EECS 545 Final Project: ThermoTwin–AI driven digital twin for electricity generation

Aayush Dutta, Jeremy Flics, Ryutaro Hashimoto, Dean Price

Abstract

Digital twinning is an area of research in nuclear engineering that seeks to increase the safety and reliability of reactor systems by simulating their behavior in real time. Traditionally, this is achieved by directly computing the dynamics equations that governs these systems. However, in this paper we present ThermoTwin, a new machine learning-informed approach to twinning a thermodynamic cycle present in reactors called the Rankine cycle. This report details our approach for generating simulated data based on the Rankine cycle dynamics equations and explains the four main functionalities of ThermoTwin. These are startup state verification, which predicts what type of accident, if any, will occur based on the initial conditions of the system; accident onset prediction which, given an accident, state predicts how long it will take for an accident to occur; steady-state sparse sensing, which uses reactor values over time to predict the energy output of the system; and startup transient twinning, which attempts to model the behavior of the whole system based exclusively on initial conditions. Overall, we found levels of success in the first three of these features, but due to the non-linearity of our dynamics equations found that initial conditions are not enough to predict the whole behavior of the system. Here is a link to our Github repository [1].

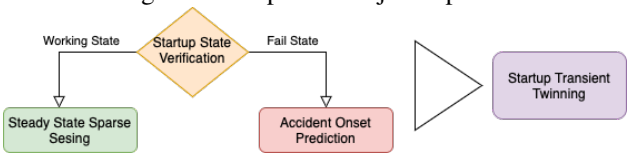
1 Introduction and Proposed Method

Digital twinning is an active area of research [6, 7] where an auxiliary simulation-based system is run alongside a physical system to fufill various diagnostic and control needs of the physical system. ThermoTwin is an AI driven digital twin which runs alongside a thermal hydraulic system used for electricity generation to provide information on potential accidents which occur in the system or provide additional monitoring of the system. Given the complexity of the governing equations for thermal hydraulic systems, the simulations which accurately predict their properties can take seconds to minutes to run. In the operation of these systems, predictions are required on the order of milliseconds. As such, machine learning (ML) is used to provide predictions on the timescales required for operation and diagnostics.

For brevity, minimal discussion of the simulation is presented here. A complete description of the governing equations and constitutive models used in the simulation is given in the progress report. The feedback on that report was to focus more on the ML models, hence, the omission of simulation details to leave room in the page limit for the requested content. However, in general, the simulation consisted of a thermal-hydraulic simulation where heat is being generated in a boiler and eventually used to generate electricity in a turbine. A series of first-order nonlinear ordinary differential equations is coupled and the Radau method is used for forward integration of these equations.

The following pipeline gives a brief overview of the proposed method discussed in the progress report and are discussed in great depth below.

Figure 1: Simplified Project Pipeline



2 Startup State Verification

2.1 Dataset Description

In the generation of the ML models used for the startup state verification module of ThermoTwin, training data was calculated using the dynamics simulation our group created. From a predefined range of various operating parameters and initial states, a simulation configuration was sampled from uniform random distributions. Then, the startup transient from that configuration was simulated. In total, there were 3 operating parameters: output power fraction, boiler heat generation and down time. Then, there were 3 initial states: inlet coolant temperature, inlet pressure and mass flow rate. To save room for the more ML-focused portions of the work, specific explanations of each variable is not included. Then, the accident condition (including no accident at all) was recorded. Figure 2.2 shows the frequency of the various accidents across to 40,000 data points. To summarize, there were 6 predictors used to classify a transient as one of four accidents. An excerpt of the data is given in the progress report but will be omitted from this paper in interest of space.

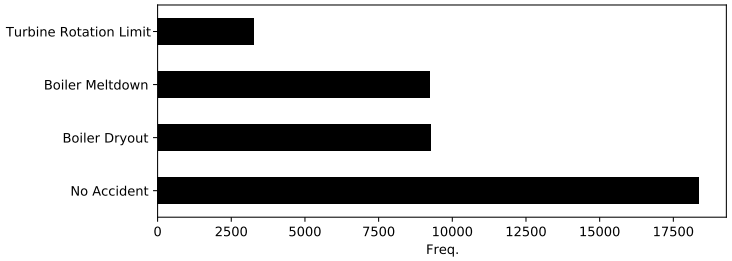
2.2 Model Selection and Performance

This first task is to classify a set of initial operating conditions according to if they lead to safe operation or they lead to an accident. Furthermore, it is useful to know which accident the set of initial conditions lead to. As such, 7 different classifiers are used to classify a set of initial conditions into an accident or safe operation. These 7 classifiers are: (1) K Nearest Neighbors (KNN), (2) Logistic Regression (LR), (3) Linear Discriminant Analysis (LDA), (4) Support Vector Machine (SVM), (5) Decision Trees (DT), (6) Gaussian Process (GP) and (7) Deep Neural Network (DNN). The training data is generated by randomly sampling initial conditions from set ranges and running the dynamics simulation to evaluate the accident condition of the system. In total, 40,000 transients are simulated to make the dataset used to train these classification models. As shown in Figure 2.2, this method for generating data for model creation and evaluation leads to unequal distributions in the labels. Therefore, simply using classification error as a loss metric is insufficient. To account for this, a balanced accuracy score (BAS) [4] is included in addition to the typical accuracy score (AS). A definition of the AS is neglected to save space, however, the BAS is given as:

$$\text{BAS} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right). \tag{1}$$

Here, TP refers to the number of true positives, FN refers to the number of false negatives, FP refers to the number of false positives and TN refers to the number of true negative classifications. To be clear, the BAS is used as the loss metric in both hyperparameter optimization and model training.

Figure 2: Frequency of occurrence of various accidents among the 40,000 transients generated for classifier training and evaluation.



Training these classifiers consists of 2 steps. First, the models themselves must be trained with some subset of the full dataset. This subset is referred to as the training set. Second, the configurations of the models, as determined by their

hyperparameters, should be adjusted such that the model gives the best performance. A subset of the dataset referred to as the validation set is used for this step. Following the training, each classifier is evaluated using a dataset that was not used for either training step, this dataset is referred to as the testing set. The sizes of the training set, validation set and testing set follow the ratio 0.5-0.25-0.25 respectively. Furthermore, due to computational constraints, the full 40,000 points in the dataset are not used for all models. The evaluation metrics given in this section were checked for convergence for whatever dataset size was used and convergence was overwhelmingly satisfied for all metrics.

To report the results of this exploration, Table 1 gives the optimized hyperparameters associated with each method. These parameters were selected by applying a Bayesian optimization technique to the parameter ranges/categories given in Table 2. Each set of hyperparameters was given 50 objective function evaluations to produce reasonable optima, in the future, this number should be greatly increased for better results—particularly for DNN. Hyperparameters were optimized by minimizing the BAS of the model on the verification data set. No hyperparameters were used for LR, LDA and GP so they are not included in this Table. For brevity, the search ranges of each of these parameters will not be reported but—particularly for the DNN—they were limited to reduce the large computational burden already dedicated to the problem. Furthermore, the BAS and AS are reported for each of these 7 models in Table 3. The total number of data points used in the hyperparameter optimization and training/verification/testing of these models is also included in this Table. These were limited to reduce the computational cost of training the models. As shown in the next section, the size of the dataset for the non-DNN models did not have significant impact on model performance. These models did not have the degrees of freedom necessary to accommodate the extra knowledge gained from the acquisition of more data.

Clearly, DNN shows the strongest performance of the 7 models. This is expected as DNN is a very flexible model type which requires extensive tuning to show good results. This tuning is done through the verification data set. Regardless, SVM shows very strong performance given that it only requires the selection of 2 hyperparameters. GP and KNN perform the weakest of these 7 algorithms, this is likely due to the lack of a "clustering" effect as can be seen in some classification problems.

Table 1: Optimized hyperparameters for accident classifiers. Naming follows parameters as given in TensorFlow [3].

KNN	SVM	DT	DNN
n_neighbors = 22	C = 3.54	criterion = "gini"	width = 56
	kernel = "poly"	splitter = "random"	nlayers = 2
		max_depth = 28	activation = "sigmoid"
			epochs = 432
			learning rate = 0.001

Table 2: Ranges used in Bayesian optimization of hyperparameters. [] indicate parameters which were optimized over a continuous range. {} indicate categorical parameters for optimization. KNN left out of table for space constraints, n_neighbors was optimized in the range 1 to 50.

SVM	DT	DNN
C = [0.1, 12]	criterion = {"gini", "entropy"}	width = [1,60]
kernel = {"poly", "linear", "rbf", "sigmoid"}	splitter = {"best", "random"}	nlayers = [1,20]
	max_depth = [1, 30]	activation = Too many to list, 9 in total
		epochs = [200, 1500]
		learning rate = {0.01, 0.001, 0.0001}

2.3 Data Complexity

An important aspect in any data-driven exploration is the evaluation of the sufficiency in the total size of the dataset. Furthermore, the hyperpameters provided in Table 1 are used for all dataset sizes. This is sufficient because the objective of this section is to show that the data set sizes used to evaluate the models is sufficient because model performance saturates before those sizes. The objective is not to show the best model performance for each dataset size, therefore, there is only a need to show saturation in the model performance parameters by the sizes shown in Table 1, where the best model performance for that dataset size *has* been approximated.

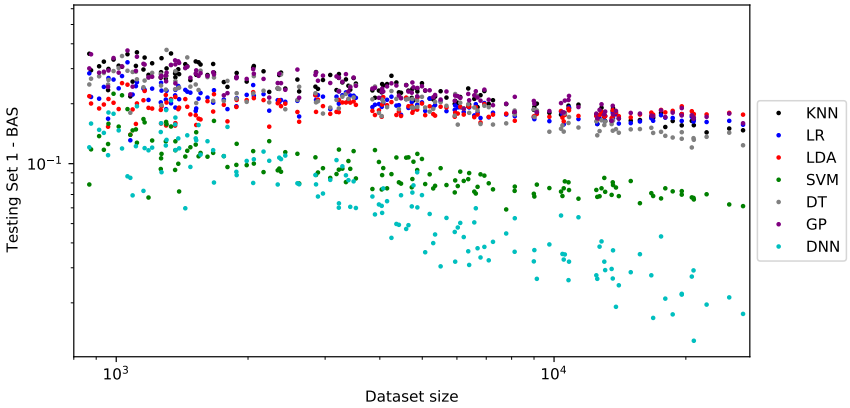
Figure 2.3 shows the behavior of 1 - BAS as a function of dataset size. These datasets are comprised of the training, verification and testing sets. The change in 1 - BAS behavior as a function of dataset size is not only due to the

Table 3: Model training error metrics and testing error metrics for 7 classifiers. Total number of data points used across training/verification/testing sets is also given.

	Train AS	Train BAS	Test AS	Test BAS	Dataset Size
KNN	0.9216	0.8726	0.8656	0.8006	10,000
LR	0.8898	0.8235	0.8784	0.8227	10,000
LDA	0.8826	0.8245	0.8752	0.8322	10,000
SVM	0.9808	0.9751	0.9576	0.9483	10,000
DT	0.9882	0.9836	0.8944	0.8619	10,000
GP	0.8952	0.7985	0.8872	0.8004	10,000
DNN	0.9898	0.9900	0.9831	0.9825	40,000

improvement of the classification models but it is also due to the improvement of the evaluation metrics because as the testing set gets larger, variability in the results from the use of these testing sets shrink. That is why overall, the width of the 1 - BAS metrics gets narrower as the dataset size increases. Of all the models, it appears that only SVM and DNN were able to leverage the increased dataset sized to achieve increases in model performance. The improvement in the SVM tends to saturate around 10,000 samples, although some minor improvement can still be observed to the 27,000 samples shown in this plot. However, this improvement is minimal despite large increases in computational burden required to optimize the hyperparameters. Therefore, 10,000 data points are used. The scaling performance of these two models is from the flexibility of these models. SVM allows for a wide range of kernels to be used to best approximate trends in the data while DNN's flexibility comes from the complex structure of these models.

Figure 3: 1 - BAS performance metric as a fucntion of dataset size. Datasets are comprised of training, validation and testing groups.



3 Accident Onset Prediction

A natural extension of our work with classification was to look at initial conditions that are likely to result in accident states and then forecast the time it takes for said accident to occur. Therefore, in this part of the project, we restrict ourselves to failing states (justified by the project pipeline in the introduction).

3.1 Dataset Desciption

The dataset used for this section is identical to that of section 2 with the addition of accident timings (produced by the simulation) which are in seconds. Furthermore, non-accident states were omitted from the training, cross validation, and testing data. This left us with a dataset of size 21760 which was then suitably split into the aforementioned categories (50-25-25 split). Upon final testing a 70-30 split was used.

3.2 Model Selection

Clearly, the problem we have at hand is a regression problem. Therefore, the models that were compared were: (1) Linear Regression, (2) Ridge Regression, (3) Lasso Regression, (4) Elastic Net Regression and (5) a DNN. All of these notions, with the exception of Elastic Net Regression, were covered in class so we will forgo technical details. Elastic Net Regression is a combination of L1 and L2 regularization where the loss function is given by

$$L(w) = \sum_{i=1}^{N} \left(y^{(i)} - w^T x^{(i)} \right)^2 + \lambda\alpha\|w\|_1 + \frac{1}{2}\lambda(1-\alpha)\|w\|_2$$

Where $\lambda \in \mathbb{R},\ \alpha \in [0,1]$ are hyperparameters to be found. Finally, the reported losses below for each model, which we will use for comparison. Is that of mean-squared-error (which we have discussed in class).

As mentioned earlier, the hyperparameter search for each of these models used a dataset split of (50-25-25 split) for training, cross validation and testing. The range of the hyper-parameter searches is reported in Table 4 followed by the optimal values of the hyper-parameters in Table 5. Do note that the final hidden layer for the DNN was a ReLU activated with a width of one (and the loss function used was mean squared error). Therefore, the reported optimal hyper-parameter for the activation function is pertinent to the hidden layers. Finally, as in section two, Bayesian Optimization techniques were used to find optimal hyperparameters. MultiTerms bleo indicates the degree of multinomial terms in the postprocessed data (note that a multinomial value of 8 transforms the data from $20000 \times 6 \mapsto 20000 \times 3003$). Finally 6 shows the performance of the models with respect to MSE.

Table 4: Ranges used in Bayesian optimization of hyperparameters. [] indicate parameters which were optimized over a continuous range. {} indicate categorical parameters for optimization

Linear	Ridge	Lasso	Elastic	DNN
-	$\lambda = [0,1]$	$\lambda = [0,1]$	$\lambda = [0,1]$	
			$\alpha = [0,1]$	width = $\{1, 2, \ldots, 78\}$
muliTerms = [1,10]	muliTerms = [1,10]	muliTerms = [1,10]	muliTerms =[1,10]	nLayers = $\{1, 2, \ldots, 20\}$
				activation: again too many to list. 6 used
				epochs : $\{200, 201, \ldots, 1500\}$
				learningRate: $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$

Table 5: Optimized hyperparameters for accident time regression. Naming follows parameters as given in TensorFlow [3].

Linear	Ridge	Lasso	Elastic	DNN
-	$\lambda = 0.0005$	$\lambda = 0.00001$	$\lambda = 0.00025$	
			$\alpha = 0.012$	width = 64
muliTerms = 8	muliTerms = 8	muliTerms = 8	muliTerms =8	nLayers = 2
				activation: "leakyRelu"
				epochs : 702
				learningRate: 0.001

Table 6: Model training, crossvalidation, and testing error given optimal hyperparameters. Note, we are using mean squared error.

	Train Loss (s^2)	Test Loss (s^2)	CV Loss (s^2)
Linear	2.3300	7.1164	6.8091
Ridge	4.7454	5.4818	5.6515
Lasso	5.3868	6.4151	6.8613
ElasticNet	4.8322	5.6132	6.1965
DNN	0.0576	1.0514	1.1502

So we observe that the DNN performs the best, by far. This result is not unexpected due to the flexibility of the model. Furthermore, we note that our system is governed by non-linear ODEs which tend to be very sensitive to initial conditions. On the contrary, multinomial functions are quite well behavied and don't exhibit the aforementioned property.

Do note that the optimal hidden layer activation function used, leakyReLU [2], was not discussed in class. However, it is quite similar to ReLU and is given by, for $0 < \alpha << 1$: $f : \mathbb{R} \to \mathbb{R}$, where $x \mapsto \begin{cases} x & x > 0 \\ \alpha x & x \geq 0 \end{cases}$

Upon final testing (with a 70-30 data-set split) the test loss of our DNN was 0.83631.

A similar process to section 2.3 was employed to confirm dataset sufficiency. Results will be omitted for the sake of brevity.

4 Steady-State Sparse Sensing

As stated in previous sections, considerable effort has been spent detecting and preventing reactor failure-states. Therefore, a logical next step is the analysis of steady-states since they can be reliably be reached when using ThermoTwin. The problem we targeted is compensating for the lack of sensors on difficult to measure quantities with time series forecasting. In particular, we seek to forecast turbine energy levels based on other sensor readings.

4.1 Dataset Description

Since this section of our project focused on time series forecasting rather than classification across initial conditions, a single transient was generated over a significantly longer time period for analysis. Initial conditions were chosen to be in a realistic range, and were checked for convergence. Additionally, to add complexity and realism to the modeling, input heat was allowed to vary up to random noise during the simulation.

4.2 Time Series Forecasting as a Supervised Learning Problem

Given that our dynamical system of interest is governed by first-order differential equations, it makes sense to not only consider current sensor readings for our features, but also readings from the recent past. As such, a long short-term memory (LSTM) network was a natural choice. However, LSTMs are typically used to for supervised learning, so our time series problem had to be converted into the proper problem domain. This was achieved by using a sliding window approach. That is, each output prediction was associated with an input matrix whose rows were sensor readings for n previous time-steps, including the current. Upon inspection, the turbine data was highly periodic for all steady-state transients, so a look-back value of just $n = 7$ was sufficient.

Another consideration when converting a time series problem into one of supervised learning is the testing-training data split needs to be informed by temporal relationships. As such, traditional training methods like k-fold cross validation can not be used. Instead, we experimented with walk forward validation [5], but eventually found that a simple testing-training split with the test set being the last 35% of the transient was more computationally efficient, but about equally accurate.

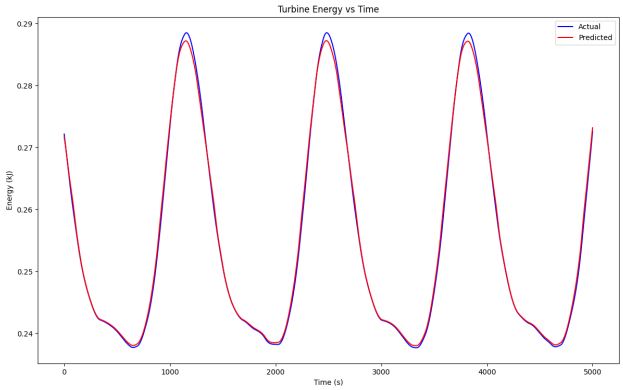
4.3 LSTM Network Architecture

While many configurations of the LSTM were tested, the repetitive nature of the data, allowed for a relatively simple model to perform quite well. The advantage of keeping things simple is twofold. First, over-fitting is less likely, and second, the training time of the model is reduced. In particular, a single LSTM layer with 64 connections was fed into a 20% droupout layer, before being sent to a dense layer. Relu was used as the activation function, and the model was trained with an ADAM optimizer minimizing mean squared error. After 50 epochs, the training loss was 3.816×10^{-4}, and the test loss was similarly low at 3.800×10^{-4}. These loss values were likely so similar because of the periodic nature of the steady-state transient. Figure 4.3 shows the last 5000 points, shortened to remove redundancy since the whole trend appear the same, of both expected and actual outcomes on the test data.

5 Startup Transient Twinning

Startup transients are predicted from initial states to verify the operation of the sensor network as well as verify the predictability of the transient. This is called "Twinning" because it is a system that runs alongside the native system monitoring systems. The ML task here is predicting future time-series trends of variables from initial states. Our ML model is expected to use 6 initial values as input features and output the predicted values at each time points.

Figure 4: Expected and predicted total turbine energy over time of LSTM over test set.



5.1 Dataset Description

Like section4, initial conditions were chosen to be in a realistic range, and were checked for convergence. The data points of data set were the initial value trials and 16436 trials in total. There were a total of 11 variables to be predicted, each with 500 data points in second. Table 7 shows the summary of our data set.

Table 7: Summary of the variables.

	N of time_pints	mean	std	min	max
qdot_boil	1	22,010.328	8,512.666	10,000.605	49,809.601
Pout_frac	1	0.276	0.129	0.050	0.499
Pout_time	1	20.7320	6.290	4.082	29.999
mdot	1	319.506	11.530	300.000	339.998
T0	1	12.657	3.992	5.002	19.998
P0	1	29.269	17.465	17.465	59.991
h1_log	500	1,334.039	438.906	118.165	2,719.731
h2_log	500	1,333.937	438.823	118.160	2,719.608
Eturb_log	500	0.425	0.696	0.00	6.071
omega_log	500	1.937	1.386	0.011	8.997
P1_log	500	12.646	3.984	5.000	19.999
P2_log	500	12.613	3.987	4.848	19.991
Pout_log	500	0.085	0.147	0.00	2.902
rho1_log	500	611.704	280.204	30.865	1,005.254
rho2_log	500	610.934	280.873	30.810	1,005.236
T1_log	500	545.222	57.867	300.000	623.146
T2_log	500	545.106	57.793	299.999	623.124
Tboil_log	500	387.932	27.321	300.000	449.986
x1_log	500	0.096	0.190	0.000	0.979
x2_log	500	0.096	0.190	0.000	0.979

5.2 Model Description

Since we can use only 6 input features and have to predict 500 time points for each variables, we used Deep Neural Network model. We built 11 separate models for each variables, but set the layers of networks, hyper-parameters and loss function in common. Our fully connected neural network has 1: input layer with ReLU activation function, 2: 2 hidden layers with 64 and 32 hidden nodes, respectively, 50% dropout and ReLU activation function, 3: output layers. We used mean squared error as loss function. We trained our network by ADAM with 0.01 learning rate for 50 epochs. All of these hyperparameters were found through rigorous testing (as in Section 2 and 3).

5.3 Model Performance

To evaluate the accuracy of the model, we split 70% into training data and 30% into test data.Then, we used mean absolute percentage error (MAPE) as metrics. While mean squared error is affected by the distribution of predicted value, MAPE is less affected and can be compared with different predicted variables. MAPE takes values equal or grater than 0, with 0 indicating no error and increasing error as it increases. The formula of MAPE is as follows.

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\|\frac{y_{true} - y_{predict}}{y_{true}}\|. \tag{2}$$

Figure 5: The distribution of absolute percent error of each data samples. Cross Marks shows the mean and red line shows the median.

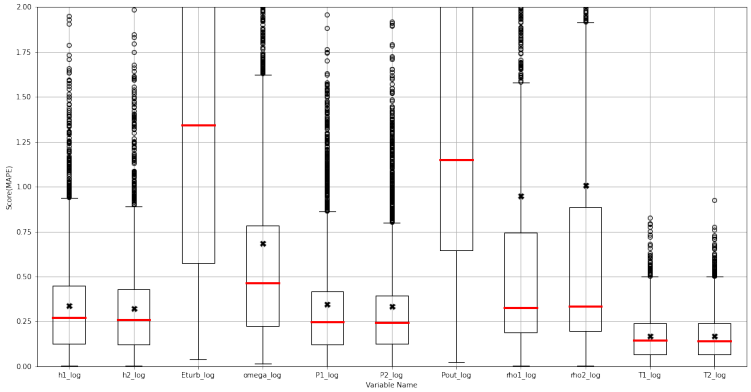


Figure 5.3 shows the distribution of absolute percentage error of our models.The cross marks indicate the actually MAPE scores. According to the figure, our model performance is not good overall. Even the best variable(T2_log) is a MAPE of 0.15.

This result is not surprising, since our model must predict 500 values with only 6 values, which is too little information and therefore less accurate. To sum up, we can conclude that the ML approach is not optimal for Twinning task

6 Related Works

In our proposal, we discussed the idea of rankine cycle simulation software such as Modelica and Simulink [8] which run computationally expensive numerical models (not ML based) to predict states. As mentioned earlier, our goal is to provide a cheaper surrogate model using the processes above.

A particular related work we would like to discuss is the following [9] project. They made a simulation to produce time series data for a different thermodynamic system and then trained a machine learning using this. In this paper, we see that the use of Deep Neural networks and LSTM produced an accuracy value of 92 %. Their approach was quite similar to our work in section 4. Our novelty here was with respect to the specific thermodymamic system we decided to apply our work to. Furthermore, the sections on startup state verification and accident onset prediction was novel and not considered (apart from producing transients for fail states and analytically distinguishing them) in the work above.

7 Conclusion, Possible Continuation and Contributions

Overall, we are thrilled with the accuracy of our models in sections 2,3,4 as they far surpassed the benchmarks we posed for ourselves in our progress report.

Our work in section 5, in contrast, did not meet our expectations. However, as discussed, the complexity/specifics of the problem somewhat justifies the poor results. As discussed in the intro, this is an active area of research and our results could be improved with the development of new techniques/theory.

Contributions were equal across all group members.

References

[1] [Online]. Available: https://github.com/deanrp2/EECS545final

[2] [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[4] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.

[5] D. Falessi, J. Huang, L. Narayana, J. F. Thai, and B. Turhan, "On the need of preserving order of data when validating within-project defect classifiers," 2018. [Online]. Available: https://arxiv.org/abs/1809.01510

[6] Y. Jiang, S. Yin, K. Li, H. Luo, and O. Kaynak, "Industrial applications of digital twins," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2207, p. 20200360, 2021.

[7] M. Liu, S. Fang, H. Dong, and C. Xu, "Review of digital twin about concepts, technologies, and industrial applications," *Journal of Manufacturing Systems*, vol. 58, pp. 346–361, 2021.

[8] U. of Liège, "Modelica: Sustainable energy conversion through the use of small-scale organic rankine cycles for waste heat recovery and solar applications." 2017, last accessed 10 March 2022. [Online]. Available: http://www.labothap.ulg.ac.be/staff/squoilin/

[9] M. I. Radaideh, C. Pigg, T. Kozlowski, Y. Deng, and A. Qu, "Neural-based time series forecasting of loss of coolant accidents in nuclear power plants," *Expert Systems with Applications*, vol. 160, p. 113699, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420305236