SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis | 979-8-3503-5554-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SCW63240.2024.00086

A Hierarchical Deep Learning Approach for Predicting Job Queue Times in HPC Systems

Austin Lovell*
Department of Computer Science
Purdue University
Indiana, USA
lovella@purdue.edu

Philip Wisniewski*
Department of Computer Science
Purdue University
Indiana, USA
pwisnie@purdue.edu

Sarah Rodenbeck
Rosen Center for Advanced Computing
Purdue University
Indiana, USA
srodenb@purdue.edu

Ashish
Rosen Center for Advanced Computing
Purdue University
Indiana, USA
ashish@purdue.edu

***Abstract*—Accurate wait-time prediction for HPC jobs contributes to a positive user experience but has historically been a challenging task. Previous models lack the accuracy needed for confident predictions, and many were developed before the rise of deep learning.**

In this work, we investigate and develop TROUT, a neural network-based model to accurately predict wait times for jobs submitted to the Anvil HPC cluster. Data was taken from the Slurm Workload Manager on the cluster and transformed before performing additional feature engineering from jobs' priorities, partitions, and states. We developed a hierarchical model that classifies job queue times into bins before applying regression, outperforming traditional methods. The model was then integrated into a CLI tool for queue time prediction. This study explores which queue time prediction methods are most applicable for modern HPC systems and shows that deep learning-based prediction models are viable solutions.

***Index Terms*—High-performance computing, machine learning, operations research, neural networks, queue management, performance optimization, computational efficiency, resource allocation**

I. Introduction

Accurate prediction of SLURM job start times in high-performance computing (HPC) clusters enables users to optimize submissions and leads to a less frustrating user experience. Queue time can be an opaque issue to users, as queue time can vary significantly between jobs, even when surface-level job characteristics are similar. This issue is amplified by the exponentially decreasing distribution of queue times, where a substantial majority of jobs on the HPC systems we investigated have a near-zero queue time, but some have days-long queue times. In turn, this makes the relatively rare jobs with notable queue times, which are the jobs where obtaining an accurate queue time prediction would be most important, difficult to isolate and model. We define queue time as the delay in minutes between when a job is eligible to run and when it starts running. We expect accurately predicting queue times will be of continued importance in HPC, especially as demand for such systems increases, both regarding the number of users accessing HPC systems and the scale of requested jobs. This is especially important in domains involving machine learning and the training of deep neural networks, which require heavy computing power. The increased demand and usage will further complicate batch scheduler algorithms and start time predictions, making accurate prediction a growing problem.

Our study focuses on the NSF-funded HPC system Anvil, which uses the SLURM scheduler configured with a fair share policy for job management [1]. This adds an additional layer of complexity compared to systems that do not employ a fair share policy, as this makes it necessary to integrate features relating to users and their history in some way. Anvil is a primarily CPU-focused system with GPU capabilities. Its resources are shared and divided among 11 separate partitions, each with slightly different and often overlapping resources. However, only seven partitions were used in the dataset for this model, as the other four partitions are generally only used for administrative testing. While the CPU partitions share nodes with each other, the single GPU partition is isolated. The vast majority of jobs are submitted to the shared partition, which was the target of 68.95% of all jobs in the dataset.

We propose a queue prediction tool (TROUT) using hierarchical neural network models: a binary classifier for quick-start predictions and a regression model for longer jobs. Quick-starting jobs are defined as jobs that will spend less than ten minutes in the queue, whereas longer jobs are anything else. The dual model approach was selected to combat the issue of skewed data. For inference, users can use a command-line tool to submit their job's ID, and TROUT will output a queue time prediction.

The main contributions of this work are threefold. First, we introduce a novel hierarchical deep learning approach that addresses the challenge of highly skewed queue time distributions in HPC environments. Second, we demonstrate

*Both authors contributed equally to this research.

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00086

the power of extensive feature engineering, including the use of interval trees, for efficient computation of overlapping job resources. Finally, we comprehensively compare our method against traditional approaches, offering valuable insights into the factors driving performance improvements in HPC queue time prediction. Despite the complexity of SLURM job scheduling, which has limited the success of previous queue prediction efforts, our study rigorously tests past feature engineering practices on modern HPC workloads and evaluates whether deep learning can deliver significant accuracy improvements.

II. Related Works

While queue time prediction is a common problem in HPC centers, few well-established best practices exist. Classical methods, such as time-series modeling [2], [3], k-nearest neighbors [4], and support vector machines [4], [5] were some of the first methods tried and are still being used in many projects. Machine learning methods, such as random forest modeling [6], decision trees [7], and XGBoost-based regression and classification models [8], have also been frequently used for this task. However, while resource-efficient, classical methods struggle to capture complex job and queue interactions, and the low accuracy limits practical utility.

Past work suggests deep neural networks [5], [9] show promise in tasks relating to job predictions. Though these methods are more capable of identifying the complex patterns needed for accurate queue or run-time predictions, over-fitting and generalizability to unseen data can be a problem for such models. In particular, these models lack transferability: they do not include features relating to the size of the cluster or resources remaining, and they also include users themselves as features, meaning they are not fully robust to changes over time.

Some previous studies have also included a separate model for predicting the runtime of existing jobs, and they have used the output of this model as a feature for the final wait time prediction model. This is a useful feature for the model to have, and it is one that most earlier works did not include. [10]. Due to the amount of variance in terms of requested time compared to actual used time for jobs submitted to HPC clusters, it is important to have additional information regarding when running jobs will finish. Knowing approximately when nodes will be free is an essential facet of wait time prediction, and this factored into our decision to use a separate runtime prediction model for this study. The runtime model used for this study is basic, and incorporating a more robust and accurate runtime prediction model into a queue time prediction tool is an interesting task that could be explored further.

Feature selection and data splitting greatly influence model efficacy. We adopt time-based splitting [10] to calculate the queue at any given time and explore numerical and categorical features to improve prediction accuracy. Time-based splitting is a commonly used approach, as predicting the queue time of jobs that are further ahead temporally when training and testing the model will more closely resemble the deployed use case of predicting newly queued jobs.

Variable	Max	Mean	Median	Std Dev	Count
Requested Time (hr)	432	12.552	4	22.4	3,880,043
Runtime (hr)	330.3	1.9	0.03	7.7	3,880,043
Wasted Time (hr)	432	10.7	3.6	20.4	3,880,043
Jobs Submitted By User	516914	839.1	43	11318.3	4,624

TABLE I
Anvil Historic Jobs Statistics

To build our deep learning model, we refined ideas from existing research on feature selection and engineering, incorporating a hierarchical structure that addresses both classification and regression, thus reducing overfitting. Our model integrates dynamic features reflecting the system's current state and recent history with innovative feature engineering that adapts to changes in system configuration and user behavior.

III. Methodology

The data for this study were gathered using SLURM's historical job accounting data on Anvil. Data were taken from September 2021 through May 2024 and included 3.8 million jobs with features relating to jobs, the partitions they were submitted to, the cluster itself, and user history.

These features were selected primarily to find which features would be available for real-time jobs in the queue which would not have access to other information, such as average CPU usage. Additionally, the SLURM documentation writes that "jobs are selected to be evaluated by the scheduler in the following order: Jobs that can preempt, Jobs with an advanced reservation, Partition PriorityTier, Job priority, Job submit time, Job ID" [11]. Preemption, the most important evaluation metric, depends on the resources requested and available in the cluster. As part of the feature engineering and experimentation process, we looked at additional features, such as DateTime and measuring resources in CPU minutes. These other features were then eliminated based on decreased performance in conjunction with looking at SHAP values. SHAP values are a method of assigning importance to each feature of a model [12]. Features with a SHAP value closer to 0 are less impactful on the model's prediction and can be removed. Some other features, such as day, month, and year, were removed due to author discretion and concern over the model not learning actual patterns of the SLURM scheduler and instead overfitting to arbitrary patterns that may not apply to jobs when deployed for real-world use. The most impactful features included in the final model were the amount of CPUs being used in running jobs by partition, the memory requested of jobs in that partition's queue at the time of submission, the time limit of the requested job, and the priority of the requested job upon submission to the queue.

Next, this raw data was used to create additional features regarding the resources requested by other running jobs and

jobs in the queue at the time of eligibility. We engineered features such as the amount of resources currently being used by running jobs and the amount of resources requested by jobs in the queue, as seen in Table 2. Eligibility was used instead of submit time as some jobs are not considered to start until other criteria are met, such as when a user schedules a job to begin at a specific time. These engineered features included measurements of total resources used by pending jobs, running jobs, and pending jobs with a higher priority than the selected job within a given partition. User features such as the amount of resources requested in the past day were also calculated. Features were combined based on the overlap between their eligibility and start time, calculated using interval trees. Trees were separated into groupings of 100,000 jobs with an overlap of 10,000 jobs between trees to reduce computation time. These trees were then merged back together after finishing. To manage the highly skewed nature of the data and reduce the input scale, a natural log transformation was applied to all features. This transformation helps normalize the data distribution, making the model's training process more efficient. A summary of the final features used can be found in Table 2. Other combinations of features were tested and compared against the model's performance but were found to detract from the model's capabilities.

Scaling methods, such as min-max scaling or box-cox scaling, were tested but found not to provide noticeable benefits in performance. This data was then used in modeling to predict the queue time of jobs. We conducted our data preprocessing and collection using Bash scripting and Python programs, utilizing a persistent PostgreSQL database hosted on the Anvil composable subsystem of the cluster to store the data. Cluster specifications, such as the number of CPUs, memory size, and CPUs per node were also included as features. The addition of these features makes TROUT specific to a specific HPC system in some aspects; however, these statistics can be easily modified without changing the overall architecture, and retraining can be performed to make the models generalize to other HPC systems.

Two densely connected feed-forward neural networks were used to estimate job start time using these features. The first is a fully connected binary classification model with two hidden layers. This model predicts whether jobs will start in ten minutes or less, while the second model is a regression model that predicts the queue time in minutes for jobs predicted to take more than ten minutes by the classification model, as seen in Fig. 1 and Algorithm 1. The process was split into these two models to allow for a greater focus on identifying long queue times, as long queue times are more consequential to users and thus more important to predict but represent a small minority of data; in the raw data, 87% of jobs had a queue time less than 10 minutes. First, identifying that the job will have a significant (>10 minutes) queue time allows the regression model to create more accurate predictions about the specific queue time. The distribution of queue times can be seen in Fig. 2 and job statistics can be seen in Table 1. Additionally, Ten minutes was determined to be a reasonable threshold for splitting the queue times based on whether the wait time would significantly impact the user experience. Based on the analysis of the predicted queue times by the model, it is likely that the classification accuracy would be similar if a different cutoff value of slightly more or less than 10 minutes were chosen.

Upon later testing, the model was evaluated at split times of 5 and 30 minutes. Splitting the data at the 5-minute mark resulted in decreased performance for the regression model, with over twice the mean absolute percentage error as opposed

Feature	Description
Priority	SLURM Priority
Timelimit Raw	Requested time limit (m)
Req CPUs	Requested CPUs
Req Mem	Requested memory (GB)
Req Nodes	Requested number of nodes
Par Jobs Ahead	Number of jobs in partition at time of eligibility with higher priority
Par CPUs Ahead	Sum of CPUs requested for jobs in partition at time of eligibility with higher priority
Par Mem Ahead	Sum of requested memory (GB) for jobs in partition at time of eligibility with higher priority
Par Nodes Ahead	Total nodes requested of all jobs in partition at time of eligibility with higher priority
Par Timelimit Ahead	Sum of requested wallclock for jobs in partition at time of eligibility with higher priority
Par Jobs Queue	Jobs in partition at time of eligibility
Par CPUs Queue	Sum of CPUs requested for jobs in partition at time of eligibility
Par Mem Queue	Sum of requested memory (GB) for jobs in partition at time of eligibility
Par Nodes Queue	Total nodes requested of all jobs in partition at time of eligibility
Par Timelimit Queue	Sum of requested wallclock for jobs in partition at time of eligibility
Par Jobs Running	Number of jobs currently running in partition at time of eligibility
Par CPUs Running	Sum of requested CPUs being used by running in partition at time of eligibility
Par Mem Running	Sum of requested memory (GB) of jobs currently running in partition at time of eligibility
Par Nodes Running	Number of nodes being used by jobs currently running in partition at time of eligibility
Par Timelimit Running	Sum of requested walltime for jobs currently running in partition at time of eligibility
User Jobs Past Day	Number of submitted jobs by user within past day
User CPUs Past Day	Number of CPUs requested by user within past day
User Mem Past Day	Sum of memory (GB) requested by user within past day
User Nodes Past Day	Total nodes requested by user within past day
User Timelimit Past Day	Sum of requested wallclock by user within past day
Par Total Nodes	Total nodes belonging to the partition
Par Total CPU	Total CPU cores belonging to the partition
Par CPU per Node	Number of CPU cores per node in partition
Par Mem per Node	Size of storage (GB) per node in partition
Par Total GPU	Total GPU units belonging to partition
Pred Runtime	Predicted runtime of job from random forest
Par Queue Pred Timelimit	Predicted runtime of all jobs currently pending in partition
Par Running Pred Timelimit	Predicted runtime of all jobs currently running in partition

TABLE II
FEATURE TABLE

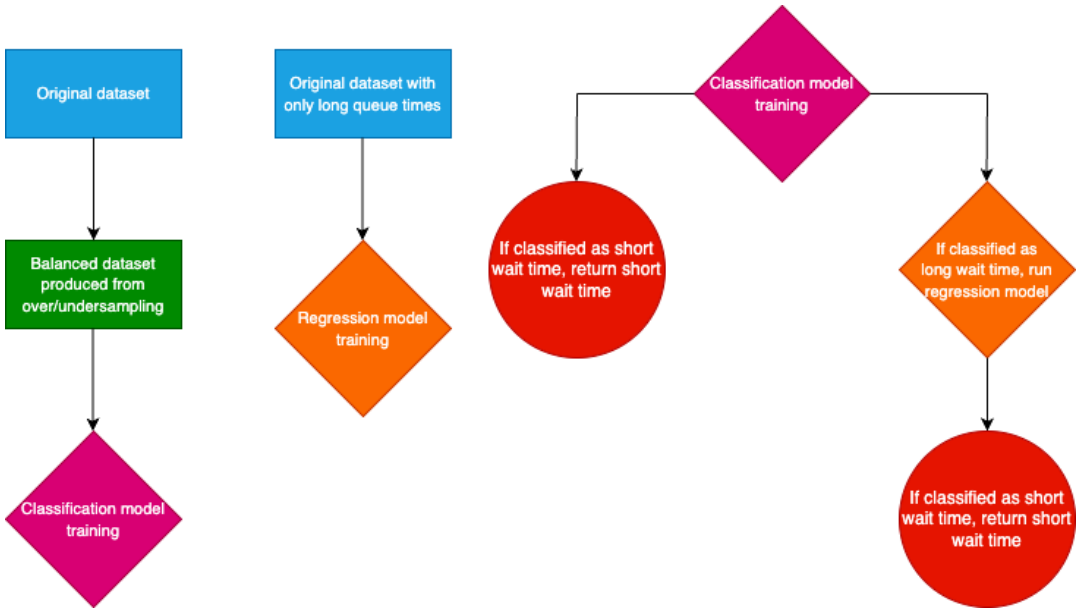

Fig. 1. Model training and implementation overview

Algorithm 1 Predict Queue Time Based on Input Job

Input: jobData - data for a specific job
Output: A string message predicting time
Let over10min **be a boolean**
over10min ← BINARYCLASSIFIERMODEL(jobData)
if over10min **then**
  Let regPrediction **be an integer**
  regPrediction ← REGRESSIONMODEL(jobData)
  return ”Predicted to start in ” `regPrediction` ” minutes”
else
  return ”Predicted to take less than 10 minutes”
end if

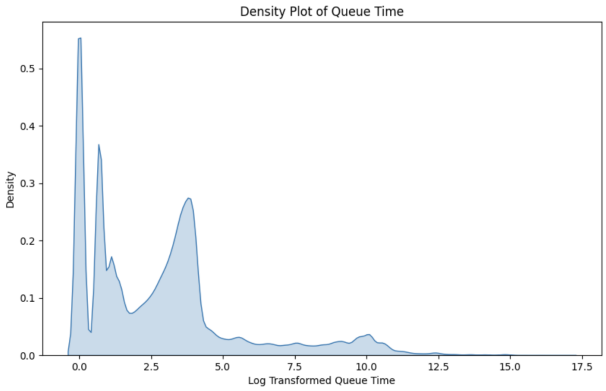

Fig. 2. Queue Time Density Graph

to the 10-minute cutoff. As for the 30-minute cutoff, the model trained on that had one fold, which outperformed the 10-minute cutoff model with a mean absolute percentage error half that of the 10-minute one, but the final fold of that model resulted in roughly the same mean absolute percentage error. Since the performance increases were only marginal for the 30-minute cutoff, we felt this threshold would negatively impact the user experience, and due to concerns over having sufficient data for the classification model to train on, we elected to go with a final cutoff of 10 minutes.

To mitigate data skew, Synthetic Minority Over-sampling Technique (SMOTE) [13] algorithms were used for under-sampling the majority class (short queue time jobs) and oversampling the minority class through artificial data creation to create balanced classes. The most recent 20% of jobs from the dataset were used as validation and test data to simulate deployment use cases where the model will predict future jobs. The model was trained using a basic pure percentage accuracy loss function, a valid training loss function due to the two classes' artificial balancing. Both models made use of the Adam optimizer [14].

The regression model's architecture contains 33 input features and three hidden layers. The exponential linear unit (ELU) activation function [15] was used for all layers except the output layer, as it achieved marginally better results than other standard activation functions, such as ReLU.

The Optuna hyperparameter framework was used to determine the best combination of hyperparameters within the model [16]. The hyperparameters investigated include the learning rate, the number of epochs to train for, the number of hidden layers for the model, the size of each layer, the size

Fig. 3. Example Time Series Split with 3 Splits

of the dropout layers to use, which features to use, and which activation function to use.

Batch normalization [17] was tested on the regression model; however, it was not selected for use. Not only did batch normalization layers not result in notably improved performance, but they also led to concerns over use in post-production. As the dataset was so large, and the range of values within each feature also varied greatly, batch normalization would have been challenging to implement and account for properly. Furthermore, as the regression model was intended to predict an exact time in minutes rather than predicting a range or bin of times, the model needed to be able to predict extremely high and extremely low values simultaneously. This was less feasible when using batch normalization, especially given the size of the hidden layers.

The model utilized the smooth L1 loss function [18], a combination of mean absolute error and mean squared error. This function is suitable for this task, as it can account for large misses due to long queue time jobs with outlier wait times and help prevent the effects of the exploding gradient problem.

The regression model used a time series cross-validation method for training, splitting the input data into five folds with a testing size of one-sixth of the dataset. This is visualized with an example time series split with three splits in Fig. 3. This splitting method was chosen for better modeling deployment use cases and due to concerns about overrepresenting the training data in the testing data when modeling, which would have given misleading results.

One limitation of the dataset of jobs submitted to the system is that many tens or hundreds of jobs would often be submitted back-to-back by the same user with the same amount of resources requested. These jobs had similar queue times, and failing to keep these jobs together during training resulted in the test set being artificially similar to the training set. This phenomenon was observed during early testing when doing a simple train-test split with shuffling, which doubled the performance of the model when compared to not shuffling the dataset due to data leakage. The solution was to use time series cross-validation to avoid this issue as well as to adhere to the principle of not training on any data from the future, as it is likely that jobs and job characteristics on the cluster will drift over time as demand for HPC resources increases.

The model was evaluated primarily based on mean absolute percentage error and compared against other selected models. These other models were chosen based on popularity and success across other studies [10]. Absolute percentage error was selected as the comparison metric due to wanting to measure the relative accuracy of predictions in relation to the scale of the output. To elaborate, predicting a low value when the actual result is large would be much worse to users than predicting a fairly large value for a large value, even if the difference in predictions vs actual values would be larger for the high prediction [e.g., predicting one minute when the true value is 10 minutes (900% off, MSE of 81) versus predicting 10 minutes when the true value is 30 minutes (200% off, MSE of 400).

IV. Results

The classification model had a binary accuracy of 90.48% with similar accuracy on both classes on a test set of the most recent 80,000 jobs. In contrast, the regression model had an average mean absolute percentage error of 97.567% over the last three test splits from the time series split (with individual mean absolute percentage errors of 69.99%, 90.87%, and 131.18%). Furthermore, analysis of the model's predicted results with the actual results showed a correlation of Pearson's r = 0.7532 for the final split (Fig. 5), as well as a visibly linear trend in the previous split (Fig. 4). This highlights TROUT's ability to perform well across the entire range of data, maintaining a fairly strong positive correlation. It is important to take the range of data into context when interpreting the overall mean percentage error: both a one-minute prediction for a delay of two minutes and a one-day prediction for a delay of two days will both yield 100% error, which highlights the ability to maintain proportionate predictive capabilities across periods (which our model did when investigating performance on different bins of time).

The regression-based neural network was compared against other model types, including an XGBoost regression model, a random forest regression model, and a k nearest neighbors

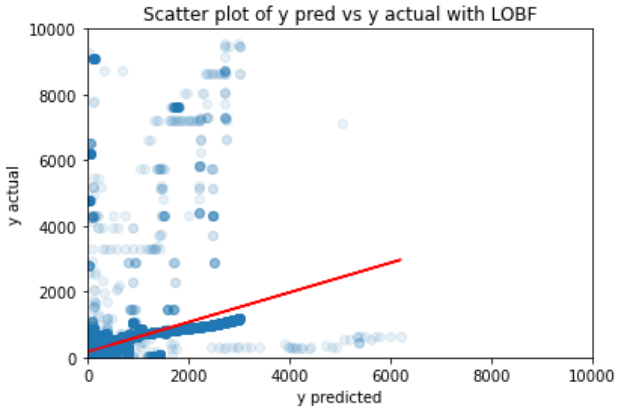

Fig. 4. Scatter plot, fold 4

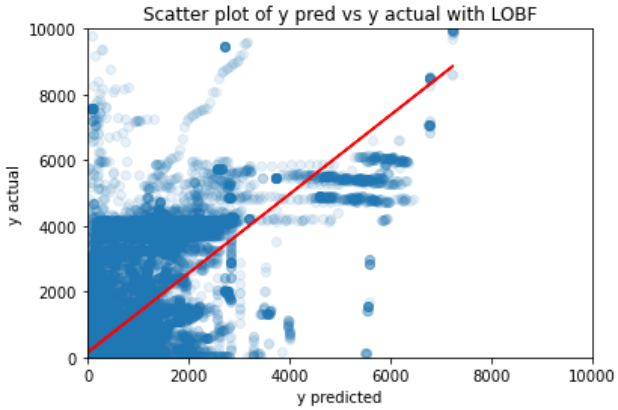

Fig. 5. Scatter plot, fold 5

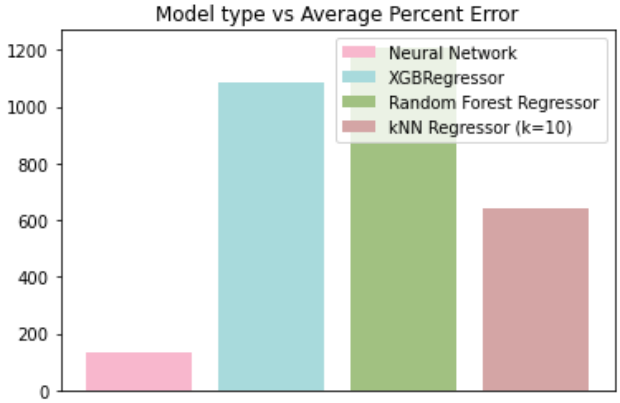

Fig. 6. Average percent error by model, fold 4

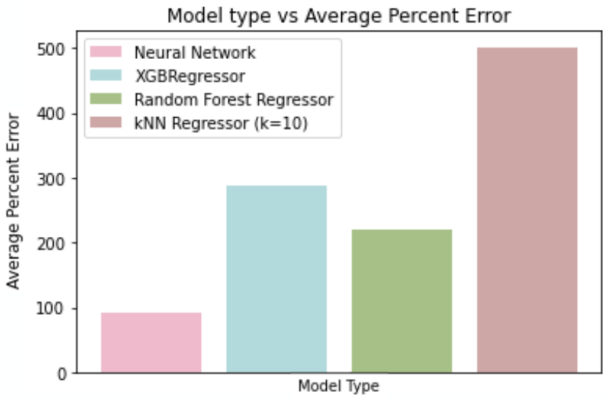

Fig. 7. Scatter plot for fold 5

regression model. These additional models were selected to serve as benchmarks based on previous work. Brown et al. [19] utilized XGBoost and kNN models to predict queue wait times. In a separate internal study performed at Purdue University, Woo et al. [20] utilized decision trees to predict job wait times and queue times. A random forest was used as a benchmark instead to reduce overfitting and have less variance in testing. All models were trained on the same data and split with the same features. Our neural network model outperformed the other types of models across all splits (Fig. 6 and Fig. 7). This supports the notion suggested in prior studies that deep neural networks are a strong choice for this task [10]. The metric used to compare the models was the average percent error. We also observed that the relative performance increase of neural networks compared to other models remained roughly the same across folds. However, there did not appear to be a significant trend between which of the other three models performed best; instead, it varied by split.

We also measured the percentage of jobs whose predicted times had less than 100% percent error (Fig. 8 and Fig. 9). We concluded the same results from this data, as the neural network consistently predicted a higher proportion of jobs to be within this threshold. However, we note that the variance between results for this metric was less than the variance of average percent error between the models. One possible reason for this is that the other models predict lower times more frequently, resulting in an increased performance due to the distribution of observed times having more instances at lower values. Furthermore, the second metric is less sensitive to strong outliers, meaning the results will tend to overestimate a model's performance.

V. Discussion and Future Work

In this paper, we have developed a tool to predict job wait times on the Anvil cluster. Our results show improved performance over previous methods of estimating queue time, supporting the idea that machine learning-based models and deep neural networks, particularly with robust feature engineering, can pick up on important features relating to job scheduling and runtime. This increased ability to predict start times can enable researchers to be more efficient and have a more positive user experience. Overall, our model marks a step forward in improving user experience users and administrators

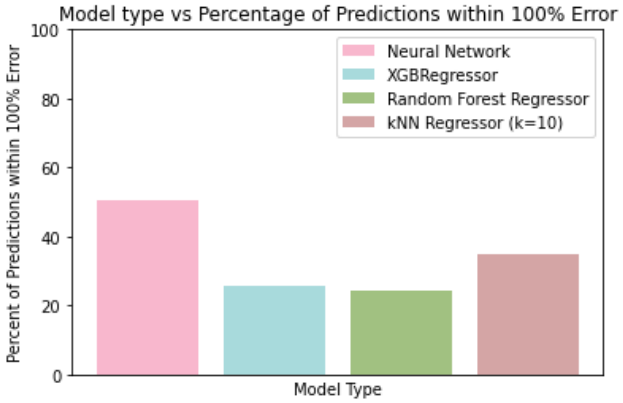

Fig. 8. Percent of predictions within 100% error, fold 4

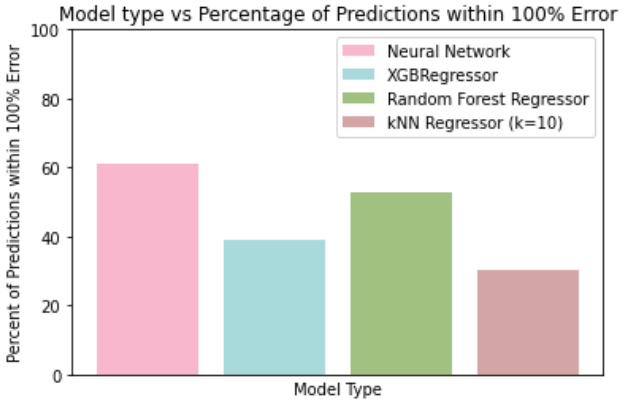

Fig. 9. Percent of predictions within 100% error, fold 5

of HPC systems, and we believe that the model is adaptable enough to generalize well to use on other clusters.

Our work highlights an innovative approach to performing feature engineering with interval trees. Many problems in artificial intelligence and HPC require large amounts of data, and performing operations and feature engineering on this data can be resource-intensive and time-consuming. This is especially true when working with time-series data or data with heavy overlap between instances. Using interval trees offers an improved solution to this problem, resulting in faster compute times for engineering features relating to overlapping jobs.

We have integrated our model into a command-line tool that takes a real, existing job in a queue on the cluster and outputs a prediction to the user based on this job. Running on a single AMD Epyc Milan CPU on Anvil, the tool takes only a few seconds to run the input through the hierarchical model and print an answer. In the future, we intend to integrate this into a user dashboard tool for increased accessibility. However, we also note that future work on integrating online learning capabilities is needed to ensure predictions stay current with the cluster changes. Code for this project can be accessed in the Github repository linked below. Using this project's code, the hierarchical model can be easily specialized for any other HPC system that utilizes SLURM through retraining with the respective historical data from that HPC system. Though the model training process is designed to be generalizable and easily retrainable, it has yet to be tested on HPC systems aside from Anvil, and we can only hypothesize that prediction accuracy will be similar on different systems. Similarly, the model design and architecture could also be extended for use with clusters that utilize a job scheduler other than SLURM, but this new dimension makes it difficult to predict how well the model would perform.

The work in this study could easily be extended to support hypothetical job queuing. This would involve a user supplying TROUT with the parameters requested for a job they wish to submit. Then, using this data and the current state of the running nodes and the queue, TROUT would return a queue time prediction for this hypothetical job, allowing users to get an estimate without actually submitting a job. This could allow users to optimize their job submissions until they achieve parameters that will result in their job running within a desired time frame.

While our hierarchical model saw superior performance compared to existing methods, some limitations remain. A significant driver of diminished model performance is the discrepancy between the resources jobs request and the resources jobs use. Users often overestimate the resources and time their job will require, adding noise to the dataset. In particular, overestimation was found to be a consistent problem for the jobs submitted to Anvil, as the average job in our data used only 15% of requested wall time, with some power users using less than 5% of requested wall time on average. This necessitates additional work to more accurately predict how long jobs in the queue will take, reducing our model's accuracy and presenting a potential opportunity for focused discussions and user training. We plan to investigate using an additional model to predict job run-times for jobs in the queue and developing additional features from there. Another area for improvement we ran into was the imbalance between partitions. Specifically, out of the roughly 3.8 million historical jobs, over 2.7 million were in the "shared" partition. This stark contrast may obfuscate unique attributes relating to prediction on these smaller queues.

Another area we hope to continue to refine is our model's support for different kinds of HPC fairshare policies. This would involve using features similar to the partition statistic features, where the model takes in details relating to specific configurations and resources available. An example of such a feature would be the duration over which jobs impact fair share in terms of time and adjusting the current feature of user jobs ran in past day for example into user jobs ran in past slurm-period. However, as of now, the model has only been trained on data from ANVIL, limiting the usefulness and feasibility of these potential features as of the moment.

Finally, as with many regression prediction models, the model struggled to predict massive outliers, like jobs that

spent days in the queue, as there was not enough data (even after synthetic data generation) to predict these extreme values correctly. Due to the closed-box nature of deep learning-based models, it is difficult to diagnose what causes widely inaccurate guesses to occur. There are some rare cases where the queue time of seemingly easy-to-predict jobs is massively over or underestimated, and ascertaining what causes this is challenging given the large number of features and weights the model uses.

Using a novel hierarchical deep-learning approach, our study explored methods for predicting job queues in high-performance computing (HPC). Our model not only performed better than existing methods but also demonstrated the potential to enhance the efficiency of HPC systems significantly. While effective in the HPC context, our techniques have broader applicability in other areas involving complex scheduling and resource management. For example, these methods could be adapted to predict waiting times in healthcare settings, improve efficiency in manufacturing operations, or optimize resource allocation in cloud computing environments. This promising result instills optimism and allows users to conduct their research more efficiently. Future studies should consider testing the applicability of our model to these different contexts, expanding its potential impact across various industries.

VI. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2005632. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] SchedMD, "Slurm workload manager overview," 2021. Retrieved Sep 15, 2024 from https://slurm.schedmd.com/overview.html.

[2] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema, "Trace-based evaluation of job runtime and queue wait time predictions in grids," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, (New York, NY, USA), p. 111–120, Association for Computing Machinery, 2009.

[3] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[4] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495–504, 2010.

[5] M. R. Wyatt, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "Prionn: Predicting runtime and io using neural networks," in *Proceedings of the 47th International Conference on Parallel Processing*, ICPP '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[6] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and underestimation rate in job runtime estimates," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 530–540, 2017.

[7] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and io traffic on high-end clusters," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 255–258, 2016.

[8] K. Menear, K. Konate, K. Potter, and D. Duplyakin, "Tandem predictions for hpc jobs," in *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*, PEARC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

[9] Z. Hou, S. Zhao, C. Yin, Y. Wang, J. Gu, and X. Zhou, "Machine learning based performance analysis and prediction of jobs on a hpc cluster," in *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 247–252, 2019.

[10] K. Menear, A. Nag, J. Perr-Sauer, M. Lunacek, K. Potter, and D. Duplyakin, "Mastering hpc runtime prediction: From observing patterns to a methodological approach," in *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, PEARC '23, (New York, NY, USA), p. 75–85, Association for Computing Machinery, 2023.

[11] SchedMD, "Slurm multifactor priority plugin," 2023. Retrieved Sep 15, 2024 from https://slurm.schedmd.com/priority_multifactor.html.

[12] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 4768–4777, Curran Associates Inc., 2017.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, p. 321–357, June 2002.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. https://doi.org/10.48550/arXiv.1412.6980.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 2016. https://doi.org/10.48550/arXiv.1511.07289.

[16] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[17] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[18] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015. https://doi.org/10.48550/arXiv.1504.08083.

[19] N. Brown, G. Gibb, E. Belikov, and R. Nash, "Predicting batch queue job wait times for informed scheduling of urgent hpc workloads," 2022. https://doi.org/10.48550/arXiv.2204.13543.

[20] J. Woo, S. Smallen, and J.-P. Navarro, "Karnak 2.0 wait time prediction." unpublished.