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: 10.21105/joss.10480

Software

- Review
- Repository
- Archive

Editor: Blake Rayfield

Reviewers:

- @robertmartin8
- @NicolasDuvernois

Submitted: 26 July 2025
Published: 16 June 2026

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 (see, e.g., Comerton-Forde et al., 2019; Grégoire & Martineau, 2022; O'Hara et al., 2014; Shkilko & Sokolov, 2020).

Despite their common conceptual structure, exchange data feeds differ in message formats, order type definitions, and exchange 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. 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 from raw feed data, supports multiple exchange-specific message formats,[2] and leverages modern Python features such as 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

[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] MeatPy currently supports all Nasdaq ITCH versions from 2.0 to 5.0 used by exchanges on the INET platform, which includes most Nasdaq-operated equity exchanges, as well as IEX DEEP.

data engineering infrastructure.

State of the Field

Several tools exist for processing limit order book data, but none provide an open-source, extensible Python framework for full order book reconstruction from raw exchange feeds. LOBSTER (Huang & Polak, 2011) is the most widely-used platform, reconstructing order books from Nasdaq ITCH data as an online service. However, its closed-source, commercial nature limits extensibility and reproducibility: researchers cannot inspect the reconstruction logic, adapt it to new exchange formats, or deploy it on their own infrastructure. Other published implementations (Clark McGehee, 2013; Gai et al., 2013) were not released as reusable open-source libraries, leaving nothing to contribute to or build upon.

In practice, most research teams rely on custom, unpublished scripts for order book reconstruction, leading to duplicated effort, inconsistent implementations, and barriers to reproducibility. Prior to MeatPy, co-author Martineau maintained a proprietary SAS-based solution that proved difficult to extend due to SAS's limited support for object-oriented programming and whose proprietary licensing complicated deployment on high-performance computing facilities.

MeatPy was built as a new framework rather than a contribution to an existing project because no open-source Python library for limit order book reconstruction existed. By providing a complete, open pipeline covering binary feed parsing, order book reconstruction, and event-driven analysis, MeatPy fills this gap and enables reproducible market microstructure research.

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

Timestamp	Message	Bid/Ask	Shares	Price	Stock	Ref Number
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 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.

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.

Software Design

MeatPy's design reflects a deliberate trade-off between computational efficiency and accessibility. We chose Python over lower-level languages to lower the barrier to entry for researchers, accepting some performance cost in exchange for readability, ease of modification, and compatibility with the scientific Python ecosystem.

The framework is built around an event-driven architecture. Format-specific parsers convert raw exchange data feeds into standardized event objects encapsulating order additions, cancellations, executions, and trading status updates. These events are streamed to the order book engine, which reconstructs a dynamically updated view of the limit order book across all price levels.

The architecture implements the observer pattern, decoupling order book state management from event handling. Multiple event handlers can subscribe to market events without modifying core reconstruction logic, enabling flexible processing pipelines. Users can define custom handlers for recording order book snapshots or event details to disk. Generic type parameters allow the framework to support different exchange formats without code duplication, while maintaining type safety through modern Python annotations. For the most common use cases, the pipeline consists of the components shown in Figure 1.

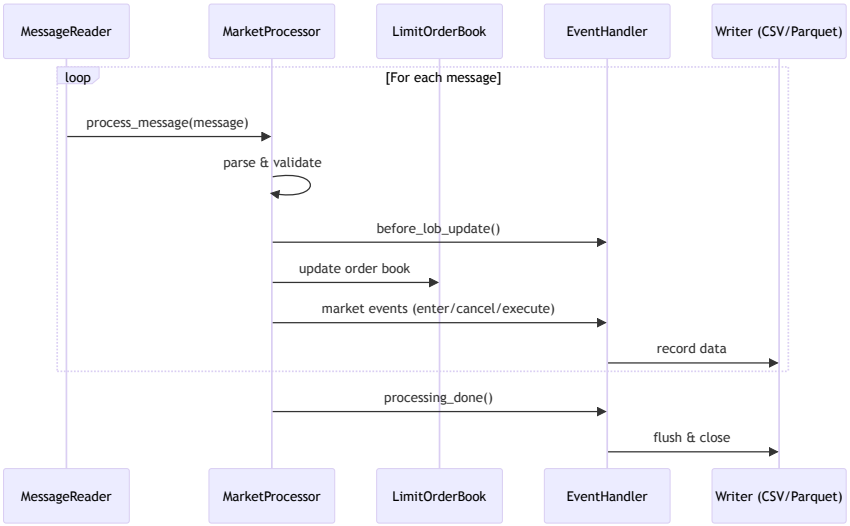

Figure 1: Sequence Diagram of MeatPy Processing Pipeline. Messages are read and parsed by the MessageReader, which passes them to the MarketProcessor. The MarketProcessor validates the messages, updates the LimitOrderBook, and notifies the EventHandler of market events. The EventHandler records data 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 modular design separates data parsing, message representation, and order book reconstruction, making it straightforward to support additional exchange formats. MeatPy minimizes external dependencies, requiring only `pyarrow` for efficient data serialization to CSV and Parquet formats. This ensures straightforward deployment across diverse environments, from laptops to high-performance computing clusters.

Research Impact Statement

MeatPy has supported peer-reviewed academic publications examining market microstructure. The framework enabled the analysis in Grégoire & Martineau (2022), investigating how earnings news propagates to stock prices using Nasdaq order book data. It contributed to Comerton-Forde et al. (2019), examining market quality under inverted fee structures, and Yaali et al. (2022), developing visualization techniques for high-frequency trading data.

The project has 29 stars and 8 forks on GitHub and was downloaded 263 times in the last month[3], indicating adoption beyond the original development team. The project has received bug reports and feature requests through GitHub Issues, demonstrating engagement from the research community.

AI Usage Disclosure

Generative AI tools were used during the development of MeatPy and the preparation of this manuscript. ChatGPT and GitHub Copilot assisted with code implementation from the time they became available. Claude Code was subsequently used for bug fixes, improving the API interface and documentation, implementing automated release workflows, and copy-editing this paper. Prior to version 0.2, MeatPy contained no AI-generated code. All AI-generated content was reviewed and edited by the authors to ensure accuracy and clarity.

[3]According to PyPI Stats, see https://pypistats.org/packages/meatpy.

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

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