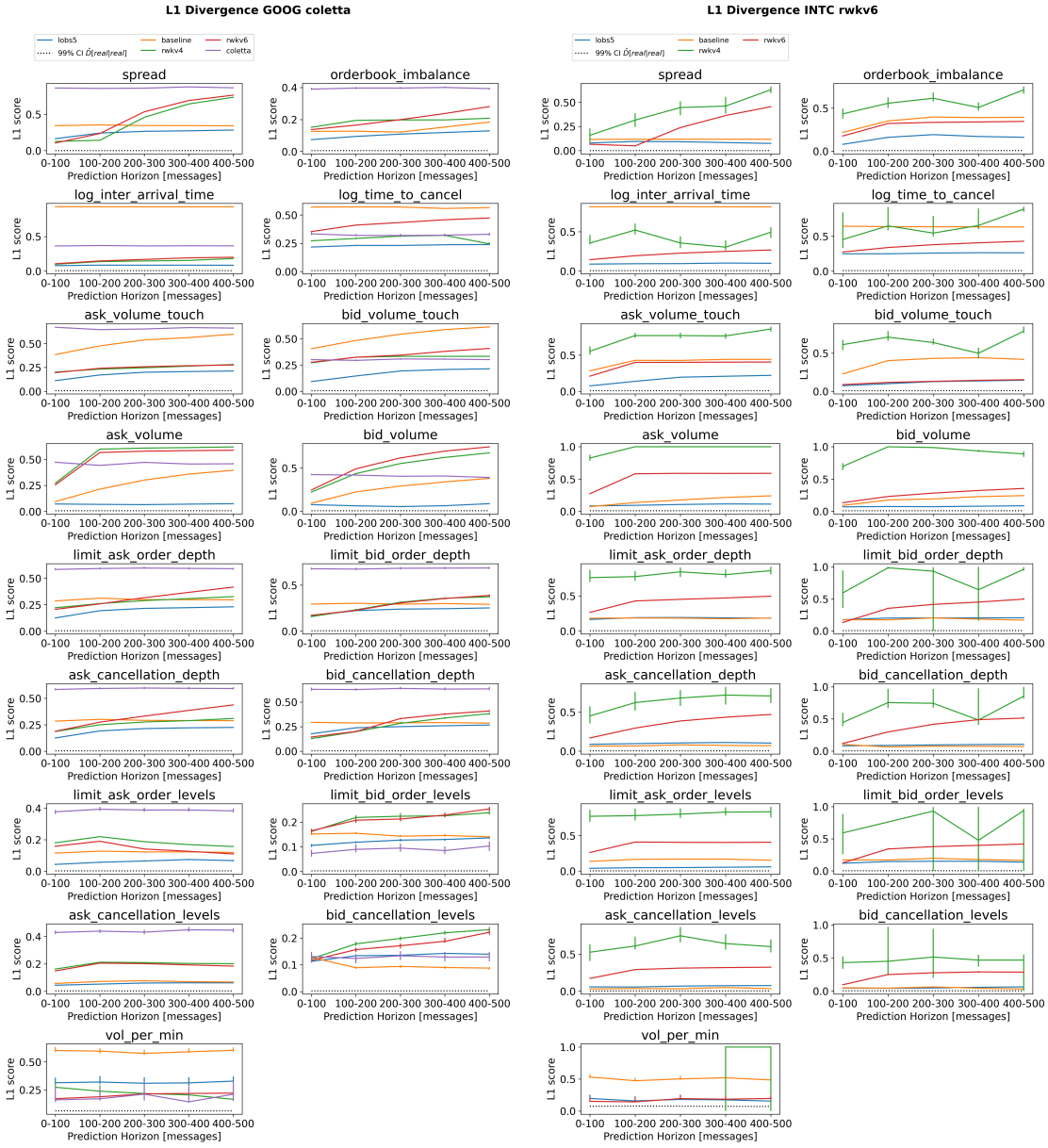
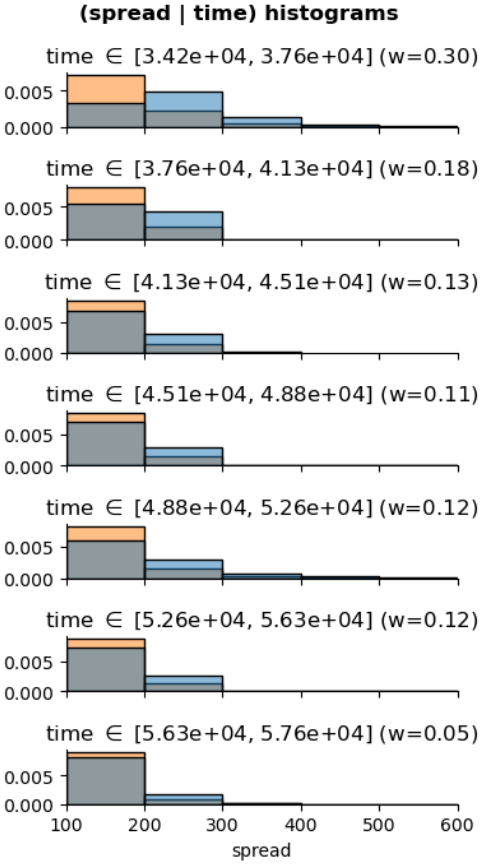


Figure B.9: ***coletta* - GOOG - histograms comparing score distributions** for real (blue) and generated (orange) LOB data for Alphabet (GOOG) and Intel (INTC) stocks.

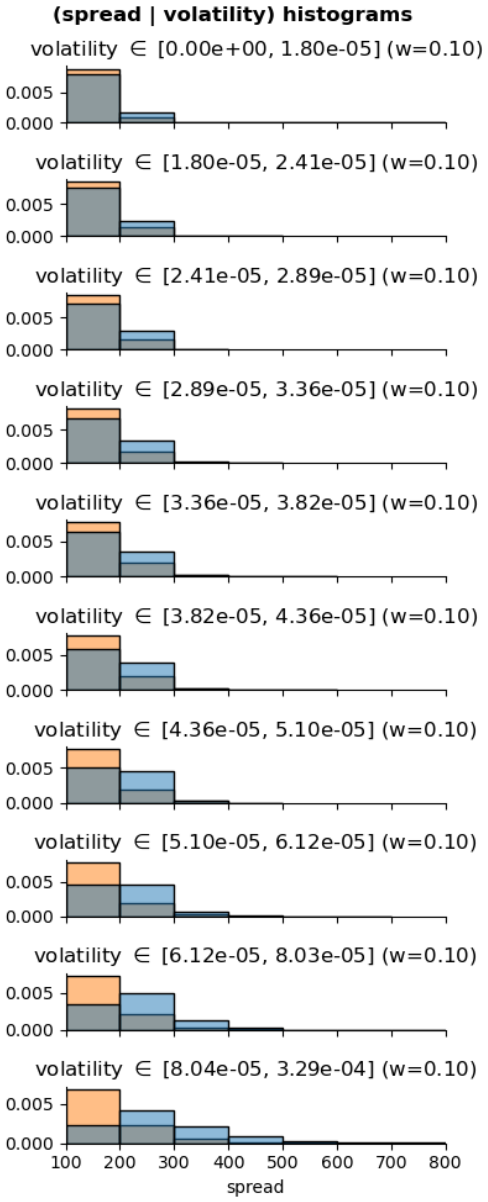


(a) GOOG (b) INTC

Figure B.10: **L1 error divergence:** comparing the L1 errors of score distributions of real data with generated data distributions at a specific horizon into the future shows accumulating model errors. This is explainable due to snowballing errors caused by teacher forcing (conditional next token loss). A good model should be able to control errors for sequence lengths as long as possible. To provide a significance threshold over pure sampling noise, the dotted lines plot the 99. percentile of L1 error between bootstrapped samples of only real data.



(a) Bid-ask spread conditional on the hour of the day: spreads are higher early in the day, where the generated data also exhibits too narrow spreads.



(b) Spread conditional on volatility: higher volatility corresponds to higher frequency of higher spreads. The model does not fully capture this change, as the higher discrepancy in high-volatility bins shows.

Figure B.11: **Histograms of conditional score distributions** for real (blue) and generated (orange) data for the Alphabet stock (GOOG). Weights w, expressing the share of data in the bin, measure the impact of the specific conditional distribution (row) on the total metric loss.

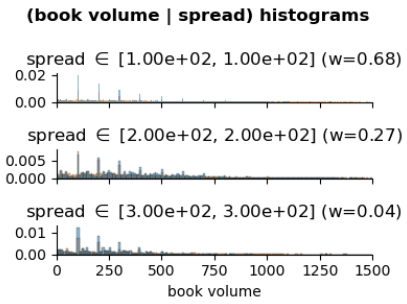


Figure B.12: **Histograms of total book volume conditional on bid-ask spread for Alphabet stock (GOOG)**. Weights w, expressing the share of data in the bin, measure the impact of the specific conditional distribution (row) on the total metric loss.

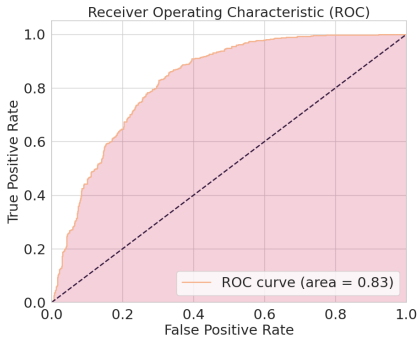


Figure B.13: **LOBS5 - ROC curve of the discriminator on test data (GOOG).** The discriminator represents a worst-case adversarial score function by learning to effectively differentiate between real and generated sequences of LOB states.

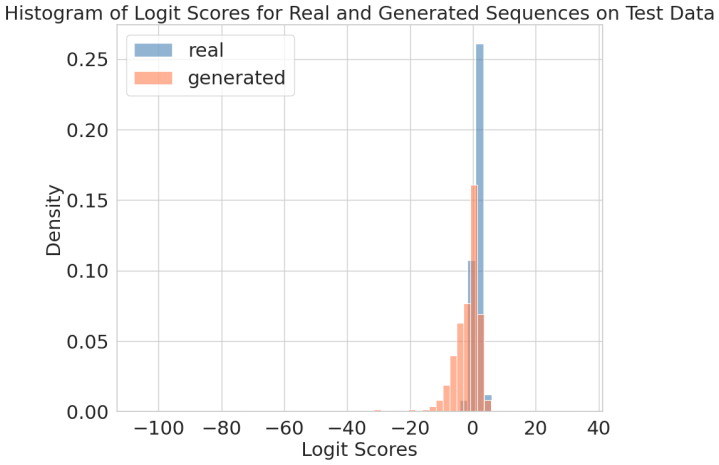


Figure B.14: **LOBS5 - Histogram of logit scores for real and generated sequences** on held-out test data (GOOG). Matching this distribution well would indicate high model quality, as even a trained discriminator network would not be able to differentiate the distributions.

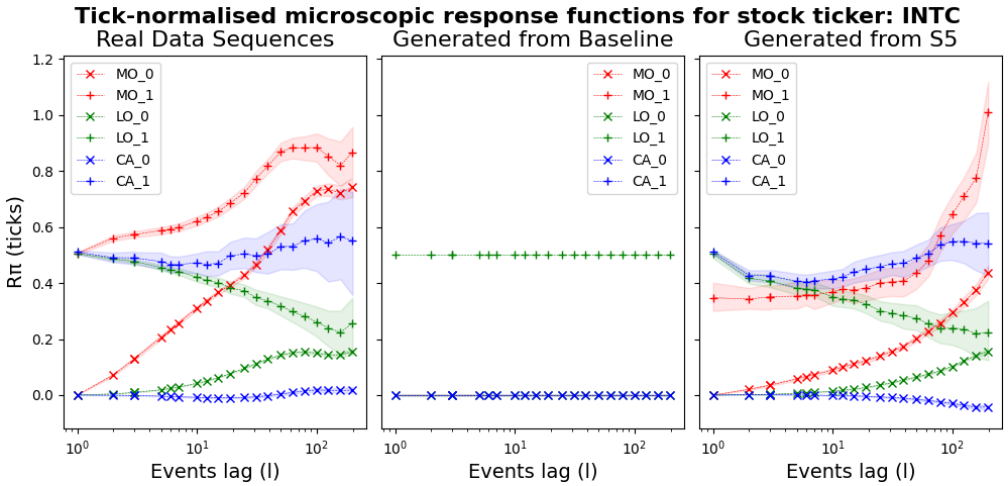


Figure B.15: **Comparison of impact response functions** for different event types between real and generated data-sets, tick-normalized mid-price response. Shaded regions are 99% confidence intervals. There is a comparison between two select models: the LOBS5 and the stochastic baseline. We see that, in contrast to the baseline, the generative model is able to reproduce much more of the expected impact function, though not as well as for GOOG.

Appendix C

Statement of Authorship

A statement of authorship is provided for each multi-authored paper included in this thesis. The statements describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication, there exists a complete statement that is filled out and signed by the candidate and supervisor.

Statement of Authorship for the paper "Machine Learning for Microstructure Data-Driven Execution Algorithms" in Chapter 1.

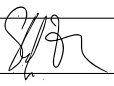
Paper title	Machine Learning for Microstructure Data-Driven Execution Algorithms
Authors	P Nagy, J Powrie, S Zohren
Publication status	Published
Publication details	Handbook of Artificial Intelligence and Big Data Applications in Investments, 100-106, 2023.

Student Confirmation

Student name	Peer Nagy	
Contribution to the paper	PN wrote the manuscript with input from all authors. All authors devised the conceptual ideas, contributed to the article and approved the submitted version.	
Signature and Date		05.07.2026

Supervisor Confirmation

By signing the Statement of Authorship, the supervisor is certifying that the candidate made a substantial contribution to the publication, and that the description above is accurate.

Supervisor name	Dr Stefan Zohren	
Supervisor comments		
Signature and Date		10.07.2026

Statement of Authorship for the paper "Asynchronous Deep Double Dueling Q-learning for Trading-Signal Execution in Limit Order Book Markets" in Chapter 3.

Paper title	Asynchronous Deep Double Dueling Q-learning for Trading-Signal Execution in Limit Order Book Markets
Authors	P Nagy, JP Calliess, S Zohren
Publication status	Published
Publication details	Frontiers in Artificial Intelligence 6, 2023.

Student Confirmation

Student name	Peer Nagy	
Contribution to the paper	PN performed the experiments and wrote the manuscript with input from all authors. All authors devised the project, the conceptual ideas, and the design of the experiments. All authors contributed to the article and approved the submitted version.	
Signature and Date		05.07.2026

Supervisor Confirmation

By signing the Statement of Authorship, the supervisor is certifying that the candidate made a substantial contribution to the publication, and that the description above is accurate.

Supervisor name	Dr Stefan Zohren	
Supervisor comments		
Signature and Date		10.07.2026

Statement of Authorship for the paper "JAX-LOB" in Chapter 4.

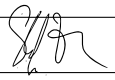
Paper title	JAX-LOB: A GPU-Accelerated Limit Order Book Simulator to Unlock Large Scale Reinforcement Learning for Trading
Authors	S Frey, K Li, P Nagy, S Sapora, C Lu, S Zohren, J Foerster, A Calinescu
Publication status	Published
Publication details	Proceedings of the Fourth ACM International Conference on AI in Finance, 583-591, 2023.

Student Confirmation

Student name	Peer Nagy	
Contribution to the paper	SF, KL, and PN contributed equally to this work. All three contributed to the conception, implementation, and writing of the paper with input from all authors. SS and CL contributed to the implementation, provided feedback, and assisted with writing and editing. SZ, JF, and AC offered guidance and supported project design and manuscript revision.	
Signature and Date		05.07.2026

Supervisor Confirmation

By signing the Statement of Authorship, the supervisor is certifying that the candidate made a substantial contribution to the publication, and that the description above is accurate.

Supervisor name	Dr Stefan Zohren	
Supervisor comments		
Signature and Date		10.07.2026

Statement of Authorship for the paper "Generative AI for End-to-End Limit Order Book Modelling" in Chapter 5.

Paper title	Generative AI for End-to-End Limit Order Book Modelling: A Token-Level Autoregressive Generative Model of Message Flow Using a Deep State Space Network
Authors	P Nagy, S Frey, S Sapora, K Li, A Calinescu, S Zohren, J Foerster
Publication status	Published
Publication details	Proceedings of the Fourth ACM International Conference on AI in Finance, 91-99, 2023.

Student Confirmation

Student name	Peer Nagy	
Contribution to the paper	PN conceptualised the project and led its implementation and manuscript preparation, with input from all authors. SF, SS, and KL contributed to implementation, provided feedback, and assisted with writing and editing. AC, SZ, and JF offered guidance and supported project design and manuscript revision.	
Signature and Date		05.07.2026

Supervisor Confirmation

By signing the Statement of Authorship, the supervisor is certifying that the candidate made a substantial contribution to the publication, and that the description above is accurate.

Supervisor name	Dr Stefan Zohren	
Supervisor comments		
Signature and Date		10.07.2026

Statement of Authorship for the paper "LOB-Bench" in Chapter 6

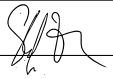
Paper title	LOB-Bench: Benchmarking Generative AI for Finance – an Application to Limit Order Book Data
Authors	P Nagy, S Frey, K Li, B Sarkar, S Vyetrenko, S Zohren, A Calinescu, J Foerster
Publication status	Published
Publication details	Proceedings of the 42nd International Conference on Machine Learning (ICML), 2025.

Student Confirmation

Student name	Peer Nagy	
Contribution to the paper	PN and SF contributed equally to this work. Both worked on the conceptualisation and implementation with input from all authors. KL and BS contributed to parts of the project implementation and manuscript preparation. SV, SZ, AC, and JF offered guidance and supported project design and manuscript revision.	
Signature and Date		05.07.2026

Supervisor Confirmation

By signing the Statement of Authorship, the supervisor is certifying that the candidate made a substantial contribution to the publication, and that the description above is accurate.

Supervisor name	Dr Stefan Zohren	
Supervisor comments		
Signature and Date		10.07.2026