Transfer Learning Workflow for I/O Bandwidth Prediction

HPC I/O in the Data Center Workshop 2023

Dmytro Povaliaiev

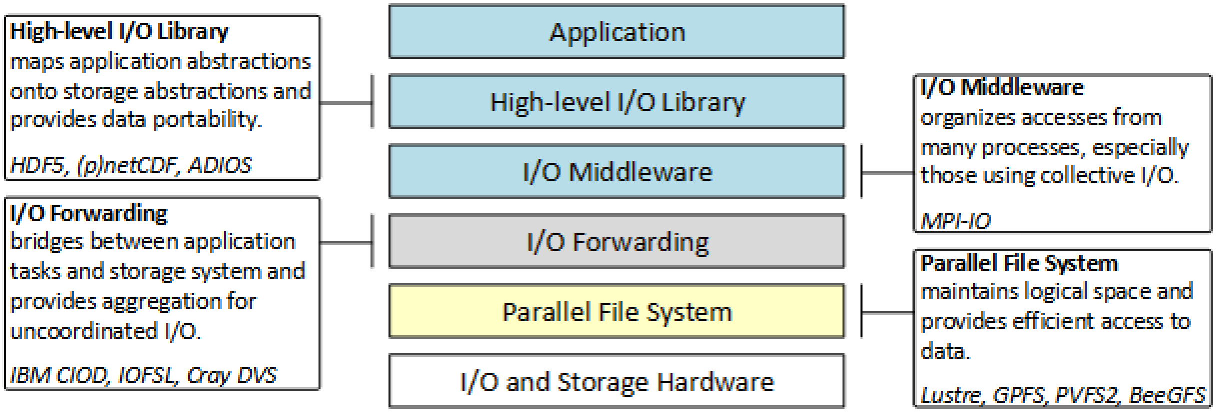
Outline

1. Background & motivation
2. Proposed workflow
    1. Preprocessing the logs
    2. Cleaning the data
    3. Training the source model
    4. Validating the results
    5. Results of the initial training
    6. Fine-tuning the model on the target dataset
    7. Results of fine-tuning
    8. Comparison of results between the transfer learning stages
    9. Preliminary results using data from ALCF Theta
3. What did the model learn?
    1. Explainable AI methods
    2. Top 10 most important features
4. Conclusion

Background & motivation

Why should we predict the I/O bandwidth of the jobs on the cluster?

- Useful for optimizing performance & efficiency
  - Identify performance anomalies
  - Tune the filesystem
  - Make better hardware procurement decisions
  - Potentially implement hardware optimizations (lower energy usage etc.)



Extracted from [1]

- BUT requires a lot of data from the specific cluster
  - Need to set up a monitoring & processing pipeline
  - Takes a lot of time to collect a significant amount

Background & motivation

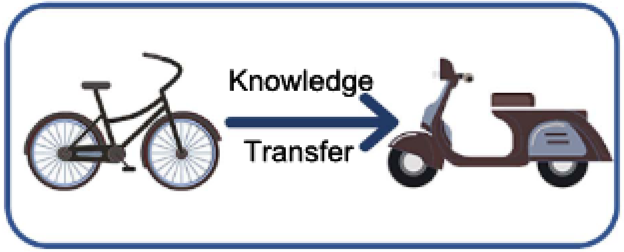
Transfer Learning for I/O Prediction

- Relies on the assumption that „different clusters might exhibit similar I/O characteristics“
  - Same filesystems
  - Same I/O APIs such as POSIX, MPI-IO, etc.
  - Similar applications (e.g. computational fluid dynamics or biomedical simulations)

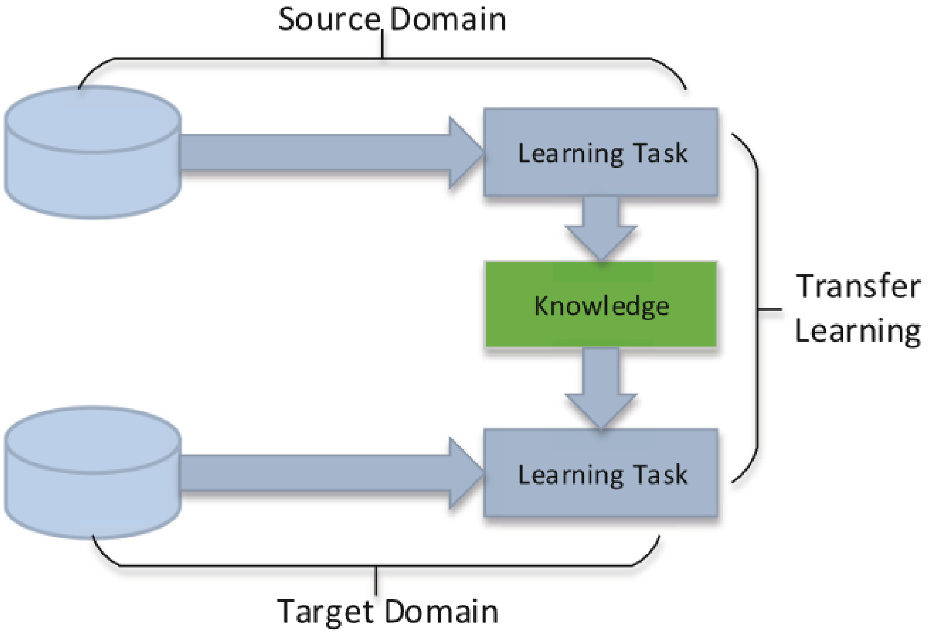
- Use an already existing dataset from another cluster
  - Years of I/O performance data
  - Real-life application runs

- Fine-tune on a small dataset collected at the target installation
  - Relatively short time to gather the data
  - Might work as a Proof-of-Concept for hardware procurement

Transfer Learning: The Idea



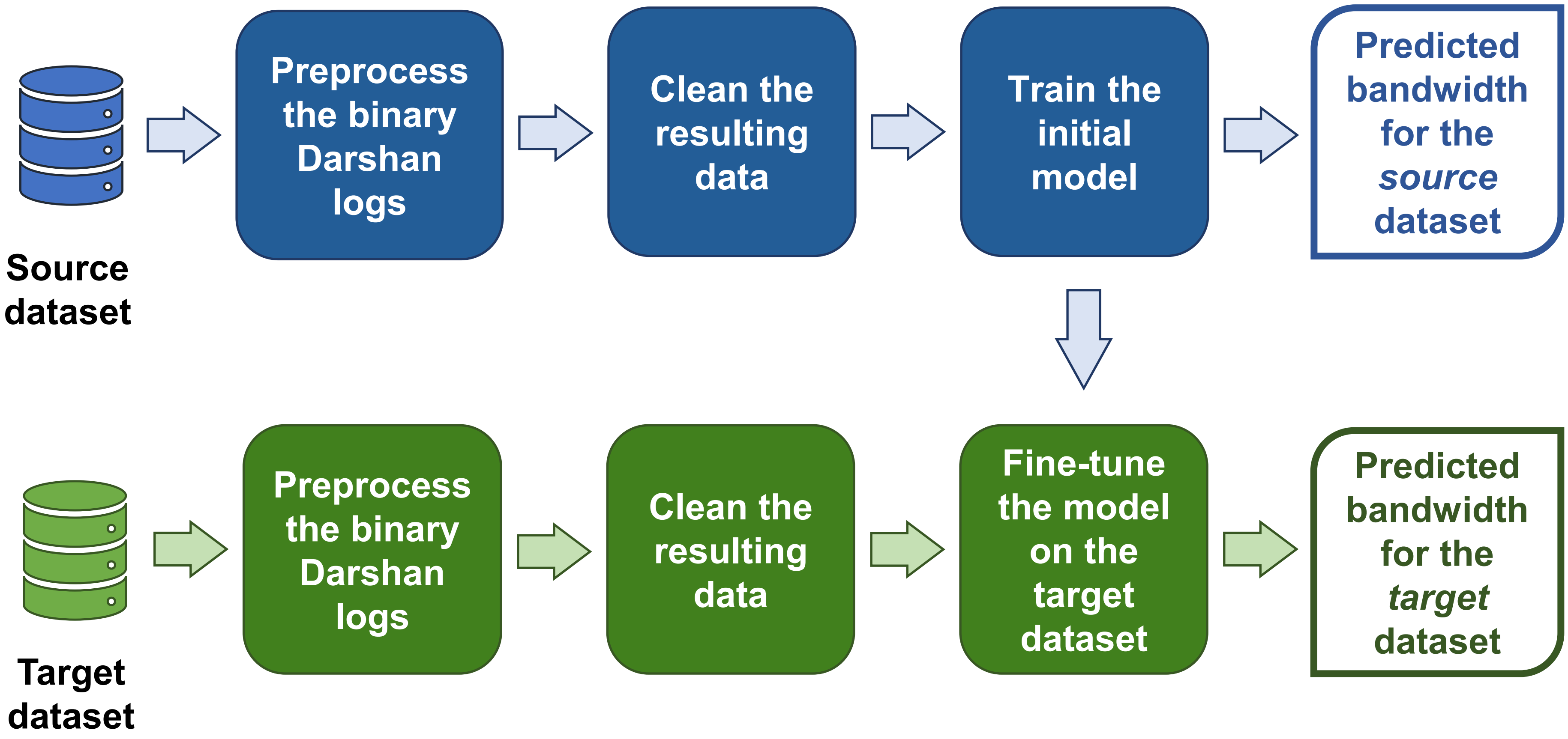
Extracted from [2]



Extracted from [3]

Try to predict the I/O bandwidth of a specific job on a specific cluster, based on the observations from another cluster

Proposed workflow

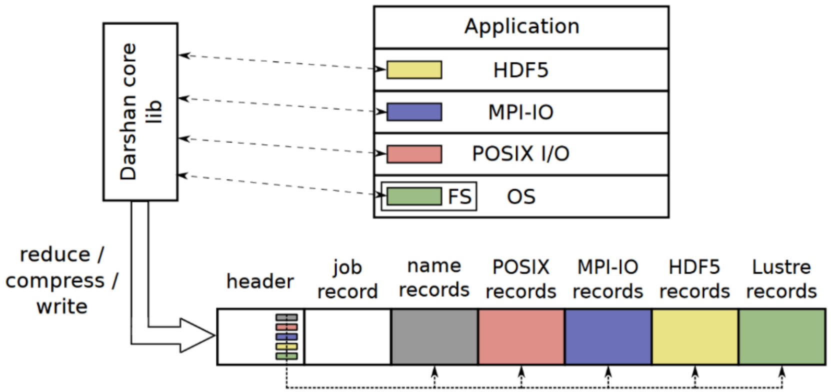


Preprocessing the binary Darshan logs

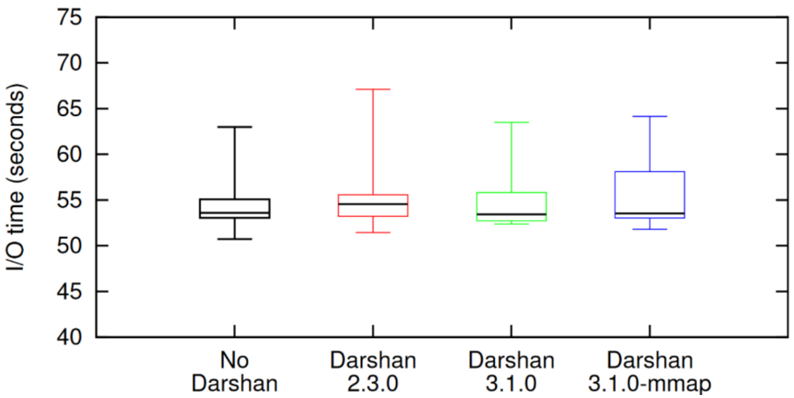
Why Darshan?

- Developed at the Argonne National Lab
- A well-known tool for the I/O performance measurement & shown to be reliable
- Minimal influence on the applications‘ I/O time (less than 3% [19])
- Binary log format allows storing significant amounts of performance data

- Several large-scale public datasets are already available



Overview of Darshan’s architecture & log format [14]



Darshan’s influence on I/O time of the instrumented application [14]

Preprocessing the binary Darshan logs

Prediction target design

- Focus on the POSIX module for now
  - De facto standard for I/O operations on Unix-like filesystems
  - MPI-IO, HDF5, and other APIs are implemented on top of it
    - Their calls are reflected in the corresponding POSIX ops counters [18]
  - Potentially more data, as using MPI-IO requires POSIX, but not vice versa [18]
  - Existing body of work to compare against

- Parse the binary logs using PyDarshan
  - Python module from the authors of Darshan
  - Provides a summary of sizes, times, the I/O histogram, and so on

  - Does not calculate the bandwidth by default → must be done separately

Preprocessing the binary Darshan logs

Datasets (both collected at the Lustre filesystem)

- **Blue Waters (source dataset)**
  - Gathered during 2012-2021 at the University of Illinois
  - More than 4.65 mln individual files
  - The subset used contains ~690k records
    - Not all logs contain POSIX performance data
    - PyDarshan supports only logs recorded with v3.21+

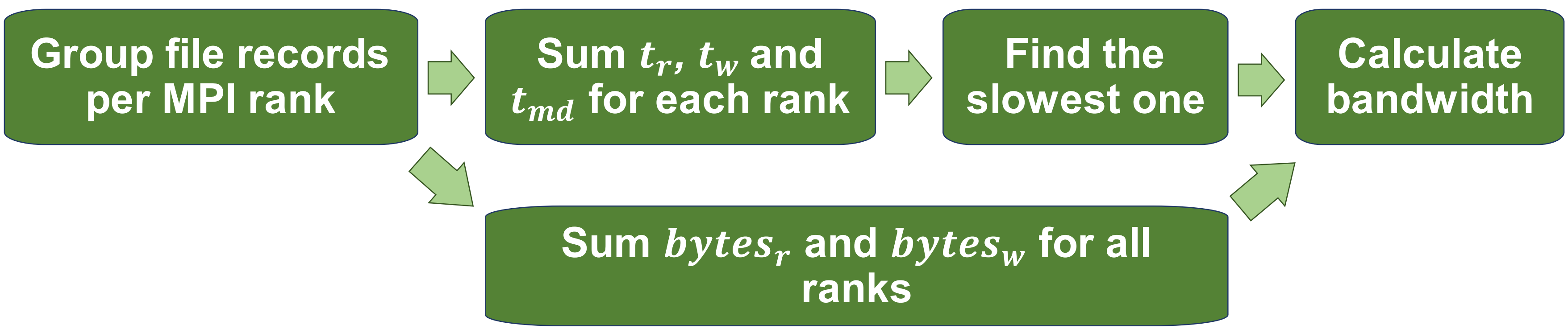
- **CLAIX (target dataset)**
  - Data from several applications:
    - C-Class NAS Parallel Benchmark from NASA
      - 4, 9, 16, 64-process variants
    - Ciao - 48, 144, 162, 240 processes
    - Quantum Espresso was considered, but removed due to the very high variance it introduced
  - Limited size

Preprocessing the binary Darshan logs

How to calculate the bandwidth?

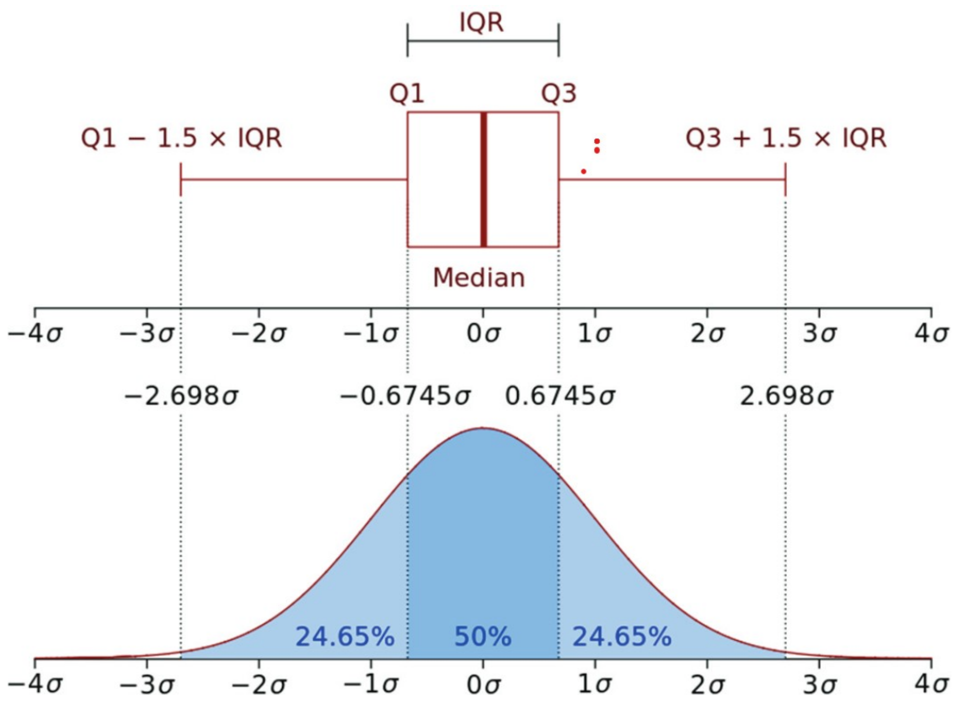
$$MiB/s = \left(\frac{\sum_{rank=0}^{n-1} (bytes_r + bytes_w)}{\max_{rank=0}^{n-1} (t_{md} + t_r + t_w)} \right)$$

Darshan’s bandwidth formula [14]



Bandwidth calculation workflow for an individual Darshan log

Cleaning the resulting data



The IQR and its projection on a normally distributed density [15]

- High number of outliers causes problems with model convergence
  - Three-stage removal process
    - Eliminate erroneous items, e.g., with negative times (similar to [18])
    - Remove all-zero features
    - Apply the Interquartile Range (IQR) method to the rest

What is the input?

Darshan job summary (by PyDarshan)

- 96 different POSIX counters + **# of processes:**
  - **Times:**
    - POSIX_F_READ_TIME, POSIX_F_WRITE_TIME …
    - POSIX_F_SLOWEST_RANK_TIME …
  - **Sizes:**
    - POSIX_BYTES_WRITTEN, POSIX_BYTES_READ
    - POSIX_SLOWEST_RANK_BYTES, POSIX_FASTEST_RANK_BYTES
    - 4 most frequently appearing **access** sizes & **strides**
  - **Ops counts:**
    - POSIX_OPENS, POSIX_SEEKS, POSIX_STATS …
    - POSIX_CONSEC_READS, POSIX_CONSEC_WRITES …
    - 4 most frequently appearing **access** sizes & **strides**
  - **I/O histogram**
    - Number and total size of read/write ops split into brackets:
      - 0-100B, 100B-1KB, …, 1GB+
  - **Alignments (file & memory)**
  - **Read/write switches**
  - **POSIX mode**
  - **Offsets etc.**

Training the source model

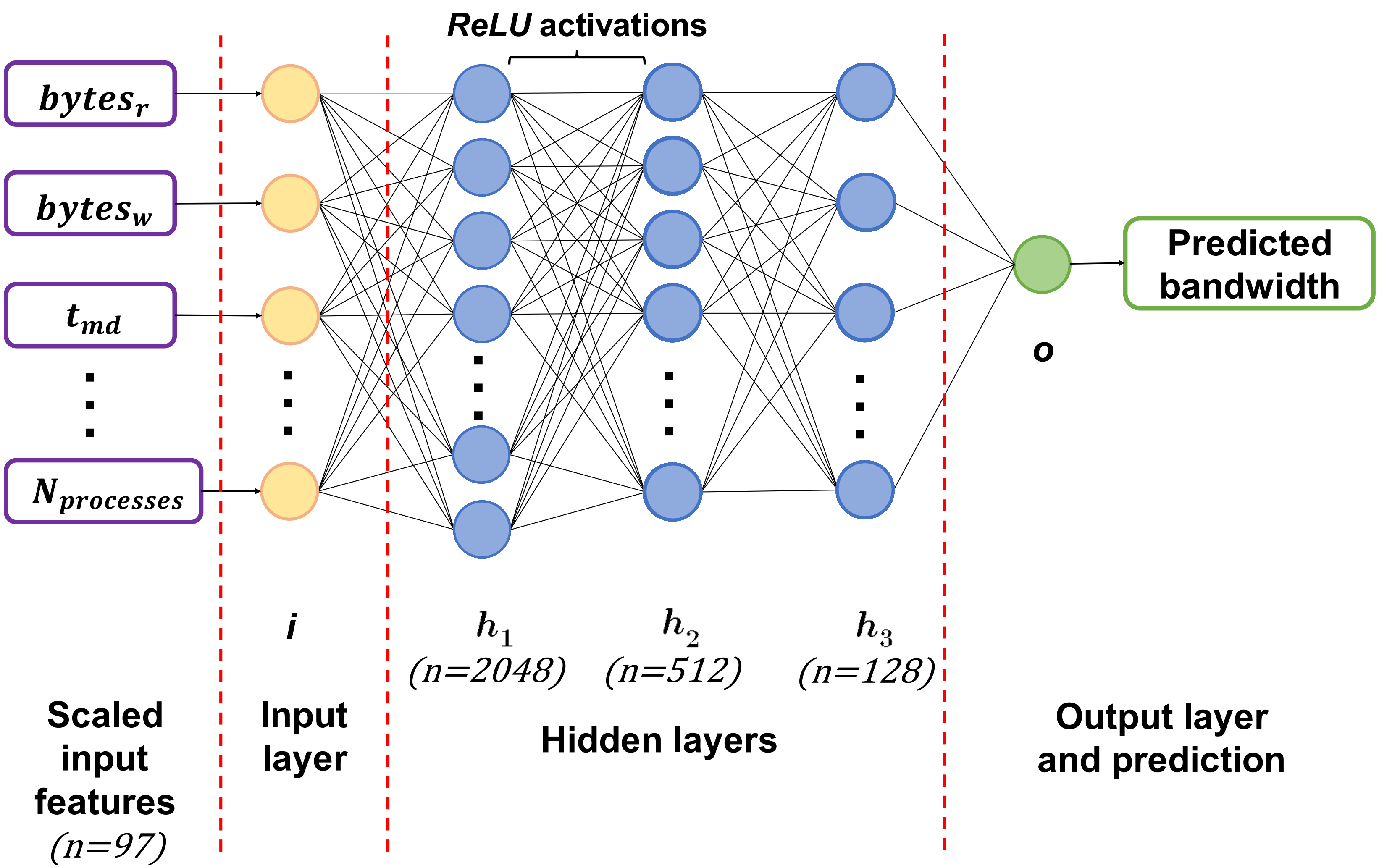
- Architecture: Multi-Layer Perceptron
  - Mathematically proven universal approximator [16]
  - No structure of the features to rely on for a CNN
  - No time series → not well-suited for an RNN

  - Efficient: total time to train ~60 mins

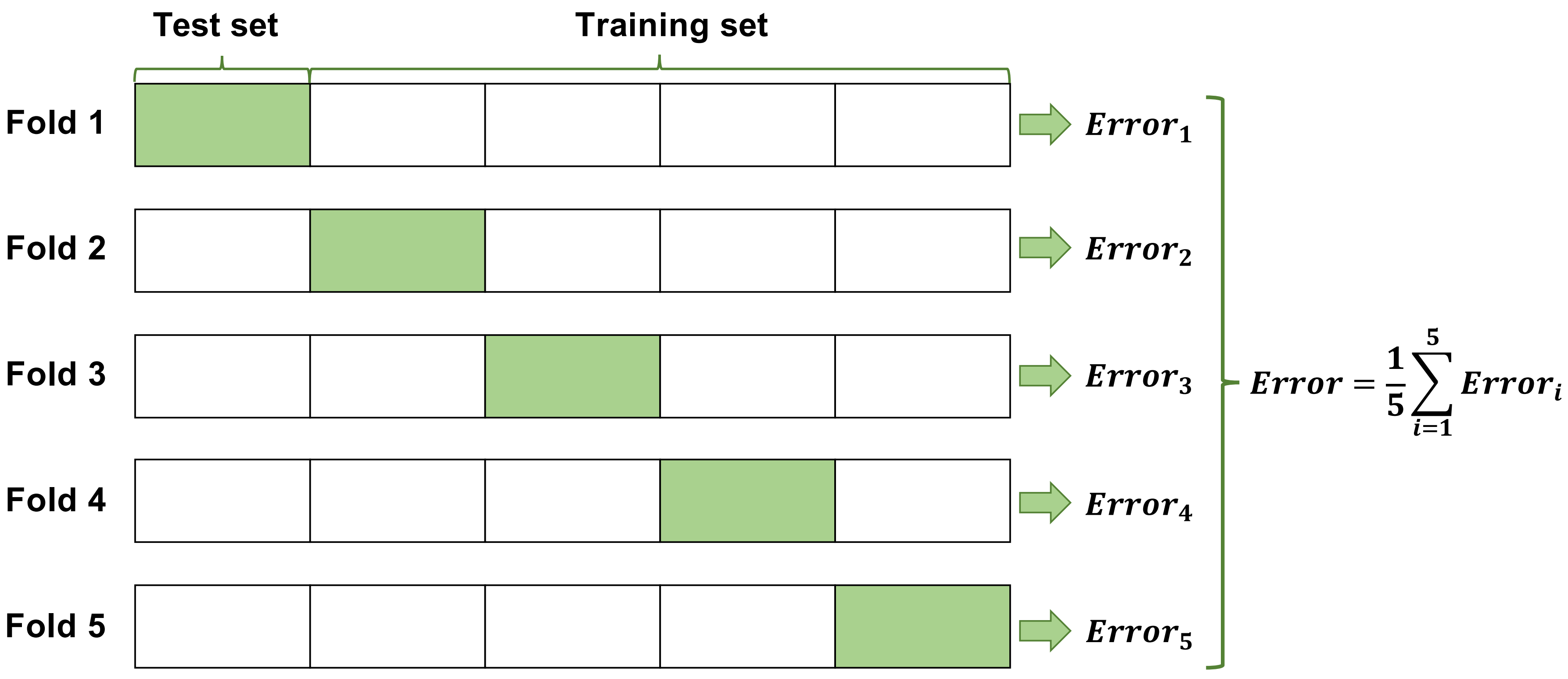
- 2 different sets of the training data:
  - Full dataset
  - Subset with the number of processes per job that appears at least once in the target data

  - Motivation: some of the jobs in the Blue Waters dataset would be physically impossible to run on the CLAIX cluster
    - The model does not need to generalize to them
    - Try to focus on more realistic data → potentially better performance

Neural network architecture

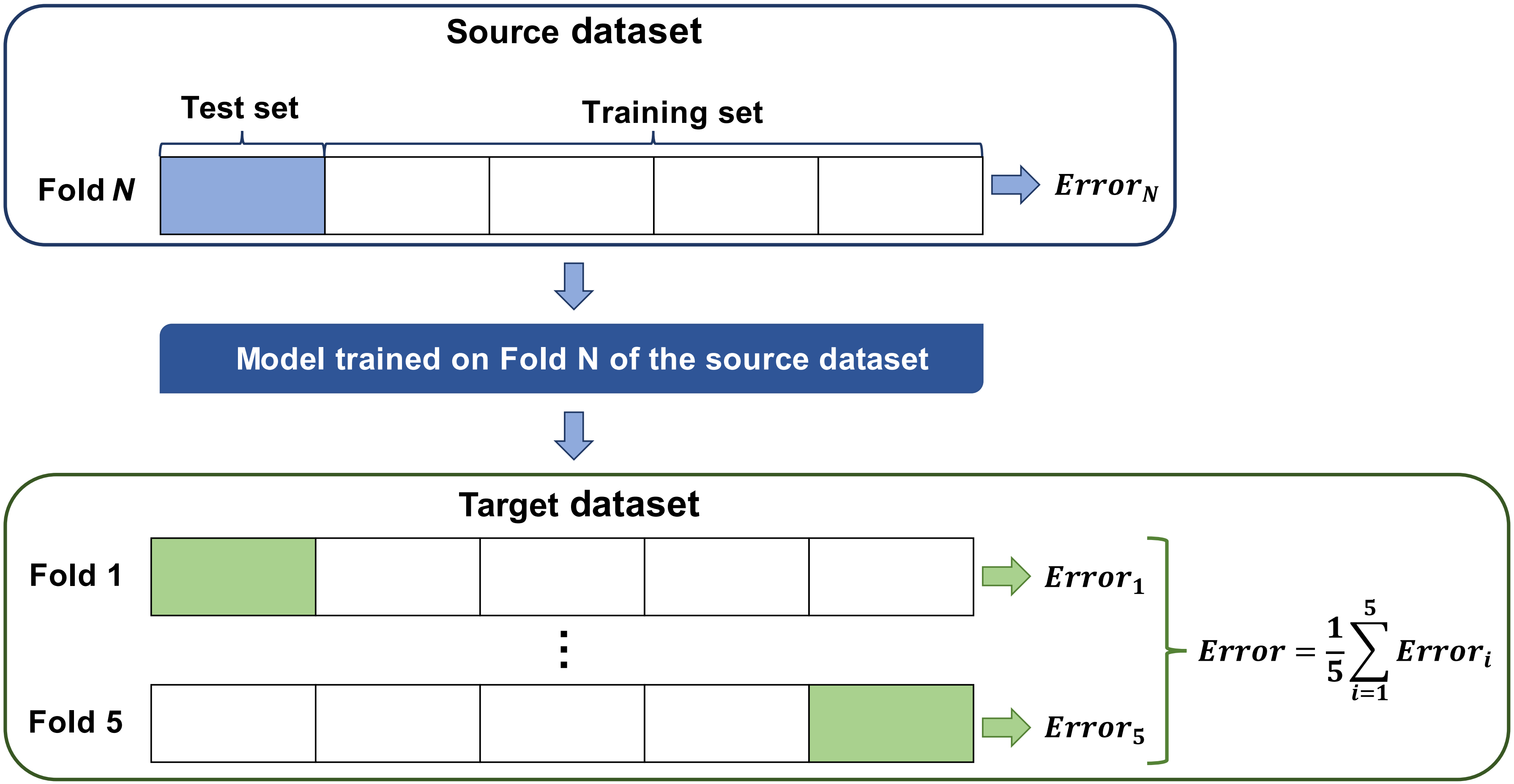


Validating the results – Initial training



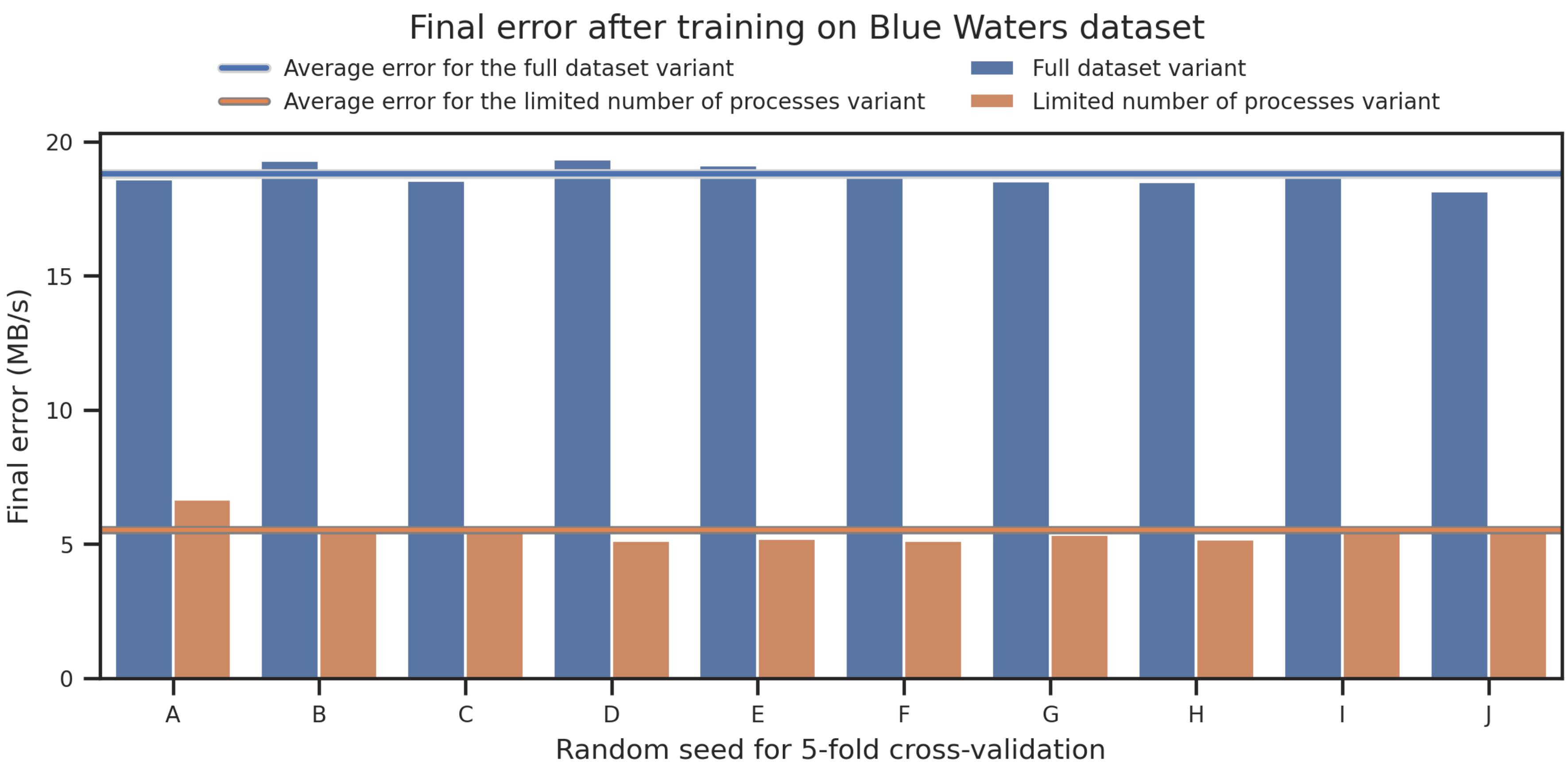
The principle of 5-fold cross-validation

Validating the results – Transfer learning



Cross-validation of the transfer learning

Results of the initial training on the Blue Waters dataset



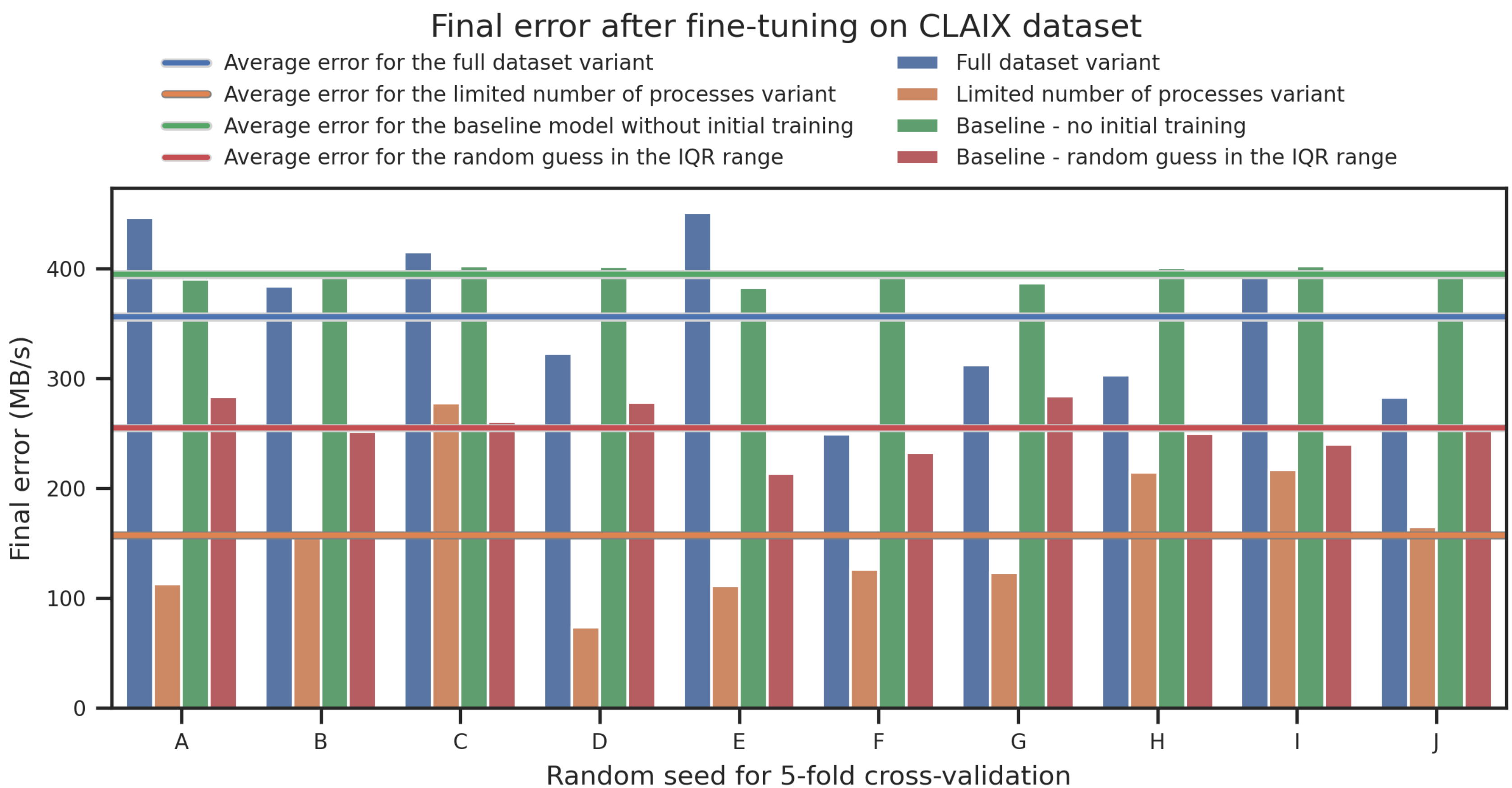
Variant	Full dataset	Limited # of processes	Random guess in IQR
MAE (*n=50*)	18.81 MB/s	**5.53 MB/s**	155.4 MB/s

Fine-tuning the model

- All models were fine-tuned using the same network-based transfer learning setup
  - Weights of the output layer reset
  - All layers unfrozen
  - Trained for 1200 epochs (vs 600 on the source dataset)

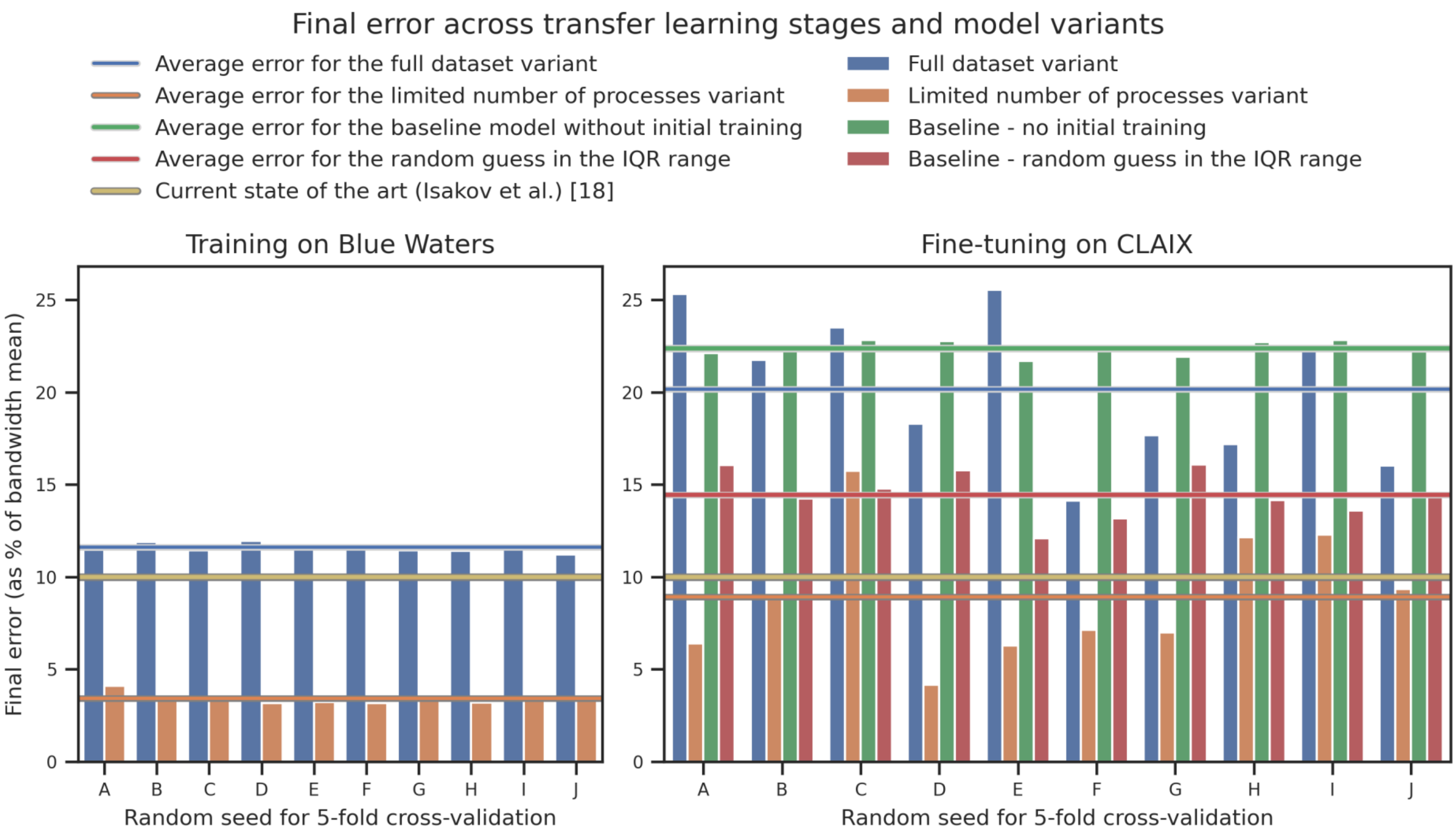
  - Fine-tuning time: <1 min on P100 GPU, ~6 mins on an Intel CPU
    - Very low resource requirements

Results after fine-tuning on the data from CLAIX



Variant	Full dataset	Limited # of processes	No initial training	Random guess in IQR
MAE	355.92 MB/s	**157.44 MB/s**	394.67 MB/s	254.75 MB/s

Comparison of results between the transfer learning stages

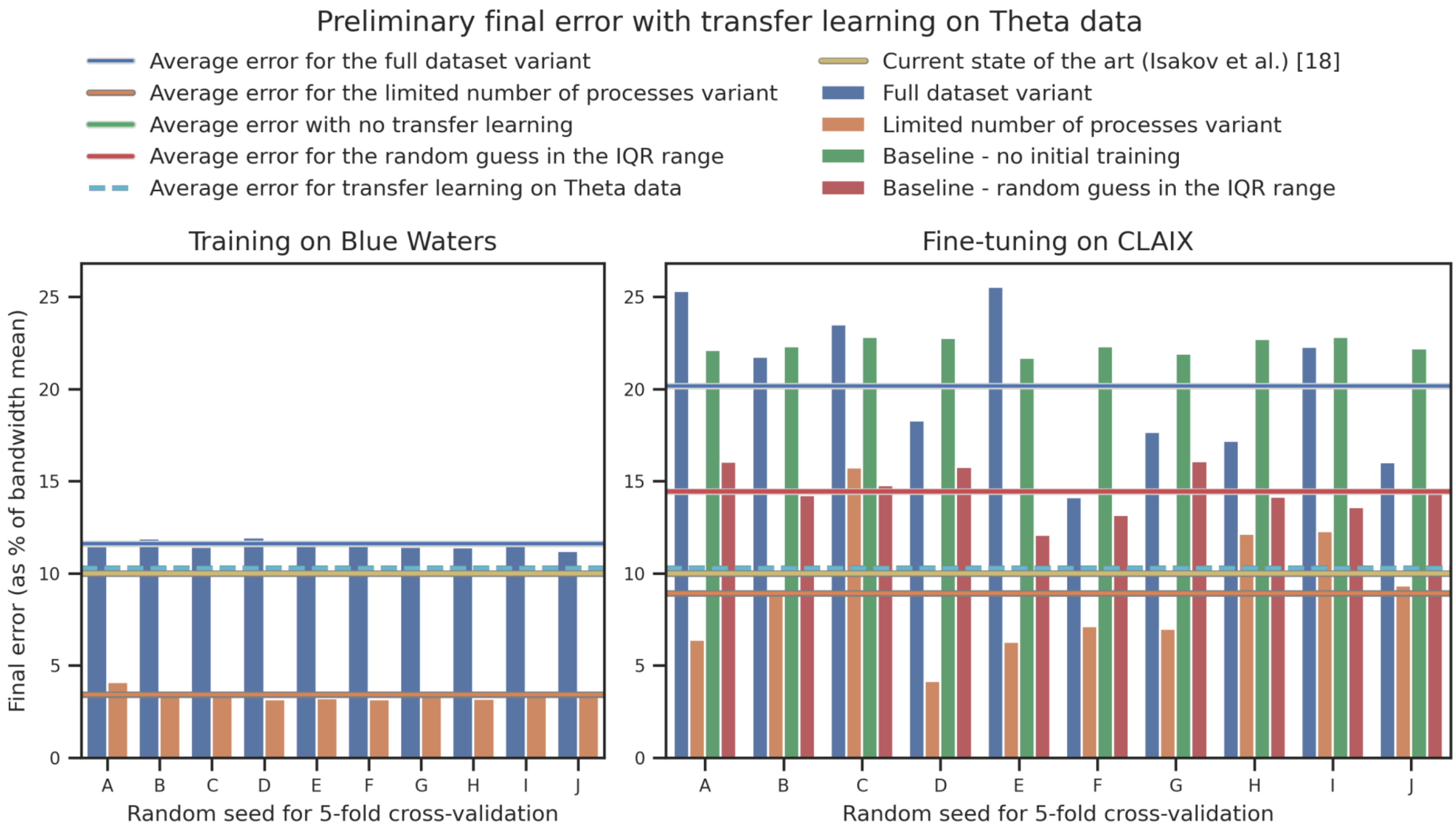


Comparison of results between the transfer learning stages (cont.)

Final errors across all stages and variants (as % of the mean bandwidth)

Variant	Initial training	Fine-tuning
Full dataset	11.6%	20.1%
Limited number of processes	**3.4%**	**8.92%**
Random guess in the IQR	95.9%	14.4%
No initial training	-	22.4%
Current state of the art (Isakov et al.) [18]	10%	10%

Preliminary results using data from ALCF Theta



What did the model learn?

- Explainable AI lets us “take a look into the black box”
- Idea: Attribute importance to the features

- Multiple approaches available:
  - Integrated Gradients [4] (with NoiseTunnel [5])
  - DeepLift [6]
  - Feature Ablation [7]
  - Shapley Value Sampling [8, 9]
  - Guided Backpropagation [10]
  - Feature Permutation [11]
  - InputXGrad [12]
  - Saliency [13]

- Use **all the approaches** above to cross-compare the attributions

Top 10 most important features

$$MiB/s = \left(\frac{\sum_{rank=0}^{n-1} (bytes_r + \boxed{bytes_w})}{\boxed{\max_{rank=0}^{n-1} (t_{md} + t_r + t_w)}} \right)$$

- **Times:**
  - POSIX_F_READ_TIME
  - POSIX_F_META_TIME
  - POSIX_TOTAL_TIME
  - POSIX_F_MAX_READ_TIME
  - POSIX_F_WRITE_TIME
  - POSIX_F_SLOWEST_RANK_TIME
- **Sizes:**
  - POSIX_ACCESS2_ACCESS
  - POSIX_SLOWEST_RANK_BYTES
  - POSIX_MAX_READ_TIME_SIZE
  - POSIX_BYTES_WRITTEN
- **# of processes**

Conclusion

- The proposed workflow is shown to work in the proof-of-concept form
  - Cross-validation results are mostly stable for both clusters
  - Explainable AI identifies the features considered by Darshan crucial for the bandwidth as the most important ones for the model
  - The results imply the produced models can outperform the current state of the art
- Several aspects require additional work in the future
  - Verify the workflow using data from MPI-IO, HDF5, and other common I/O APIs
  - Try to target different filesystems (e.g., BeeGFS)
  - Increase the diversity of applications in the target dataset
  - Evaluate MAPE as the measurement of model accuracy
    - Has its own drawbacks → try to use it as a part of a two-component error function:
      - MAE for the low-bandwidth jobs
      - MAPE for the high-bandwidth jobs
  - Test the proposed workflow on the data from additional clusters
  - Experiment with alternative outlier removal techniques or the ways to increase the robustness of the models to outliers
  - Use additional FS information to make more informed predictions
  - Remove all the time-based features & try to predict the execution time for a job

Thank you for your attention!

More details:

https://publications.rwth-aachen.de/record/958007

References

[1] S. Neuwirth and A. K. Paul, 'Parallel I/O Evaluation Techniques and Emerging HPC Workloads: A Perspective', in 2021 IEEE International Conference on Cluster Computing (CLUSTER), Sep. 2021, pp. 671–679. doi: 10.1109/Cluster48925.2021.00100.

[2] F. Zhuang et al., 'A Comprehensive Survey on Transfer Learning', Proceedings of the IEEE, vol. 109, no. 1, pp. 43–76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

[3] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, 'A Survey on Deep Transfer Learning', in Artificial Neural Networks and Machine Learning – ICANN 2018, Cham, 2018, pp. 270–279. doi: 10.1007/978-3-030-01424-7_27.

[4] M. Sundararajan, A. Taly, and Q. Yan, 'Axiomatic attribution for deep networks', in Proceedings of the 34th International Conference on Machine Learning - Volume 70, Sydney, NSW, Australia, Aug. 2017, pp. 3319–3328.

References

[5] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, 'SmoothGrad: removing noise by adding noise'. arXiv, Jun. 12, 2017. doi: 10.48550/arXiv.1706.03825.

[6] A. Shrikumar, P. Greenside, and A. Kundaje, 'Learning important features through propagating activation differences', in Proceedings of the 34th International Conference on Machine Learning - Volume 70, Sydney, NSW, Australia, Aug. 2017, pp. 3145–3153.

[7] L. Merrick, 'Randomized Ablation Feature Importance'. arXiv, Oct. 01, 2019. doi: 10.48550/arXiv.1910.00174.

[8] J. Castro, D. Gómez, and J. Tejada, 'Polynomial calculation of the Shapley value based on sampling', Computers & Operations Research, vol. 36, no. 5, pp. 1726–1730, May 2009, doi: 10.1016/j.cor.2008.04.004.

References

[9] E. Strumbelj and I. Kononenko, ‘An Efficient Explanation of Individual Classifications using Game Theory’, J. Mach. Learn. Res., vol. 11, pp. 1–18, Mar. 2010.

[10] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, ‘Striving for Simplicity: The All Convolutional Net’, presented at the ICLR (workshop track), 2015.

[11] J.-B. Yang, K.-Q. Shen, C.-J. Ong, and X.-P. Li, ‘Feature Selection for MLP Neural Network: The Use of Random Permutation of Probabilistic Outputs’, IEEE Transactions on Neural Networks, vol. 20, no. 12, pp. 1911–1922, Dec. 2009, doi: 10.1109/TNN.2009.2032543.

[12] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, ‘Not Just a Black Box: Learning Important Features Through Propagating Activation Differences’. arXiv, Apr. 11, 2017. doi: 10.48550/arXiv.1605.01713.

References

[13] K. Simonyan, A. Vedaldi, and A. Zisserman, ‘Deep inside convolutional networks: visualising image classification models and saliency maps’, Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[14] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, “Modular HPC I/O Characterization with Darshan,” in 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT), Nov. 2016, pp. 9–17. doi: 10.1109/ESPT.2016.006.

[15] H. Perez and J. H. M. Tah, “Improving the Accuracy of Convolutional Neural Networks by Identifying and Removing Outlier Images in Datasets Using t-SNE,” *Mathematics*, vol. 8, no. 5, Art. no. 5, May 2020, doi: 10.3390/math8050662.

[16] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

References

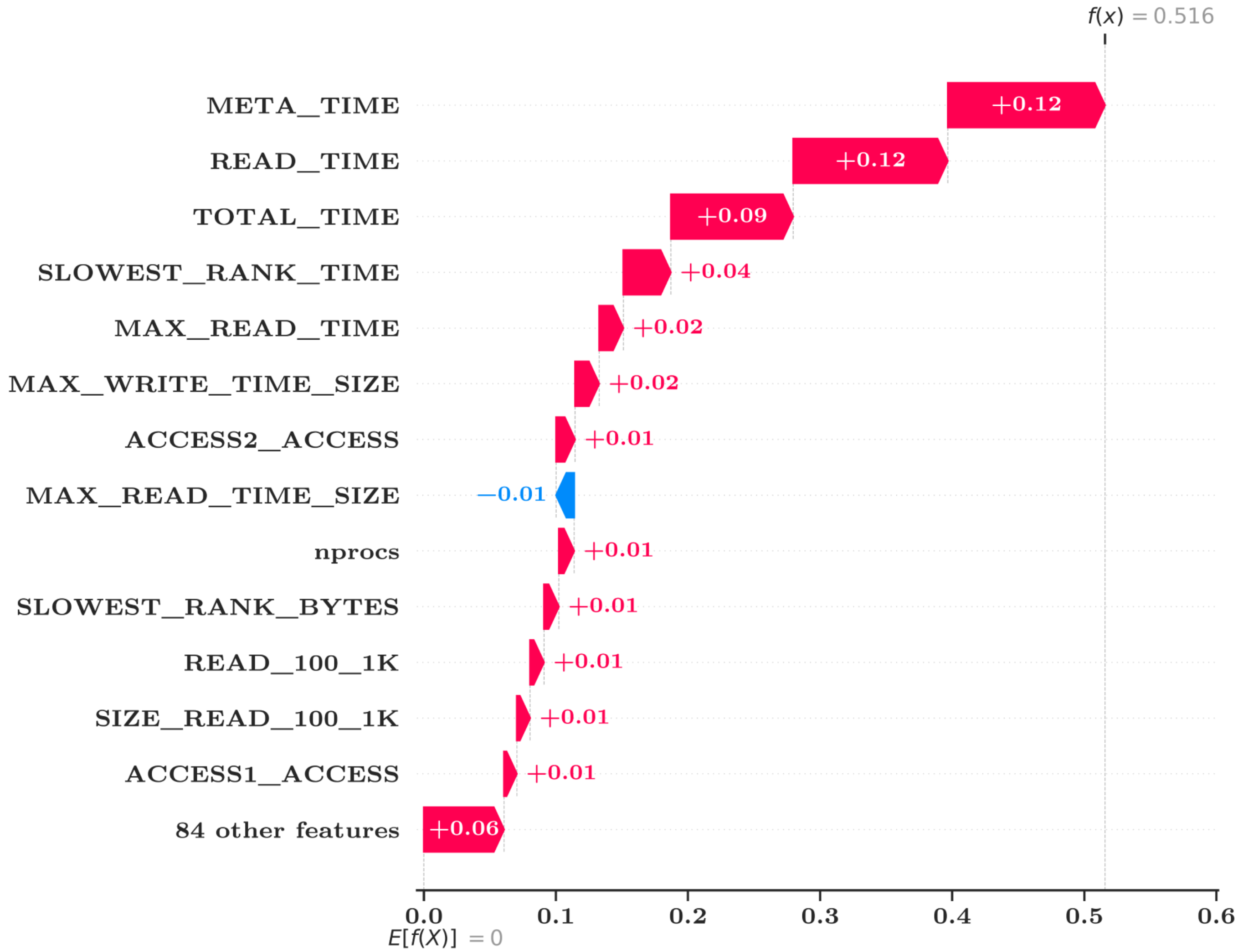
[18] M. Isakov, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy, “HPC I/O Throughput Bottleneck Analysis with Explainable Local Models,” p. 13.

[19] P. Carns et al., “Understanding and Improving Computational Science Storage Access through Continuous Characterization,” ACM Trans. Storage, vol. 7, no. 3, p. 8:1-8:26, Oct. 2011, doi: 10.1145/2027066.2027068.

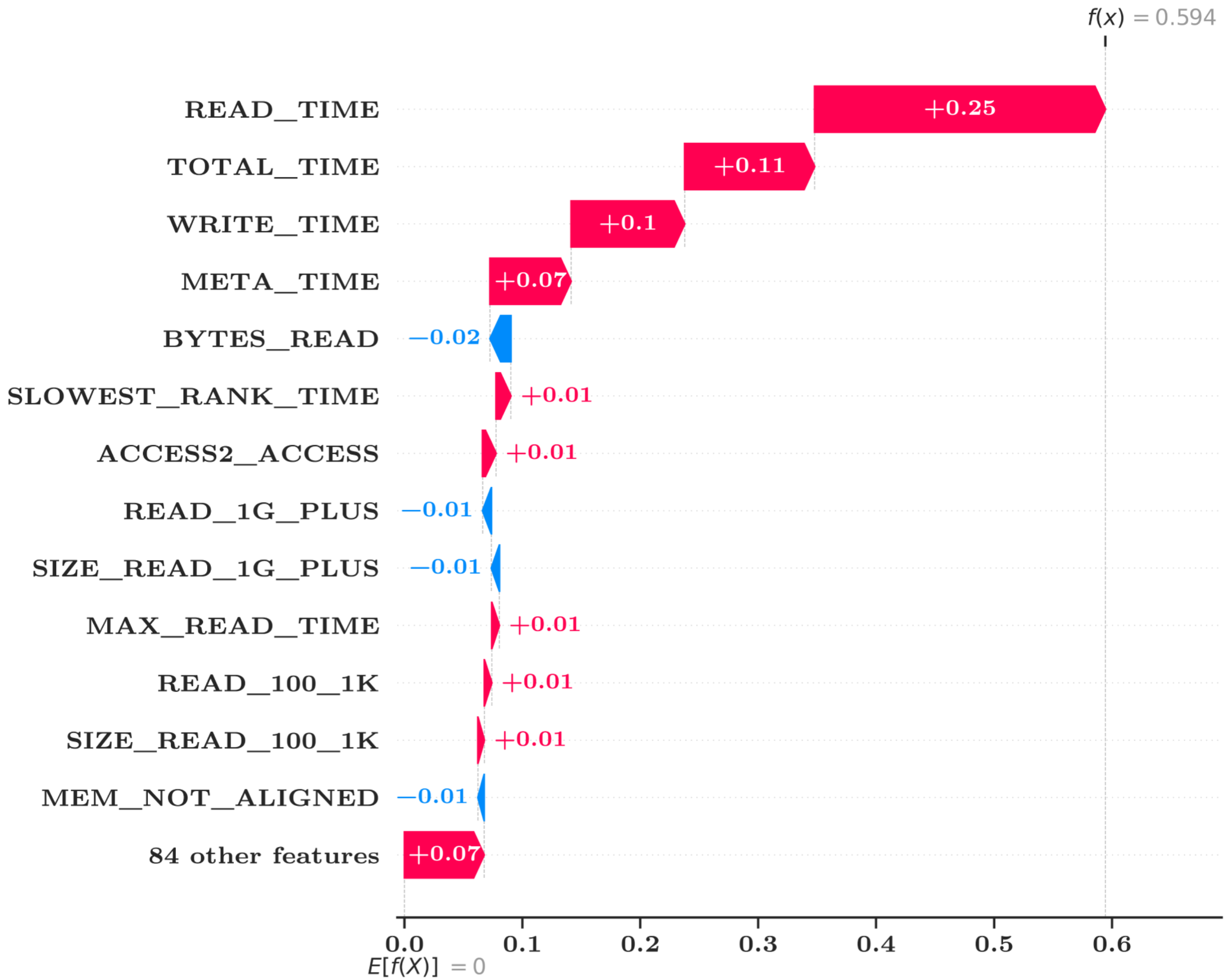
[20] B. T. Shealy, F. A. Feltus, and M. C. Smith, “Intelligent Resource Provisioning for Scientific Workflows and HPC,” in 2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS), Nov. 2021, pp. 9–16. doi: 10.1109/WORKS54523.2021.00007.

[21] D. J. Gabriel, “I/O Throughput Prediction for HPC Applications Using Darshan Logs,” Master’s Thesis, University of Nevada, Reno, 2022.

Appendix – Most important features (full dataset variant)



Appendix – Most important features (limited # of processes variant)



Appendix – Detailed model input

- POSIX_OPENS
- POSIX_FILENOS
- POSIX_DUPS
- POSIX_READS
- POSIX_WRITES
- POSIX_SEEKS
- POSIX_STATS
- POSIX_MMAPS
- POSIX_FSYNCS
- POSIX_FDSYNCS
- POSIX_RENAME_SOURCES
- POSIX_RENAME_TARGETS
- POSIX_RENAMED_FROM
- POSIX_MODE
- POSIX_BYTES_READ
- POSIX_BYTES_WRITTEN
- POSIX_MAX_BYTE_READ
- POSIX_MAX_BYTE_WRITTEN
- POSIX_CONSEC_READS
- POSIX_CONSEC_WRITES
- POSIX_SEQ_READS
- POSIX_SEQ_WRITES
- POSIX_RW_SWITCHES
- POSIX_MEM_NOT_ALIGNED
- POSIX_MEM_ALIGNMENT
- POSIX_FILE_NOT_ALIGNED
- POSIX_FILE_ALIGNMENT
- POSIX_MAX_READ_TIME_SIZE
- POSIX_MAX_WRITE_TIME_SIZE
- POSIX_SIZE_READ_0_100
- POSIX_SIZE_READ_100_1K
- POSIX_SIZE_READ_1K_10K
- POSIX_SIZE_READ_10K_100K
- POSIX_SIZE_READ_100K_1M
- POSIX_SIZE_READ_1M_4M
- POSIX_SIZE_READ_4M_10M
- POSIX_SIZE_READ_10M_100M
- POSIX_SIZE_READ_100M_1G
- POSIX_SIZE_READ_1G_PLUS
- POSIX_SIZE_WRITE_0_100
- POSIX_SIZE_WRITE_100_1K
- POSIX_SIZE_WRITE_1K_10K
- POSIX_SIZE_WRITE_10K_100K
- POSIX_SIZE_WRITE_100K_1M
- POSIX_SIZE_WRITE_1M_4M
- POSIX_SIZE_WRITE_4M_10M
- POSIX_SIZE_WRITE_10M_100M
- POSIX_SIZE_WRITE_100M_1G
- POSIX_SIZE_WRITE_1G_PLUS
- POSIX_STRIDE1_STRIDE
- POSIX_STRIDE2_STRIDE
- POSIX_STRIDE3_STRIDE
- POSIX_STRIDE4_STRIDE
- POSIX_STRIDE1_COUNT
- POSIX_STRIDE2_COUNT
- POSIX_STRIDE3_COUNT
- POSIX_STRIDE4_COUNT
- POSIX_ACCESS1_ACCESS
- POSIX_ACCESS2_ACCESS
- POSIX_ACCESS3_ACCESS
- POSIX_ACCESS4_ACCESS
- POSIX_ACCESS1_COUNT
- POSIX_ACCESS2_COUNT
- POSIX_ACCESS3_COUNT
- POSIX_ACCESS4_COUNT
- POSIX_FASTEST_RANK
- POSIX_FASTEST_RANK_BYTES
- POSIX_SLOWEST_RANK
- POSIX_SLOWEST_RANK_BYTES
- READ_0_100
- READ_100_1K
- READ_1K_10K
- READ_10K_100K
- READ_100K_1M
- READ_1M_4M
- READ_4M_10M
- READ_10M_100M
- READ_100M_1G
- READ_1G_PLUS
- WRITE_0_100
- WRITE_100_1K
- WRITE_1K_10K
- WRITE_10K_100K
- WRITE_100K_1M
- WRITE_1M_4M
- WRITE_4M_10M
- WRITE_10M_100M
- WRITE_100M_1G
- WRITE_1G_PLUS
- rank
- POSIX_F_READ_TIME
- POSIX_F_WRITE_TIME
- POSIX_F_META_TIME
- POSIX_TOTAL_TIME
- POSIX_F_MAX_READ_TIME
- POSIX_F_MAX_WRITE_TIME
- POSIX_F_FASTEST_RANK_TIME
- POSIX_F_SLOWEST_RANK_TIME