Accelerating Data Science Workflows with Modin

Planning / Implementation

Pandas implements its DataFrame API on top of core data structures like the BlockManager and relies on a single-process, single-threaded execution model. Most operations (e.g., read_csv, groupby, apply) execute serially on one CPU core within the Python process, which can underutilize multicore hardware and limit scalability as data sizes grow.

In contrast, Modin retains Pandas' API but employs a partitioned DataFrame model and a query compiler to decompose operations into tasks on many smaller Pandas DataFrame partitions. These tasks are scheduled across cores or nodes via pluggable execution backends such as Ray or Dask, enabling parallel I/O and compute (e.g., parallel CSV parsing, local aggregations with a combine step) while introducing initialization and scheduling overheads.

This architectural shift means that short-lived or small-DataFrame workflows may still favor Pandas' low-overhead single-threaded engine, whereas large-scale workloads with parallel-friendly operations can benefit from Modin's distributed execution with minimal code changes.

The following figure shows that simply switching the import from `import pandas as pd` to `import modin.pandas as pd` keeps the familiar Pandas API but swaps the single threaded Pandas execution engine for Modin's distributed engine.

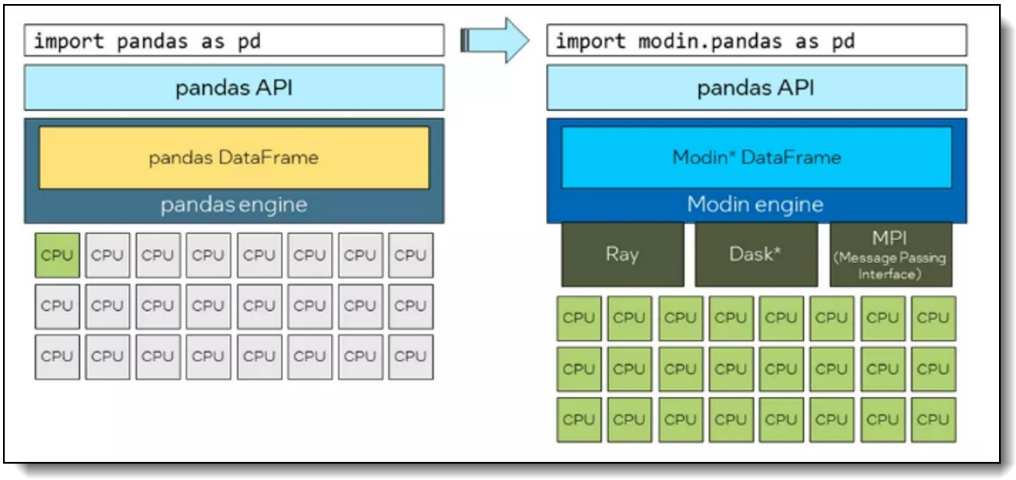

Figure 1. Architecture difference between Pandas and Modin

In this first paper of a series, we systematically benchmark Pandas versus Modin on representative workloads, focusing on execution time (measured via Python's time module) under data processing scenarios. We outline our methodology, including dataset characteristics, measurement procedures, and warm-up considerations to account for backend initialization costs. We then present detailed results for operations such as CSV ingestion, simple calculation, GroupBy tasks, and custom function application, interpreting speedups and overheads in the context of Modin's parallel architecture.

Series of papers

This paper is Part 1 of a three-part series on CPU-first acceleration of data science workloads.

- Part 2 extends the investigation to model training using Intel Extension for scikit-learn
- Part 3 evaluates the full pipeline from data processing to ML training, plus a comparison performance across 4th Gen, 5th Gen and 6th Gen Intel Xeon CPUs. (coming soon)

Methodology

To rigorously compare Pandas and Modin, we designed a repeatable benchmarking workflow that mirrors real world data science tasks across varying dataset scales.

Topics in this section:

Dataset generation

We generate synthetic DataFrames of varying sizes to simulate realistic workloads while controlling schema and reproducibility. For each target row count (100k, 200k, 400k, 800k, 1.6M) we create a DataFrame with 102 columns:

- **100 float64 columns**: values drawn from a uniform distribution via NumPy's Generator, for example:

  ```
  rng = np.random.default_rng(seed) and rng.random((n, 100))
  ```

- **1 int64 column**: values from a specified integer range; for example:

  ```
  rng.integers(low=0, high=1_000_000, size=n, dtype=np.int64))
  ```

- **1 object (string/categorical) column**: for group-by benchmarks, generate a limited set of categories, for example 10–20 distinct string labels chosen uniformly at random via `rng.choice([...], size=n)` to ensure meaningful aggregation groups.
  After generation, each DataFrame is saved to CSV (with a fixed delimiter and no index) so that "read CSV" benchmarks use files of approximate sizes: ~187MB, ~376 MB, ~753 MB,~ ~1.5 GB, and ~2.4 We record the exact file sizes for reference. Use a fixed random seed to ensure reproducibility across Pandas and Modin runs. DataFrame construction leverages Pandas for initial CSV writing; later, reading uses Pandas or Modin as appropriate.

Evaluation metrics

We focus on the following primary metric:

- **Execution Time**: measured using Python's time module before/after the operation to capture wall-clock durations for each operation. We run each benchmark operation multiple times (e.g., 10 iterations) and report the mean. For Modin, we include an initial "warm-up" run to initialize its execution backend (This is a onetime system overhead), then measure subsequent runs for steady-state performance. We separately record the backend startup time to understand its impact in short-lived scripts.

Operations benchmarked

We select representative operations reflecting common data workflows:

1. **Reading and loading data**:
    - Measure time to load the CSV file into a DataFrame (pd.read_csv(...) vs. modin.pandas.read_csv(...)). For Modin, ensure the execution engine is preconfigured (e.g., MODIN_ENGINE=ray) and warmed up. Record backend initialization separately.
2. **Simple calculation**:
    - Compute a new column derived from arithmetic on existing float columns, e.g., sum or weighted sum across several columns.
    - This tests vectorized arithmetic performance. We measure time for creating one or more such derived columns.
3. **GroupBy aggregation**:
    - Group by the object (categorical) column and compute aggregations (mean, sum) across the float columns.
    - This exercise stresses partitioned aggregation: Modin will perform local partial aggregations on partitions followed by a combine step. We measure performance for each dataset size.
4. **Lambda / custom-function application** :
    - Apply a Python-level function row-wise or element-wise. For example, a function combining numeric and string columns.
    - This tests how Pandas vs. Modin handle Python function serialization and execution across processes/partitions. We measure time and note expected overhead in Modin due to inter-process communication.
5. **For each operation type, we did the following**:
    1. Run on each dataset size.
    2. Perform multiple iterations, with collect() and, for Modin, a warm-up step.
    3. Record execution time.

Performance evaluation

This section highlights key performance trends observed when comparing Modin and Pandas across varying data sizes and operations, illustrating where parallel execution yields benefits and where single-threaded simplicity may suffice. These insights help determine at what scale and for which workloads Modin’s architecture outweighs its overhead compared to standard Pandas workflows.

The following figure demonstrates that Modin dramatically outperforms Pandas in data processing execution time, and the speed up widens as the dataset scales from 100 k to 1.6 M rows.

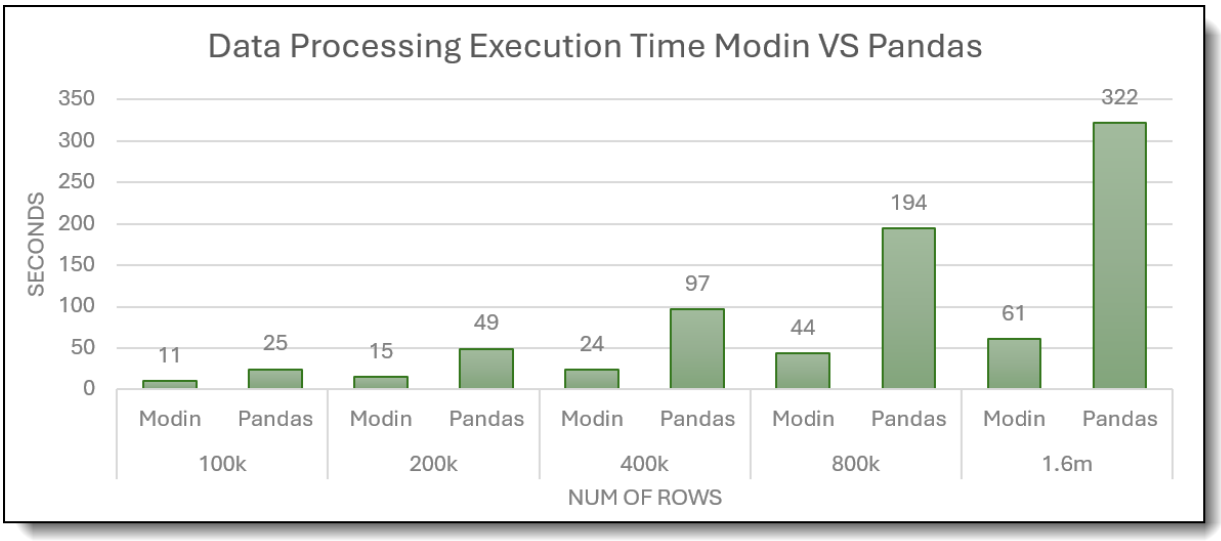

Figure 2: Total Execution Time of 5 data processing for Modin and Pandas

The benchmark exposes two standout advantages of Modin versus Pandas:

- **Significant speedups on large workloads**: For datasets on the order of ~1.6 million rows, Modin completes the five core processing steps over 5× faster than Pandas, demonstrating clear benefits for large-scale data pipelines.
- **Superior scalability**: When data volume increases tenfold, Modin's total processing time grows by approximately 6×, whereas Pandas exhibits nearly linear growth. This sub-linear scaling highlights Modin's ability to leverage parallel execution more effectively as datasets expand.

The following figure breaks down performance by operation, showing Modin's execution time ratios (blue bars) relative to Pandas' baseline (orange = 1×).

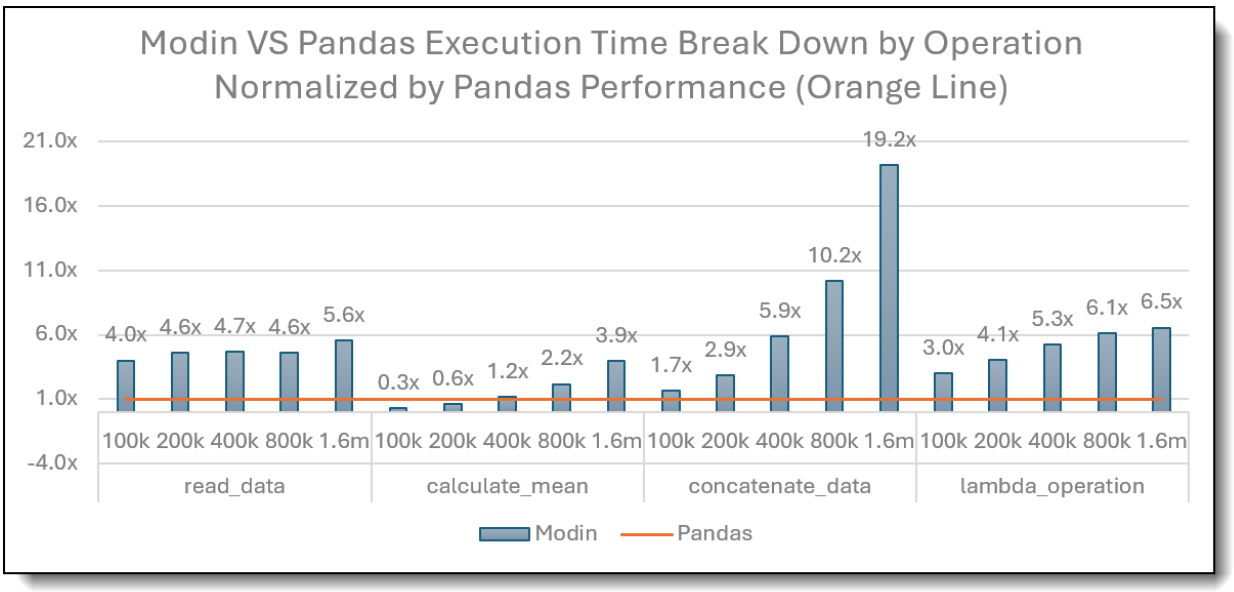

Figure 3: Modin vs Pandas Break Down for 4 data processing and normalized by Modin execution time

Digging into the per-operation benchmarks reveals a nuanced performance story:

- Modin delivers substantial speedups for I/O-bound and many transformation tasks as data grows:
  - For operations like reading data (read_data), concatenation (concatenate_data), and custom/apply patterns (lambda_operation), Modin outperform Pandas consistently. For

example, at ~1.6 M rows, Modin can be 3–19× faster than Pandas depending on the operation. This reflects Modin's parallel CSV parsing and partitioned task execution across cores.

- Certain operations retain Pandas' advantage due to lower overhead or more efficient single-threaded paths:
    - Simple arithmetic on small data:

      For small DataFrames (e.g., 100k rows), Pandas often outperforms Modin on lightweight vectorized computations because Modin's parallel engine startup and scheduling overhead dominate at this scale. However, Modin overtakes once size grows beyond a few hundred thousand rows.

    - Group-by aggregation:

      Across all tested sizes, Pandas shows much lower execution time for the group-by operation, indicating that for this specific pattern, Pandas' optimized single-threaded path (with minimal scheduling overhead) remains more efficient than Modin's partitioned aggregation, which incurs inter-process coordination costs.

      This suggests keeping Pandas for small-scale arithmetic tasks and for group-by–heavy workloads, while using Modin for large-scale I/O and transformation pipelines where its parallelism yields clear benefits.

Conclusion

The execution-time benchmarks demonstrate that Modin's parallel, partitioned architecture yields substantial benefits for large-scale data processing, while Pandas remains preferable for smaller workloads and certain operations. Specifically, for moderate-to-large datasets (e.g., hundreds of thousands to millions of rows),

Modin consistently outperforms Pandas on I/O-bound tasks (e.g., CSV/Parquet reads) and compute-intensive transformations (e.g., vectorized arithmetic, concatenation, and custom-function apply), often achieving multiple-fold speedups as data size grows. Moreover, Modin exhibits sub-linear scaling: as data volume increases tenfold, its total processing time grows by a factor significantly less than ten, whereas Pandas' single-threaded execution shows near-linear growth, causing processing times to escalate rapidly.

Conversely, for small datasets or low-overhead operations, Pandas' minimal startup and scheduling overhead allows faster execution than Modin, which must initialize its engine and manage task scheduling.

These findings imply a clear break-even point:

- For workflows routinely handling data at or above several hundred thousand rows, adopting Modin can markedly reduce runtime
- For lighter workloads or highly interactive analyses, continuing with Pandas avoids unnecessary overhead.

Practitioners should benchmark their own pipelines to identify this threshold, pre-initialize Modin in longer-running scripts to amortize startup costs, and selectively combine Pandas and Modin based on operation type and dataset size.

Test environment

Our test server had the hardware and software configuration listed in the following table.

Part 3 in our series of papers (coming soon) will benchmark servers with 3rd/4th Gen Xeon processors against servers with 5th/6th Gen Xeon processors to spotlight Modin's generation-over-generation performance gains. We'll first run the tests on an older server to quantify the baseline, then recommend upgrading to the latest CPUs.

- Demonstrate how Modin accelerates Pandas workloads on Xeon processors
- Highlight the further speed-ups delivered by the newest Xeon generations

Table 1. Test environment

Component	**Specification**
Server configuration	
Platform	ThinkSystem SR650 V2
CPU	Intel Xeon Silver 4310 processor, 24 cores / 48 threads @ 3.0 GHz
Memory	16x 16 GB DDR4 RAM
OS	Ubuntu 22.04.5 LTS (Linux kernel 6.8.0-59-generic)
Software packages	
Python	Version 3.11.11
Pandas	Version 2.2.3
Modin	Version 0.33.1
NumPy	Version 1.26.4

References

See the following web pages for more information:

- Intel Modin Documentation
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-of-modin.html
- Modin Getting Started Guide
  https://www.intel.com/content/www/us/en/developer/articles/guide/distribution-of-modin-getting-started-guide.html
- An Easy Introduction to Modin: A Step-by-Step Guide to Accelerating Pandas
  https://www.intel.com/content/www/us/en/developer/articles/technical/modin-step-by-step-guide-to-accelerating-pandas.html
- Data Science at Scale with Modin
  https://medium.com/intel-analytics-software/data-science-at-scale-with-modin-5319175e6b9a
- Ray vs Dask: Lessons learned serving 240k models per day in real-time
  https://emergentmethods.medium.com/ray-vs-dask-lessons-learned-serving-240k-models-per-day-in-real-time-7863c8968a1f
- Modin Github
  https://github.com/modin-project/modin

Authors

Kelvin He is an AI Data Scientist at Lenovo. He is a seasoned AI and data science professional specializing in building machine learning frameworks and AI-driven solutions. Kelvin is experienced in leading end-to-end model development, with a focus on turning business challenges into data-driven strategies. He is passionate about AI benchmarks, optimization techniques, and LLM applications, enabling businesses to make informed technology decisions.

David Ellison is the Chief Data Scientist for Lenovo ISG. Through Lenovo's US and European AI Discover Centers, he leads a team that uses cutting-edge AI techniques to deliver solutions for external customers while internally supporting the overall AI strategy for the Worldwide Infrastructure Solutions Group. Before joining Lenovo, he ran an international scientific analysis and equipment company and worked as a Data Scientist for the US Postal Service. Previous to that, he received a PhD in Biomedical Engineering from Johns Hopkins University. He has numerous publications in top tier journals including two in the Proceedings of the National Academy of the Sciences.

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service. Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
8001 Development Drive
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary. Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk. Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document, LP2267, was created or updated on July 29, 2025.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP2267
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP2267.

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. A current list of Lenovo trademarks is available on the Web at https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:
Lenovo®
ThinkSystem®

The following terms are trademarks of other companies:

Intel® and Xeon® are trademarks of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the U.S. and other countries.

Other company, product, or service names may be trademarks or service marks of others.