The Journal of Open Source Software

MeatPy: A Python Framework for Limit Order Book Reconstruction and Analysis

Vincent Grégoire [1*¶] **and Charles Martineau** [2*]

1 Department of Finance, HEC Montréal, Canada **2** Rotman School of Management and UTSC Management, University of Toronto, Canada ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 26 July 2025
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

MeatPy is a Python framework specifically developed for processing and analyzing financial market data, with a primary focus on reconstructing and examining limit order books. Limit order books are central to financial markets, recording all outstanding buy and sell orders for securities at different price levels. MeatPy offers robust support for high-frequency trading data formats, notably the Nasdaq ITCH standard. With an event-driven object-oriented architecture and strong type safety via modern Python typing, MeatPy provides a flexible environment for market data analysis.

Statement of Need

High-frequency financial markets now operate at sub-millisecond time scales, with individual order placements, cancellations, and executions occurring in nanoseconds. To understand market microstructure, liquidity provision, and price formation, researchers increasingly rely on historical order book data that record every message sent by the exchange. Several exchanges, including Nasdaq, IEX, and the Australian Stock Exchange and Chi-X Australia, make their high-frequency data feeds available for academic research, often at no cost or a reduced fee.[1] These feeds offer unprecedented granularity, capturing not only trades and their direction, but also order placements, cancellations, halts, circuit breakers, and specialized exchange-specific order types. As such, they have been used to study a wide range of market microstructure-related questions (Comerton-Forde et al., 2019; Grégoire & Martineau, 2022.; see, e.g., O'Hara et al., 2014; Shkilko & Sokolov, 2020)

Despite their common conceptual structure, exchange data feeds differ in message formats, order type definitions, and exchanges rules which affect processing logic. A single day of Nasdaq TotalView ITCH data can exceed ten gigabytes in compressed binary form and contain billions of messages. Processing these heterogeneous and high-volume data streams has traditionally relied on writing custom scripts and programs which are kept private by the research team (see, e.g. Clark McGehee, 2013; Gai et al., 2013) or on using third-party processing tools or platforms such as LOBSTER by Huang & Polak (2011).[2] This situation limits reproducibility and slows research progress.

MeatPy addresses these challenges by providing an open-source Python framework for parsing and analyzing high-frequency financial market data. It reconstructs full limit order books

[1] Nasdaq data can be obtained through their "Academic Waiver Policy"; IEX provides free historical data, called HIST, on a T+1 basis on their website; ASX and Chi-X Australia data can be accessed via SIRCA by their academic subscribers

[2] LOBSTER is available to academic institiutions as a flat-fee online service.

DRAFT

from raw feed data, supports multiple exchange-specific message formats,[3] and leverages modern Python features such a generators, context managers, and type annotations to improve reliability and developer productivity. By abstracting away the complexity of heterogeneous feed formats and offering efficient data processing primitives, MeatPy enables researchers to focus on designing and executing market microstructure analyses rather than building low-level data engineering infrastructure.

Challenges in Processing Limit Order Book Data

Processing limit order book (LOB) data poses significant technical and conceptual challenges compared to working with more conventional tabular financial datasets. While stock prices, trades, or aggregated quotes are typically available as simple time series, raw exchange feeds record every event affecting the state of the order book, often at sub-millisecond intervals. Because these events depend on one another, the full state of the market at any given time must be inferred by dynamically applying these events in sequence rather than reading a single row of static information.

Commercial data feeds such as Nasdaq TotalView ITCH amplify this challenge by optimizing bandwidth through highly efficient message formats. For example, a new order addition may specify a stock symbol, price, and size as in the first row of Table 1. Subsequent messages referencing the same order—such as cancellations or executions omit the stock symbol and price entirely, providing only the unique order identifier. This design is optimized for low-latency transmission but complicates downstream processing: the consumer must maintain a mapping between active order identifiers and their associated attributes to correctly interpret later modifications or removals. Without reconstructing and maintaining this state, it is impossible to know, for example, which stock an execution message pertains to or whether it affects liquidity on the bid or ask side.

Table 1: Sample Nasdaq ITCH Messages

Timestamp	Message	Bid/Ask	Shares	Price	Stock	Ref Number
53435.759668667	Add	Ask	100	120.00	AAPL	335531633
53437.259368363	Add	Bid	310	119.00	AAPL	335531640
53447.159138313	Cancel		50			335531633
53460.229122363	Exec		50			335531640
53461.119111364	Add	Bid	75	159.00	IBM	401304823
55591.122214484	ExecHid	Ask	100	119.90	AAPL	(hidden)

Another major challenge is producing consistent snapshots of the order book. Researchers frequently need to analyze market depth or price formation at specific points in time, yet order book data evolves through a continuous stream of incremental updates rather than periodic summaries. Building a snapshot at time t requires processing all preceding messages to construct the state as it would have been observed in real time, and taking a snapshot before the limit order book is updated when a message arrives after t. MeatPy addresses this by implementing the observer pattern that allows users to register callbacks for specific events, such as order book updates or trade executions, enabling them to capture snapshots at arbitrary points in time without needing to manually manage the underlying state.

The sheer size of the input files also creates practical difficulties in terms of data storage, memory usage, and processing speed. Unlike pre-aggregated quote data, raw LOB feeds record every micro-level event, and processing them often requires parallelization, chunked reading, or specialized binary parsing strategies. These technical complexities make working with raw

[3] MeatPy currently supports TotalView-ITCH versions 4.1 and 5.0 used by exchanges on the INET platform which includes most Nasdaq-operated equity exchanges. Support for IEX DEEP+ is under developement.

DRAFT

limit order book data a substantial barrier to entry for researchers and practitioners who lack specialized tooling. While MeatPy does not implement parallel processing, it is designed to filter and process messages in a memory-efficient manner, allowing users to split the processing of large files across multiple MeatPy instances.

Finally, the lack of standardized APIs or libraries for limit order book reconstruction means that researchers often need to implement custom solutions for each exchange format they encounter. This leads to duplicated effort, inconsistent implementations, and difficulties in sharing code across research teams. MeatPy aims to solve this problem by providing a unified framework that abstracts away the differences between exchange formats while maintaining the flexibility needed for specific use cases.

These challenges motivated the design of MeatPy, which prioritizes flexibility and ease of use over raw computational efficiency. By sacrificing some low-level performance optimizations in favor of transparent abstractions and modern Python type safety, MeatPy allows researchers to quickly focus on economic questions and market microstructure insights. This design choice lowers the barrier to entry for academic and industry users who need accessible tools but do not wish to invest significant effort in implementing and maintaining custom parsing and reconstruction pipelines.

Architecture

MeatPy is built around an event-driven architecture that allows efficient and flexible processing of high-frequency financial market data. At its core, the framework uses format-specific parsers to convert raw exchange data feeds, such as Nasdaq TotalView ITCH, into standardized event objects. These event objects encapsulate individual market messages, including order additions, cancellations, executions, and trading status updates.

Once parsed, events are streamed to the order book engine, which reconstructs a complete and dynamically updated view of the limit order book across all price levels. This engine maintains the state of outstanding buy and sell orders and updates it incrementally as new messages arrive, enabling researchers to analyze liquidity and price dynamics with full market depth. This engine implements the observer pattern, allowing multiple event handlers to subscribe to market events such as order book updates, trade executions, and cancellations. This lets users define custom logic for handling these events, such as recording order book snapshots or other event details to disk.

The data flows through a pipeline that can be easily customized and extended to accommodate new data sources or processing requirements. For the most common use cases, the pipeline consists of the components shown in Figure 1.

DRAFT

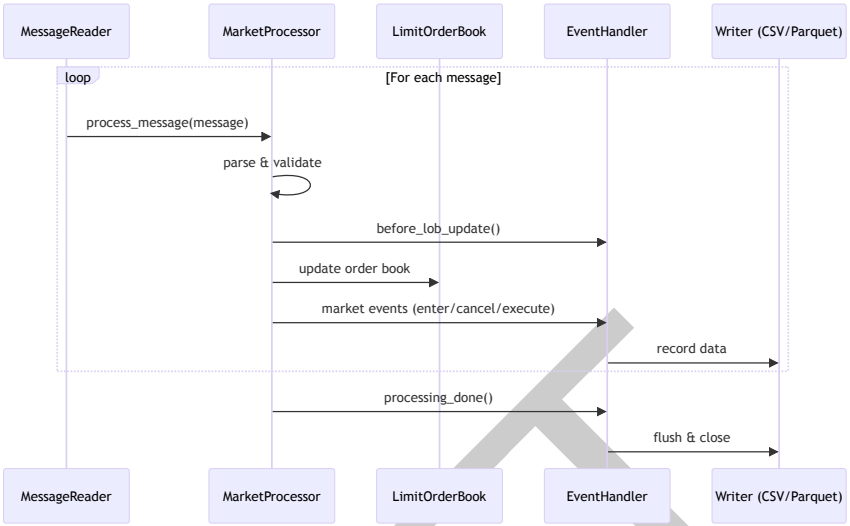

Figure 1: Sequence Diagram of MeatPy Processing Pipeline. Messages are read and parsed by the MessageReader, which passes them to the MarketProcessor. The MarketProcessor validates the messages, updates the LimitOrderBook, and notifies the EventHandler of market events. The EventHandler records data to a Writer, which can output to CSV or Parquet formats. At the end of processing, the Writer flushes the remaining in-memory data and closes the output files.

This object-oriented modular design separates data parsing, message representation, and order book reconstruction, making it straightforward to support additional exchange formats or extend functionality. MeatPy doesn't require any external dependencies beyond Python's standard library, except for the `pyarrow` package required for efficient data writing to CSV and Parquet formats. This ensures that MeatPy can be easily deployed in diverse environments, from local machines to high-performance computing clusters.

Research Applications

MeatPy has supported several academic studies, including the investigation of earnings news dissemination and its impact on stock prices (Grégoire & Martineau, 2022), the analysis of market quality under different fee structures and tick sizes (Comerton-Forde et al., 2019), and visualization techniques for exploring high-frequency trading data (Yaali et al., 2022).

Acknowledgements

Seoin Kim and Javad YaAli provided valuable research assistance on the project. MeatPy development benefited from the financial support of IVADO GRANT #PRF-2019-3059794586.

References

Clark McGehee, L. E. H. (2013). An Object-Oriented Library for Real-Time Processing of NASDAQ Order Book Data. *Journal of Computer Engineering & Information Technology*, *02*(01). https://doi.org/10.4172/2324-9307.1000101

Comerton-Forde, C., Grégoire, V., & Zhong, Z. (2019). Inverted fee structures, tick size, and market quality. *Journal of Financial Economics*, *134*(1), 141–164. https://doi.org/10.1016/j.jfineco.2019.03.005

DRAFT

Gai, J., Choi, D. J., O'Neal, D., Ye, M., & Sinkovits, R. S. (2013). Fast construction of nanosecond level snapshots of financial markets. *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, 1–4. https://doi.org/10.1145/2484762.2484825

Grégoire, V., & Martineau, C. (2022). How is earnings news transmitted to stock prices? *Journal of Accounting Research*, *60*(1), 261–297. https://doi.org/10.1111/1475-679X.12394

Huang, R., & Polak, T. (2011). LOBSTER: Limit Order Book Reconstruction System. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1977207

O'Hara, M., Yao, C., & Ye, M. (2014). What's Not There: Odd Lots and Market Data. *The Journal of Finance*, *69*(5), 2199–2236. https://doi.org/10.1111/jofi.12185

Shkilko, A., & Sokolov, K. (2020). Every Cloud Has a Silver Lining: Fast Trading, Microwave Connectivity, and Trading Costs. *The Journal of Finance*, *75*(6), 2899–2927. https://doi.org/10.1111/jofi.12969

Yaali, J., Grégoire, V., & Hurtut, T. (2022). HFTViz: Visualization for the exploration of high frequency trading data. *Information Visualization*, *21*(2), 182–193. https://doi.org/10.1177/14738716211064921

DRAFT