Accelerating Data Science Workflows with Intel Extension for Scikit-learn

Planning / Implementation

The first paper of this series demonstrated how Modin alleviates Pandas I/O and transformation bottlenecks. Once data is prepared, model training becomes the next compute-intensive stage. The Intel extension for scikit-learn and boost trees offers a **drop-in upgrade**—one line of code—that unlocks multicore performance without rewriting pipelines.

Once data is cleansed, model training becomes the next computational hotspot. Rather than re-implementing algorithms, a single call to **patch_sklearn()** turns Intel Extension for Scikit-learn into a true drop-in accelerator, tapping every CPU core without touching the rest of your pipeline. Intel's extension re-uses scikit-learn's Python API yet dispatches heavy-lifting to vectorized C++ kernels. This paper quantifies those benefits.

These experiments were conducted on Lenovo ThinkSystem SR650 V4. The Lenovo ThinkSystem SR650 V4 is an ideal 2-socket 2U rack server for customers that need industry-leading reliability, management, and security, as well as maximizing performance and flexibility for future growth. The SR650 V4 is based on two Intel Xeon 6700-series or Xeon 6500-series processors, with Performance-cores (P-cores), formerly codenamed "Granite Rapids-SP".

The SR650 V4 is designed to handle a wide range of workloads, such as databases, virtualization and cloud computing, virtual desktop infrastructure (VDI), infrastructure security, systems management, enterprise applications, collaboration/email, streaming media, web, and HPC.

Figure 1. Lenovo ThinkSystem SR650 V4

Benchmark methodology

To quantify the benefit of sklearn-intelex over vanilla scikit-learn we followed a controlled, repeatable workflow that isolates the effects of **hardware**, **library implementation**, and **workload mix**. The methodology is organized around three building blocks—**metrics**, **algorithms & datasets**, and **procedure**—described below.

Topics in this section:

Metrics

The list below defines the performance and quality signals we record for every experiment:

- **Primary**– *Training wall-clock time* (median of 5 cold-start runs).
- **Guardrail**– *Predictive quality*: accuracy (classification), Mean Square Error (regression), silhouette (clustering).

Algorithms and datasets

We chose widely used classical algorithms paired with publicly available datasets large enough to stress a multi core CPU yet still runable on a single server. The following table groups them by ML task and shows which metric serves as the speed target (primary) and which protects model fidelity (guardrail).

Table 1. Algorithms and datasets

Task	Algorithm	Workload	Primary Metric	Guardrail Metric
Classification	Logistic Regression	CIFAR-100	Training & Inference Time	Log Loss
	KNeighborsClassifier	MNIST	Training & Inference Time	Accuracy
	Random Forest Classifier	Rain in Australia	Training & Inference Time	Accuracy
Regression	Random Forest Regressor	Yolanda	Training & Inference Time	Mean Square Error
	Linear Regression	YearPredictionMSD	Training & Inference Time	Mean Square Error
Clustering	K-Means	Spoken arabic digit	Training & Inference Time	Inertia
	DBSCAN	Spoken arabic digit	Training Time	Davies-Bouldin score
Booting Tree	XGBoost	Synthetic Multiclass Data	Inference Time	Accuracy
	LightGBM	Synthetic Multiclass Data	Inference Time	Accuracy
	CatBoost	Synthetic Multiclass Data	Inference Time	Accuracy

Procedure

The following run-loop is executed for **each algorithm × dataset** combination to generate the timing and accuracy numbers reported later. By repeating the *exact* workflow twice—once with stock scikit-learn ("vanilla") and once with Intel's patched version—we isolate the speed-up attributable solely to sklearn-intelex.

1. Preprocess dataset and cache in memory.

   Load the raw dataset, perform any required cleaning/encoding, and hold the resulting arrays in memory so file-I/O never contaminates timing results.

2. Select one of the following implementations:
   - Intel run: call from sklearnex import patch_sklearn; patch_sklearn() to monkey-patch scikit-learn estimators with oneDAL-backed, multi-threaded versions.
   - Baseline ("vanilla") run: skip the patch—i.e., import and use scikit-learn exactly as shipped on PyPI—so all algorithms execute on a single core.

3. Measure training & inference.

   Fit the model five cold-start times, wrapping both fit() and immediate predict() calls with timer(). Record the median wall-clock duration along with the selected guard-rail metric (accuracy, MSE, etc.)

4. Evaluate on 20 % hold-out set.

   Captures guard-rail quality metrics (accuracy, MSE, silhouette, etc.)

5. Shut down the Python process between Intel and baseline runs.

   Flushes OS caches and prevents warm-cache bias.

Results

This section presents the quantitative (Tables 2 & 3) and visual (Figures 2 & 3) outcomes of our benchmarking campaign, detailing how sklearn-intelex affects **training speed, inference speed, and predictive accuracy** across every algorithm–dataset pair. We first summarize aggregate speed-ups and accuracy preservation, then highlight noteworthy patterns and edge cases observed during the experiments.

Training time speed-ups

To understand where Intel Extension for scikit-learn delivers the most value, we broke training latency down by task and algorithm.

The table below reports the median wall-clock training time (in ms) for each workload **with sklearn-intelex activated ("Intel Accelerator") versus unmodified scikit-learn ("Vanilla Algorithm")** and the resulting speed-up factor. Use these values in tandem with Figure 2 to see how acceleration varies across model families.

Table 2. Training-time speed-ups

Task	Algorithm	Intel Accelerator (ms)	Vanilla Algorithm (ms)	Speed-up x
Clustering	DBSCAN	5080	445,238	87.6x
	K-means	558	1408	2.5x
Classifier	KNeighborsClassifier	609	202	0.3x
	Logistic Regression	16,748	177,503	10.6x
	Randon Forest Classifier	402	3383	8.4x
Regression	Linear Regression	5	184	34.7x
	Randon Forest Regressor	8188	125,242	15.3x

Figure 2 shows how much Intel's patch slashes training compared to Vanilla Sklearn

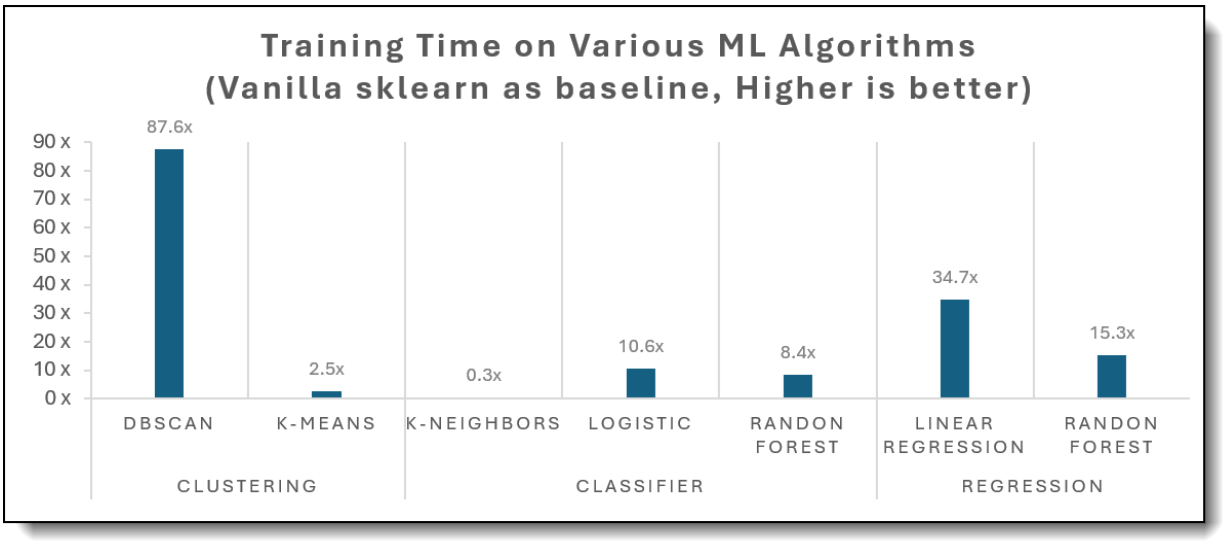

Figure 2. Training Times comparison

Inferencing time speed-ups

After training speed, the next question for production pipelines is how fast each model can deliver predictions. We therefore timed the predict() call for every algorithm–dataset pair under the same cold-start conditions used in the training benchmarks.

The table below summarizes the results. For each workload it lists the median wall-clock latency with Intel Extension for scikit-learn (“Intel Accelerator”) versus unmodified scikit-learn (“Vanilla Algorithm”) and the resulting speed-up factor.

Table 3. Inferencing-time speed-ups

Task	Algorithm	Intel Accelerator (ms)	Vanilla Algorithm (ms)	Speed-up x
Clustering	K-means	1.6	7	4.5x
Classifier	KNeighborsClassifier	319	7001	21.9x
	Logistic Regression	81	164	2.0x
	Randon Forest Classifier	504	411	0.8x
Regression	Linear Regression	1	2	2.5x
	Randon Forest Regressor	110	400	3.6x
Boosting Trees	XGBost	1.6818	4.7616	2.8x
	LightGBM	0.704	4.9127	7.0x
	CatBoost	1.687	2.4129	1.4x

The following figure shows how Intel’s extension cutting inference latencies.

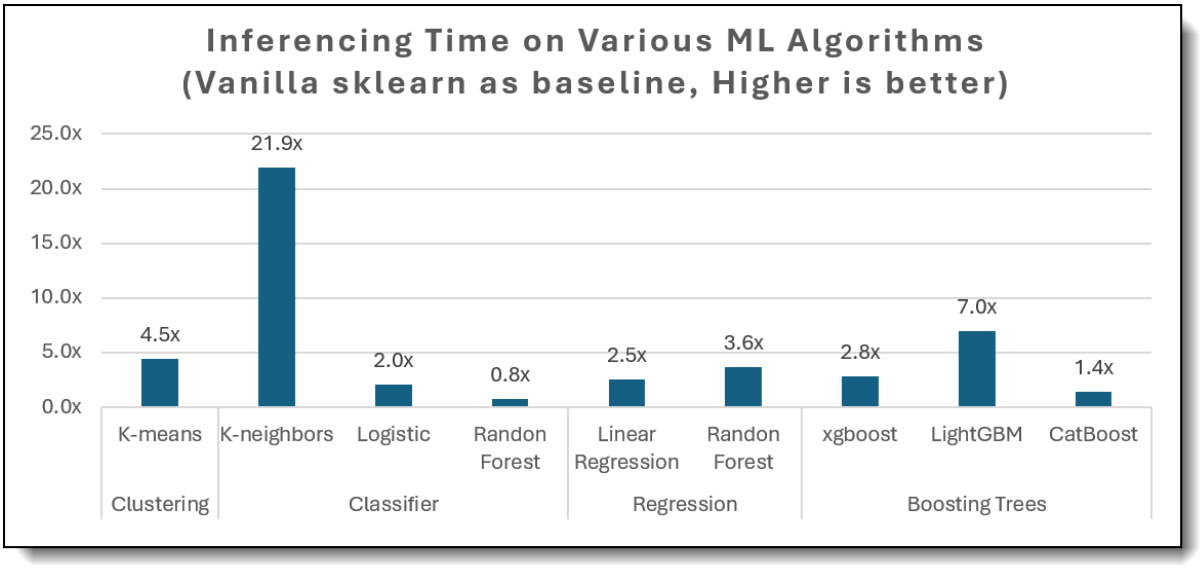

Figure 3. Inferencing Time comparison

Accuracy preservation

Across all tasks the accuracy/MSE deviation relative to vanilla scikit-learn remained within ±0.2%, validating the extension's numerical equivalence.

The following table confirms that the Intel Extension preserves model quality. Across all workloads, the guard rail metrics (accuracy, log loss, MSE, inertia, etc.) differ from vanilla scikit-learn by ≤ 0.2 %.

Table 4. Accuracy Preservation

Algorithm	Metric	Intel Extension result	Vanilla Algorithm result	Metric difference %
DBSCAN	Davies-Bouldin score	0.85	0.85	0.00%
K-means	Inertia	13381.46	13377.72	0.03%
KNeighborsClassifier	Accuracy	0.96	0.96	0.00%
Logistic Regression	Log Loss	3.71	3.71	0.05%
Randon Forest Classifier	Accuracy	0.86	0.85	0.05%
Linear Regression	MSE	36.39	36.43	0.10%
Randon Forest Regressor	MSE	83.65	83.80	0.19%
XGBoost	Accuracy	0.976	0.976	0.00%
LightGBM	Accuracy	0.96	0.96	0.00%
CatBoost	Accuracy	0.977	0.977	0.00%

Observations

Together, these results distill into three headline takeaways:

- **Training efficiency soars** (Table 1).

  Intel Extension for Scikit-learn cuts fit-time by double- to triple-digit factors: **6× for DBSCAN**, 34.7× for Linear Regression, 10.6× for Logistic Regression, and 8.4× for Random ForestClassifier—while K-NN lags (0.3×) because that routine is still un-optimized.

- **Inference gets quicker too** (Table 2).

  Prediction latency shrinks **4-22×** for most workloads (21.9× on K-NN, 4.5× on K-means, 3.6× on Random ForestRegressor, 2.5× on Linear Regression). The only regression is a mild 0.8× slow-down on Random Forest Classifier, where patch-overhead outweighs gains on the small test set.

- **Model quality is preserved** (Table 3).

  Accuracy, MSE, inertia, and Davies-Bouldin all stay within **±0.2%** of vanilla scikit-learn, confirming that the speed-ups come with *virtually zero* impact on predictive performance.

A single line call to Intel Extension for Scikit-learn instantly unleashes full multicore power without altering your existing pipeline. For organizations already using pandas DataFrames and scikit-learn, migrating preprocessing to Modin and model training to scikit-learn-intelex forms a cohesive optimization path entirely on CPU, mitigating GPU scarcity and cost.

Conclusion

The combined results in Tables 1 and 2 show that optimization gains vary by phase. Algorithms such as k-Nearest Neighbors train ~0.3× slower with Intel Extension yet deliver ≈22× faster inference, making them attractive for prediction-heavy workloads. In contrast, Random Forest Classifier enjoys an ≈8× training boost but a slight 0.8× slowdown at inference on the small test set, favoring experimentation cycles over low-latency scoring unless additional tuning is applied. Most other methods (DBSCAN, Linear/Logistic Regression, K-means, Random Forest Regressor) accelerate both phases.

Choosing where to deploy the extension should therefore consider the complete fit to predict pipeline and the dominant cost in production.

Overall, on a dual-socket Intel Xeon platform, Intel Extension for Scikit-learn delivers **2-87×** faster training times on mainstream ML algorithms while preserving baseline accuracy. These gains translate directly to higher experiment throughput, faster iteration cycles, and improved server utilization for production retraining workloads.

Future work - Part 3

Building on the speed-ups demonstrated in Part 1 and Part 2 (this document), the next stage evaluate the end-to-end pipeline how Intel Accelerator boosts the data science projects.

The objectives of Part 3 of this series will be the following:

- Evaluate the full pipeline—from Modin-accelerated data processing through Intel Extension for Scikit-learn model training to Intel Optimization for XGBoost inference.
- A comparison performance across 4th Gen, 5th Gen and 6th Gen Intel Xeon CPUs, highlighting generation-over-generation efficiency gains.

Test environment

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

Table 5. Test environment

Component	**Specification**
Server configuration	
Platform	Lenovo ThinkSystem SR650 V4
CPU	Intel Xeon 6787P processor, 86 cores / 172 threads @ 3.2 GHz
Memory	16x 64 GB DDR5 RAM
OS	Ubuntu 22.04.5 LTS (Linux kernel 6.8.0-59-generic)
Software components	
Python	Version 3.11.11
Pandas	Version 2.2.3
scikit-learn	Version 1.5.0
scikit-learn-intelex	Version 2025.1

References

See the following web pages for more information:

- Lenovo ThinkSystem SR650 V4 product guide
  https://lenovopress.lenovo.com/lp2127-thinksystem-sr650-v4-server
- Lenovo ThinkSystem SR650 V4 datasheet
  https://lenovopress.lenovo.com/datasheet/ds0194-lenovo-thinksystem-sr650-v4
- Intel Extension for Scikit-learn – GitHub repository
  https://github.com/uxlfoundation/scikit-learn-intelex/tree/main/examples/notebooks
- Intel Extension for Scikit-learn documentation
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/scikit-learn.html
- Intel oneAPI Data Analytics Library
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/onedal.html#gs.nmjeoe
- scikit-learn User Guide
  https://scikit-learn.org/stable/user_guide.html

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, LP2268, was created or updated on July 31, 2025.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP2268
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP2268.

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.