University College London

Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Finance, University College London

LOB-AI
Limit Order Book message simulation using Deep Learning

Author
Leonardo Guerra

Academic Supervisor
Dr Paris Pennesi
Department of Computer Science
University College London

Industrial Supervisor
Z. Tong & P. Kshetrimayume
eFX Trading
HSBC

September 9, 2024

Abstract

This thesis explores the application of AI and large language model (LLM)-inspired techniques to the modeling of limit order books (LOBs) in financial markets. The accurate simulation of LOBs is essential for understanding market microstructure, price formation, and liquidity, making it a crucial tool for both academic research and practical trading applications. We aimed to investigate whether the integration LLM-derived AI methodologies could enhance the realism and predictive accuracy of LOB simulations.

Our research was driven by three key questions: how to benchmark the current reference model on our dataset, which model architecture yields the best performance, and how stress testing impacts message flow simulation. Through a series of experiments, we assessed the performance of various model architectures, focusing on their ability to predict event types, timing, and mid-price movements.

The results showed that while our models could accurately predict event types and timing, and introduce some variance in returns, they struggled with mode collapse—a critical issue where the model fails to generate a diverse range of tokens, limiting its ability to replicate complex market behaviors. We developed a method for stress-testing models, which further highlighted this challenge. In response, we proposed potential solutions, including latent space encoding and customized loss functions, to address mode collapse and improve model robustness.

Our work underscores the potential of AI and LLM approaches in financial modeling, while also identifying key areas for future research. By refining these models, we can move closer to creating sophisticated simulations that accurately capture the dynamics of financial markets, offering valuable insights for both researchers and practitioners.

Acknowledgments

I would like to express my deepest gratitude to my thesis supervisor, Prof. Paris Pennesi, whose guidance and expertise have been invaluable throughout this journey.

I am also profoundly grateful to the project supervisors, Zhaozhen Tong and Punicha Kshetrimayum, for their continuous support and invaluable feedback during our weekly meetings. Their dedication of time and effort was instrumental in the success of this project.

A special thanks to my thesis partner, Jason, without whom this project would have been too daunting to undertake. His collaboration and support were essential, and I am truly grateful for his valuable contributions.

I want to thank my girlfriend Bella for being by my side throughout this process. Despite the many hours I had to dedicate to this project, she never once complained and remained my steadfast source of strength and love.

To my parents, I am deeply thankful for their encouragement to pursue this Master's degree and for their unwavering support throughout the year. I extend my heartfelt thanks to my entire family—my aunt, uncle, grandparents, and little cousin—for their unconditional love and support back home.

I also want to acknowledge my friends back in Italy, who always make me feel at home whenever I return, as if I had never left. Finally, I am immensely grateful for the incredible new friends I have made during this year in the UK. They have made this year remarkable, and I look forward to deepening our bonds in the years to come.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective

Stock exchanges operate through a sophisticated system known as the Limit Order Book (LOB), which records and processes buy and sell orders in real-time. These orders, represented as messages, update the LOB by managing all pending transactions and facilitating the matching of compatible trades. The accurate simulation of the LOB is paramount for understanding market dynamics, optimizing trading strategies, and ensuring market efficiency. Effective LOB simulation enables traders to maximize their execution strategies while minimizing market impact, a critical aspect of high-frequency trading and algorithmic strategies.

Despite extensive studies on the statistical properties and dynamics of the LOB, the complexity introduced by varying trading objectives and styles of numerous market participants presents ongoing challenges. In fact, each participant, whether a retail trader or a high-frequency trading firm, interacts with the LOB in ways that can significantly affect price formation and liquidity. As markets evolve, so too must our models of these interactions.

Recent advances in Generative AI, including models such as Generative Adversarial Networks (GAN) [3], have paved the way for innovative approaches using Deep Learning. This thesis focuses on a specific subclass of these models, State Space Deep Learning models, to generate predictive and reactive limit order book messages.

1.2 Motivation Behind the Thesis

The primary motivation for this thesis is to advance the understanding and predictive accuracy of LOB simulations, a crucial tool in the ever-evolving landscape of financial markets. By implementing and refining a State Space Deep Learning model based on the research done by Peer Nagy et al. [4], we aim to develop a robust simulator capable of

not only forecasting market movements but also analyzing the market's reaction to trade executions in real-time. This simulator will generate realistic LOB messages that, when integrated with the JAX-LOB simulator, will enable dynamic updates and predictions of LOB states. Additionally, this work bridges the gap between LOB simulation and Large Language Models (LLMs), offering a novel approach to leverage vast amounts of data and capture complex market behaviors.

The relevance of this research is reflected in several dimensions::

- **Benchmarking and Enhancing Model Performance:** This research seeks to evaluate our model's performance against established datasets, identify areas for improvement, and refine its architecture to better capture the nuances of market dynamics.
- **Deepening Understanding of Market Dynamics:** Through accurate simulation, we aim to gain insights into how various trading activities influence the market. This understanding is crucial for developing strategies that optimize order execution while minimizing adverse market impacts.
- **Testing under Adverse Market Conditions:** By assessing the model's performance under simulated adverse market conditions, we will evaluate its robustness and the stability of the LOB during periods of high volatility or market stress.

This research is not only about enhancing the technical performance of LOB simulations but also about contributing to a deeper understanding of the financial markets. In a world where markets are increasingly influenced by automated trading systems, improving these simulations is essential for ensuring that trading strategies are both effective and resilient under a wide range of conditions. This work lays the groundwork for future explorations into more advanced and adaptable market simulation tools.

1.3 Content of the Thesis

This thesis is structured to systematically explore and answer the research questions posed. Chapter 2 provides a comprehensive literature review, setting the stage by examining existing theories and practices in LOB simulations. Chapter 3 introduces the research questions and outlines the methodology, including the design and implementation of the State Space Deep Learning model. In Chapter 4, we describe the dataset and detail the experimental setup used to evaluate the model. After this we move on to describing the architecture of the model and its functioning. Chapter 6 discusses the experiments we conducted and their results, providing insights into model performance and answering the research questions. Finally, Chapter 7 concludes the thesis by summarizing the findings, discussing their implications, and suggesting avenues for future research.

Chapter 2

Related Works

In this chapter, we will provide the reader with the necessary background to understand the concepts presented this thesis and the current state-of-the-art systems related to this work. While a thorough discussion about the concepts of Machine Learning and Deep Learning is out of the scope of this thesis, a brief introduction to the main ideas may benefit the reader.

2.1 Machine Learning and Deep Learning

Machine Learning can be defined as a the part of artificial intelligence and computer science that is centered around the use of data and algorithms to replicate the human learning process in a continuous gradual improvement of accuracy [5]. By using statistical methods, algorithms are fitted to accomplish classification or prediction tasks, uncovering key information on the task at hand. The insights gained through this information subsequently drives decision making within applications and businesses, ideally impacting key growth metrics. Deep Learning is a sub-field of Machine Learning that eliminates some of data pre-processing that is typically involved with machine learning and reduces the manual human intervention required in typical machine learning to extract features from the data at hand [6]. Deep learning models are composed of many simple linear models or nodes arranged in layers with intervening non-linearities to allow for complex information representation. This architecture gives DL models a hierarchical layer structure (from here the term "deep") that creates the ability to learn complex concepts.

2.1.1 State Space Models

State Space Models (SSMs) are a class of deep learning models used to describe systems governed by a hidden state and observed through noisy measurements. This thesis employs SSMs to model the generation of limit order book messages within a dynamic system. The system's state evolves over time and is inferred in part from the transactions that occur.

These models are particularly suitable for situations where one needs to infer an internal state not directly observable from the input data.

In our work, we utilize a discretized version of SSM known as the S5 model, as defined by Smith et al. [7]. The S5 model, unlike the continuous form, operates in discrete time and is mathematically formulated as:

$$\begin{aligned} x_{k+1} &= \bar{A}x_k + \bar{B}u_k \qquad \text{(State Evolution)} \\ y_k &= \bar{C}x_k + \bar{D}u_k \qquad \text{(Output Equation)} \end{aligned} \tag{2.1}$$

where:

- x_k: Hidden state at discrete time step k.
- u_k: Input to the system at time step k.
- y_k: Output of the system at time step k.
- $\bar{A}$: Discrete state transition matrix.
- $\bar{B}$: Discrete input matrix.
- $\bar{C}$: Discrete output matrix.
- $\bar{D}$: Discrete feedthrough (or direct transmission) matrix.

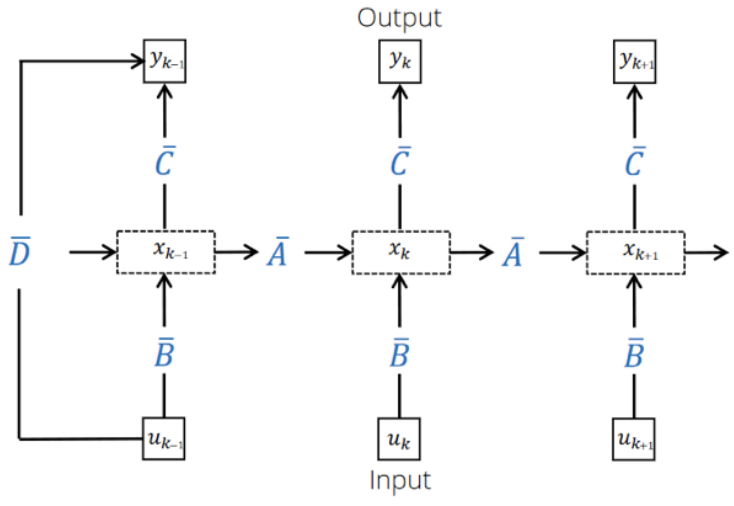

Figure 2.1: Discretized SSM model diagram [1]

The S5 model leverages the HiPPO framework—a method for maintaining and updating state memory over time [8]—alongside deep learning techniques to efficiently process long sequences of data. This discrete formulation allows the model to operate efficiently in both training and inference phases, particularly when dealing with large-scale sequential

data such as limit order book messages. Our architecture employs stacked S5 layers to capture the intricate dynamics of financial markets, as will be detailed in the following chapters.

2.2 Order Book Modelling

As introduced earlier, order book modeling has emerged as a significant area of research. A well-functioning order book simulator can predict market movements and the impact of trades, which are critical for executing optimal trading strategies. Given the importance of these simulations, there is substantial interest in employing cutting-edge technologies to model order book dynamics. Traditionally, the order book has been modeled in four main ways. These include direct simulation of the order book itself [9], agent-based simulations that capture the behavior of market participants [10], zero-intelligence models that use stochastic processes to represent the LOB [11], and simulations that focus on the flow of trade messages, with a simulator managing these interactions. In their paper Konark Jain et al.[12] review the state of order book modelling and it emerges that historically, the first three ways have dominated the literature. Studies [13] have already demonstrated strong performance in predicting mid-price movement. However, with advancements in generative AI and deep learning, a new and more flexible approach is gaining popularity: directly generating order messages and integrating them into the order book. This method consists of two components: a generative AI model that produces the messages and a simulator that applies these messages to update the order book's state.

2.3 Order Book Simulator: JAX-LOB

The order book model is commonly represented as structured around discrete price levels, each distinguished by a tick, the smallest allowable price increment often regulated by market authorities. Each price level can host a non-negative integer number of orders, which may vary in size and be classified as either buy or sell orders. The purpose of each order is to trade at a specific price level, indicating both the quantity (order size) and the trade direction (buy or sell). Orders intended for buying are positioned at lower price levels, forming the bid' side, while sell orders are placed at higher levels, forming the 'ask' side of the book.

For example, consider the aggregated order sizes at each price level for a stock on a random trading day. The bid price, $p_{b,1}$, represents the highest price a buyer is willing to pay, and the ask price, $p_{a,1}$, represents the lowest price at which a seller is willing to sell. The formulas for these are:

$$\text{Bid price } (p_{b,1}) : \text{Highest buy order price} \tag{2.2}$$

$$\text{Ask price } (p_{a,1}) : \text{Lowest sell order price} \tag{2.3}$$

The difference between $p_{a,1}$ and $p_{b,1}$ is known as the spread, S, and the mid-price, p_m, is calculated as the average of these two prices:

$$S = p_{a,1} - p_{b,1} \tag{2.4}$$

$$p_m = \frac{p_{a,1} + p_{b,1}}{2} \tag{2.5}$$

We consider LOB orders that fall into three categories:

- **Limit Orders**: An order to buy or sell a set amount of an asset at a specified price, different from the current best matching price on the opposite side of the LOB. These orders do not guarantee execution but typically incur lower transaction costs since they provide liquidity.
- **Market Orders**: These orders aim to buy or sell at the best available current price and, if the order's volume exceeds the available volume at the best quote, the remainder is executed at deeper price levels. Market orders generally incur higher transaction costs as they remove liquidity from the market.
- **Cancellation / Deletion Orders**: Orders that either fully or partially remove an existing limit order without execution, typically not incurring transaction costs.

To implement such LOB model the JAX-LOB simulator [14] was used. JAX-LOB is a GPU-accelerated tool designed to model the dynamics of financial markets and order books, focusing on the reinforcement learning (RL) domain. The simulator handles three primary operations: adding new orders, canceling existing ones, and matching incoming orders with those already in the LOB. When a new order arrives, it is either matched with an existing order using a price-time priority algorithm or added to the LOB if no immediate match is found. The implementation is built on JAX [15], a machine learning library that enables just-in-time (JIT) compilation and automatic gradient calculation (auto-grads), allowing the simulator to leverage the parallel processing power of GPUs. The LOB is represented using fixed-size arrays, with separate arrays for the bid and ask sides, where each order is stored with 6 attributes price, quantity, order ID, Trader ID and timestamps (seconds and nanoseconds). The use of fixed-size arrays ensures that the simulator can maintain the structural requirements of JAX while efficiently handling the addition, cancellation, and matching of orders. The simulator uses this array-based architecture to avoid the overhead associated with reordering data upon order removals. The design ensures that even complex operations like order matching, which require multiple iterations to match large orders, are optimized for parallel execution, allowing for a significant speedup in processing time, particularly when handling large-scale simulations involving thousands of parallel LOBs.

2.4 Order Book Simulation using Generative AI

The recent advancements in generative AI have led to significant research efforts across various fields, including finance. These techniques have been increasingly applied to order book modeling, as highlighted in a recent study [12], which underscores the growing shift towards the use of Generative Adversarial Networks (GANs) driven by advances in deep learning. One notable example is the study by Rama Cont et al. [3] where they developed a GAN-based simulator designed to model the net transitions of the order book state. Their approach involves training a probabilistic model to learn the distribution of future LOB states conditioned on the current state, effectively capturing realistic market impact patterns. They demonstrate that their model not only provides accurate mid-price predictions but also adheres to market impact properties, such as the square-root law in relation to trade size, which are commonly observed in financial markets.

2.5 Order Book Message generation Using State Space model

2.5.1 State space model approach

A novel approach to order book modeling was introduced by Peer Nagy et al. in their pre-print paper *"Generative AI for End-to-End Limit Order Book Modelling"* [4]. Unlike traditional methods that directly model order book states, their approach models the order message flow and feeds it into the JAX-LOB order book simulator. This paper serves as the foundational reference for this thesis. We implemented the architecture based on the authors' publicly available code [16] and made further refinements and improvements to it. The key innovation in their work is the use of an LLM-inspired tokenization approach to encode order messages, bridging the gap between NLP techniques and LOB simulation tasks and potentially harnessing the vast amounts of data available to some institutions. To implement this, they use an autoregressive generative model architecture based on the S5 State Space model [7] to generate the message sequence that is fed to the order book simulator, producing new states and simulating its behaviour. The S5 state space models are particularly well-suited for handling long sequences [17], making them ideal for capturing the complex and lengthy sequences typical in financial data, such as those found in order book message flows. As this architecture forms the basis of our own implementation, a detailed examination of the model will be provided in Chapter 5 of this work.

2.5.2 Reference paper methodology and data

The model presented in the paper was trained and evaluated using the LOBSTER dataset [18], focusing specifically on data from Alphabet (GOOG) and Intel (INTC). The dataset contains the flow of order book messages for the respective stocks. It spans 102 days of training data (from 1 July 2022 to 11 November 2022), with 12 days allocated for validation (28 November to 13 December 2022) and another 12 days for testing (14 December to 30 December 2022). These two stocks were selected as they represent examples of small-tick (GOOG) and large-tick (INTC) LOBs, which exhibit different market dynamics. The dataset includes a variety of message types, but for this study, the focus was narrowed to four primary types: new limit orders, partial order cancellations, full order deletions, and visible order executions. Messages related to hidden order executions, auction trades, and trading halts were excluded because they either do not affect the visible dynamics of the LOB or occur outside regular trading hours.

The data is then preprocessed and tokenized. This involves converting prices from dollar values to ticks relative to the previous mid-price, truncating extreme prices and order sizes to ensure the data stays within a manageable range, and normalizing the data to create a more stationary representation. Referential messages, such as cancellations and executions, are augmented with additional details about the original orders to maintain the relationships between messages. During the tokenization phase, each message is broken down into a sequence of tokens. Fields like event type, direction, price, and size are encoded into distinct tokens, with prices split into tokens representing the sign and tick distance from the mid-price. Time-related data is tokenized in groups of three digits to capture precise timing down to the nanosecond level.

2.5.3 Inference and Results

The inference process and findings reported by Peer Nagy et al. are particularly relevant as they are the basis of this thesis and what we aim to benchmark. The inference phase of their model leverages the JAX-LOB simulator [14] to reconstruct Limit Order Book (LOB) states from the generated message sequences. This reconstruction is crucial for simulating the realistic dynamics of a financial market, as the simulator processes these messages according to standard market rules, ensuring that the resulting LOB states behave as they would in an actual market.

Their experimental results show that the model is capable of generating data that closely approximates the target distribution, validated through several key metrics. First, they found a strong alignment between the unconditional marginal distributions of the generated data and actual market data. Second, they assessed the model's ability to match conditional distributions by calculating the correlation between generated mid-price returns and those observed in real market data. The model demonstrated forecasting per-

formance, with correlations comparable to those achieved by other deep learning models explicitly trained for mid-price prediction. Finally, they evaluated the model's overall performance using perplexity scores, a common metric in language modeling, which confirmed the model's ability to generate realistic and coherent sequences of order book messages.

Given that this paper serves as the foundation for the architecture implemented in this thesis, the subsequent chapters will analyze more in depth the data, architecture, and results. Building on this initial implementation, our work begins by benchmarking and stress-testing the model. We then explore modifications to the architecture and experiment with different models to evaluate their performance in generating realistic order book data. Through this iterative process, we aim to refine the approach and contribute further to the development of generative models in financial markets.

Chapter 3

Research Questions

The previous chapters have reviewed the relevant literature, discussed state-of-the-art projects, and introduced key concepts necessary to understand the field of order book modeling. With this foundational knowledge established, we now turn to the core objective of this thesis: to explore the challenges of simulating order book states through the generation of order book message flows, and to establish a baseline for future research in this area.

To guide this exploration, we have formulated several research questions that will structure our inquiry and focus our efforts on the most critical aspects of this problem:

1. **How can we benchmark the current reference model on our dataset and what evaluation metrics can we use?**

   Benchmarking is a critical first step in our research, as it allows us to assess the performance of the reference model on our specific dataset. By identifying appropriate evaluation metrics, we can quantitatively measure how well the model reproduces realistic order book states from generated message flows. These metrics might include measures such as distributional similarity, predictive accuracy of mid-price movements, and computational efficiency. This question is essential because it establishes the baseline performance of the model and provides a standard against which any modifications or new models can be compared.

2. **What model architecture allows for the best performance?**

   The architecture of the model plays a pivotal role in its ability to generate accurate and realistic order book messages. Different architectural setups can impact the quality of the generated data. This question seeks to identify the optimal model architecture size that maximizes the fidelity of the order book simulation. By experimenting with different architectural choices and varying the model size, we aim to discover the configurations that best capture the nuances of order book dynamics and offer the highest quality reproduction of LOB states.

3. **How do adverse market condition affect the message flow simulation? Can the model be used to simulate trade impact on market?**

   Stress testing involves subjecting the model to extreme conditions or scenarios to evaluate its robustness and reliability. This question explores how the model behaves under such conditions, particularly in terms of simulating trade impact. By understanding how the model responds to high levels of activity with large orders, we can assess its ability to replicate the complexities of real-world trading environments. Additionally, this question investigates whether the model can be used as a tool to simulate trade impacts, providing insights into how large trades might affect market dynamics and liquidity. The ability to accurately simulate these impacts would be valuable for both academic research and practical applications in finance.

These research questions are designed to address the key challenges in the field of order book modeling. By systematically investigating each question, this thesis aims to contribute to the development of more accurate and robust generative models for financial markets. The insights gained from this research will not only advance our understanding of the problem but also provide a foundation for future work in the area, ultimately improving the tools available for market analysis and trading strategy development.

Chapter 4

Datasets

The aim of this thesis is to implement and benchmark a model capable of generating realistic LOB messages based on previous ones and order book states. Achieving this goal requires high-quality data on which to train the model. Specifically, we need a dataset composed of two elements: the message flow of orders and the corresponding flow of order book states. To achieve this, we obtained a reduced version of the LOBSTER dataset [18] which contains message files and order book data for two NASDAQ stocks: Cisco Systems (CSCO) and Priceline Group (PCLN).

4.1 LOBSTER Dataset

Our LOBSTER dataset consists of 62 days of Level-3 data for both CSCO and PCLN, spanning from January 2, 2015, to March 24, 2015. In the context of LOBs, the terms "Level-2" and "Level-3" data refer to different depths of market data information [19]:

- **Level-2 Data**: Provides detailed information about the order book at various price levels, including the best bid and ask prices and the aggregated volume of orders at each level. However, it does not track individual orders; instead, it aggregates the volume at each price level, showing the total volume available at the best bid or ask prices without detailing the individual orders that comprise this volume.
- **Level-3 Data**: Offers a more granular view, including all the information available in Level-2 data, but with the added detail of tracking individual orders within the order book. Level-3 data includes the price, size, time of placement, and order ID of each individual order, enabling a complete reconstruction of the order book's state over time. It captures every action that affects the LOB, such as order submissions, cancellations, modifications, and executions.

The Level-3 data for these 62 days results in a total of around 50k samples for CSCO and 10k samples for PCLN where each sample is composed of non-overlapping sequences

of 500 messages and corresponding states each.

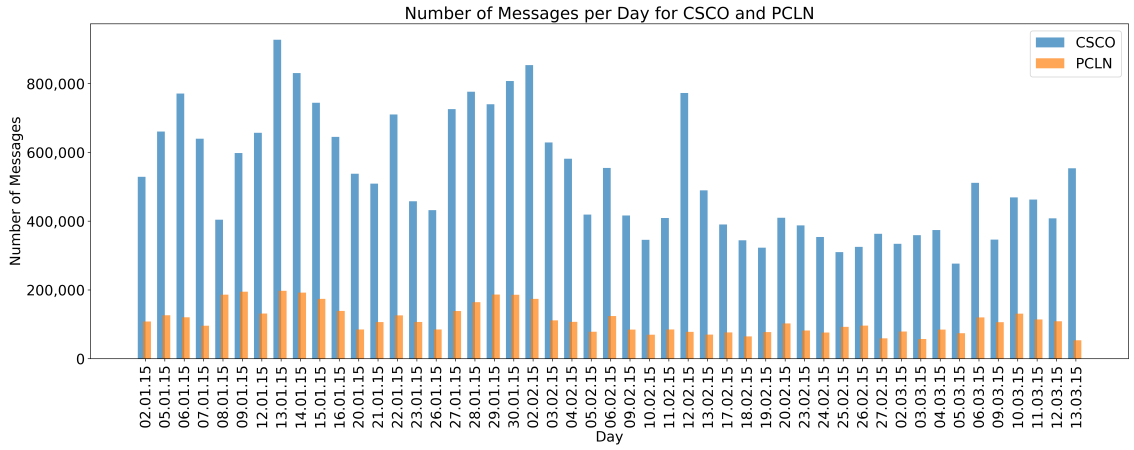

Figure 4.1: Messages per day in the training set for CSCO and PCLN

Notably, the volume of total messages is significantly higher for CSCO, highlighting that CSCO is a small-tick stock, while PCLN can be considered a large-tick stock.

- **Small-tick stocks**: These are stocks where the tick size (the minimum price increment between consecutive price levels, for example \$0.01) is small relative to the stock price. Small-tick stocks tend to have more frequent price changes and a larger number of orders at each price level, leading to a more densely populated order book.
- **Large-tick stocks**: Large-tick stocks are those where the tick size is large relative to the stock price. These stocks tend to have less frequent price changes, with larger gaps between price levels, resulting in a sparser order book with a more stable spread over time and a less dense message flow.

The dataset includes only trades during regular NASDAQ trading hours, on working days between 09:30 and 16:00 US East Coast time. LOBSTER data contains seven types of messages: new limit orders, partial order cancellations, full order deletions, visible order executions, hidden order executions, auction trades, and trading halts. However, only the first four types are utilized, as hidden orders and trading halts do not impact the visible dynamics of the order book. Each message comprises six 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).

Figure 4.2: LOB message

4.2 Data Preprocessing

The data preprocessing follows the approach outlined in our reference study [4] to prepare the Level-3 data for integration into the model loop. Initially, raw message and book data are loaded, with the message data filtered to include only the relevant types: new limit orders, cancellations, deletions, and executions. The next step involves converting prices from their original dollar values into ticks (cents) relative to the previous mid-price. This conversion starts by calculating the mid-price, defined as the average of the best bid and ask prices at a given time. The original prices are then transformed into integers representing their distance, in ticks, from this mid-price. If the mid-price falls between valid tick levels, it is rounded down to the nearest tick. To maintain consistency, prices beyond a predefined range—specifically, more than 999 ticks above or below the mid-price—are truncated to ensure the data remains within a finite and manageable range.

A new feature is also added as the inter-arrival time, the time difference between the arrival of two consecutive messages. By calculating this difference, we obtain a new feature that captures the temporal spacing between orders, rather than the absolute time at which each order was placed. This feature is much more informative and easier for the model to learn from, as it directly reflects the intensity and rhythm of trading activity.

For referential messages (cancellations, deletions, and executions), which need to reference existing orders in the book, additional details are appended to preserve the correct relationships between orders. Specifically, each referential message is enriched with information about the original order it refers to, in particular, the price, size, and the timestamp of the original order. This additional information helps the model correctly identify and link these messages to the corresponding orders within the order book, ensuring the integrity and accuracy of the order flow sequence during both training and inference. In the case of a new limit order, the reference fields receive an NA value.

The message data is then tokenized, meaning that each field (e.g., event type, price, size) is broken down into smaller units, or tokens, that can be understood and processed by the model. This conversion transforms complex numerical or categorical values into a sequence of simpler elements. Meanwhile, the book data is processed into a sparse representation of liquidity around the mid-price.

To represent the order book state, a compact vector representation that captures essential market information is used. The vector has a dimensionality of $4 \times k + 1$, where k is the number of price levels around the mid-price, and the additional dimension tracks the mid-price change from the previous observation. The order book data contains k=10 price levels, so the resulting total vector size corresponds to 40+1.

We begin by calculating the mid-price, p_m, defined as the average of the best ask and bid prices. The price levels, $p_{a,j}$ and $p_{b,j}$, are then converted from absolute to relative values by subtracting the previous state's mid-price. Finally, the change in mid-price, Δp_m

and all price features are rounded to the nearest tick size, resulting in adjusted prices $\tilde{p}_{a,j}$ and $\tilde{p}_{b,j}$.

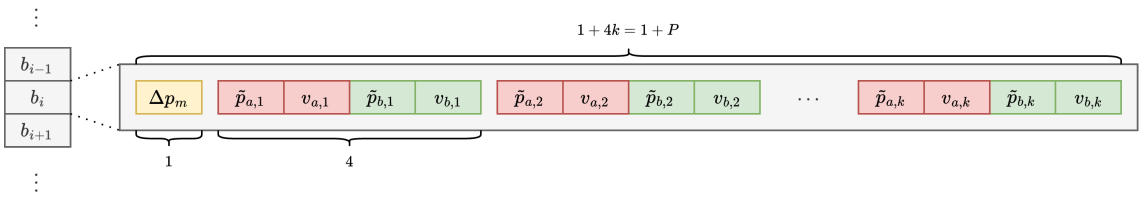

Figure 4.3: Processed order book representation [2]

The preprocessing steps are essential to make sure that the data is structured in an efficient and stable format and content so that the model can effectively learn and then replicate market dynamics.

Table 4.1: Structure of an Order Book Message and an Order Book State

Field	**Description**
Order Book Message	
`time`	Timestamp in nanoseconds after midnight, indicating when the message was generated
`event_type`	Type of event: 1 = New order, 2 = Partial cancellation, 3 = Full deletion, 4 = Visible execution
`order_id`	Unique identifier for each order in the order book
`size`	Size of the order, representing the number of shares
`price`	Price of the order, expressed in ticks relative to the mid-price
`direction`	Direction of the trade: 1 = Buy, 2 = Sell
`inter arrival time`	Timestamp after previous order
`reference order price`	Price of the referenced order in ticks
`reference order size`	Size of the referenced order
`reference order time`	Timestamp of the referenced order
Order Book State	
`Order book vector`	Array representing volume and price level (in ticks) around the mid-price
`mid_price_change`	Change in mid-price from the previous observation

4.3 Encoding

A key step of the preprocessing phase is the encoding where the data is encoded from the raw LOBSTER dataset format to a tokenized format.

4.3.1 Tokenization

Tokenization is a fundamental process in many machine learning tasks, particularly those involving sequence data like natural language processing (NLP) [20] and, in this case, financial market data. The essence of tokenization lies in converting raw data into a format that a machine learning model can effectively understand and process. By breaking down the data into smaller, discrete units known as tokens, the model can learn to recognize patterns, make predictions, and generate outputs based on these sequences of tokens.

In the context of Limit Order Book (LOB) modeling, tokenization serves to transform complex, multi-dimensional financial data into a structured sequence of numerical values that represent different aspects of market activity. Tokens allow the model to work with the data at a granular level, capturing the dynamics that are essential for accurate simulation and prediction. Without tokenization, the raw data would be too complex and unstructured for the model to process efficiently, making it difficult to achieve high levels of performance in tasks such as order book simulation and order flow prediction.

The tokenization phase is crucial because 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.

4.3.2 Encoding process

The encoding and tokenization process for LOB data begins after the data has been preprocessed and structured into a standardized format as specified in the previous section. Each field within a message from the LOBSTER dataset, such as event type, price, size, and time, is transformed into a sequence of tokens that can be fed into the model. These tokens are then used by the model to learn the conditional distributions of future market states based on past and present data.

The process involves several key steps:

- **Mapping Fields to Tokens**: Each field in a LOB message is mapped to a set of tokens. The mapping is based on a predefined vocabulary that assigns a unique token to each possible value within the field.

- **Splitting and Encoding Values**: For fields that contain numerical values, such as prices and times, the values are often split into multiple tokens. For example, a price might be split into tokens representing its sign (positive or negative) and its distance from a reference point, such as the mid-price.

- **Handling Special Values**: Special values, such as unused data, masked fields or hidden values, are assigned specific tokens to ensure that the model can recognize and appropriately handle these cases during training and inference.

- **Combining Tokens**: The tokens generated from each field are then combined into a single sequence representing the entire LOB message. This sequence is what the model ultimately uses to learn the patterns in the data.

4.3.2.1 Tokenization of Separate Fields

Each field in the dataset requires a specific tokenization method to represent its value. Below, we outline the approach for tokenizing each field.

Event Type The event type field indicates the specific action occurring in the order book, such as the placement of a new order, a cancellation, a deletion, or an execution. Each possible event type is directly mapped to a unique token representing the specific event.

Direction The direction field, which denotes whether the order is a buy or sell order, is also directly encoded into a single token, with distinct tokens representing the buy (1) and sell (2) directions, enabling the model to differentiate between the two types of orders.

Price The price field requires a more intricate encoding process. Since prices in financial data are often expressed with high precision, they are first converted into ticks relative to the mid-price (calculated as the average of the best bid and ask prices). The price is then split into two tokens: the first token represents the sign (indicating whether the price is above or below the mid-price), and the second token captures the tick distance from the mid-price.

Size The size field represents the number of shares in an order. Since sizes can also vary greatly, they are truncated to a maximum value to ensure consistency and then encoded into a single token.

Time Time is one of the most granular fields in LOB data, often recorded in nanoseconds. Given this precision, time fields are split into multiple tokens, with each token representing a specific segment of the time value (e.g., seconds and nanoseconds). This approach ensures that the model can capture the fine-grained temporal patterns that are essential for understanding the sequence of market events. The same tokenization is applied to the inter-arrival time field.

Referential Fields In messages that reference previous orders, such as cancellations or executions, additional fields are included to link back to the original order. These include the referenced price, size, and time, which are encoded in a manner similar to the primary fields. The referenced order ID may also be included to maintain the correct relationships between orders.

Table 4.2: Tokenization of each field in a message

Field	Original Data Type	Tokens Generated
`event_type`	Integer	1 token
`direction`	Integer	1 token
`price`	Integer	2 tokens (sign + tick distance)
`size`	Integer	1 token
`time`	Seconds + Nanoseconds	5 tokens (2 for seconds, 3 for nanoseconds)
`inter_arrival_time`	Seconds + Nanoseconds	4 tokens (1 for seconds, 3 for nanoseconds)
`referenced_price`	Integer	2 tokens (sign + tick distance)
`referenced_size`	Integer	1 token
`referenced_time`	Seconds + Nanoseconds	5 tokens (2 for seconds, 3 for nanoseconds)

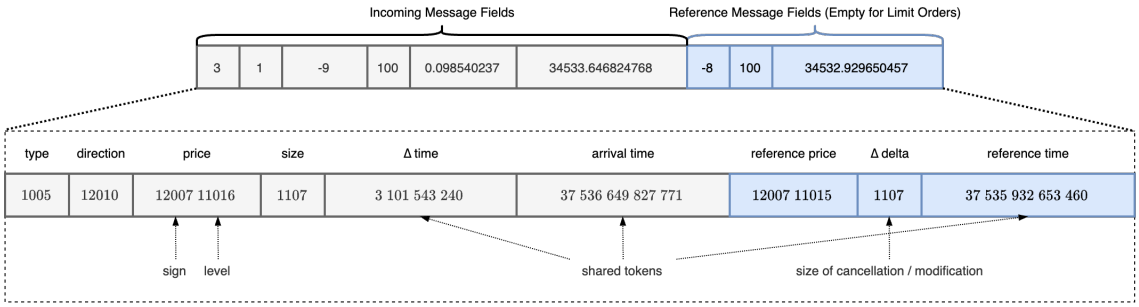

Figure 4.4: Representation of message tokenization [2]

In our specific dataset, the tokenization process results in a vocabulary comprising **12,011 distinct tokens**. Each LOB message, after being fully tokenized, is represented by a sequence of **22 tokens**. The sequence encodes all relevant aspects of the order, including event type, direction, price, size, time, and any referential information.

4.4 Train-Test Split

All this steps give us tokenized datasets, one relative to CSCO and the other relative to PCLN. As we will go over in Chapter 6, the two datasets were used to train different versions of the model to understand its behaviour in predicting large and small tick stocks respectively. To perform training, each dataset is split into three sets, a training set, a validation set and a test set. Validation and Test set accounted for 6 trading days worth of messages each, while Training set accounted for the remaining 50.

Chapter 5

Models & Approach

In this chapter we will go over the approach, the model structure and training used in this project. Then we will go over how the inference of new sample works and briefly how we performed message injection to stress test the model.

5.1 Architecture

In our research project, we began with the reference architecture and made several modifications to create a revised version for testing on our data. First, we will outline the architecture of the model presented in the reference paper [4], referred to as our "Null Model", followed by an explanation of the changes we introduced to its structure.

5.1.1 Null Model

The "Null model" architecture is built using a deep network of simplified structured state-space layers (S5). This model takes in two types of input sequences, the tokenized LOB messages and volume images of the level-2 LOB states that flow into two different sections of the model. The message sequence $m \in \mathbb{V}^{22n}$, where $\mathbb{V} \subset \mathbb{N}$ represents the token vocabulary, and n is the sequence length which was set to $n = 500$ messages. The order book sequence $b \in \mathbb{B}^n$ includes P volumes and the previous mid-price change.

Mathematically we can define the model as a function g_θ, which takes the input sequences m (the message sequence) and b (the order book state sequence) and maps them to an output vector $\hat{y}$. This output vector $\hat{y}$ belongs to $\mathbb{R}^v$, where v represents the size of the token vocabulary. Formally, this can be expressed as:

$$g_\theta : x \to \hat{y} \in \mathbb{R}^v$$
$$\text{where } x = (m, b)$$

The output $\hat{y}$ represents a set of logits that are transformed into a probability distribution over the vocabulary $\mathbb{V}$. The model learns this distribution for the masked token,

given the input sequences. The training process involves adjusting the parameters θ to minimize the difference between the predicted distribution and the actual distribution of the tokens using cross-entropy loss.

The implementation, illustrated in Figure 5.1, consists of two parallel input branches designed to process distinct types of data: tokenized message sequences and order book state sequences. Each branch leverages a deep network composed of simplified structured state-space (S5) layers combined with dense layers, providing the model with the ability to handle complex, long-range dependencies in sequential data.

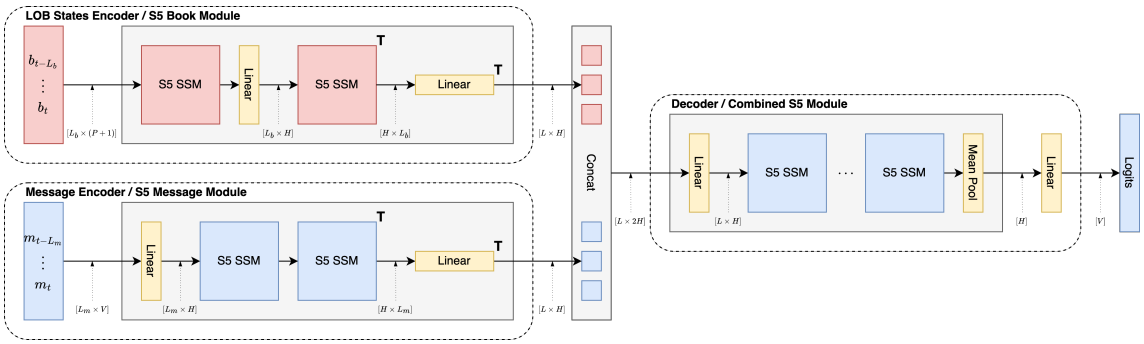

Figure 5.1: Model architecture [2]

Message Branch:

- **Input:** The message branch receives sequences of tokenized messages, where each message is encoded into a sequence of tokens.
- **Processing:** The branch starts with a linear embedding layer that projects each one-hot encoded token vector into a higher-dimensional space, defined by the hidden dimension parameter `d_model`. Following this, the embedded sequence moves through a series of stacked S5 layers. These S5 layers, leveraging a state-space formulation, are well-suited for capturing long-range dependencies and sequential patterns in the data.

Book State Branch:

- **Input:** The book state branch processes sequences of order book states, which are not tokenized but rather consist of continuous numerical values representing liquidity at different price levels.
- **Processing:** Each order book state observation, represented as a (40+1)-dimensional vector, is initially passed through an S5 layer. The output is then projected into the same hidden dimension `d_model` as the message branch, ensuring that both input types are mapped into a comparable feature space.

Concatenation After the individual processing of the message and book state sequences, their outputs are projected to a common sequence length L. The two sequences are then concatenated along the feature dimension, creating a unified representation that encapsulates both the message and book state information. The concatenated sequence is passed through an additional block of stacked S5 layers, further refining the combined representation and allowing the model to learn complex interactions between the message and book state data. Finally, the processed sequence is passed through a dense layer, projecting it to the output space defined by the vocabulary size v. This projection is followed by a softmax operation, which produces a probability distribution over the possible output tokens.

5.1.2 Revised Model Implementation

In our revised model implementation, we introduce an additional enhancement by incorporating `gelu` activation functions at key points in the architecture. Such modifications aim to improve the non-linear representation capability of the model, allowing it to better capture complex relationships in the input data.

GELU Activation After Encoders:

- **Message Encoder:** After the message sequences are processed by the stacked S5 layers in the message encoder, we now apply a `gelu` activation function after projecting the sequence to the `d_model` size. This non-linear activation introduces additional flexibility into the model's ability to represent features learned from the message sequences.
- **Book Encoder:** Similarly, after the order book state sequences pass through their dedicated S5 layer in the book encoder, we apply a `gelu` activation function. The activation function is situated after the projection to the `d_model` size, ensuring that the book state features are enhanced through a non-linear transformation before integration with the message sequences.

The original model did not include these `gelu` activations after the message and book encoders. Instead, the sequences were projected directly to the `d_model` size without any non-linear transformation. By introducing `gelu` activations at these critical junctures, the data undergoes a more robust transformation, potentially allowing the model to learn more nuanced patterns within the input sequences [21].

5.2 Training Process

The training process employs a masked language modeling approach, where a random token in the last message is masked using a special `MSK` token. All tokens following this

masked token are replaced with `HID` tokens to prevent the model from conditioning on them. Notably, time tokens of new messages are not masked since they are computed from generated inter-arrival times Δt and thus are not prediction targets.

5.2.1 Cross-Entropy Loss Application

The model parameters θ are trained by minimizing the cross-entropy loss over the training data. Cross-entropy is a commonly used loss function in classification tasks, especially in sequence modeling, where the objective is to minimize the difference between the predicted probability distribution and the actual distribution of tokens.

For a given input sequence x, the model produces a probability distribution over the vocabulary for each token. Let y denote the actual token (in one-hot encoded format) and $\hat{y}$ denote the predicted probability distribution over the vocabulary. The cross-entropy loss L for a single token prediction is defined as:

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

Where:

- v is the size of the token vocabulary.
- y_i is the actual token value (1 if the token is correct, 0 otherwise).
- $\hat{y}_i$ is the predicted probability for token i.

During training, the total loss is computed as the sum of the cross-entropy losses for all tokens in the sequence. This sum is then averaged over the entire batch of sequences. The loss is minimized using gradient descent, specifically the Adam optimizer, which adjusts the model parameters θ to reduce the difference between the predicted and actual tokens over multiple epochs.

5.2.2 Training Loop

In each training epoch, the model processes a batch of sequences consisting of tokenized LOB messages and corresponding order book states. The sequences are passed through the network, where the message and book modules process them separately using S5 layers and dense layers. The sequences are then concatenated and further processed by additional S5 layers in the combined module.

For each token in the masked message sequence, the model predicts the corresponding token based on the previous tokens in the sequence. The predicted token distributions are compared against the actual tokens using the cross-entropy loss function described above. The gradients of the loss with respect to the model parameters are computed, and the

parameters are updated using the Adam optimizer [22] for regular layer parameters, and AdamW for SSM layer parameters [23].

The model is trained on sequences of $n = 500$ messages and LOB states. Each message is represented by 22 tokens, resulting in an encoded sequence length of 11,000 tokens. The book states are not tokenized, so their sequence length is 500 observations. To vary the input sequences and the position of the `MSK` token, a random number of observations is skipped from the start of each day during each training epoch.

5.3 Inference

During inference, the model generates new order book (LOB) states through a multi-step autoregressive process, utilizing both the message sequences and an LOB simulator. The process begins with the model receiving a tokenized message sequence, where the first token of the last message is set to `MSK` and the remaining tokens to `HID`. Tokens are sampled sequentially in a left-to-right manner using the softmax over the output logits, with syntactically valid tokens being selected based on the predicted scores. Once the inter-arrival time (Δt) tokens are sampled, the arrival time of the message is calculated and added to the sequence and the time related tokens are skipped. The new message is then passed to the LOB simulator then updates the state by applying the generated message according to the LOB matching rules. To ensure accuracy, the simulator employs an error correction mechanism that addresses any inconsistencies, particularly in referential messages such as cancellations, deletions, and executions. The correction mechanism is an important aspect for maintaining the integrity of the order flow, as it ensures that the generated messages accurately reflect the real-world LOB dynamics.

5.3.1 Error Correction Mechanism in Inference

During the inference phase, the model generates messages that are applied to the Limit Order Book (LOB) to update its state. However, because the model's predictions can occasionally produce errors, especially when generating messages that reference non-existent or incorrect orders, an error correction mechanism is vital for ensuring the validity and coherence of the LOB updates.

The error correction process begins by decoding the generated message into a structured format that can be applied to the LOB. For new limit orders, the process checks whether the order is immediately marketable—i.e., executable against the current best bid or ask prices. If the order is marketable and so invalid in this context, it is discarded to prevent incorrect updates to the LOB.

For messages that modify, delete, or execute existing orders, the process attempts to locate the referenced order in the current state of the LOB. If the referenced order cannot be found—possibly because it does not exist or has already been executed—the error

Algorithm 1 Message Generation Process Pseudocode

Input: Message sequence $\mathbf{m}_i$, book state sequence $\mathbf{b}_i$, JAXLOB simulator $\oplus$, model $f(\cdot)$
while `message` not valid **do**
    $\mathbf{m}'_i \leftarrow \mathbf{m}_i$ ▷ reset $\mathbf{m}_i$ if generated message is invalid
    `tokens` $\leftarrow$ [`<HID>`] $\times$ 22 ▷ initialise message with list of 22 `<hid>` tokens
    for each token position i in the new message **do**
        Mask the i-th token with the `<MSK>` token.
        `new_token` $\leftarrow f(\mathbf{m}'_i, \mathbf{b}_i)$ ▷ tokens generated with top-k random sampling
        `tokens[i]` $\leftarrow$ `new_token`
        $\mathbf{m}'_i \leftarrow$ `update`$(\mathbf{m}_i,$ `new_token`$)$
    end for
    `message` $\leftarrow$ `parse`(`tokens`)
    if `error_correct(message)` success **then**
        $b_t \leftarrow \mathbf{b}_i$`[-1]`
        $b_{t+1} \leftarrow b_t \oplus$ `message` ▷ update LOB state with simulator
        $\mathbf{b}_i \leftarrow \mathbf{b}_i$`[1:]` $+ b_{t+1}$ ▷ Roll window forward
        $\mathbf{m}_i \leftarrow \mathbf{m}'_i$
        `break`
    end if
end while
Step 2: Repeat the above process until the required number of prediction steps are completed.

correction mechanism adjusts the message. This may involve finding a similar order that does exist or referencing the initial volume remaining at the specified price level. The objective is to ensure that the message can still be applied in a manner that preserves the logical integrity of the LOB.

Once the message is corrected, it is re-encoded into the tokenized format used by the model, ensuring that the entire inference pipeline continues to function smoothly. This error correction process is essential for allowing the model to generate meaningful messages and ensure consistent reference to previous orders, where small errors could otherwise propagate and result in significant discrepancies or in the simulated market.

Chapter 6

Experiments

In this chapter we will present the experiments that were conducted and illustrate the results obtained. However before diving into those aspect we will go over how we evaluated our model's performance.

6.1 Evaluation

To assess the performance of our generative model, we employ several evaluation metrics, following the approach in the original reference study [4]. In addition to these, we introduce new metrics to provide a more comprehensive understanding of the model's behavior and its ability to generate realistic market microstructure data. The evaluation methodologies are the following:

- **Comparing Distributions:** The first metric focuses on the event type and timing aspects of the message flow. We evaluate these properties by analyzing the unconditional marginal distributions in both the generated and real data. This is accomplished by plotting and comparing these distributions against the corresponding empirical distributions from the actual data. The visual comparison is represented through probability-probability (P-P) plots and histograms, allowing us to see any deviations from the original data.

- **Perplexity:** We use perplexity, a common measure in evaluating large language models (LLMs), to quantify the model's uncertainty in predicting the next token in a sequence. Perplexity P is defined as:

$$P = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 p(x_i)}$$

  where N is the number of tokens in the sequence, and $p(x_i)$ is the probability assigned by the model to the i-th token in the sequence. In practice, and in this work, we often compute perplexity based on the average log-likelihood over the sequence:

$$P = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right)$$

Lower perplexity values indicate a better fit to the data, reflecting the model's ability to generate tokens that closely resemble those in the actual data. Perplexity was analyzed in two ways: one, presented in our reference study by Peer Nagy et at. [4], followed a rolling window approach on a per message basis where the perplexity value of a message is the average value across tokens. The other implementation consists in a per-token perplexity.

- **Per-Token Evaluation:** In addition to per-message perplexity, we introduced new metrics that assess model performance on a per-token basis. These metrics include Cross-Entropy, Accuracy, Top-10 Accuracy, Cosine Similarity, and Perplexity for each token type. As discussed in the following sections, these metrics will be crucial for diagnosing specific issues within the model, as they allow us to dissect the predicted message and identify where errors occur.

- **Mid-Price Return Analysis:** To further understand the model's performance, we track fluctuations in mid-price returns over 100 generated messages, starting from each sequence's initial message. By comparing the generated and realized returns distributions, we assess how well the model captures the dynamics of mid-price movements, a vital aspect of market microstructure modeling. The overlap of these distributions, particularly within the 95% confidence intervals, serves as a key indicator of the model's ability to replicate the complex behavior of financial markets.

6.2 Experiments

After explaining the evaluation metrics employed, we now present the experiments conducted during this thesis and discuss the results obtained.

6.2.1 Experiment 1 - Benchmark

In the first experiment, we benchmarked the reference model, referred to as the "Null model", on our two datasets. As explained in Chapter 4, the dataset was split into Training, Validation and Test sets. The model was trained on the Training set with the Validation set used to monitor loss and the Test set employed for performance evaluation. Training was conducted for 50 epochs on both the CSCO and PCLN datasets until convergence was achieved. After training, we generated 500 sequences of 100 messages each using the test set. Additionally the model generated each sequence 100 message sequence twice, introducing variations in token selection methods (sampling between top k logits

instead of just the maximum). The sequences were then averaged to reduce the model's error-prone nature.

6.2.1.1 CSCO "Null model" Results

For the CSCO dataset, Figure 6.1, 6.2 and 6.3 illustrate how the "Null model" predicts event types and time-related tokens across the 500 test sequences.

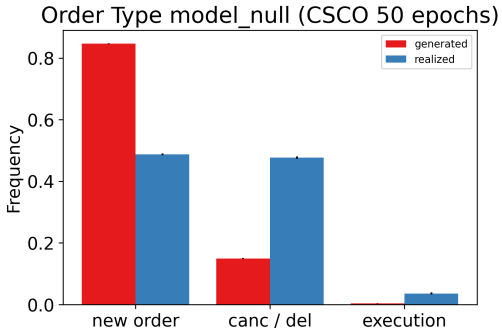

Figure 6.1: CSCO "Null" - Order Type frequency for generated and actual messages

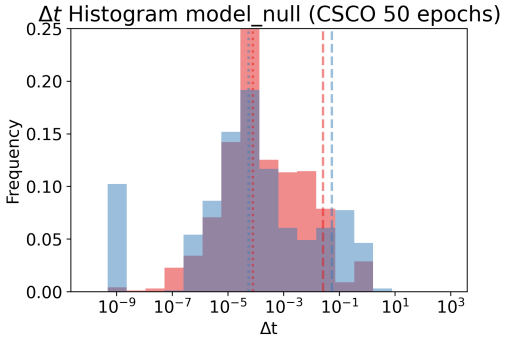

Figure 6.2: CSCO "Null" - Inter-arrival times frequency for generated and actual messages

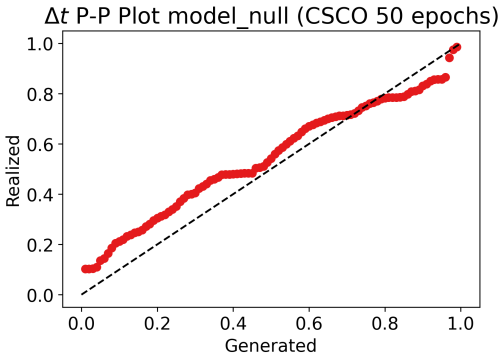

Figure 6.3: CSCO "Null" - P-P plot of inter-arrival time for generated and actual messages

The figures show that the model is able to correctly model the time aspect of messages: the histogram and P-P plot show that the model generates Δt closely resemble those in the real data. However, the model fails to generate the correct proportion of message types for execution and deletion/cancellation orders.

In terms of price and quantity, Figures 6.4, 6.5, and 6.6 reveal that the model's performance is insufficient for predicting meaningful information about returns and mid-price movements.

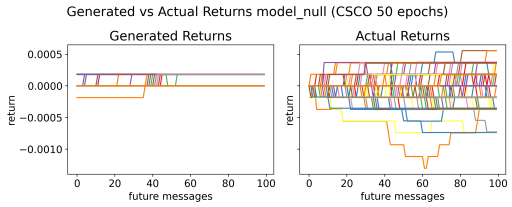

Figure 6.4: CSCO "Null" - Mid-Price Movement Sequences comparison between generated and actual

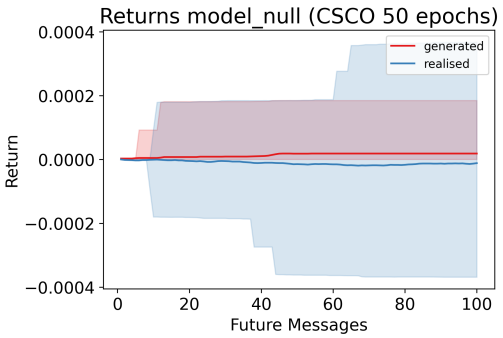

Figure 6.5: CSCO "Null" - Average movement of mid price for generated compared to actual messages

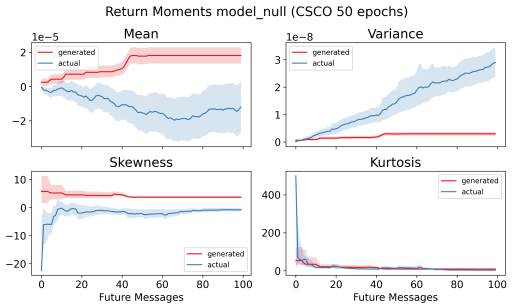

Figure 6.6: CSCO "Null" - Moments of the mid-price returns for generated and actual messages

It is thus evident that the mid-price movement does not reproduce the real movement. While some movement from the starting price is introduced, the variance is too low compared to real-world scenarios, indicating that the model underfits the data and fails to predict price and quantity tokens effectively.

6.2.1.2 PCLN "Null model" Results

On the large-tick PCLN dataset, similar results are observed, as shown in Figures 6.7, 6.8, and 6.9. The model reasonably predicts the timing but falls short in generating the correct event types, with a significant underrepresentation of cancellation/deletion orders.

As with the CSCO dataset, the model struggles to predict price and quantity effectively, resulting in mid-price movements that do not accurately replicate real market dynamics. The generated moments do not align with actual market conditions.

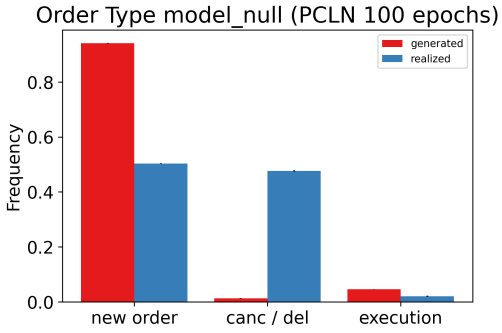

Figure 6.7: PCLN "Null" - Order Type frequency for generated and actual messages

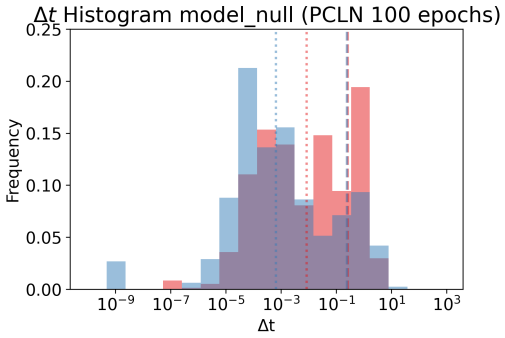

Figure 6.8: PCLN "Null" - Inter-arrival times frequency for generated and actual messages

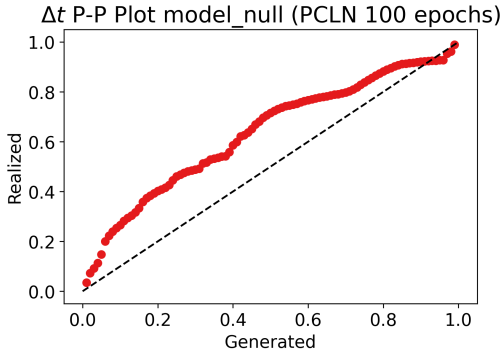

Figure 6.9: PCLN "Null" - P-P plot of inter-arrival time for generated and actual messages

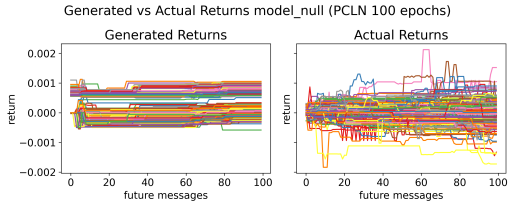

Figure 6.10: PCLN "Null" - Mid-Price Movement Sequences comparison between generated and actual

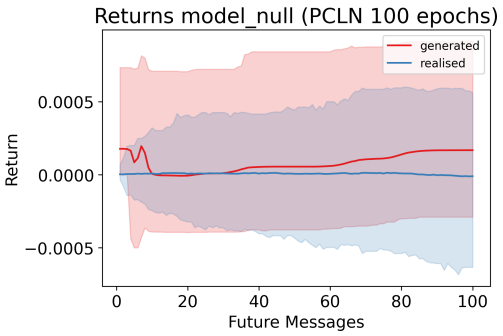

Figure 6.11: PCLN "Null" - Average movement of mid price for generated compared to actual messages

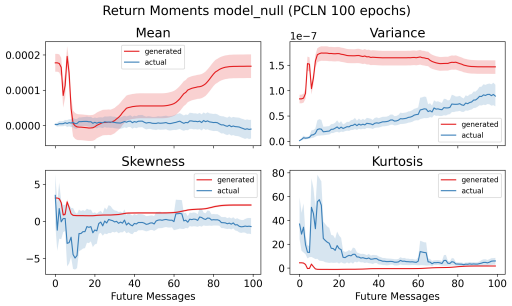

Figure 6.12: PCLN "Null" - Average movement of mid price for generated compared to actual messages

6.2.2 Experiment 2 - Best Model Architecture

The results of Experiment 1 revealed significant flaws in the model, demonstrating that, while it managed to replicate some properties of actual order message flow, it was not realistic enough to serve as a reliable generator or predictor of mid-price movements. To address these issues, we modified the "Null model" in two ways: first, by adding an additional layer of activation functions, and second, by varying the `d_model` parameter to explore whether increasing the model size could lead to improvements.

6.2.2.1 Role of `d_model`

The `d_model` parameter is a critical hyperparameter that determines the dimensionality of the feature space in which both the message tokens and book state observations are represented. This parameter controls the size of the hidden layers within the S5 and dense layers, effectively dictating the model's capacity to capture and represent complex patterns in the data.

Varying the `d_model` parameter allows for the adjustment of the model's complexity and capacity. Larger values of `d_model` increase the model's representational power but also raise computational requirements and the risk of overfitting. Conversely, smaller values reduce the model's capacity but can improve generalization and efficiency.

Table 6.1: Model Sizes and Corresponding Parameters

d_model Hyperparameter Value	Total Model Parameters
16	593,921
32	1,170,977
64	2,328,161
128	4,654,817

6.2.2.2 Results

Following the same procedure as in Experiment 1, we trained models with `d_model` values of 16, 32, 64, and 128 on both the CSCO and PCLN datasets. For the `d_model` = 32 model, the only change from the "Null model" was the addition of activation functions after the encoding modules.

The training metrics, shown in Figure 6.13, indicate that larger models have lower training and validation errors compared to the "Null model". The observed peaks in training loss are due to the use of a cosine annealing learning scheduler, which increases the learning rate at the start of each epoch to help escape local minima.

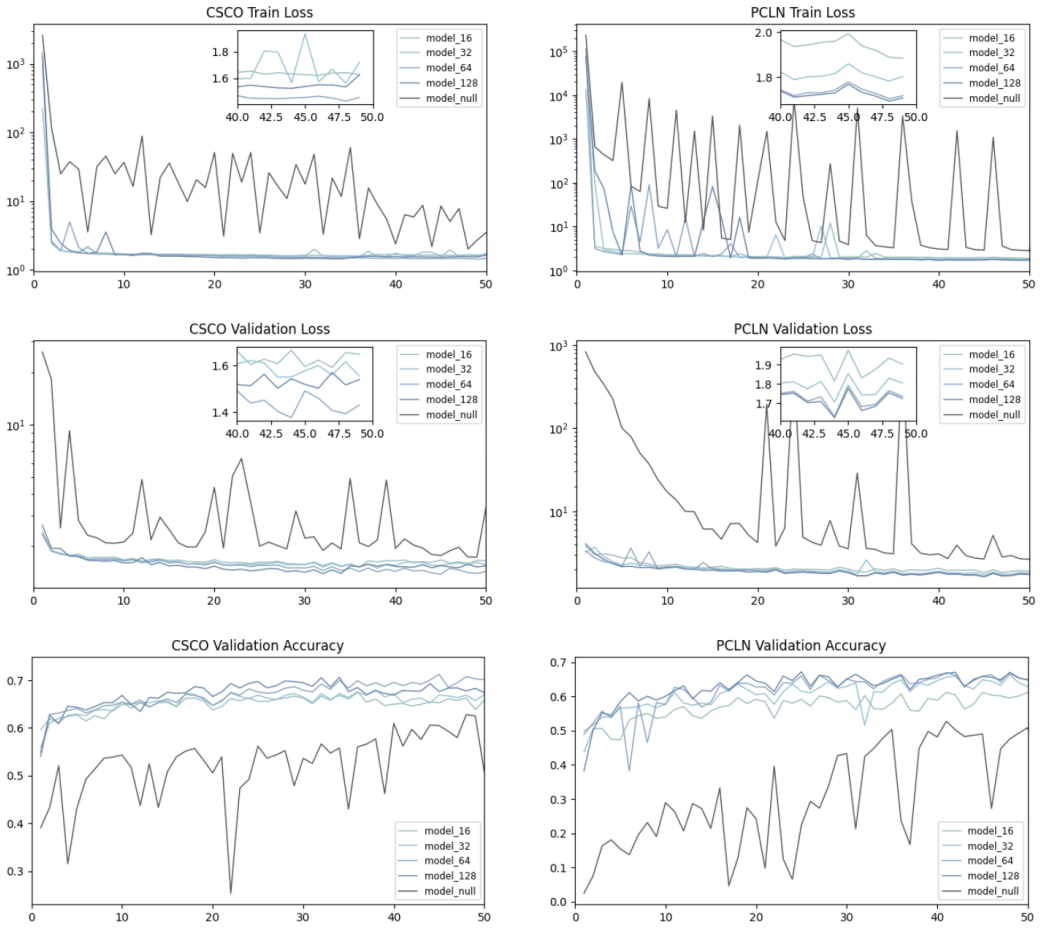

Figure 6.13: Training Loss metrics on all models trained for CSCO and PCLN [2]

On the testing side, we generated the same 500 sequences for each model and analyzed the results as in Experiment 1. The table below reports the per-message perplexity of each trained model.

Table 6.2: Mean and Median Perplexity for Various Models

Name	Mean Perplexity	Median Perplexity
csco_null_model	10.25	10.25
csco_model_16	7.83	7.90
csco_model_32	7.50	7.62
csco_model_64	7.64	7.71
csco_model_128	**7.23**	**7.36**
pcln_null_model	17.19	17.25
pcln_model_16	13.65	13.58
pcln_model_32	14.17	14.08
pcln_model_64	13.22	13.17
pcln_model_128	**12.5**5	**12.49**

The perplexity scores reveal that our new models performs better than the original reference models and highlight that larger models have lower scores, although the differences are not substantial enough to provide a clear performance advantage. To explore model behavior in depth, we analyzed the impact of generated messages on the LOB, their properties, and mid-price fluctuations during the forecasted message sequence. The following paragraphs, report the results for `d_model` = 16 and `d_model` = 128 for the CSCO and PCLN datasets. Results for `d_model` = 32 and `d_model` = 64 are included in Appendix A, as they display similar behavior.

6.2.2.3 CSCO

For CSCO we observed improvements in the generation of event types, as we see that they more closely match the actual proportions. The same happened for the modelling of inter-arrival times, which closely match the actual distributions as we can see from Figures 6.14 to 6.17.

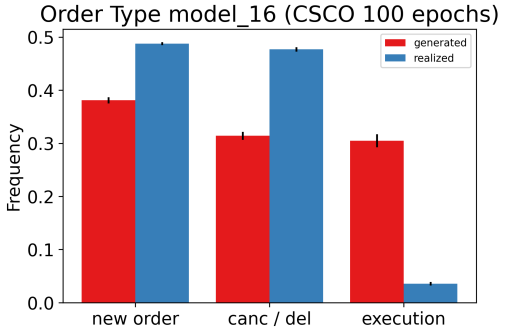

Figure 6.14: CSCO 16 - Order Type frequency for generated and actual messages

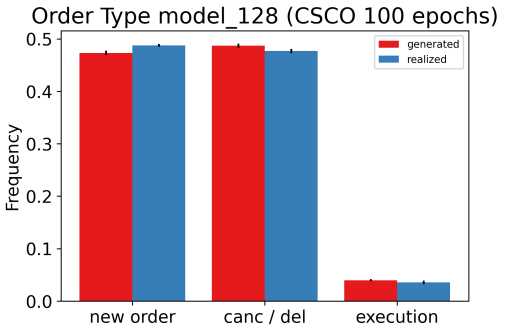

Figure 6.15: CSCO 128 - Order Type frequency for generated and actual messages

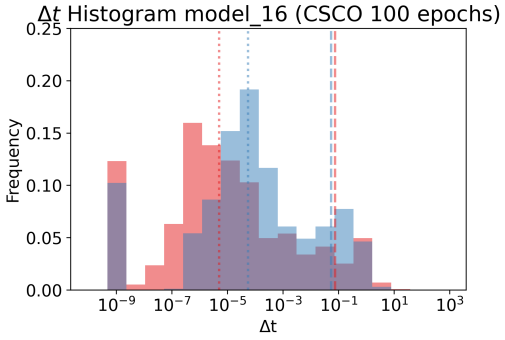

Figure 6.16: CSCO 16 - Inter-arrival times frequency for generated and actual messages

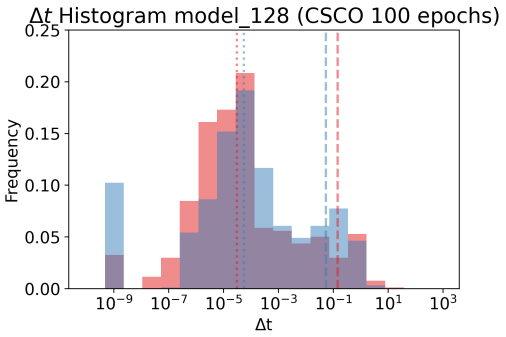

Figure 6.17: CSCO 128 - Inter-arrival times frequency for generated and actual messages

However, this improvement did not translate into better predictions of mid-price movements. Although the lower-dimensionality model (`d_model` = 16) showed better variance compared to Experiment 1, it was still insufficient to replicate real market dynamics. The figures below detail the mid-price movement for the `d_model` = 16 model. Graphs for the other model sizes (32, 64, 128) are included in the appendix and show similar results.

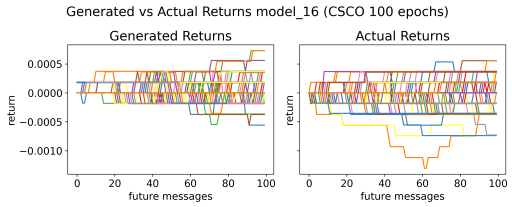

Figure 6.18: CSCO 16 - Mid-Price Movement Sequences comparison between generated and actual

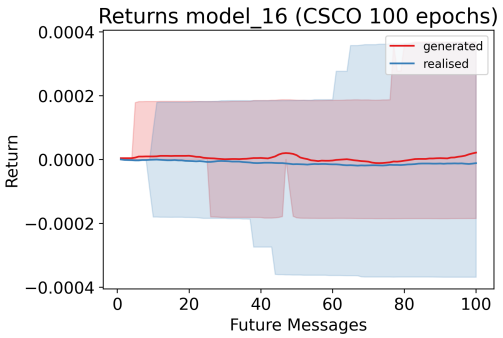

Figure 6.19: CSCO 16 - Average movement of mid price for generated compared to actual messages

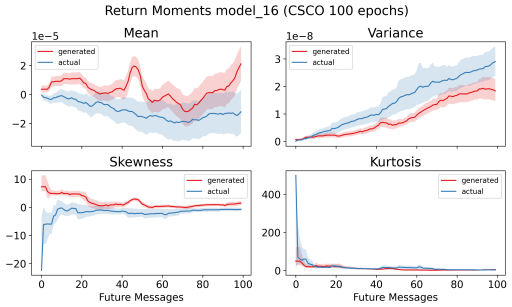

Figure 6.20: CSCO 16 - Moments of the mid-price returns for generated and actual messages

6.2.2.4 PCLN

For models trained on PLCN data we observed subpar generation of time and event type parameters compared to other models, but better performance in generating quantity and price. The analysis of the PCLN `d_model` = 16 model, shown in the figures below, indicates that it generates more varied mid-price fluctuations with better representation of variance, although the moments of the distributions remain inaccurate. Other PCLN models display similar behavior, with graphs available in the appendix.

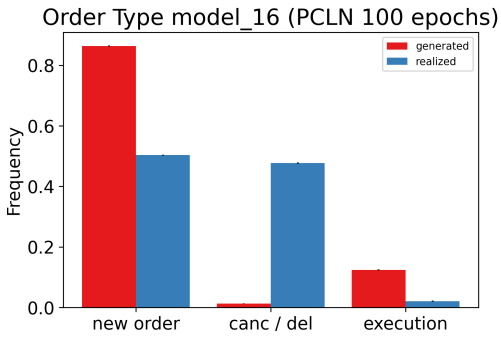

Figure 6.21: PCLN 16 - Order Type frequency for generated and actual messages

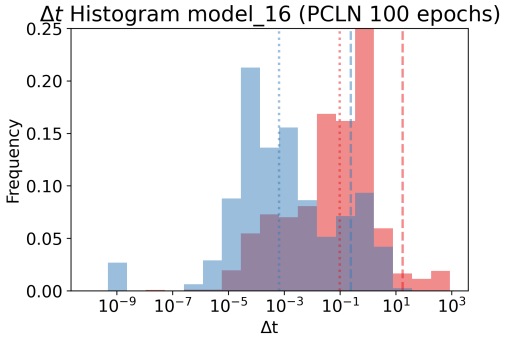

Figure 6.22: PCLN 16 - Inter-arrival times frequency for generated and actual messages

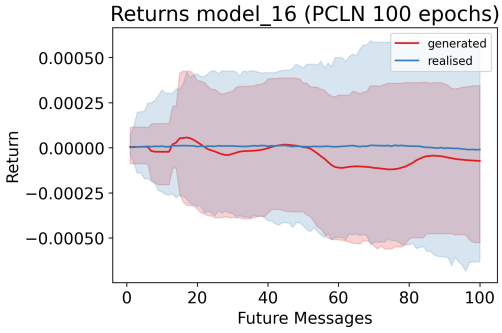

Figure 6.23: PCLN 16 - Average movement of mid price for generated compared to actual messages

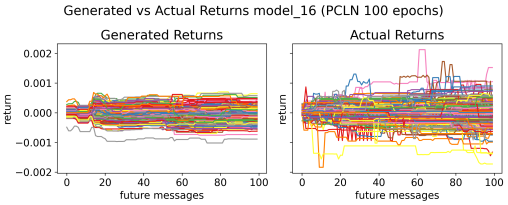

Figure 6.24: PCLN 16 - Mid-Price Movement Sequences comparison between generated and actual

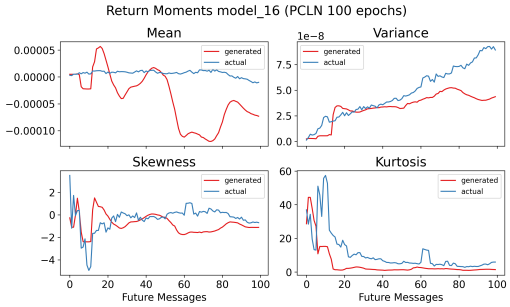

Figure 6.25: PCLN 16 - Moments of the mid-price returns for generated and actual messages

6.2.2.5 Conclusions

All models trained failed to consistently replicate the dynamics of a real-world LOB simulation and the properties of an actual message flow, even after modifying and increasing model size. To identify the root cause, we analyzed the per-token metrics (reported for CSCO model 16, 128 and PLCN model 16, while the remaining are in the appendix) in Table 6.5 to 6.8: the errors in predicting price and quantity tokens were significantly higher, indicating that the model struggled with these tokens compared to others.

Table 6.3: Per-token metrics model_16 on CSCO Dataset [2]

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.6187	0.6693	0.8394	0.7072	5.0467
time	3.3076	0.5146	0.5544	0.5301	27.3199
event_type	0.7960	0.5365	1.0000	0.6997	2.2167
size	2.5283	0.4822	0.8321	0.5264	12.5326
price	1.1620	0.6256	1.0000	0.6779	3.1965
sign	0.0365	0.9923	1.0000	0.9921	1.0372
direction	0.5619	0.7336	1.0000	0.7866	1.7541

Table 6.4: Per-token metrics model_128 on CSCO Dataset [2]

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.5213	0.6930	0.8472	0.7281	4.5780
time	3.2527	0.5146	0.5538	0.5324	25.8593
event_type	0.7618	0.5722	1.0000	0.7125	2.1420
size	2.1027	0.5763	0.8855	0.6196	8.1883
price	1.0730	0.6643	1.0000	0.7105	2.9240
sign	0.0372	0.9923	1.0000	0.9922	1.0379
direction	0.5139	0.7568	1.0000	0.8050	1.6719

Table 6.5: Per-token metrics model_128 on PCLN Dataset [2]

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.0298	0.6033	0.7274	0.6427	7.6125
time	3.6382	0.4740	0.5173	0.4868	38.0216
event_type	0.7776	0.6316	1.0000	0.7162	2.1763
size	2.5942	0.4634	0.6098	0.4881	13.3856
price	4.9328	0.0000	0.1463	0.0990	138.7621
sign	0.0124	1.0000	1.0000	0.9999	1.0125
direction	0.6512	0.5714	1.0000	0.7308	1.9178

Due to the encoding scheme and training process, out of the 12011 possible vocabulary tokens, only a few values were well-predicted, while more complex fields with greater variation proved difficult for the model. Consequently, the model fell into local minima, learning to predict the "simple tokens" while disregarding others, leading to mode collapse and the inability to predict the tokens essential for accurate mid-price movements, irrespective of model size.

Nonetheless, we successfully replicated market properties related to event distribution and inter-arrival times, and achieved promising results for some models concerning returns.

This represents a significant step forward in LOB modeling, particularly as it demonstrates the potential benefits of linking message generation to a large language model framework. Moreover, it paves the way for further research.

6.2.3 Experiment 3 - Message Injection

The third experiment aimed to address our third research question about testing the model under adverse conditions. To achieve this, we implemented a new framework where we injected the message sequence context with six large market orders, rolled the context window forward, and updated the LOB after each order. Generation then proceeded as usual, allowing us to observe the model's response to a series of large orders. We tested this method on different trained models, injecting large market orders before the start of generation. Below, we report the findings for the `d_model` = 64 model on the CSCO dataset.

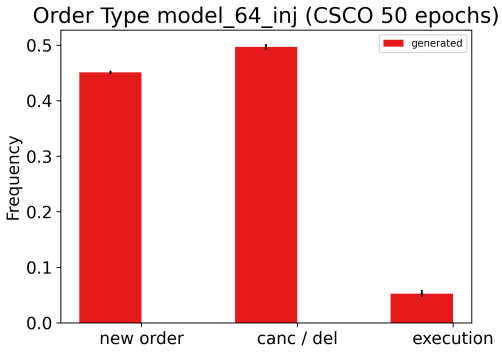

Figure 6.26: CSCO 64 after Injection - Order Type frequency after injection

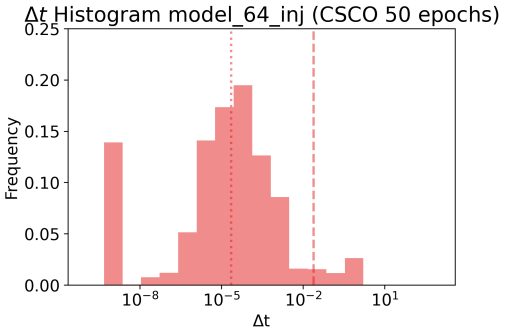

Figure 6.27: CSCO 64 after Injection - Inter-arrival times frequency for generated messages after injection

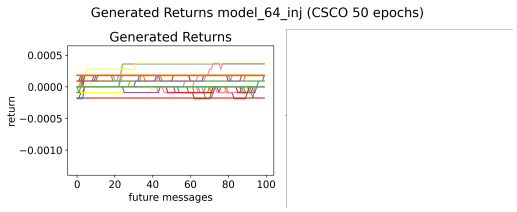

Figure 6.28: CSCO 64 after Injection - Generated Mid-Price Movement Sequences after injection

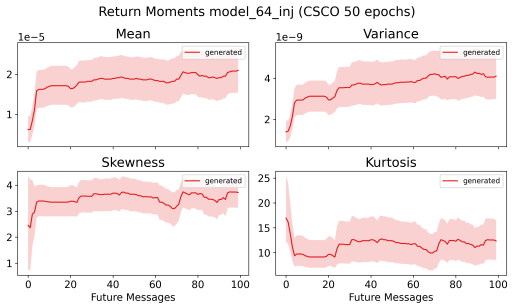

Figure 6.29: CSCO 64 after Injection - Moments of the mid-price after injection

As expected, due to the previously mentioned issues of mode collapse and the general underperformance of the model, the results do not improve. In fact, we observe no significant difference (just a slight bump in variance and movement of returns) on the returns and thus in the prediction of fields (price and quantity) where the new orders should have had an impact. The same issues identified in the earlier experiments remain the main

obstacles to the model's success. Nevertheless, this experiment provides a new method for stress testing LOB message simulation models, which can be applied in future research.

6.3 Mode Collapse

Despite extensive modifications to the model's dimensions, the addition of activation functions, and adjustments to the training loop and learning rate in an attempt to escape local minima, our analysis revealed persistent issues with mode collapse. The models consistently failed to predict all token types with sufficient accuracy to replicate market properties. As shown in Table 6.5 to 6.8, there is a notable difference in performance metrics across the generation of different tokens. Furthermore, as illustrated in the table below, the models encountered only 603 and 256 unique tokens out of the 12,011 possible on the test CSCO and PCLN datasets, respectively, and managed to generate only around 60 unique tokens, with the larger models performing slightly better. This stark reduction in the diversity of generated tokens highlights a significant issue: the model is collapsing into a narrow range of outputs, failing to capture the full complexity and variability of the real market data.

Table 6.6: Overview of unique tokens generated in the test set [2]

Model	**Dataset**	**Actual Unique Tokens**	**Generated Unique Tokens**
null	csco	603	45
16	csco	603	61
32	csco	603	83
64	csco	603	84
128	csco	603	93
null	pcln	256	33
16	pcln	256	52
32	pcln	256	48
64	pcln	256	48
128	pcln	256	62

The fact that the model is generating only a fraction of the unique tokens it has seen during training is a clear indication of mode collapse. In a well-functioning model, we would expect a broad and diverse range of token outputs, reflecting the variety found in the actual data. However, when the model collapses to a limited set of tokens, it suggests that the model is becoming overly confident in a narrow range of outputs, neglecting the broader possibilities that are critical for accurately modeling complex systems like market microstructure. This limitation severely hampers the model's ability to generate realistic sequences that mimic the true variability and dynamics of the market.

6.3.1 Causes & Possible Solutions

We have identified four primary causes of the mode collapse issue and propose potential solutions:

- **Loss Function:** One of the root causes of mode collapse is the equal treatment of all tokens in the loss function, leading the model to prioritize "easy" tokens. Enhancing the loss function by shifting from standard cross-entropy to a custom weighted loss could compel the model to focus more on predicting size and quantity tokens, which are critical for replicating market mid-price fluctuations. Additionally, rebalancing the training set to address token imbalance would improve model performance by ensuring that the model does not overfit to the more frequently occurring "easy" tokens.

- **Data:** Large language models (LLMs) excel at token prediction but require vast amounts of data to effectively learn from various scenarios. Our dataset, covering only two months for each stock, is insufficient for the model to grasp market nuances. As seen in Table 6.3, the test sets for CSCO and PCLN contain only 603 and 256 unique tokens, respectively, with the model predicting even fewer. Expanding the dataset to include more trading days and aggregating different stocks into a single training dataset could enhance performance. Similar to how LLMs can benefit from multilingual training, a larger, more diverse dataset of a wide range of different stocks, across a wide range of days, could help the model learn the differences across stocks, potentially improving its generalization and ability to replicate market conditions.

- **Error Correction:** Understanding the impact of the error correction module is crucial for accurately modeling dynamics. The current error correction method can skew results, significantly affecting the generated message flow. Since error correction is only applied during inference, it can distort the results post-training. A comprehensive redesign of the architecture—one that does not encode reference orders as token groups—could simplify message generation, especially for reference orders outside the context window. This would improve the model's robustness and reduce errors by ensuring that the error correction process does not inadvertently reinforce the mode collapse.

- **Encoding:** The model's failure to generate a sufficient number of unique tokens stems from the large, underrepresented vocabulary, where many tokens are seldom or never seen. Re-encoding entire messages into a latent space —representing each message as a single token or using dedicated tokens for the message and reference—could simplify the model's task by reducing the need to generate a sequence of 22 separate

tokens for each message. Instead, the model would focus on producing a single, coherent latent representation. This type of encoding could have a series of benefits:

- **Capturing Structure and Relationships:** The latent space encoding would allow the model to better capture the underlying structure and relationships within the data, leading to more consistent generation of unique tokens. The model would work with more abstract representations that are less prone to overfitting to specific token sequences.
- **Balancing the Dataset:** Latent space encoding could also help balance the dataset, as coverage of the latent space could be more easily assessed and managed. This would provide deeper insight into training balance and coverage.
- **Simplifying Error Correction:** This approach would simplify error correction by enabling adjustments based on the proximity of latent representations rather than exact token matches, resulting in more stable and accurate message generation.
- **Generalizing Across Contexts:** Encoding messages in a latent space could enhance the model's ability to generalize across different market contexts. Since the latent representation captures the context and relationships between tokens, the model could learn market nuances more effectively, leading to more robust predictions that better reflect the complexities of market microstructure.
- **Integrating Additional Features:** Latent space encoding would allow for the integration of additional features, such as market sentiment or macroeconomic indicators, without increasing token complexity. This could enhance the model's ability to generate realistic market behaviors and avoid the pitfalls of mode collapse.

To create this latent space, an encoder model, such as a transformer-based encoder or variational autoencoder, could be employed. This encoder would map each message into a compact, high-dimensional latent vector that encapsulates the essential information needed for accurate market simulation. The generative model would then use these latent representations to produce the next sequence of messages, reducing task complexity and mitigating the risks of mode collapse.

Chapter 7

Conclusion

The key motivation behind this thesis was to explore how modeling limit order books (LOB) using AI, particularly through a large language model (LLM)-inspired approach, could advance our understanding and simulation of financial markets. Accurate modeling of LOBs is crucial as it plays a vital role in understanding market microstructure, price formation, and liquidity, which are all essential for both academic research and practical applications in trading and risk management. By integrating AI with AI techniques, we aim to push the boundaries of traditional models and create more dynamic and realistic simulations of market behaviors, laying the groundwork for future research in this domain.

To achieve this, in Chapter 3, we asked ourselves three questions:

- How can we benchmark the current reference model on our dataset and what evaluation metrics can we use?
- What model architecture allows for the best performance?
- How do adverse market condition affect the message flow simulation?

In Chapter 6, we designed a series of tests aimed at addressing these research-driving questions.

Our first experiment focused on benchmarking the current reference model, often referred to as the "Null model," on our dataset. We employed a variety of evaluation metrics, including perplexity, distribution comparison, and mid-price return analysis, to assess the model's performance. The results revealed shortcomings, particularly in the model's ability to generate a diverse range of tokens and accurately replicate the market's dynamic properties. This highlighted the limitations of the "Null model" and set the stage for further experimentation.

To answer the second research question, we explored different model architectures in our second experiment. We introduced modifications such as adding activation functions, adjusting the model's dimensionality, and implementing latent space encoding to improve

performance. While these changes led to some improvements, especially in event type prediction and inter-arrival times, the models continued to struggle with generating realistic price and quantity variations. The analysis suggested that mode collapse and local minima were significant issues, preventing the models from fully capturing the complexity of market microstructure.

The third experiment addressed the impact of large orders in message flow simulation. By injecting large market orders into the sequence context and observing the model's response, we aimed to evaluate the robustness of our models under adverse conditions. However, the persistent issues identified in the first two experiments, particularly mode collapse, limited the effectiveness of this stress testing. The results underscored the need for further refinements in model architecture and training techniques to enhance the reliability of market simulations.

The three experiments revealed that we succeeded in creating a model capable of accurately predicting event types and timing, with some degree of variance in returns. However, the model also exhibited a significant issue with mode collapse. Alongside these findings, we developed a method for stress-testing models, which helped to uncover the critical challenge of mode collapse. Importantly, we have also proposed potential solutions to address this issue, paving the way for continued research in the right direction.

7.1 Limitations

Several limitations influenced the outcomes of this research. One of the primary challenges was the encoding structure used in the model, which treated messages as sequences of individual tokens. This approach contributed to the model's difficulty in generating diverse and coherent outputs, as it struggled to effectively represent the complexity of market data through this tokenized format. Additionally, the use of a standard loss function may have led the model to prioritize easier tokens, exacerbating the issue of mode collapse. A more customized loss function that better reflects the varying importance of different tokens could have potentially mitigated this problem.

Another significant limitation was the dataset itself. The data was limited to only two months per stock, which restricted the model's ability to learn the full spectrum of market behaviors. This lack of data diversity likely hindered the model's ability to generalize and perform well across different market scenarios. Moreover, limited computational resources posed a constraint on the scope of our experiments. With more powerful computing capabilities, we could have explored larger models, increased batch sizes, and extended training durations, all of which might have led to better model performance.

7.2 Future Work

Looking ahead, several key areas for future research have emerged from this thesis. First and foremost, acquiring a more extensive and diverse dataset is crucial. Expanding the dataset to cover a broader range of stocks and longer time periods would likely improve the model's ability to generalize and enhance its accuracy in simulating market behaviors. Additionally, exploring more advanced model architectures presents another promising avenue for future work. Techniques such as transformers with enhanced encoding strategies or hybrid models that combine generative approaches with reinforcement learning could provide significant improvements in model performance.

Further investigation into latent space encoding is also recommended. This approach could help address the issue of mode collapse by improving the diversity of generated tokens, leading to more realistic and varied market simulations. Another area for future exploration is the refinement of the error correction mechanism. Enhancing this component to be more stable and accurate, particularly under varying market conditions, would strengthen the model's robustness and reliability.

Finally, integrating additional features into the model, such as market sentiment or macroeconomic indicators, could offer a more comprehensive understanding of market dynamics. By incorporating these elements, the model could produce more informed and accurate predictions, ultimately leading to a more sophisticated and realistic simulation of market microstructure.

7.3 Final Thoughts

This thesis has highlighted both the potential and the challenges of using AI and LLM-inspired approaches to model limit order books and simulate market microstructure. While significant challenges remain, particularly in addressing mode collapse and enhancing model generalization, the progress made in predicting event types, timing, and some aspects of price movement demonstrates that this is a promising direction for future research. By continuing to refine our models and methodologies, we can develop generative systems that more accurately reflect market behaviors, offering valuable tools for financial analysis, trading strategies, and academic research. The work done here is a step toward that goal, and it lays the foundation for future advancements in the field.

The integration of AI with traditional financial models offers a glimpse into a future where market simulations are not only more accurate but also more adaptable to the ever-changing landscape of global finance. As technology continues to evolve, so too does our ability to capture the nuances of the market, opening the door to innovations that could revolutionize trading strategies, risk management, and economic research.

As physicist Niels Bohr wisely remarked: *"Prediction is very difficult, especially if*

it's about the future." Through the course of this work, I have come to appreciate the truth of this statement. Predicting financial markets is indeed fraught with uncertainty and complexity. Yet, it is precisely this challenge that makes the endeavor so vital and compelling.

Bibliography

[1] Huggingface SSM. Introduction to space state models (ssm). URL https://huggingface.co/blog/lbourdois/get-on-the-ssm-train.

[2] Candidate Number: FNDC6. Language model for limit order book message simulation. Master's thesis, University College London, 2024.

[3] Andrea Coletta, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch. 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*, ICAIF '22. ACM, October 2022. doi:10.1145/3533271.3561753. URL http://dx.doi.org/10.1145/3533271.3561753.

[4] Peer Nagy, Sascha Frey, Silvia Sapora, Kang Li, Anisoara Calinescu, Stefan Zohren, and Jakob Foerster. 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. 2023.

[5] Machine Learning. Ibm machine learning definition. URL https://www.ibm.com/cloud/learn/machine-learning.

[6] Deep Learning. Ibm deep learning definition. URL https://www.ibm.com/cloud/learn/deep-learning.

[7] Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling, 2023. URL https://arxiv.org/abs/2208.04933.

[8] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. Hippo: Recurrent memory with optimal polynomial projections, 2020. URL https://arxiv.org/abs/2008.07669.

[9] Rama Cont, Mihai Cucuringu, Jonathan Kochems, and Felix Prenzel. Limit order book simulation with generative adversarial networks. *SSRN Electronic Journal*, 01 2023. doi:10.2139/ssrn.4512356.

[10] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. Abides: Towards high-fidelity market simulation for ai research, 2019. URL https://arxiv.org/abs/1904.12066.

[11] Jean-Philippe Bouchaud. Chapter 7 - agent-based models for market impact and volatility. In Cars Hommes and Blake LeBaron, editors, *Handbook of Computational Economics*, volume 4 of *Handbook of Computational Economics*, pages 393–436. Elsevier, 2018. doi:https://doi.org/10.1016/bs.hescom.2018.02.002. URL https://www.sciencedirect.com/science/article/pii/S1574002118300029.

[12] Konark Jain, Nick Firoozye, Jonathan Kochems, and Philip Treleaven. Limit order book simulations: A review, 2024. URL https://arxiv.org/abs/2402.17359.

[13] Heting Yan, Giray Okten, Lingjiong Zhu, and Jinfeng Zhang. *Deep Learning for Limit Order Book Trading and Mid-Price Movement Prediction*. PhD thesis, 2020. AAI27740225.

[14] Sascha Frey, Kang Li, Peer Nagy, Silvia Sapora, Chris Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. Jax-lob: A gpu-accelerated limit order book simulator to unlock large scale reinforcement learning for trading, 2023. URL https://arxiv.org/abs/2308.13289.

[15] 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. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[16] LOB S5 Github Repo. Lob s5 github repo. URL https://github.com/peernagy/LOBS5.

[17] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

[18] LOBSTER Dataset. Lobster dataset. URL https://lobsterdata.com/index.php.

[19] L2 vs L3 data. Tick-level order book data. URL https://www.coinapi.io/blog/coinapi-tick-level-order-book-data#.

[20] Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. Toward a theory of tokenization in llms, 2024. URL https://arxiv.org/abs/2404.08335.

[21] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Appendix A

Appendix

A.1 Model Metrics

A.1.1 csco d_model = 16

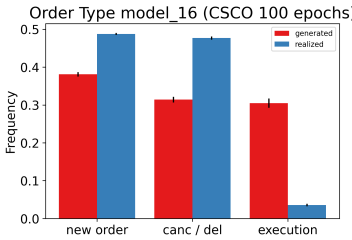

Figure A.1: Order Type Frequency

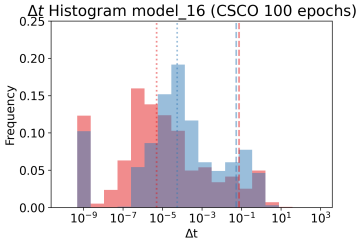

Figure A.2: Inter-arrival Times Frequency

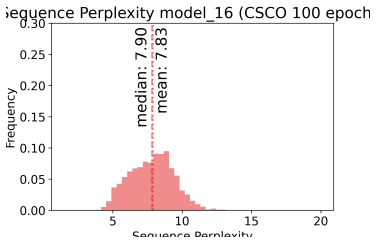

Figure A.3: Message sequence perplexity

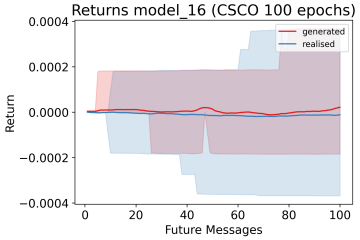

Figure A.4: Average movement of mid price

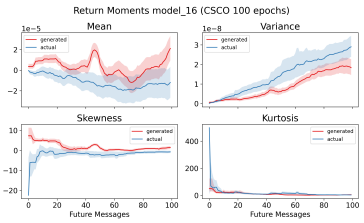

Figure A.5: Moments of the mid-price returns

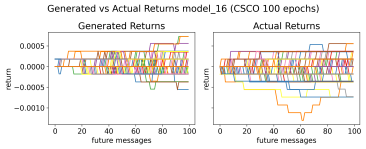

Figure A.6: Mid-Price Movement Sequences

Table A.1: Per-token metrics CSCOC model_16

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.6187	0.6693	0.8394	0.7072	5.0467
time	3.3076	0.5146	0.5544	0.5301	27.3199
event_type	0.7960	0.5365	1.0000	0.6997	2.2167
size	2.5283	0.4822	0.8321	0.5264	12.5326
price	1.1620	0.6256	1.0000	0.6779	3.1965
sign	0.0365	0.9923	1.0000	0.9921	1.0372
direction	0.5619	0.7336	1.0000	0.7866	1.7541

A.1.2 CSCO D_MODEL = 32

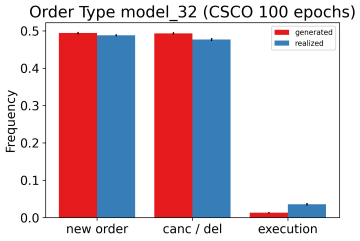

Figure A.7: Order Type Frequency

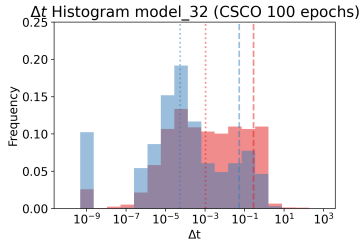

Figure A.8: Inter-arrival Times Frequency

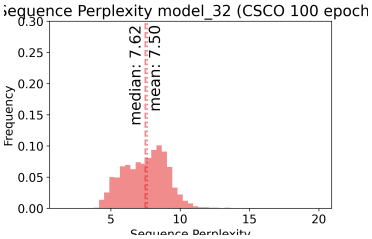

Figure A.9: Message sequence perplexity

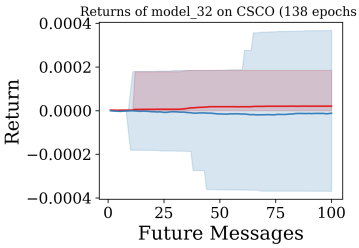

Figure A.10: Average movement of mid price

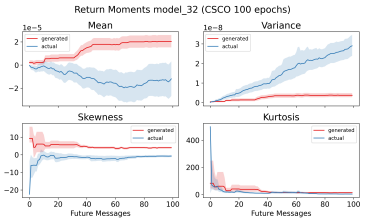

Figure A.11: Moments of the mid-price returns

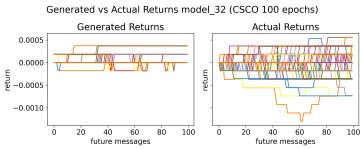

Figure A.12: Mid-Price Movement Sequences

Table A.2: Per-token metrics CSCO model_32

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.5649	0.6818	0.8438	0.7177	4.7822
time	3.2902	0.5165	0.5557	0.5312	26.8491
event_type	0.7882	0.5472	1.0000	0.6996	2.1994
size	2.2517	0.5407	0.8588	0.5807	9.5038
price	1.1273	0.6449	1.0000	0.6884	3.0873
sign	0.0367	0.9923	1.0000	0.9925	1.0373
direction	0.5447	0.7336	1.0000	0.7933	1.7241

A.1.3 CSCO D_MODEL = 64

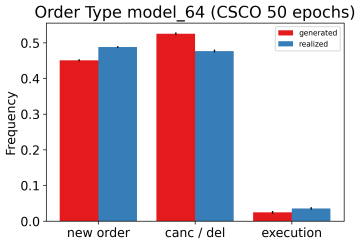

Figure A.13: Order Type Frequency

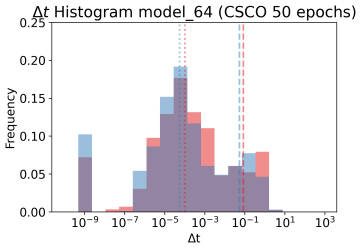

Figure A.14: Inter-arrival Times Frequency

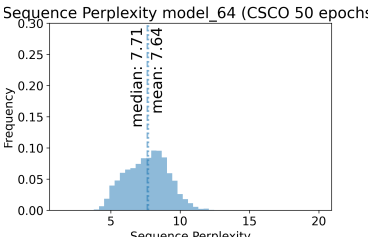

Figure A.15: Message sequence perplexity

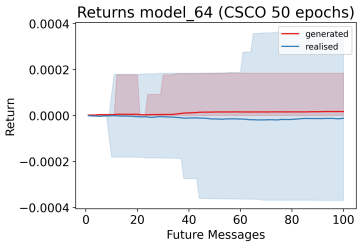

Figure A.16: Average movement of mid price

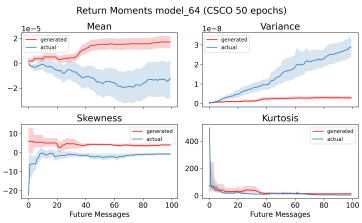

Figure A.17: Moments of the mid-price returns

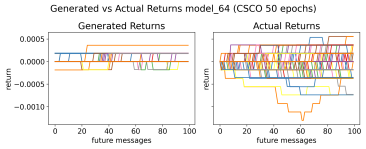

Figure A.18: Mid-Price Movement Sequences

Table A.3: Per-token metrics CSCO model_64

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.5718	0.6807	0.8436	0.7168	4.8152
time	3.2894	0.5165	0.5557	0.5305	26.8271
event_type	0.7976	0.5348	1.0000	0.6948	2.2201
size	2.2945	0.5293	0.8588	0.5747	9.9196
price	1.1251	0.6594	0.9976	0.6938	3.0806
sign	0.0375	0.9923	1.0000	0.9923	1.0382
direction	0.5414	0.7413	1.0000	0.7965	1.7184

A.1.4 CSCO D_MODEL = 128

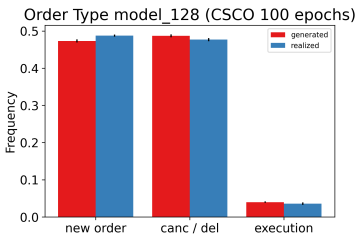

Figure A.19: Order Type Frequency

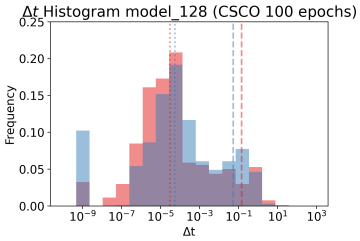

Figure A.20: Inter-arrival Times Frequency

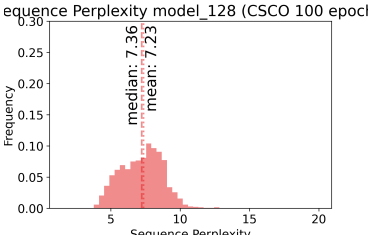

Figure A.21: Message sequence perplexity

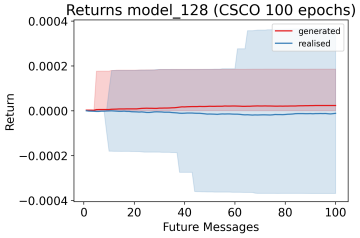

Figure A.22: Average movement of mid price

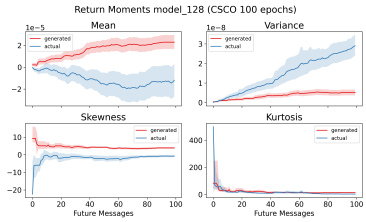

Figure A.23: Moments of the mid-price returns

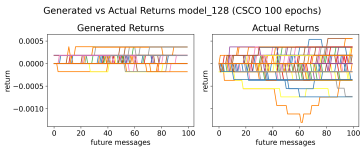

Figure A.24: Mid-Price Movement Sequences

Table A.4: Per-token metrics CSCO model_128

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.5213	0.6930	0.8472	0.7281	4.5780
time	3.2527	0.5146	0.5538	0.5324	25.8593
event_type	0.7618	0.5722	1.0000	0.7125	2.1420
size	2.1027	0.5763	0.8855	0.6196	8.1883
price	1.0730	0.6643	1.0000	0.7105	2.9240
sign	0.0372	0.9923	1.0000	0.9922	1.0379
direction	0.5139	0.7568	1.0000	0.8050	1.6719

A.1.5 CSCO "Null" Model

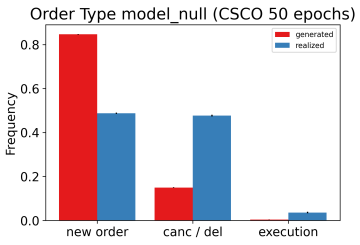

Figure A.25: Order Type Frequency

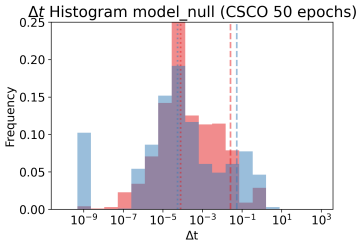

Figure A.26: Inter-arrival Times Frequency

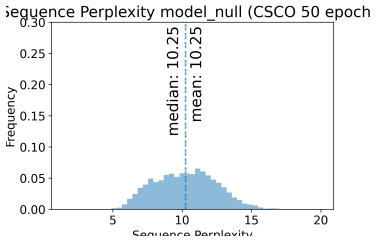

Figure A.27: Message sequence perplexity

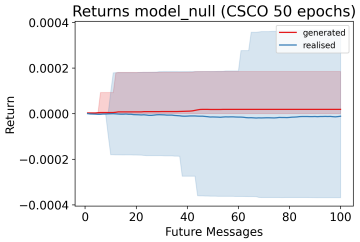

Figure A.28: Average movement of mid price

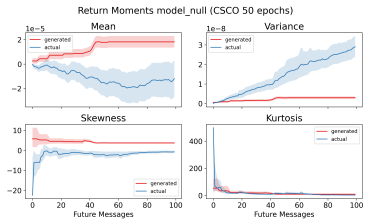

Figure A.29: Moments of the mid-price returns

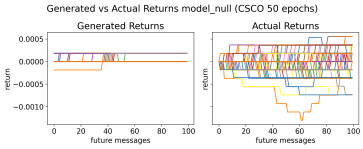

Figure A.30: Mid-Price Movement Sequences

Table A.5: Per-token metrics CSCO model_null

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	1.8400	0.6061	0.8244	0.6569	6.2968
time	3.5147	0.4910	0.5488	0.5044	33.6058
event_type	0.8671	0.4831	1.0000	0.6730	2.3799
size	3.2287	0.3244	0.7481	0.3708	25.2470
price	1.5204	0.5556	0.9928	0.6004	4.5739
sign	0.0418	0.9871	1.0000	0.9898	1.0427
direction	0.7546	0.5154	1.0000	0.6785	2.1267

A.1.6 PCLN D_MODEL = 16

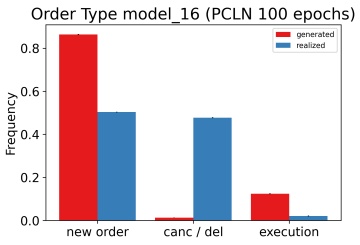

Figure A.31: Order Type Frequency

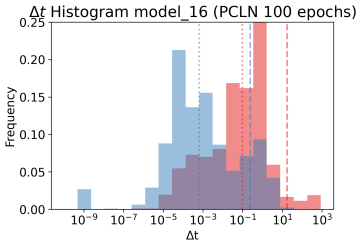

Figure A.32: Inter-arrival Times Frequency

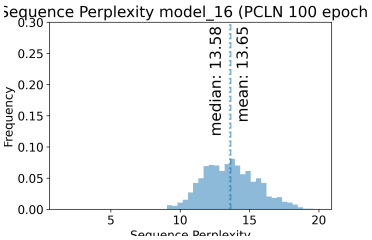

Figure A.33: Message sequence perplexity

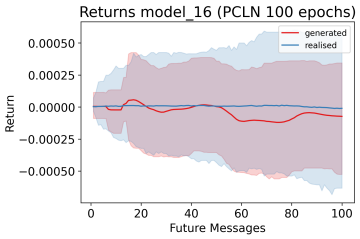

Figure A.34: Average movement of mid price

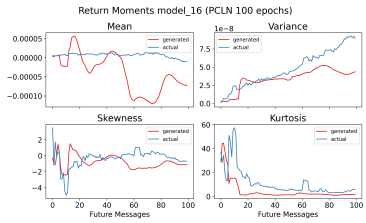

Figure A.35: Moments of the mid-price returns

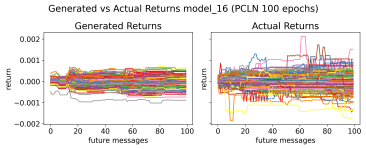

Figure A.36: Mid-Price Movement Sequences

Table A.6: Per-token metrics PCLN model_16

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.1432	0.5987	0.7210	0.6324	8.5266
time	3.7508	0.4480	0.5145	0.4651	42.5554
event_type	0.7852	0.6211	1.0000	0.7046	2.1928
size	3.0566	0.4573	0.5610	0.4690	21.2547
price	5.0480	0.0122	0.1707	0.0938	155.7175
sign	0.0160	1.0000	1.0000	0.9997	1.0161
direction	0.6473	0.6134	1.0000	0.7373	1.9104

A.1.7 PCLN D_MODEL = 32

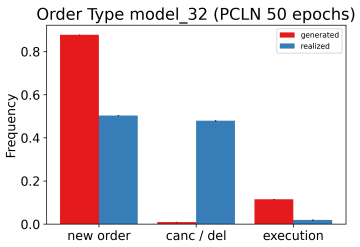

Figure A.37: Order Type Frequency

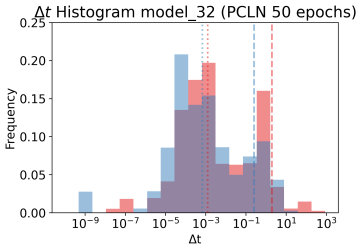

Figure A.38: Inter-arrival Times Frequency

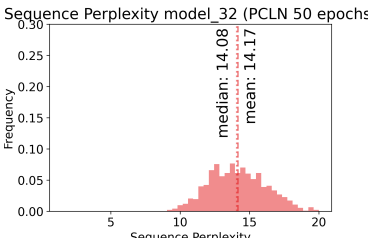

Figure A.39: Message sequence perplexity

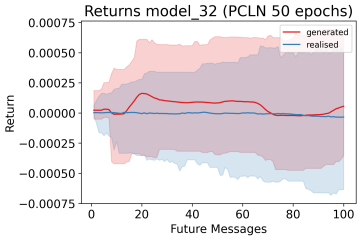

Figure A.40: Average movement of mid price

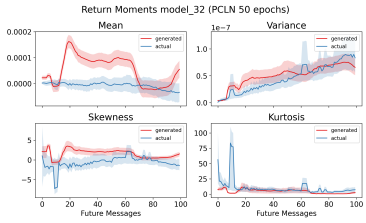

Figure A.41: Moments of the mid-price returns

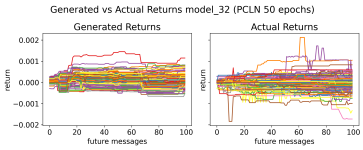

Figure A.42: Mid-Price Movement Sequences

Table A.7: Per-token metrics PCLN model_32

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.1665	0.5978	0.7174	0.6292	8.7276
time	3.7278	0.4682	0.5202	0.4746	41.5862
event_type	0.8033	0.5474	1.0000	0.6939	2.2328
size	3.0942	0.4512	0.5549	0.4565	22.0703
price	5.1841	0.0000	0.1220	0.0834	178.4173
sign	0.0909	0.9839	1.0000	0.9791	1.0951
direction	0.6431	0.6387	1.0000	0.7388	1.9024

A.1.8 PCLN D_MODEL = 64

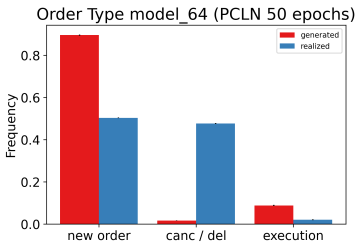

Figure A.43: Order Type Frequency

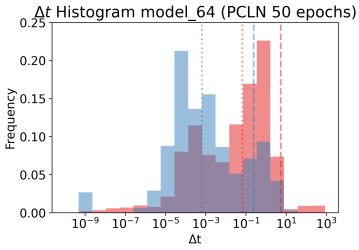

Figure A.44: Inter-arrival Times Frequency

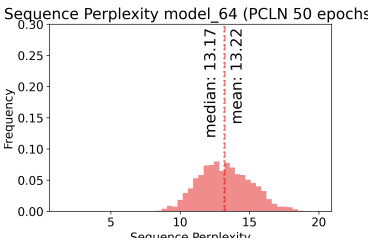

Figure A.45: Message sequence perplexity

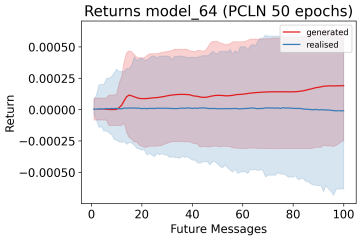

Figure A.46: Average movement of mid price

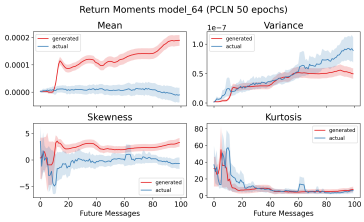

Figure A.47: Moments of the mid-price returns

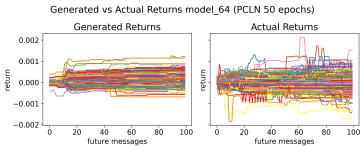

Figure A.48: Mid-Price Movement Sequences

Table A.8: Per-token metrics PCLN model_64

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.1286	0.6078	0.7192	0.6390	8.4034
time	3.6997	0.4711	0.5173	0.4813	40.4335
event_type	0.7789	0.6211	1.0000	0.7087	2.1790
size	3.0752	0.4573	0.5793	0.4709	21.6548
price	5.0799	0.0000	0.0976	0.0897	160.7581
sign	0.0162	1.0000	1.0000	0.9998	1.0164
direction	0.6220	0.6387	1.0000	0.7484	1.8627

A.1.9 PCLN D_MODEL = 128

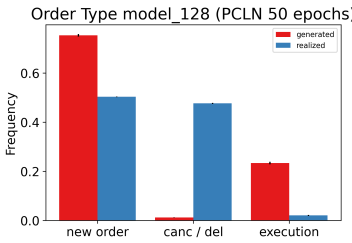

Figure A.49: Order Type Frequency

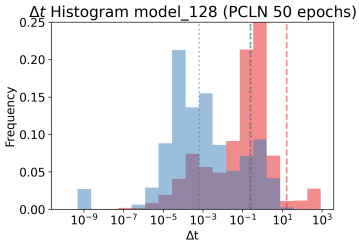

Figure A.50: Inter-arrival Times Frequency

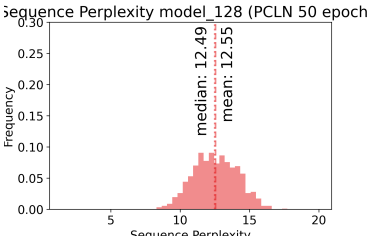

Figure A.51: Message sequence perplexity

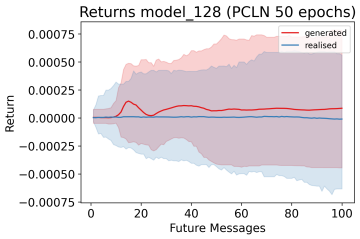

Figure A.52: Average movement of mid price

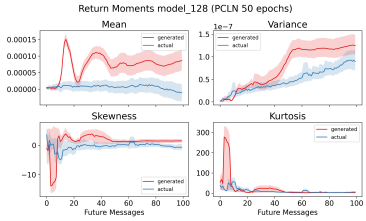

Figure A.53: Moments of the mid-price returns

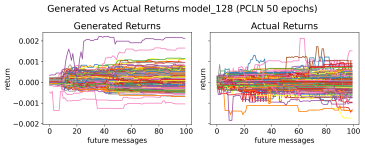

Figure A.54: Mid-Price Movement Sequences

Table A.9: Per-token metrics PCLN model_128

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.0298	0.6033	0.7274	0.6427	7.6125
time	3.6382	0.4740	0.5173	0.4868	38.0216
event_type	0.7776	0.6316	1.0000	0.7162	2.1763
size	2.5942	0.4634	0.6098	0.4881	13.3856
price	4.9328	0.0000	0.1463	0.0990	138.7621
sign	0.0124	1.0000	1.0000	0.9999	1.0125
direction	0.6512	0.5714	1.0000	0.7308	1.9178

A.1.10 PCLN "Null" Model

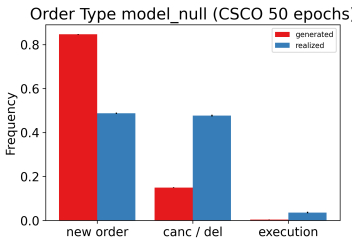

Figure A.55: Order Type Frequency

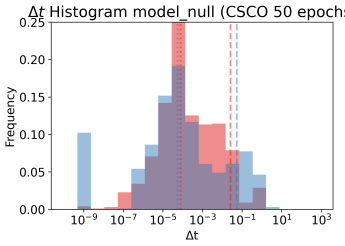

Figure A.56: Inter-arrival Times Frequency

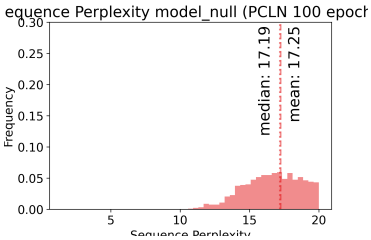

Figure A.57: Message sequence perplexity

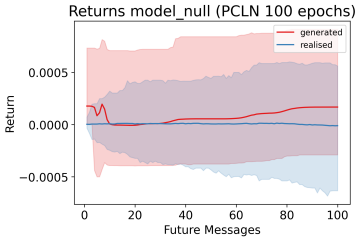

Figure A.58: Average movement of mid price

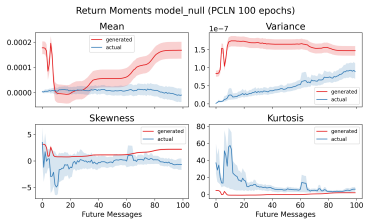

Figure A.59: Moments of the mid-price returns

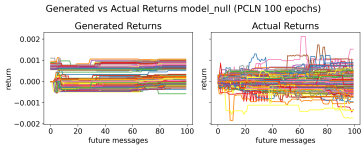

Figure A.60: Mid-Price Movement Sequences

Table A.10: Per-token metrics PCLN model_null

Field	Cross Entropy	Accuracy	Top-10 Accuracy	Cosine Similarity	Perplexity
all	2.3716	0.5543	0.6938	0.5946	10.7145
time	4.0396	0.3931	0.4682	0.4125	56.8040
event_type	0.7812	0.6316	1.0000	0.7136	2.1841
size	3.6119	0.4573	0.5122	0.4567	37.0381
price	5.0961	0.0122	0.1098	0.0872	163.3776
sign	0.6859	0.6935	1.0000	0.7139	1.9855
direction	0.6904	0.5210	1.0000	0.7104	1.9945