**Introduction**

Time series forecasting has become a rapidly developing problem for some time, specifically in economics and finance. Historically, these problems have been solved by using statistical methods or machine learning models [1]. However, with the success of deep learning models in various fields such as computer vision and natural language processing [6], time series forecasting models have shifted from these traditional methods to more complex models such as LSTMs and transformers [4]. Such methods have seen much success in time series forecasting [1, 4, 5], but it has been shown that basic linear models can often outperform these complex models on a variety of basic evaluation tasks [7]. For more complex tasks, there is empirical evidence that deep learning approaches perform better for a variety of reasons [4, 5]. Since there is evidence of success in time series forecasting for both simple methods, such as ARIMA [5], and for complex deep learning methods, such as PatchTST [4], we aim to present our determined two best methods for imputing missing resource usage data from a supercomputer.

**Dataset**

The dataset that we work with contains resource usage data from a supercomputer. There are three types of data as labeled by their csv files: training data, test data, and grading data. The training data is complete with no missing values and is used to train our models. The data that is labeled test data includes two files: one with missing values to impute and another without missing values to evaluate the performance of the imputations. We utilize these files as validation data to determine the time series forecasting root mean squared error (RMSE) of our models on unseen data. Finally, the grading data is the test data for the purposes of our task. The grading data contains several missing values. We use our models to perform data imputation on the grading data and submit the results to the Technical Leads for accuracy evaluation.

The resource data in the datasets comes from multiple hosts, where each host can have multiple jobs. For each job that runs on a host, there are six recorded data values labeled as following in the data: cpuuser, memused, gpu\_usage, nfs, block, and memused\_minus\_diskcache.

We observe that the training data’s ACF values are nonlinear and note that multiple jobs are run in parallel. Block and nfs have large ranges due to some extreme outliers. Between outliers, their distributions show more typical time series variation from time step to time step. The remaining data may exhibit some outliers, albeit much less extreme, and vary relatively closely around a moving average.

***Data Processing***

For our Data Processing step, we note that the data is a collection of metrics for many jobs running on multiple hosts. Therefore, the data in its entirety cannot be considered as a sequential dataset. As we load the data into our models for training, we split the data by host. Doing so ensures that the model does not treat the entire dataset as a single time series, but rather, multiple time series of jobs running on different hosts. We note that for most of the time series data is of length 50 to 70.

We then train a time series preprocessor on the training data which formats the data for supervised learning, impute missing data, and prepare the dataset into the correct format to be used to train our models.

***Evaluation Metric***

We select the root mean square error (RMSE) as the evaluation for our models. RMSE is defined as

,

where is the amount of data in the evaluation dataset (the provided grading dataset), is the ground truth (actual value), and is the value predicted by our model. A lower RMSE value implies that the model’s predictions are more accurate.

**Our Approach**

For imputing missing data values for supercomputer resource usage data, we select an LSTM-LightGBM hybrid model and PatchTST models. Both models, as shown in [1, 4], have proven successful in time series forecasting and are able to capture some internal trends in the data that traditional statistical methods cannot. As such, we utilize these models to evaluate our ability to impute missing resource usage data.

**LSTM and LightGBM** Our first approach uses a LSTM-LightGBM hybrid model. LSTMs are a type of recurrent neural network that solves the vanishing gradient problem, thus allowing the model to better retain information across time [2]. It utilizes memory cells to determine whether information over long sequences should be remembered or forgotten, and gates to control how the remembered information flows through the network [2]. LightGBM is a gradient boosting framework that uses a more efficient sampling and training method than other gradient boosting decision tree models [3]. Unlike these methods, LightGBM grows the decision trees from the leaves first rather than branching out uniformly, which improves its performance as compared to other models such as XGBoost [3].

Our model takes the training data and fits an LSTM model to it. Then, we pass the training data through it to obtain more features that capture more information from previous time points in the time series. We then add these new features into the training data and fit a LightGBM model to this new training data. We take this approach to combine the LSTM’s ability to capture long term trends in sequential data with LightGBM’s ability to capture complex data and make accurate predictions.

***Strengths***

Our model obtains all the benefits of an LSTM combined with the LightGBM model. The LSTM is able to capture long term trends in the data to create more features to train the LightGBM predictive model on. In doing so, we can learn some hidden dependencies in the data, that cannot be captured via traditional data processing methods.

The LightGBM model has been proven to capture complex trends in data and make efficient and correct predictions. Furthermore, LightGBM trains relatively quickly as compared to other decision tree methods, thus increasing its utility if this model were to be deployed. Our model obtains all of these benefits as well since we use LightGBM to perform time series forecasting.

***Weaknesses***

Unfortunately, this pipeline is highly complex with many hyperparameters, making it difficult to maintain or debug in the case that the model needs to be refit to new data. Part of the complexity comes from the LSTM, which is not as efficient to train as the LightGBM model; as such, our model might face difficulties if there were significantly limited compute power. We also note that LSTMs can easily overfit on smaller datasets; this is not applicable in our use case, but it does imply that this model requires much data to train on to perform optimally. Lastly, we note that our predictive model, LightGBM, is highly dependent on the LSTM’s outputs. If the LSTM were to not be trained properly, the model would not work as intended. The incorrectly extracted features from the LSTM would be passed into the LightGBM model, which would likely yield unexpected results.

**PatchTST**

PatchTST is a transformer-based model for time series forecasting. It introduces new concepts into time series forecasting. Unlike other models that utilize just one data point at a time, it utilizes patches in the time series data in order to capture more semantic information across the time series. This allows the model to interpret the data across time rather than learn on a single point in time at a time. PatchTST can also predict multiple channels at a time, meaning that for the supercomputer resource usage dataset, we can predict all the data values at the same time with just one model.

***Strengths***

PatchTST has numerous strengths over other methods of time series forecasting. Because of its transformer backbone, it can capture nonlinear trends in the data well. Furthermore, it can efficiently learn from a longer lookback window in the training data than other models. Because PatchTST utilizes patches, it can capture multiple time steps worth of data in just a single datapoint that it uses to train on. In addition, this longer lookback window takes up less memory when training and reduces the time complexity of the model. On larger datasets, PatchTST can reduce the runtime by at least 22x [4].

***Weaknesses***

Unfortunately, like other transformer models, PatchTST requires a large dataset to train on and still takes longer to train than simpler methods such as ARIMA. Its complexity may lead to the model overfitting if there is insufficient training data, capturing each of the outliers present in the training data. Furthermore, like all transformer-based models, PatchTST is also not an interpretable model due to its complexity. Finally, PatchTST assumes that all the time intervals between the data points are constant; however, this may not prove true in all use cases in the real world. As such, for the purpose of this dataset, using PatchTST is reasonable; however, if the time intervals were not evenly spaced, PatchTST could yield unexpected results.

**Results**

The PatchTST model was not able to achieve exciting performance with relatively little training time after hyperparameters were tuned. With 15 epochs of training using the training dataset, the average validation data RMSE across the six features being predicted was 26.5. The PatchTST model has potential to excel in predicting the six features. Outliers exhibited in the distributions for disk and nfs were particularly difficult to capture using the PatchTST; however, with additional computing resources and time, the performance of PatchTST could likely be improved to capture such outliers.

The Hybrid LSTM model also proved to be very effective at imputing the missing data. It took more training time compared to the PatchTST model, requiring 20 epochs. However, the model still achieved an impressive average validation data RSME of 3.84. It particularly excelled at calculating block and nfs, being able to accurately predict the large outliers of the two features.

**Discussion and Concluding Remarks**

We showcase the time series forecasting power of the LSTM and PatchTST models. We evaluate the efficacy of our models on the supercomputer resource usage dataset. Both methods are able to perform reasonably well, with our hybrid LSTM-LightGBM model having an average RMSE of 3.84, slightly the PatchTST model, which has an average RMSE of 26.5. We attribute this to the fact that the LSTM-LightGBM model is able to capture hidden long term trends in the data. Our work demonstrates the utility of these models for real world applications and proves that these models are able to reasonably capture trends in complex datasets.

**References**

[1] Elsworth, S., & Güttel, S. (2020). *Time series forecasting using LSTM networks: A symbolic approach.* arXiv. <https://arxiv.org/abs/2003.05672>

[2] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[3] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. (2017). *LightGBM: A highly efficient gradient boosting decision tree*. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 30). Curran Associates, Inc. <https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf>  
[4] Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). *A time series is worth 64 words: Long-term forecasting with transformers*. arXiv. <https://arxiv.org/abs/2211.14730>

[5] Siami‐Namini, Sima et al. “A Comparison of ARIMA and LSTM in Forecasting Time Series.” 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (2018): 1394-1401.

[6] Tan, F., Feng, S., & Ordonez, V. (2019). *Text2Scene: Generating compositional scenes from textual descriptions*. arXiv. <https://arxiv.org/abs/1809.01110>

[7] Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2022). *Are transformers effective for time series forecasting?* arXiv. <https://arxiv.org/abs/2205.13504>