Attention-Informed Surrogates for Navigating Power-Performance Trade-offs in HPC

Ashna Nawar Ahmed[1] **Banooqa Banday**[1] **Terry Jones**[2] **Tanzima Z. Islam**[1]
[1]Texas State University [2]Oak Ridge National Laboratory
ashna.ahmed@txstate.edu, banooqa@txstate.edu, trjones@ornl.gov, tanzima@txstate.edu

Abstract

High-Performance Computing (HPC) schedulers must balance user performance with facility-wide resource constraints. The task boils down to selecting the optimal number of nodes for a given job. We present a surrogate-assisted multi-objective Bayesian optimization (MOBO) framework to automate this complex decision. Our core hypothesis is that surrogate models informed by attention-based embeddings of job telemetry can capture performance dynamics more effectively than standard regression techniques. We pair this with an intelligent sample acquisition strategy to ensure the approach is data-efficient. On two production HPC datasets, our embedding-informed method consistently identified higher-quality Pareto fronts of runtime-power trade-offs compared to baselines. Furthermore, our intelligent data sampling strategy drastically reduced training costs while improving the stability of the results. To our knowledge, this is the first work to successfully apply embedding-informed surrogates in a MOBO framework to the HPC scheduling problem, jointly optimizing for performance and power on production workloads.

1 Introduction

Exascale HPC systems face a fundamental trade-off: minimizing job runtime to accelerate science while reducing power consumption for sustainability. This tension plays out in everyday decisions–scientists must choose how many nodes to request, often guessing between too few (risking termination) and too many (wasting energy and resources). Traditional scheduling heuristics, such as First-Come, First-Served with backfilling, optimize for a single objective (most commonly runtime) and lack the flexibility to adapt to workload variability or guide resource selection.

Recent work in HPC scheduling has focused on runtime prediction using ML models [1, 2, 3, 4, 5, 6, 7, 8, 9], single-objective reinforcement learning [10, 11, 12, 13, 14], or offline parameter tuning using surrogate-based Bayesian optimization (BO) [15, 16, 17, 18, 19]. While these approaches show promise, they optimize a single objective, assume full access to clean input data, and do not expose runtime–power trade-offs. Because runtime and power are inherently conflicting objectives, it is important to capture the trade-off space, as no one solution may optimize both. As such, power-performance trade-off modeling can be formulated as a multi-objective optimization problem. The MOBO literature and acquisition strategies [20], improve sample efficiency. However, these methods have not been explored in the context of real-world and noisy HPC telemetry data, characterized by missing entries, inconsistent counters, and large variability across jobs. This observation reveals a key limitation in prior work: the lack of methods for modeling conflicting objectives that operate effectively on large-scale, noisy, and irregular telemetry data.

HPC telemetry presents three core challenges: (1) it is massive, noisy, and irregular, making surrogate training unstable [21]; we address this challenge with **intelligent sample acquisition** to guide selective sampling; (2) it is multimodal and variable in scale, with log-transformed targets amplifying sensitivity to noise [22]; in response, we propose **attention-based embeddings** to highlight

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

informative features and suppress redundancy; and (3) runtime and power are conflicting objectives, which traditional heuristics cannot jointly optimize; we model the power-performance trade-off by leveraging the **MOBO** method. In practice, our use of MOBO is intended as a decision-support tool: it can recommend the number of nodes a user should request for a job, a choice that is otherwise difficult to make manually and often leads to suboptimal runtime or wasted power.

These challenges motivate two hypotheses tested in this study: **H1:** attention-based embeddings improve surrogate quality compared to direct regressors. **H2:** MOBO captures runtime–power trade-offs more effectively than single-objective BO (SOBO) or random baselines.

Contributions: In this work, we (1) Propose a surrogate-driven MOBO framework that integrates attention-based embeddings and intelligent sample acquisition for robust modeling from irregular HPC telemetry; (2) Demonstrate across two large-scale HPC datasets that both **H1** and **H2** hold consistently; (3) Provide a reproducible pipeline for surrogate training and optimization that explicitly balances runtime and power trade-offs for decision support.

2 Related Work

There exists a large body of work that leverages ML for predicting performance for HPC systems using single metric performance prediction [2, 3, 4, 5, 6, 7, 8, 9]. More recently, Ramachandran et al. [1] propose a hybrid Genetic Algorithm (GA) and ML approach for runtime prediction. Their method replaces inaccurate user-supplied runtimes with GA-defined runtime classes, enabling models such as KNN, SVR, XGBoost, and DNN to achieve $R^2 > 0.8$. While effective for runtime prediction, this work does not incorporate power modeling or multi-objective trade-offs.

In parallel, the optimization community has advanced the MOBO methodology to handle conflicting objectives by learning surrogate models and guiding the search toward Pareto-optimal trade-offs. Daulton et al.[20] introduced qEHVI, a differentiable acquisition function that efficiently improves hypervolume in parallel MOBO settings. Such methods aim to approximate the *Pareto front*, the set of solutions where no objective (e.g., runtime) can be improved without worsening another (e.g., power). Pareto optimization is valuable because it exposes trade-offs directly, giving decision-makers multiple balanced options rather than a single outcome[23]. However, these approaches have not yet been applied to HPC job scheduling.

Jannach et al. [24] survey multi-objective recommender systems and highlight the role of Pareto optimization [23] in balancing competing goals such as accuracy and diversity. Similar to their ranking of items for end users, our framework produces ranked recommendations of candidate resource configurations in HPC scheduling. The difference lies in the domain: they target user-facing recommendation systems, while we focus on runtime–power trade-offs in supercomputing workloads.

Surrogate-based Bayesian optimization (BO) has been widely applied to offline autotuning in HPC, including compiler and thread parameters [15], scalable asynchronous searches [16], file system and storage tuning [17, 18], and large-scale parallel optimization [19]. These approaches improve subsystem efficiency but remain single-objective and do not capture conflicting runtime–power trade-offs. Our work fills these gaps by using attention to improve surrogate model quality on irregular HPC telemetry, and applying these models in MOBO to find better runtime–power trade-offs.

3 Our Approach

We design a surrogate-driven MOBO framework to address the scale, heterogeneity, and conflicting objectives of HPC scheduling. First, to cope with massive but irregular telemetry, we adopt intelligent sample acquisition based on active learning (see Appendix: Intelligent Sample Acquisition Overview). Instead of training on all raw logs, the surrogate iteratively selects the most informative samples, improving efficiency and robustness in the presence of noisy data.

To handle heterogeneity in HPC telemetry, we use attention-based embeddings to highlight informative features and suppress irrelevant or redundant inputs. We implement this using TabNet [25], which learns feature-level attention directly from structured data. We hypothesize that such embeddings enable lightweight regressors–such as Random Forest [26], XGBoost [27], and LightGBM [28]–to generalize better, train faster, and remain more interpretable. While transformer-based models have shown promise for modality fusion and long-range dependencies [29, 30, 31], they are resource-

intensive, require large training sets, and lack interpretability, which limits their practicality for deployment in HPC systems.

Finally, to model runtime-power trade-off space, we integrate trained surrogates into the MOBO framework. This allows us to capture runtime–power trade-offs explicitly, predict performance for candidate node allocations, and generate Pareto fronts that quantify the balance between runtime and power consumption. In HPC, this often involves balancing runtime and power consumption–improving one typically worsens the other. The Pareto front captures all such non-dominated solutions, providing a spectrum of efficient choices for informed decision-making. By comparing against single-objective and random baselines, we show that the MOBO framework produces higher-quality Pareto fronts–capturing more balanced runtime–power trade-offs.

End-to-end, as illustrated in Fig. 1 in the Supplementary Material section, the pipeline consists of pre-processing HPC datasets, selecting informative samples, extracting embeddings, fitting surrogate models, integrating them into the MOBO framework, predicting runtime and power outcomes, and visualizing Pareto fronts for decision-making in HPC scheduling.

4 Experimental Setup

Datasets. We evaluate on two real-world HPC job-log datasets: PM100 [32], which contains 231,238 job records with 35 features, and Adastra [33], which contains 15,285 job records with 35 features. Both datasets include categorical fields (e.g., queue, partition), numerical metrics (e.g., runtime, memory, power), and system-level telemetry derived from production supercomputers. **Pre-processing.** We pre-process these datasets including handling missing values, aggregating node-level power to job-level totals, and convert categorical, numeric, and time-series–derived signals (e.g., GPU bursts, I/O spikes) into a single structured input for modeling. **Surrogate Models.** We compare two classes of surrogates: (1) Transformer-based models using TabNet, and (2) light-weight regressors–Random Forest [26], XGBoost [27], and LightGBM [28] trained using attention-based embeddings. All surrogates are embedded in a MOBO loop that generates Pareto fronts capturing runtime–power trade-offs. **Baselines.** We compare against three baselines: (1) SOBO (runtime-only), (2) SOBO (power-only), and (3) Random Search. SOBO quantifies the impact of optimizing one objective in isolation, while Random estimates the value of surrogate-guided acquisition. **Compute Environment.** Experiments are run on the Stampede3 supercomputer at TACC [34] using CPU nodes for training and MOBO evaluations. Hyperparameter and training details are in Tables 11 and 12.

5 Results

Table 1: Summary of results supporting Hypotheses 1 and 2, and dataset sizes under sampling.

Dataset	Metric	Hypothesis 1 (Embeddings vs. TabNet)	Hypothesis 2 (MOBO vs. Baselines)	Sample size
PM100	HV	✓ Embeddings have *orders of magnitude* higher HV than Regressor	✓ Improved HV 24% vs. SOBO–Runtime (TabNet Regressor)	50%→73,983; 75%→77,278; 100%→109,202
	Spread	✓ Embeddings have ∼99% lower Spread than Regressor	✓ MOBO best in 3/4 families (75%)	Original→231,238
Adastra	HV	✓ Embeddings have 37% more HV than Regressor	✓ Improved HV 37% vs. SOBO–Runtime (TabNet Regressor)	50%→3,964; 75%→4,163; 100%→4,547
	Spread	✓ Embeddings have ∼90% lower Spread than Regressor	✓ MOBO best in 3/4 families (75%)	Original→15,285

Impact of Attention-Based Embeddings. This experiment tests **H1** by evaluating whether attention-based embeddings improve surrogate quality by enabling simpler models to generalize better. Observations. Table 1 and Fig. 4 (Supplementary Material section) show that embedding-informed simple surrogate models outperform a transformer architecture across both datasets in most settings, supporting *H1*. Simpler, tree-based models outperform transformer-based models in our setting because subsampling significantly reduces dataset size, limiting the effectiveness of deep architectures that require large amounts of data to generalize well. Moreover, we study the impact of these high

quality surrogates on the downstream pipeline and find that: (1) On PM100, embeddings reduce Spread by $\sim$99% and yield orders-of-magnitude higher HV under SOBO; (2) On Adastra, they improve HV by 37% and reduce Spread by $\sim$90%. These results can be explained by the fact that attention mechanism isolates relevant features while filtering out noise, making tree-based models to train faster and remain robust.

Impact of Intelligent Sample Acquisition. This experiment evaluates whether intelligent sample acquisition improves surrogate stability and optimization quality for HPC job scheduling. Observations. As summarized in Table 1, embedding-informed surrogates consistently outperform TabNet regressors in both HV and spread, supporting *H1*. MOBO generally outperforms SOBO and Random, with clearer gains under sampling (H2). Across both PM100 and Adastra, intelligent sample acquisition enables stable and accurate surrogates using only 50–75% of the data, significantly reducing training overhead. At 50%, both datasets maintain consistent hypervolume (HV) and spread; at 75%, surrogate accuracy (MAPE ≈ 0.99) plateaus (Supplementary Material section, Table 13). Compared to full-data training, intelligent sample acquisition yields faster convergence and lower variance (Supplementary Material section: Fig. 2, 3). On PM100, it improves Pareto diversity: spread drops from 2.5×10^5 to below 10^4. On Adastra, it suppresses erratic HV values (from 10^{16} to 10^{13}), producing smoother trade-offs.

Impact on Timing Overheads. This experiment evaluates whether intelligent sample acquisition reduces the computational cost of surrogate-driven MOBO by lowering runtime overhead across preprocessing, training, and optimization stages. Observations. Table 10 (Supplementary Material section) shows that sampling consistently reduces total execution time. On PM100, runtime drops from 6480s to 6009s, with the largest savings in surrogate training. On Adastra, the reduction is more pronounced—from 2699s to 1962s (27% decrease). While MOBO evaluation time increases slightly due to repeated acquisition steps, this cost is offset by the reduction in data volume. The dominant gains come from fewer training samples, which shorten preprocessing and model training. These results reinforce that intelligent sampling not only improves surrogate stability and optimization quality, but also reduces end-to-end runtime.

Limitations and Future Work. Our present study evaluates two production traces and focuses on runtime-power trade-offs; going forward we will (i) *deployment & latency*: profile end-to-end overheads by reporting surrogate inference time and BO cycle wall-time, and tighten the loop via cached embeddings and warm-started retraining to meet scheduler budgets; (ii) *scalability in objectives*: extend beyond two objectives and assess complexity/quality trade-offs using scalarization warm-starts and NEHVI/log-NEHVI variants with controlled candidate sets; (iii) *model choice*: examine when lightweight regressors suffice versus transformer-based surrogates under different data regimes, with an emphasis on stability, small-sample behavior, and compute/latency footprints; and (iv) *generalization across systems*: study portability across clusters via re-embedding or adapter heads with minimal calibration runs, and quantify the impact of observed domain gaps. These steps concentrate on practical readiness and breadth while preserving the simplicity that makes the current pipeline deployable.

Broader Impacts. This research will have a significant impact on computational scientists by automating the complex decision of how many nodes to use for their HPC jobs. By embedding this intelligence into the scheduler, users are freed from this error-prone task, allowing them to focus on science. Building on our previous work [3], which shows that accurate runtime prediction can cut time-to-science by 71% and resource usage by 42%, we can extend the work further to balance runtime with power consumption. Such a capability will accelerate scientific discovery, boost the efficiency of multi-million-dollar HPC systems, and lower their operational cost.

6 Conclusion

This work introduces an end-to-end framework for data-efficient, multi-objective decision-making in HPC scheduling. We improve the deployability of AI-driven performance models by using intelligent sample acquisition for data reduction and attention-based embeddings to improve prediction accuracy. These enhanced surrogates serve as the foundation for a MOBO-based optimization engine that effectively models the power-performance trade-off space. The result is a scalable performance modeling methodology that can be integrated into next-generation HPC schedulers to automate complex configuration decisions and optimize multiple operational goals simultaneously.

7 Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science under Award Number DE-SC0022843. Additional support was provided by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program under Contract Number DE-AC05-00OR22725.

References

[1] S. Ramachandran, M. L. Jayalal, M. Vasudevan, S. Das, and R. Jehadeesan. Combining machine learning techniques and genetic algorithm for predicting run times of high performance computing jobs. *Applied Soft Computing*, 165:112053, 2024.

[2] Tanzima Z Islam, Jayaraman J Thiagarajan, Abhinav Bhatele, Martin Schulz, and Todd Gamblin. A machine learning framework for performance coverage analysis of proxy applications. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 538–549. IEEE, 2016.

[3] Arunavo Dey, Neil Antony, Aakash R. Dhakal, Kowshik Thopalli, Jayaraman J. Thiagarajan, Tapasya Patki, Aniruddha Marathe, Tom Scogland, Jae-Seung Yeom, and Tanzima Islam. ModelX: A Novel Transfer Learning Approach Across Heterogeneous Datasets. In *The 34th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2025. Accepted (19% acceptance rate).

[4] Tanzima Islam, Alexis Ayala, Quentin Jensen, and Khaled Ibrahim. Toward a programmable analysis and visualization framework for interactive performance analytics. In *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*, pages 70–77. IEEE, 2019.

[5] Tarek Ramadan, Tanzima Z Islam, Chase Phelps, Nathan Pinnow, and Jayaraman J Thiagarajan. Comparative code structure analysis using deep learning for performance prediction. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 151–161. IEEE, 2021.

[6] Tarek Ramadan, Ankur Lahiry, and Tanzima Z. Islam. Novel representation learning technique using graphs for performance analytics. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 1311–1318, 2023.

[7] Jae-Seung Yeom, Jayaraman J Thiagarajan, Abhinav Bhatele, Greg Bronevetsky, and Tzanio Kolev. Data-driven performance modeling of linear solvers for sparse matrices. In *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), International Workshop on*, pages 32–42. IEEE, 2016.

[8] Arunavo Dey, Aakash Dhakal, Tanzima Z. Islam, Jae-Seung Yeom, Tapasya Patki, Daniel Nichols, Alexander Movsesyan, and Abhinav Bhatele. Relative performance prediction using few-shot learning. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1764–1769, 2024.

[9] Daniel Nichols, Alexander Movsesyan, Jae-Seung Yeom, Abhik Sarkar, Daniel Milroy, Tapasya Patki, and Abhinav Bhatele. Predicting cross-architecture performance of parallel programs. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 570–581, 2024.

[10] Yuping Fan, Zhiling Lan, Taylor Childers, Paul Rich, William Allcock, and Michael E. Papka. Deep reinforcement agent for scheduling in hpc, 2021.

[11] Yuping Fan and Zhiling Lan. Dras-cqsim: A reinforcement learning based framework for hpc cluster scheduling. *arXiv preprint arXiv:2105.07526*, 2021.

[12] Qiqi Wang, Hongjie Zhang, Cheng Qu, Yu Shen, Xiaohui Liu, and Jing Li. Rlschert: An hpc job scheduler using deep reinforcement learning and remaining time prediction. *Applied Sciences*, 11(20):9448, 2021.

[13] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. Rlscheduler: An automated hpc batch job scheduler using reinforcement learning. In *Proceedings of [Conference Name]*, pages xx–xx, 2019.

[14] Abel Souza, Kristiaan Pelckmans, and Johan Tordsson. A hpc co-scheduler with reinforcement learning. *arXiv preprint arXiv:2401.09706*, 2024.

[15] Harshitha Menon, Abhinav Bhatele, and Todd Gamblin. Auto-tuning parameter choices in hpc applications using bayesian optimization. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 831–840. IEEE, 2020.

[16] Anh Tran. Scalable^3-bo: Big data meets hpc - a scalable asynchronous parallel high-dimensional bayesian optimization framework on supercomputers. *arXiv preprint*, arXiv:2108.05969, 2021.

[17] Kyurae Kim, Youngjae Kim, and Sungyong Park. A probabilistic machine learning approach to scheduling parallel loops with bayesian optimization. *IEEE Transactions on Parallel and Distributed Systems*, 2022. Preprint at arXiv.

[18] Matthieu Dorier, Romain Egele, Prasanna Balaprakash, Jaehoon Koo, Sandeep Madireddy, Srinivasan Ramesh, Allen D. Malony, and Rob Ross. Hpc storage service autotuning using variational-autoencoder-guided asynchronous bayesian optimization. *arXiv preprint arXiv:2210.00798*, 2022.

[19] Adrian Perez, Seth Ockerman, Tristan Aikman, and Khaled Z. Ibrahim. Parallelizing autotuning for hpc applications: Unveiling the potential of speculation strategy in bayesian optimization. *The International Journal of High Performance Computing Applications*, 2025. In press.

[20] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 9851–9864. Curran Associates, Inc., 2020.

[21] Byung H. Park, Saurabh Hukerikar, Ryan Adamson, and Christian Engelmann. Big data meets hpc log analytics: Scalable approach to understanding systems at extreme scale. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 758–765, 2017.

[22] Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021.

[23] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, New York, NY, USA, 2001.

[24] Dietmar Jannach and Himan Abdollahpouri. A survey on multi-objective recommender systems. *Frontiers in Big Data*, 6:1157899, 2023.

[25] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1027–1035. ACM, 2020.

[26] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[27] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 785–794, New York, NY, USA, 2016. ACM.

[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pages 3146–3154, 2017.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pages 5998–6008, 2017.

[30] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[31] Andrew Jaegle, Felix Gimeno, Andrew Brock, Oriol Vinyals, Andrew Zisserman, and João Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, pages 4651–4664, 2021.

[32] Francesco Antici, Mohsen Seyedkazemi Ardebili, Andrea Bartolini, and Zeynep Kiziltan. Pm100: A job power consumption dataset of a large-scale production hpc system. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '23, page 1812–1819, New York, NY, USA, 2023. Association for Computing Machinery.

[33] Adastra: Log data from the hpe-cray ex supercomputer at cines. `https://www.top500.org/system/180047/`, 2024. Accessed: 2025-08-18.

[34] Texas Advanced Computing Center. Stampede3 user guide. `https://docs.tacc.utexas.edu/hpc/stampede3/`, 2025. Accessed: 2025-08-18.

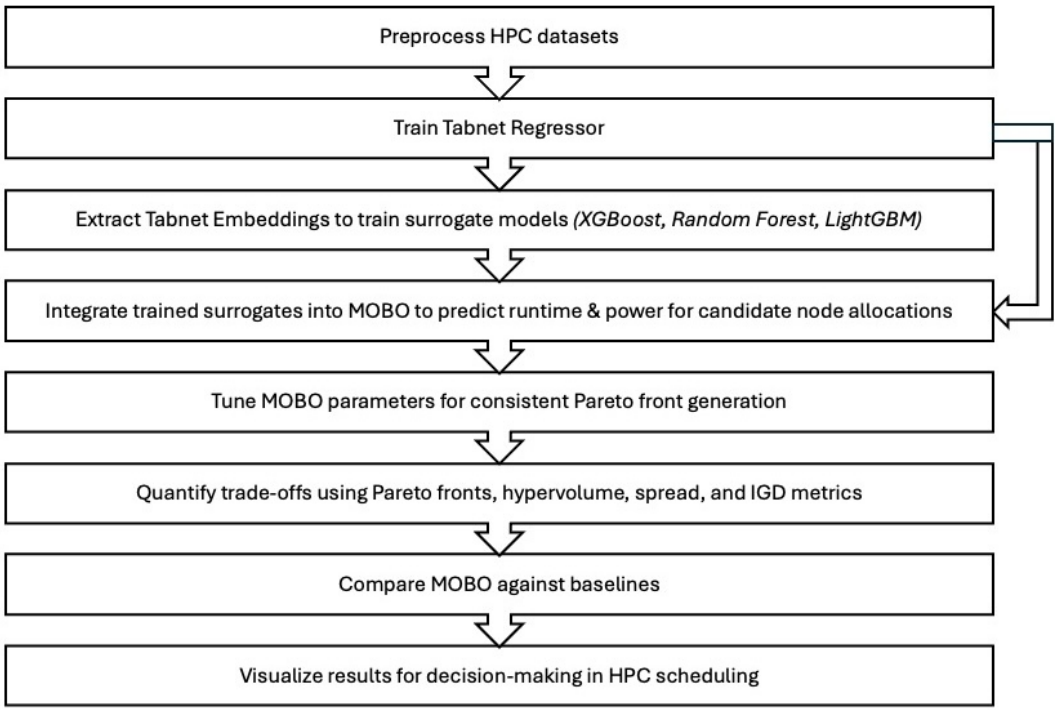

Figure 1: End-to-end pipeline for surrogate-driven MOBO on HPC scheduling.

A Supplementary Material

The full pipeline is shown in Figure 1.

Intelligent Sample Acquisition Overview We propose a loss-proportional subset sampler for mixed regression–classification targets. Given a table $D = \{(x_i, y_i)\}_{i=1}^{n}$ partitioned into regression targets T_r and classification targets T_c, we first construct a numeric sampling view V by (i) reducing structured power arrays to scalar totals, (ii) converting datetimes to epoch seconds and deriving durations (wait, queue, job duration), (iii) casting string numerics and imputing missing values, and (iv) label-encoding remaining categoricals. We then assign each sample a scalar difficulty L_i by training lightweight per-target predictors on V and aggregating their per-sample errors: absolute error for regression and soft error $1 - \Pr(\text{correct} \mid x_i)$ for classification. Sampling probabilities are defined by a linearly scaled and clipped map $p_i = \text{clip}(\lambda L_i,\, p_{\min},\, 1)$ with a small auto-tuning loop on λ to match an expected sampling rate τ while capping the fraction of saturated points at probability 1. Finally, we draw $z_i \sim \text{Bernoulli}(p_i)$ independently and return the full-fidelity subset $D_{\text{sub}} = D[\{i : z_i = 1\}]$. This emphasizes informative (higher-loss) regions while preserving exploration via the floor $p_{\min}$.

The full results are shown in Tables 2 and 3 (no sampling); 4 and 5 (fraction = 0.5); 6 and 7 (fraction = 0.75); and 8 and 9 (fraction = 1.0). The timing overhead with and without sample selection is shown in Table 10.

Table 2: Hypervolume and Spread Results (No Sampling): TabNet Regressor and TabNet Embedding + XGBoost

Dataset / Metric	TabNet Regressor				TabNet Embedding + XGBoost			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	8.58E+03	5.50E+03	9.41E+03	2.87E+09	0.00E+00	2.56E+08	2.56E+08	4.65E+08
PM100 – Spread	9.60E+04	8.40E+03	2.57E+05	9.53E+04	0.00E+00	1.34E+05	1.34E+05	3.21E+05
Adastra – Hypervolume	3.92E+06	3.91E+06	3.91E+06	1.06E+14	0.00E+00	4.81E+12	4.81E+12	4.97E+11
Adastra – Spread	1.42E+05	2.61E+04	1.43E+05	1.42E+05	0.00E+00	2.57E+07	2.57E+07	3.79E+07

The figures 2 and 3 show the convergence of surrogates across different models.

Table 3: Hypervolume and Spread Results (No Sampling): TabNet Embedding + LightGBM and RF

Dataset / Metric	TabNet Embedding + LightGBM				TabNet Embedding + RF			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	0.00E+00	2.73E+08	2.73E+08	4.51E+08	0.00E+00	2.61E+08	2.61E+08	4.52E+08
PM100 – Spread	0.00E+00	1.43E+05	1.43E+05	3.29E+05	0.00E+00	1.35E+05	1.35E+05	3.24E+05
Adastra – Hypervolume	0.00E+00	6.50E+12	6.50E+12	3.94E+12	1.16E+04	5.96E+12	5.96E+12	6.09E+12
Adastra – Spread	0.00E+00	3.48E+07	3.48E+07	3.54E+07	7.98E+04	3.15E+07	3.15E+07	3.71E+07

Table 4: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 0.5): TabNet Regressor and TabNet Embedding + XGBoost

Dataset / Metric	TabNet Regressor				TabNet Embedding + XGBoost			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	1.34E+10	1.34E+10	1.30E+10	2.05E+12	0.00E+00	2.73E+08	2.73E+08	4.65E+11
PM100 – Spread	1.60E+08	3.69E+06	3.78E+06	1.28E+08	0.00E+00	1.41E+05	1.41E+05	4.65E+11
Adastra – Hypervolume	1.60E+06	1.43E+06	6.39E+04	6.13E+13	0.00E+00	6.49E+12	6.49E+12	4.20E+12
Adastra – Spread	2.79E+04	5.76E+03	5.67E+03	2.85E+04	0.00E+00	3.48E+07	3.48E+07	2.00E+07

The figures 4, 5, and 6 show evaluation results across two HPC datasets, showing surrogate quality and trade-off performance,

Licensing of Assets. The PM100 dataset [10] is distributed under the **Creative Commons Attribution 4.0 (CC BY 4.0)** license, as noted in the SC'23 proceedings. The Adastra system documentation and log data [11] are publicly described by CINES/GENCI, but no explicit license is specified; we assume standard institutional terms for research use. The Stampede3 user guide [12] is published on the TACC documentation portal and made available under TACC's usage policy; no formal open license is displayed.

Table 5: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 0.5): TabNet Embedding + LightGBM and RF

Dataset / Metric	TabNet Embedding + LightGBM				TabNet Embedding + RF			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	0.00E+00	2.51E+08	2.51E+08	4.61E+11	0.00E+00	2.75E+08	2.75E+08	4.56E+11
PM100 – Spread	0.00E+00	1.38E+05	1.38E+05	1.02E+06	0.00E+00	1.42E+05	1.42E+05	9.99E+05
Adastra – Hypervolume	0.00E+00	6.18E+12	6.18E+12	4.69E+12	1.43E+07	6.26E+12	6.26E+12	4.21E+12
Adastra – Spread	0.00E+00	3.32E+07	3.32E+07	2.08E+07	1.44E+04	3.35E+07	3.35E+07	1.99E+07

Table 6: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 0.75): TabNet Regressor and TabNet Embedding + XGBoost

Dataset / Metric	TabNet Regressor				TabNet Embedding + XGBoost			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	1.15E+03	1.15E+03	1.15E+03	2.58E+09	0.00E+00	2.49E+08	2.49E+08	4.69E+11
PM100 – Spread	3.34E+02	7.73E+01	2.86E+02	3.31E+02	0.00E+00	1.29E+05	1.29E+05	9.87E+05
Adastra – Hypervolume	1.05E+02	1.05E+02	1.05E+02	1.54E+16	0.00E+00	6.02E+12	6.02E+12	4.17E+13
Adastra – Spread	4.82E+02	4.82E+02	4.82E+02	0.00E+00	0.00E+00	3.22E+07	3.22E+07	4.75E+07

Table 7: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 0.75): TabNet Embedding + LightGBM and RF

Dataset / Metric	TabNet Embedding + LightGBM				TabNet Embedding + RF			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	0.00E+00	2.70E+08	2.70E+08	4.58E+11	0.00E+00	2.78E+08	2.78E+08	4.56E+11
PM100 – Spread	0.00E+00	1.36E+05	1.36E+05	1.01E+06	0.00E+00	1.44E+05	1.44E+05	9.99E+05
Adastra – Hypervolume	0.00E+00	6.34E+12	6.34E+12	2.55E+13	3.44E+03	6.26E+12	6.26E+12	3.56E+13
Adastra – Spread	0.00E+00	3.40E+07	3.40E+07	4.03E+07	3.39E+04	4.77E+07	4.77E+07	3.33E+07

Table 8: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 1): TabNet Regressor and TabNet Embedding + XGBoost

Dataset / Metric	TabNet Regressor				TabNet Embedding + XGBoost			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	4.29E+04	3.47E+04	4.36E+04	9.25E+10	1.38E+02	2.55E+08	2.56E+08	4.63E+11
PM100 – Spread	4.10E+05	2.17E+04	4.72E+05	3.91E+05	2.65E+03	1.30E+05	1.30E+05	8.04E+05
Adastra – Hypervolume	4.01E-02	2.93E-02	2.93E-02	2.49E+13	3.31E+03	1.18E+13	1.18E+13	1.24E+14
Adastra – Spread	4.70E-01	4.70E-01	4.70E-01	0.00E+00	4.03E+04	6.31E+07	6.31E+07	5.62E+07

Table 9: Hypervolume and Spread Results with Intelligent Sample Selection (fraction = 1): TabNet Embedding + LightGBM and RF

Dataset / Metric	TabNet Embedding + LightGBM				TabNet Embedding + RF			
	MOBO	SOBO (Runtime)	SOBO (Power)	Random	MOBO	SOBO (Runtime)	SOBO (Power)	Random
PM100 – Hypervolume	1.38E+02	2.84E+08	2.84E+08	4.30E+11	0.00E+00	2.70E+08	2.70E+08	4.55E+11
PM100 – Spread	0.00E+00	1.46E+05	1.46E+05	7.81E+05	0.00E+00	1.40E+05	1.40E+05	8.12E+05
Adastra – Hypervolume	3.31E+03	1.99E+13	1.99E+13	1.91E+14	1.58E+04	6.54E+12	6.54E+12	8.23E+12
Adastra – Spread	3.30E+00	1.08E+08	1.08E+08	9.10E+07	6.42E+05	3.50E+07	3.50E+07	3.69E+07

Table 10: Comparison of timing overhead (seconds) for PM100 and Adastra datasets with and without intelligent sampling (fraction = 1).

Step	PM100		Adastra	
	No Sampling	With Sampling (Fraction = 1)	No Sampling	With Sampling (Fraction = 1)
Preprocessing	10.82	6.36	0.80	0.23
Runtime Model	2359.40	1264.97	237.73	68.02
Power Model	2002.76	959.08	183.99	53.62
Preproc. MOBO	10.20	6.53	0.64	0.23
MOBO	1883.77	3633.60	2065.70	1642.01
SOBO Runtime	104.86	71.11	120.39	95.58
SOBO Power	108.97	67.81	90.18	101.99
TOTAL	6480.78	6009.46	2699.43	1961.68

Table 11: Common experimental settings used across both pipelines (Adastra; PM100 uses the same settings unless noted).

Setting	**Value / Notes**
Datasets	Adastra (15 days log trace); PM100 (SC-W'23)
Sampling fractions	0.5, 0.75, 1.0 (active-learning guided)
Design variable	`num_nodes_alloc`
MOBO acquisition	`qLogExpectedHypervolumeImprovement` (`logEHVI`)
MOBO iterations	300 (`q`=1 candidate/step; `optimize_acqf`: 5 restarts, 32 raw samples)
GP model	Multi-output GP (BoTorch), reference point inferred online
Baselines	SOBO (runtime-only), SOBO (power-only), Random (5 seeds; split budget)
Random baseline	`N_RANDOM_SEEDS`=5; total points $\approx$ MOBO budget (evenly split per seed)
Figure naming	"Transformer-based model" in figures denotes TabNet regressor (text aligned accordingly)

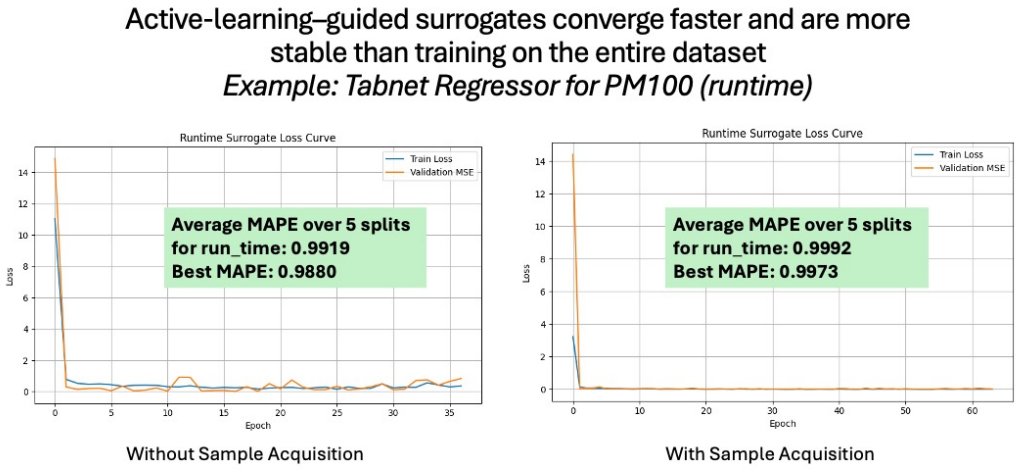

Figure 2: Runtime surrogate convergence across different models.

Table 12: Pipeline-specific training and model hyperparameters. PM100 uses the same settings; only the dataset changes.

Component	**Regressor Pipeline (TabNet only)**	**Embeddings Pipeline (TabNet + RF/XGB/LGBM)**
TabNet training (fit parameters)		
Max epochs	1000	100
Early stopping	Patience = 20	Patience = 10
Batch size	1024	1024
Virtual batch size	128	128
Eval metric	MSE (valid)	RMSE (valid)
Seed	`seed` (set in notebook)	`random_state=42` (downstream); TabNet uses default seed unless overridden
Target transforms	Robust/log transforms supported; inverse-transform for eval	Same; targets transformed once, meta stored for inverse transforms
Downstream regressors (used only in embeddings pipeline)		
Random Forest	—	`n_estimators=100`, `max_depth=10`, `random_state=42`, `n_jobs=-1`
XGBoost	—	`n_estimators=100`, `max_depth=6`, `learning_rate=0.10`, `random_state=42`
LightGBM	—	`n_estimators=100`, `max_depth=6`, `learning_rate=0.10`, `random_state=42`
MOBO loop (both pipelines)		
Acquisition	`qLogExpectedHypervolumeImprovement` (logEHVI), Sobol sampler with 128 normal samples	
Optimization	`optimize_acqf`: 5 restarts, 32 raw samples; $q = 1$	
Reference point	Inferred dynamically from current Y (minimization space with negated objectives)	
Pareto metrics	Hypervolume, Spread (raw minimization space; signs handled consistently)	

Table 13: Effect of Active Learning (sampling fractions) on surrogate accuracy and optimization metrics.

Dataset	**Fraction**	**Train Size**	**MAPE**	**Observed Effect (HV/Spread)**
PM100	50%	73,983	≈0.99	HV stable; Spread drops from 2.5×10^5 to $< 10^4$
	75%	77,278	≈0.99	Accuracy plateaus; HV/Spread stable
	100%	109,202	≈0.99	No further gains vs. 75%
Adastra	50%	3,964	≈0.99	HV stable; erratic values ($10^{16} \rightarrow 10^{13}$) suppressed
	75%	4,163	≈0.99	Accuracy plateaus; smoother trade-offs
	100%	4,547	≈0.99	No further gains vs. 75%

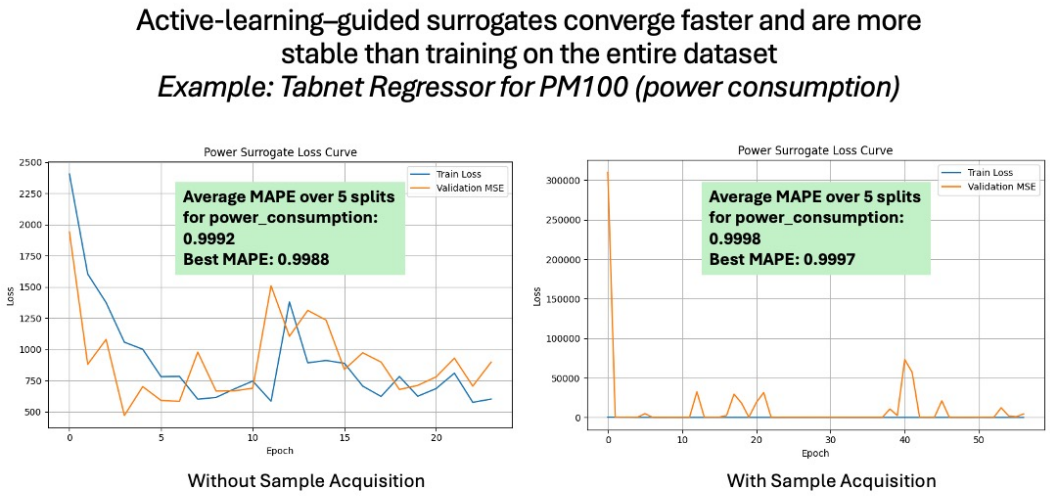

Figure 3: Power surrogate convergence across different models.

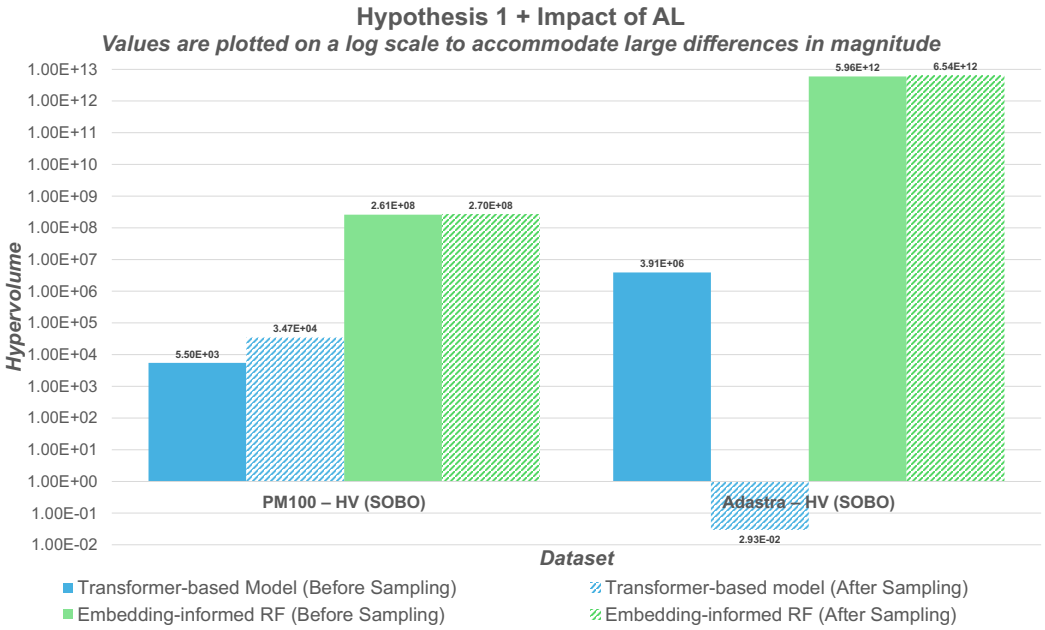

Figure 4: Embedding-informed surrogates yield higher HV than TabNet models on PM100 and Adastra.

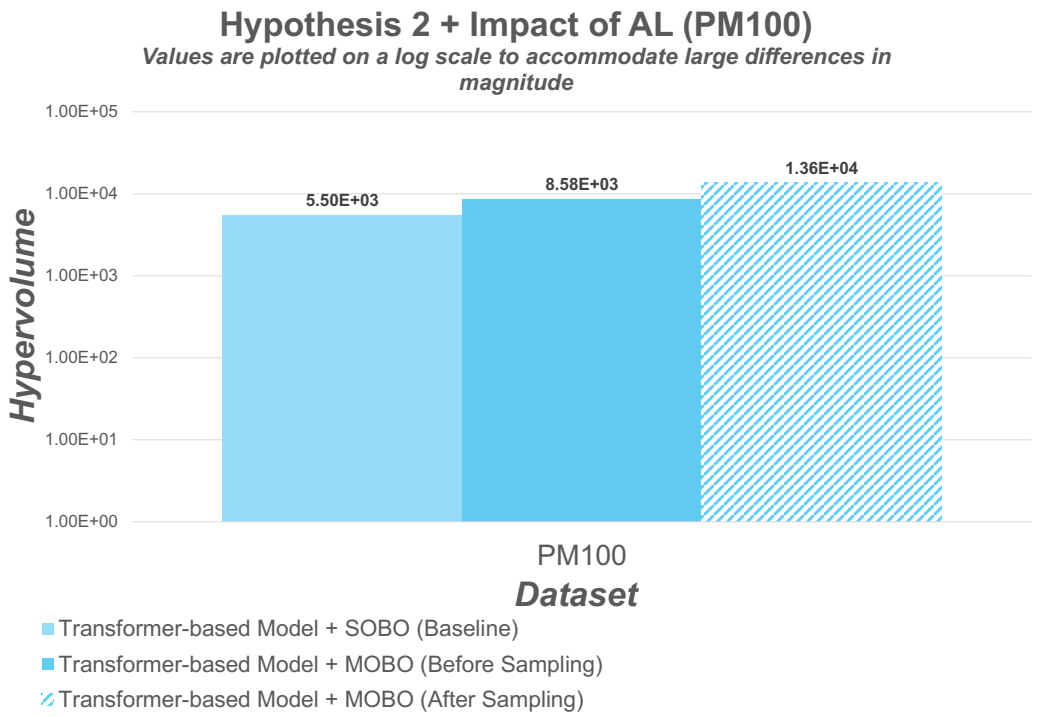

Figure 5: MOBO achieves higher HV than SOBO on PM100, revealing better runtime–power trade-offs.

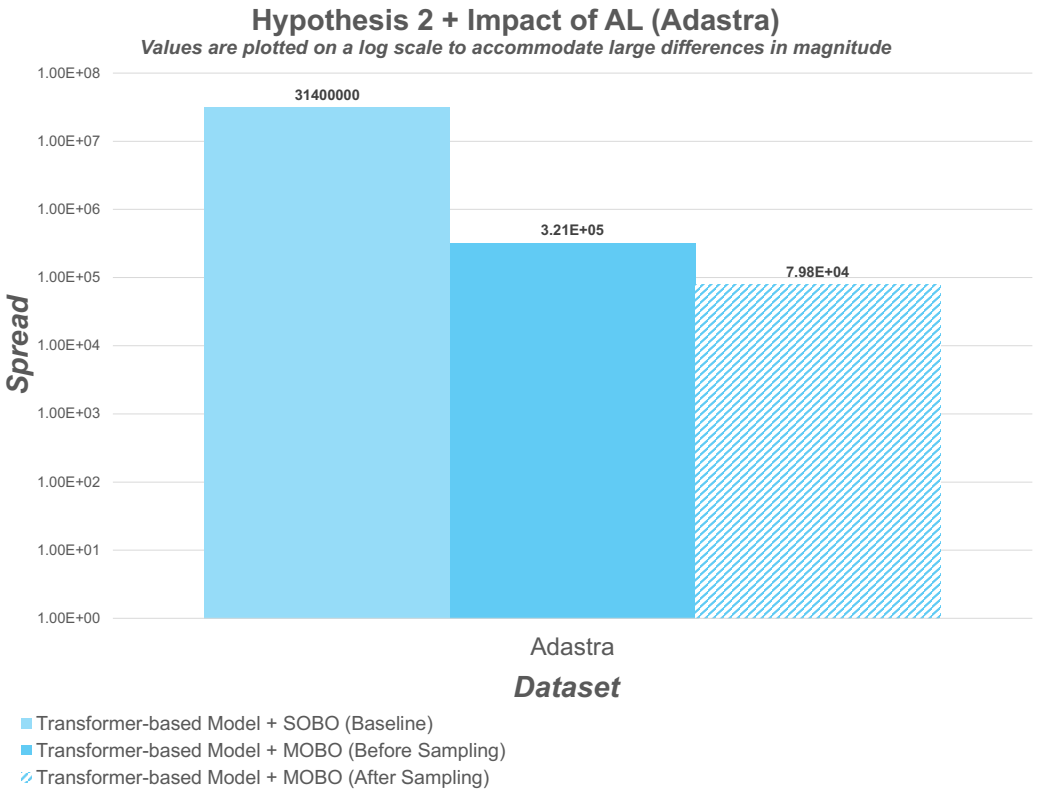

Figure 6: MOBO achieves lower Spread than SOBO on Adastra, indicating more balanced Pareto fronts.