TabTransformer for Datacenter Equipment Failure Monitoring using SMOTE and Drift Detection

Brandon McCoy
github.com/McCoyBrandon/AI_Capstone

December 8, 2025

1 Abstract

Unplanned datacenter downtime can cause significant financial losses, operational disruption, and reputational damage. This project proposes a proactive machine failure prediction pipeline using tabular diagnostic data from industrial equipment. A TabTransformer architecture was implemented to capture feature dependencies through multi-head self-attention, combined with SMOTE to address severe class imbalance and TorchDrift for distribution shift monitoring. Models were evaluated on the AI4I predictive maintenance dataset with binary and multi-class failure targets. Results demonstrate strong binary failure detection accuracy and substantial improvements in minority class recall after SMOTE augmentation. Confusion matrix analysis highlights remaining challenges in distinguishing specific failure types, indicating a need for expanded datasets and continued learning pipelines. The method shows potential for enabling lightweight deployment for real-time monitoring and early intervention in large-scale datacenter environments.

2 Introduction

Datacenter downtimes can cause huge financial losses and reputational damage. Notable examples include in 2017 when Amazon's S3 service was down for a few hours with an estimate of roughly $300 million in lost revenue for S&P 500 and U.S. financial-service companies. Another being in Australia in 2023 with the Optus outage that lasted about 11–12 hours. Optus is widely used by the government and private sectors, resulting in crucial services in public safety and logistics causing failed communications and heavy delays. Ultimately costing the company A$ 2 billion in stock, senate inquiry, and large reputation lost.

These downtimes can often be a result of cascading failures, in which one problem may spark a domino effect in poorly managed systems that results in a catastrophic failure incident. But having humans track second-by-second diagnostics for a datacenter with thousands of machines on a 24/7 basis can be expensive and may cost as much as the potential downtimes themselves. The need for a proactive machine failure tracking system that can be run by a small team and is more cost-efficient is increasingly important in the modern information age and cloud computing world.

3 Related Work

To start is the paper that the AI4I dataset was published with.[1] The authors published this dataset because it is hard to get datacenter diagnostic data, as it is hard to get this type of dataset from the private sector that keeps it as proprietary. It was helpful in understanding the parameters that they were providing and some context they found in their own exploratory data analysis.

I then sought out literature on approaches that have attempted to utilize this data for various approaches. The first paper, "Leveraging Safe and Secure AI for Predictive Maintenance of Mechanical Devices Using Incremental Learning and Drift Detection," this paper was an interesting read on their methods with random forest, SVM, and DNN. They used incremental learning and drift detection and was great introduction into some of the difficulties they had with using this data. I found, "Exploring ML for Predictive Maintenance Using Imbalance Correction techniques and SHAP'[3] to also be helpful. Where the both papers addressed the class imbalance issue with the dataset utilizing the SMOTE method, the second paper provided comparisons of with and without SMOTE to show the clear advantages of over sampling for the training of their models.

Dr. Flavio Esposito recommended I read the paper "Attention is All You Need"[4]. I found this paper intriguing, and I noticed that the previous work I read on this dataset did not mention transformers. Therefore, I did some more research on transformers and specifically found papers on transformers that work well with tabular data, [5][6]. And to differentiate my work frome previous related work, I decided I wanted to attempt a model that not only provided the binary classification, but attempt to address the failure types and see how how an attention model handles it.

4 Problem Definition and Solution Design

Given diagnostic data in a tabular format for devices in a datacenter, the objective is to create a transformer that uses attention to accurately track the risk of devices failing. However, we also address issues of class imbalance due to low classification counts in comparison to the overall data. The classification should come in two forms, firstly a binary 'True' or 'False' for the risk of machine failure, and secondly a multi-class estimate for the type of failure. This includes tool wear failure (TWF), heat dissipation failure (HDF), power failure (PWF), and overstrain failure (OSF). Additionally, there needs to be an output that can specify the machine ID that is being flagged with the predicted probability of failure. Ideally, the prediction model should be able to run on a lightweight device.

5 Methodology and Implementation Details

This project implements a machine-learning pipeline designed to predict machine failure events and classify specific failure types using tabular diagnostic data from datacenter equipment. The workflow begins with extensive data pre-processing, including the removal of random failures, standardizing numerical features, and encoding the categorical machine types. Because failure events are extremely rare relative to normal operating observations, the dataset exhibited a substantial class imbalance. Which would otherwise cause traditional models to overfit to the majority class and overlook critical minority patterns.

To address this, I utilized Synthetic Minority Oversampling Technique (SMOTE) generates artificial samples by interpolating between an existing minority instance x_i and one of its k nearest minority neighbors x_{nn}. This can represented with the features of the AI4I dataset as the following.

Let the numeric features be

$$\mathbf{x}^{(n)} = [AirTemp,\ ProcTemp,\ RPM,\ Torque,\ Wear],$$

and let the categorical feature be

$$x^{(c)} \in \{L, M, H\}.$$

SMOTE–NC generates a synthetic instance according to

$$AirTemp_{new} = AirTemp_i + \delta(AirTemp_{nn} - AirTemp_i),$$

$$ProcTemp_{new} = ProcTemp_i + \delta(ProcTemp_{nn} - ProcTemp_i),$$

$$RPM_{new} = RPM_i + \delta(RPM_{nn} - RPM_i),$$

$$Torque_{new} = Torque_i + \delta(Torque_{nn} - Torque_i),$$

$$Wear_{new} = Wear_i + \delta(Wear_{nn} - Wear_i),$$

$$x^{(c)}_{new} = mode(x^{(c)}_{nn_1}, \dots, x^{(c)}_{nn_k}),$$

where $\delta \sim U(0,1)$ interpolates only the numeric components, and the categorical feature is assigned by the

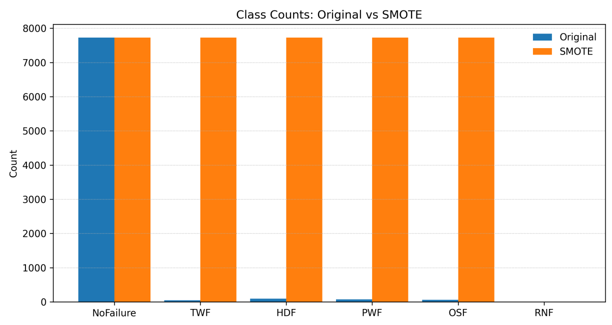

Figure 1: SMOTE Before and After

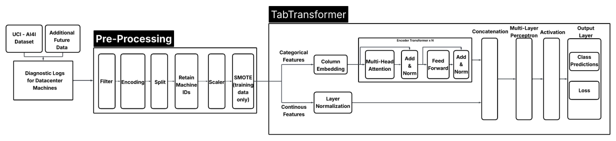

Figure 2: Machine Failure Prediction Pipeline

mode of the k nearest minority neighbors. Resulting in new minority class observations that lie within the local feature space of the minority class rather than duplicating existing instances. This improves class balance during training while preserving the statistical structure of the minority distribution. As a result, this creates a larger training set that includes more observations for minorty classes as can be seen in Figure 1. As a note, since RNF represents random failures that are not able to be addressed with explanation, I removed these observations and did not run SMOTE on them to avoid noise in the training data. The data was divided into training, validation, and separated test sets to reflect chronological behavior. The core predictive model utilizes a TabTransformer supervised learning architecture, which leverages multi-head self-attention to learn contextual relationships between tabular features. This approach allows the network to capture dependencies and interactions that linear and tree-based models may miss.

TabTransformer embeds the categorical feature ($Type$) into a d–dimensional space and applies multi–head self–attention to produce a contextualized representation $\mathbf{z}$, which captures dependencies across operational conditions. The numeric features ($AirTemp$, $ProcTemp$, RPM, $Torque$, $Wear$) are normalized and concatenated with $\mathbf{z}$ to form the final input vector,

$$\mathbf{x}_{final} = concat\big(\mathbf{x}^{(n)}, \mathbf{z}\big).$$

These inputs are then put through a multi-layer perceptron before reaching the activation and loss. As can be seen in the architecture diagram in figure 2. Seeing how it addresses the numerical values based on attention for categorical variables I believe helps with addressing the nuances of how machines of different quality (L/M/H) may have differing performing metrics within the diagnostic data that may be harder to distinguish in other models. For this model, RELU was utilized as the activation function. And the output included the probability for each machine failure classification type, including NoFailure. Although I did have concerns about how well model will hold up in real world environments when my model was heavily reliant on the generative SMOTE method. Additionally the fact that the categorial L/M/H is based on quality of the machine manufacturing and quality in equipment can change. Therefore, since machine behavior and sensor distributions can evolve over time, a drift detection module was incorporated to monitor statistical distribution shifts and performance decay during inference. This could also be used to create indicators for model retraining or pipeline recalibration before significant degradation occurs.

6 Experimental Evaluation: Settings and Results

Model performance was evaluated using macro-F1 score, recall, precision, accuracy, and AUC-ROC, with additional focus on confusion matrix analysis to verify improvements in failure type classification sensitivity. Metrics were benchmarked both before and after SMOTE augmentation to quantify and justify its impact. I found that the SMOTE method showed drastic improvement in my models. The hyperparameters for my selection for the best model that I was able to find having reasonable scores and consistent results can be seen in Table 1.

Hyperparameter	**Value**
Batch size	256
Training epochs	40
Learning rate (Adam)	0.0005
Model dimension (D_{MODEL})	128
Number of attention heads ($NHEAD$)	2
Feedforward dimension	128
Number of transformer layers	3
Dropout rate	0.0
Weight decay	1×10^{-5}
Label smoothing	0.05
Activation	ReLU

Table 1: Hyperparameters used for training the TabTransformer model.

In figure 3 we can see the confusion matrix of the checkpoint with highest macro-AUC score. I'm finding a high accuracy score in the binary classification of Machine Failure being true or false. With the recall score being lower than liked due to false positives.

However, this is a vast improvement from the pre-SMOTE model runs in which the model was over predicting false positives by the hundreds. With false positives being primarily an issue with Tool Wear Failures, I believe there is a need for more data to further train the model and to improve the recall score, but these failure flags can still be used to track potential failures before they happen.

When working on a binary 'Failure' vs 'NoFailure' using the attention-based tab transformer it was able to get high accuracy scores with relatively low adjustments and hyperparameter tuning. But it became more difficult when trying to address recall accuracy on the multi-classifier for type of failure. This has to do greatly with the class imbalance issue I ran into. The SMOTE method was great for getting a reasonable accuracy score for a baseline model. But I believe that more data and iterative learning will need to be applied to this model in order to get the failure type classifications scores higher.

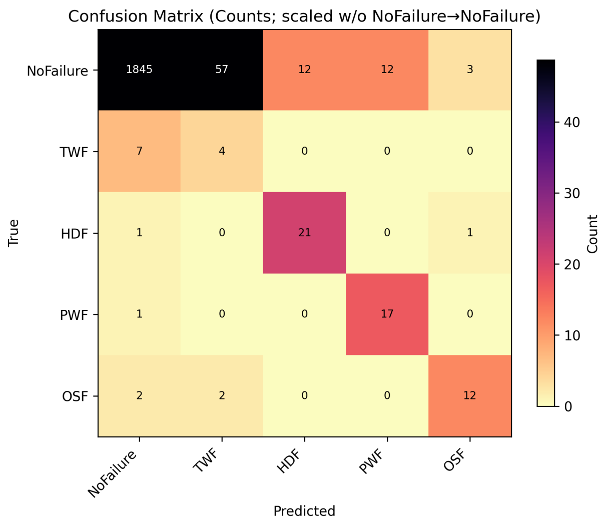

Figure 3: Failure Type Confusion Matrix

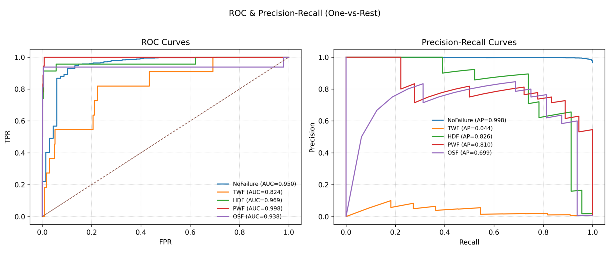

Figure 4: ROC and Precision-Recall Curves

7 Discussion, Limitations, Ethical Considerations

First and foremost, this project has been a great way to learn about how to deal with datasets that use continuous data that has a large amount of observations for 'normal' states (NoFailure for AI4I data) and a small amount of targeted events, such as machine failures for a datacenter with a large number of machines. The binary classifier of 'Failure' vs 'NoFailure' is reasonably achievable as it uses the sum of all the sub-categories of failures. But when getting granular with the failure types, using traditional class balancing technique such as weighting the class split for the train/text/validation datasets will not suffice.

This is where the Synthetic Minority Oversampling Technique (SMOTE) came in very helpful. However, one of the concerns about using the SMOTE method is that real world changes in equipment or run-times may result in changing conditions in which the prediction effectiveness degrades, or original training data doesn't fully represent real scenarios. That is why drift detection was additionally important in exploring and I believe should be expanded with in future work. For this iteration it allows to diagnosis for model performance, but in future developments of continued learning it could be utilize to tune the model.

The binary machine failure prediction as I saw in previous papers is great creating a system that may know when to automatically turn off a machine. But by having the failure types, we help IT teams to create better action plans and address maintenance in a more meaningful way. For example, have a power shortage causing a power failure would need to be addressed differently than a possible tool wear failure. And knowing how much strain a machine can handle can help with machine priority in computing or taking offline to avoid cascading failures when machines try to compensate for the loss of overs and causes an overstrain issue. However, in order to implement such action plans with a level of confidence this model would need the future development of continued learning methods and new data as I saw in the literature but was not able to implement at this time.

I don't believe there is any egregious ethical issues with this way of implementing AI either. In this case we are taking a job that would be nearly impossible to do by a human, and instead enhancing the ability of IT professionals efficiently and effectively address potential issues.

8 Conclusion

When working on a binary 'Failure' vs 'NoFailure' using the attention-based tab transformer it was able to get high accuracy scores with relatively low adjustments and hyperparameter tuning. But it became more difficult when trying to address recall accuracy on the multi-classifier for type of failure. This has to do greatly with the class imbalance issue I ran into. The SMOTE method was great for getting a reasonable accuracy score for a baseline method. And I was able to include TorchDrift to assist with further model evaluation and tracking. But I believe having more data and moving the model to integrate a continued learning enforcement will help further increase recall scores and accurate tracking types of failures and when machines start driving into possible error states.

9 References

[1] S. Matzka, “Explainable Artificial Intelligence for Predictive Maintenance Applications,” in *Proc. 2020 3rd Int. Conf. Artificial Intelligence for Industries (AI4I)*, 2020, pp. 69–74.

[2] P. B. S, M. K. M. V., N. Almuraqab, and P. B. H, “Leveraging Safe and Secure AI for Predictive Maintenance of Mechanical Devices Using Incremental Learning and Drift Detection,” Comput. Mater. Contin., vol. 83, no. 3, pp. 4979–4998, 2025.

[3] K. Patel and A. D. Shanbhag, “Exploring ML for Predictive Maintenance Using Imbalance Correction Techniques and SHAP,” in *Proc. 2022 Int. Conf. Electrical, Computer and Energy Technologies (ICECET)*, 2022, pp. 1–10.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” arXiv preprint arXiv:1706.03762, 2023.

[5] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, “Revisiting Deep Learning Models for Tabular Data,” arXiv preprint arXiv:2106.11959, 2023.

[6] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, “TabTransformer: Tabular Data Modeling Using Contextual Embeddings,” arXiv preprint arXiv:2012.06678, 2020.

10 Appendix 1. Reproducibility Checklist Using Docker

The complete end-to-end pipeline (train → eval → infer). Additional notes in GitHub README.

10.1 Build the Image

From the repository root:

```
docker build -t tabtransformer-ai4i .
```

10.2 Evaluate a Trained Model

```
docker run --rm -v "$(pwd)/runs:/app/runs" tabtransformer-ai4i \
  python -m src.training.eval \
  --ckpt runs/Test_run_1/model.ckpt \
  --out runs/Test_run_1/eval_metrics.json
```

10.3 Inference Demo

```
docker run --rm \
  -v "$(pwd)/runs:/app/runs" \
  -v "$(pwd)/src/data:/app/src/data" \
  tabtransformer-ai4i \
  python -m src.deploy.infer \
  --ckpt runs/Test_run_1/model.ckpt \
  --csv src/data/ai4i2020.csv \
  --out runs/Test_run_1/demo_metrics.json \
  --failures-csv runs/Test_run_1/flagged_failures.csv
```

10.4 Docker Summary

```
docker build -t tabtransformer-ai4i .
docker run --rm -v "$(pwd)/runs:/app/runs" tabtransformer-ai4i
docker run --rm -v "$(pwd)/runs:/app/runs" tabtransformer-ai4i \
  python -m src.training.eval \
  --ckpt runs/Test_run_1/model.ckpt \
  --out runs/Test_run_1/eval_metrics.json

docker run --rm \
  -v "$(pwd)/runs:/app/runs" \
  -v "$(pwd)/src/data:/app/src/data" \
  tabtransformer-ai4i \
  python -m src.deploy.infer \
  --ckpt runs/Test_run_1/model.ckpt \
  --csv src/data/ai4i2020.csv \
  --out runs/Test_run_1/demo_metrics.json \
  --failures-csv runs/Test_run_1/flagged_failures.csv
```

11 Appendix 2. Team Contribution Statement

My partner dropped out about half way through the semester. Up to that point, they had not contributed much, other than contributing to the initial project planning document and creating of the basic repository directory template as provided in class. But I am responsible for contributing the code and end product of this project.