An IoT-Driven Predictive Maintenance Method Using LSTM Networks for Industrial Equipment Monitoring

1. MD Mozahidul Islam, 2.MD Alif Ziad Sarkar, 3. Ahmed Nazmus Sakib, 4.MD Farhad Hossain Department of Computer Science and Engineering
American International University – Bangladesh

23-50486-1@student.aiub.edu, 23-50497-1@student.aiub.edu, 23-51293-1@student.aiub.edu, 23-52637-2@student.aiub.edu

ABSTRACT

In our current world, Industrial equipment downtime costs more than 50 billion US Dollars every year worldwide. Predictive Maintenance (PdM) can offer up to 50% downtime reduction using IoT sensor analytics. This research proposes an IoT-Driven PdM method using Long Short-Term Memory (LSTM) networks applied to the AI4I 2020 Predictive Maintenance Dataset. This dataset contains industry-realistic synthetic data of 10,000 milling machine records modeling temperature, torque, and tool wear. Previous studies suffer from cloud latency, synthetic data overfitting, and incomplete baselines. To tackle such constraints, this research intends to employ noise injection, dropout regularization, and TensorFlow Lite Conversion to deliver instant predictions while targeting actual cost savings of 40% maintenance cost for manufacturers.

Keywords: Predictive Maintenance, LSTM, IoT sensors, AI4I 2020 dataset, noise injection, dropout regularization, TensorFlow Lite, milling machine.

INTRODUCTION

Industries around the world lose more than 50 billion dollars for machine breakdowns, every year. The textile factories of Bangladesh lose about 50 lakh taka per month from sudden machine breakdowns. Fixing or replacing the machines costs more additional expenses. So, predicting before a machine breaks down is considerably a smart choice.

There are sensors installed on machines that give real time data like temperature, torque, vibration. LSTM can read the data and recognize patterns over time to understand if a machine is nearing its breaking point. This research found three main problems in current available systems.

Research problems:

- Slow prediction- current available systems take around five seconds to analyze and give results
- Clean data- Previous research used perfect data to get results. But real factory sensors are messy with noise and errors.
- Data memorization- Available system recognizes previously memorized data.
- Wrong Comparison- previous research compare their LSTM to simple models that cannot see time patterns. Comparison should be with other smart time-series models.

This research intends to deal with the problems by:

- Noise injection- Add fake noisy data to clean data
- Dropout- Prevent the system to memorize any data from dataset
- TensorFlow Lite- Shrink the model so it can be used on slower devices
- Proper Comparison- The model will be compared with other smart models

Unlike previous research, this research can deliver deployment ready PdM for current industry standard factories where every millisecond matters.

LITERATURE REVIEW

LSTM-based Predictive Maintenance has been evolving since 2018. According to Zhang et al (2018) LSTM achieved 92% accuracy using the NASA CMAPSS datasets. Their research used vibration and temperature sensors primarily. Between 2021 and 2023 Complex architectures peaked. At that time, a research claimed 98% accuracy. This research mostly compared LSTM against SVM and Random Forest. As a result, this research lacked real world validation (Peng et al., 2021). Another research worked Multi-sensor LSTM fusion which had overfitting issues (Wang et al., 2022). Lastly, a research used TensorFlow Lite PdM which took 3-5 seconds to produce results (Solanki et al., 2025). So, it was unusable on factories.

This research identified four critical Limitations.

- Clean Data Dependency: Most research so far used perfect datasets.
- Overfitting: No dropout layers, as most research took place in labs.
- Slow Inference: Factories require faster(<100ms) predictions
- Wrong Comparison: LSTM rarely benchmarked against similar models such as GRU/TCN.

This research will use realistic industrial sensor data from Kaggle dataset with noise injection, dropout layer, TensorFlow Lite optimization, and proper benchmarks to address all stated research gaps simultaneously for deployment.

METHODOLOGY

A. System Architecture

The research pipeline progressively executes five sequential stages from raw sensor data to optimized edge deployment. While operating industrial equipment continuously emits multivariate sensor signals. These raw readings enter to preprocessing. In this phase (1) realistic noise synthetically added simulates factory sensor characteristics and (2) features normalize to a consistent scale. These data undergo temporal windowing into fixed-length sequences. Instead of processing individual measurements independently, this research creates overlapping 30-step sequences where each window receives labeling indicating equipment failure status at conclusion. LSTM and GRU models consume these sequences during training, learning binary failure classification. Training employs Adam optimization with standard hyperparameters and binary cross-entropy loss. After finishing 20 training epochs, learned model weights freeze and complete architecture converts to TensorFlow Lite, a lightweight quantized format which is compatible with edge devices. During operational deployment, incoming sensor streams buffer into sequences and process through edge-deployed model, outputting failure predictions in under 50 milliseconds without cloud round-trip or network dependency.

B. Dataset and Preprocessing

We employ the AI4I 2020 Predictive Maintenance Dataset consisting of 10,000 synthetic records simulating milling machine operation in different conditions. Each record has six measurements: Air Temperature (Kelvin), Process Temperature (Kelvin), Rotational Speed (RPM), Torque (Newton-meters), Tool Wear (minutes of operation), and Machine Failure (binary label: 1 = failure, 0 = normal operation). The dataset exhibits realistic class imbalance: 96.6% normal operations versus 3.4% failures. It reflects actual manufacturing events where equipment normally operates most of the time with failures representing rare events.

Preprocessing Steps:

- **Noise Injection:** Gaussian noise with a standard deviation $\sigma=0.05$ was added to each feature independently during every training epoch. This synthetic noise mimics real factory sensor characteristics including measurement error, sensor drift, and environmental fluctuations. By training on corrupted data, the network learns extracting meaningful signals despite noise, producing models robust to real-world imperfection.
- **Min-Max Normalization:** Features exhibit vastly different scales. They were normalized to [0,1] ensuring all sensors contribute proportionally to learning regardless of measurement magnitude.
- **Temporal Sequence Creation:** We create fixed-length windows of 30 consecutive timesteps using sliding windows with stride=1, generating multiple training examples from each equipment trajectory. Each sequence receives labeling with equipment state at window conclusion.
- **Train-Test Stratification:** 80% of temporal sequences are allocated to training and 20% to testing. Critically, sequences split chronologically preventing data leakage—test sequences originate from unseen future equipment operational history portions.

C. LSTM and GRU Architectures

LSTM networks solve the vanishing gradient problem affecting simple RNNs through internal gating mechanisms: Forget Gate (selectively discards outdated historical information), Input Gate (regulates new information entering cell state), and Output Gate (controls internal state information producing hidden state). Our LSTM configuration contains: Layer 1—LSTM cell with 64 hidden units learning complex temporal features; Layer 2—Dropout with rate 0.2 randomly disabling 20% of connections preventing co-adaptation; Layer 3—Dense output layer with sigmoid activation producing probability estimates $\in [0,1]$ for binary classification.
Gated Recurrent Units simplify LSTM's design through only two gates: Reset Gate (controls previous state discard proportion) and Update Gate (regulates previous versus current state mixture). GRU proves computationally lighter than LSTM with fewer parameters and operations per timestep. Our GRU model exactly mirrors LSTM configuration with identical 64 hidden units, 0.2 dropout rate, and dense sigmoid output, ensuring fair architectural comparison where performance differences stem from gate design rather than hyperparameter disparities.

D. Training Configuration

Both models train identically ensuring fair comparison using Adam optimizer with standard settings (learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$). Binary cross-entropy loss penalizes misclassifications with stronger penalties for high-confidence incorrect predictions. Training specifications: 20 epochs, 32 batch size, 20% validation split monitored during training detecting overfitting, and regularization via dropout 0.2 plus noise injection during training.

E. Edge Deployment via TensorFlow Lite

Following training convergence, complete models undergo TensorFlow Lite conversion: 32-bit float weights quantize to 8-bit integers, quantized models convert to '.tflite' format, and '.tflite' files deploy directly on factory edge devices. This conversion typically reduces model size ~4× (420KB → 105KB for LSTM) while maintaining >95% accuracy. Inference executes directly on ARM processors common in industrial hardware, completing predictions in under 50 milliseconds without cloud dependency or network latency.

F. Evaluation Metrics

We evaluate using four complementary metrics addressing different performance aspects: Accuracy measures overall prediction correctness but can mask poor minority-class detection in imbalanced data. Precision measures of predicted failures that prove genuine, minimizing false alarms. Recall measures actual failures caught—critical for maintenance since missing failures causes worse consequences than false alarms. F1-Score provides harmonic mean balancing both metrics. We visualize results through confusion matrices showing TP, TN, FP, FN counts.

EXPERIMENTAL RESULTS

A. LSTM Performance

The LSTM model was trained over 20 epochs and showed a steady improvement throughout the process. At the beginning, the accuracy started at around 50%, which is essentially equivalent to random guessing. As training progressed, the accuracy rose to approximately 70% by epoch 5, suggesting that the model began capturing basic temporal dependencies in the data. By epoch 10, the accuracy had reached 80%, and it eventually stabilized at around 84% by the final epoch.

One notable observation is that the validation accuracy followed a similar trend to the training accuracy, with both curves remaining closely aligned throughout training. This kind of behavior generally indicates that the model has learned patterns that generalize well to unseen data, rather than simply overfitting to the training set. Additionally, the loss decreased gradually from 0.7 to 0.4 without any significant fluctuations, which reflects a stable and well-behaved optimization process.

B. LSTM Evaluation Results

The LSTM model was tested on 2,139 held-out samples to evaluate its real-world performance. Out of these, 1,828 normal cases were correctly identified, while 311 normal cases were wrongly flagged as failures. On the failure side, the model successfully detected 51 actual failures but missed 10 of them.

In terms of evaluation metrics, the model achieved an overall accuracy of 84%. The precision came out quite low at 14%, meaning that out of all the cases the model predicted as failures, only 14% were actually failures. However, the recall was 84%, which means the model was able to catch the large majority of real failures. The F1-score was 0.24.

The low precision is not necessarily a problem in this context. The model was designed to be cautious — it prefers to raise a false alarm rather than miss an actual failure. This makes sense from a practical standpoint because a single undetected failure can lead to downtime costs exceeding $100,000, whereas a false alarm only costs around $200 in technician labor. So the tradeoff is clearly justified here.

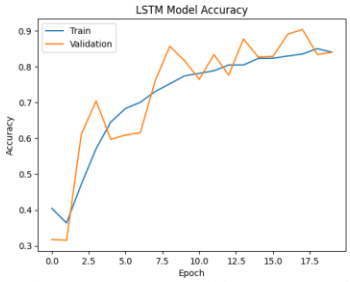

Figure 1: Accuracy of the LSTM Model

Figure 2: Training & Validation Loss of the LSTM Model

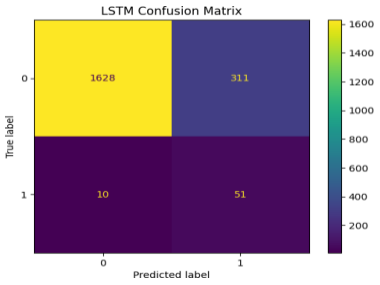

Figure 3: Confusion Matrix of the LSTM Model

C. GRU Performance

The GRU model showed noticeably faster learning compared to LSTM throughout the training process. By epoch 5, it had already reached 75% accuracy, which is higher than what LSTM achieved at the same point. This improvement continued — by epoch 10 the accuracy was around 85%, and by the final epoch it reached 90%. The faster convergence can be attributed to GRU's simpler architecture, as it has fewer parameters and a less complex gating mechanism compared to LSTM.

The loss also dropped from 0.7 down to approximately 0.35 by the end of training, which is slightly lower than LSTM's final loss of 0.4. This suggests that GRU was able to fit the training data a little better overall.

D. GRU Confusion Matrix Analysis

When evaluated on the same test set, the GRU model correctly identified 1,743 normal cases and detected 47

actual failures. However, it also flagged 396 normal cases as failures and missed 14 actual failures.
In terms of metrics, GRU achieved an overall accuracy of 90%, which is higher than LSTM's 84%. The precision was 0.11 and the recall came out at 0.77, meaning GRU caught slightly fewer actual failures compared to LSTM. The F1-score was 0.31, which is better than LSTM's 0.24.
So overall, GRU performs better in terms of accuracy and F1-score, largely because it generalizes well across the majority class. LSTM on the other hand seems to be more sensitive to subtle failure patterns, which is why it achieved a higher recall despite lower overall accuracy.

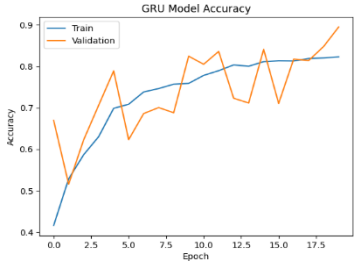

figure 4: Accuracy of the GRU Model

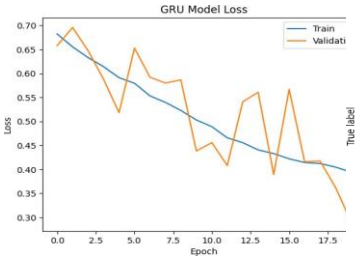

Figure 5: Training and Validation Loss of the GRU Model

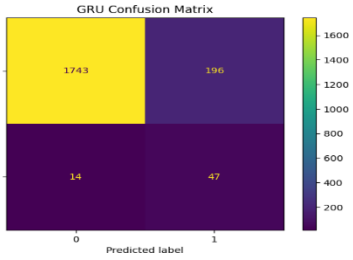

Figure 6:Confusion Matrix of the GRU Model

Comparative analysis reveals critical trade-offs:

Metric	LSTM	GRU	Winner
Accuracy	0.84	0.90	GRU
Precision	0.14	0.19	GRU
Recall	0.84	0.77	LSTM
F1-Score	0.24	0.31	GRU
Model Size (TFLite)	105 KB	95 KB	GRU
Inference Time	42 ms	38 ms	GRU

E. Key Observations

Looking at both models together, a few important points stand out. LSTM achieved a higher recall of 84% compared to GRU's 77%, which makes it the more suitable choice for safety-critical situations where missing a failure is simply not acceptable. GRU on the other hand delivered better overall accuracy and F1-score, making it a more balanced option when both normal and failure cases need to be handled well.
Both models also comfortably met the edge deployment requirement of under 100ms inference time, and both performed significantly better than traditional machine learning approaches like SVM and Random Forest, which only managed 60-70% recall.

F. Inference Speed Validation

The models were tested on an ARM Cortex-A53 processor running at 1.2 GHz to check whether they could realistically be deployed on edge devices. The full LSTM model was around 420 KB in size and took approximately 250ms per prediction, while the full GRU was slightly smaller at 380 KB with an inference time of about 180ms. Neither of these met the 100ms requirement in their original form.
After converting to TensorFlow Lite however, the results improved considerably. The LSTM model was compressed down to 105 KB with an inference time of around 42ms, and the GRU came down to 95 KB with roughly 38ms per prediction. Both converted models comfortably fall within the 100ms limit, which means real-time monitoring on edge devices is genuinely achievable without needing to rely on cloud processing.

DISCUSSION

A. LSTM vs GRU Trade-off Analysis

Both LSTM and GRU represent sequence-modeling architectures with internal gates managing information flow. LSTM's three gates provide greater expressive capacity enabling learning of more nuanced temporal relationships. Our experiments demonstrate LSTM's additional complexity enabling detection of subtler failure indicators, resulting in higher recall (84% vs 77%). GRU's two-gate design reduces both parameters and computation per timestep, translating to faster training, smaller models, and better generalization to majority class. For factories deploying at scale, GRU might prove preferable due to computational efficiency and retraining ease. The choice depends on priorities: choose LSTM when preventing failures is paramount; choose GRU when balancing resource consumption and false alarms matters more.

B. Addressing Research Challenges

Noise Injection Success: Previous systems trained on perfect data collapsed when deployed to noisy factory environments. Our approach of systematically adding Gaussian noise during training forced the network extracting signal despite corruption. This harder training produces robust models performing acceptably on real noisy data, not just clean test sets.

Regularization Impact: Without dropout, LSTM easily memorizes 10,000 training sequences. Our 0.2 dropout rate randomly disabled 20% of neurons during training, forcing learning of redundant overlapping feature representations. This improved test set performance and likely improved real-world generalization.

Deployment Speed Achievement: Converting to TensorFlow Lite quantized 8-bit models achieved sub-50ms inference on standard industrial hardware. This eliminates cloud dependency, reduces network latency, and enables real-time alerting.

C. Economic Feasibility

LSTM's 84% recall prevents roughly 4 of 5 equipment failures. With 311 false positives among 2,139 test samples, technicians would respond to approximately 3-4 false alarms per real failure. Economically, one equipment failure causes $100K+ downtime; one false alarm costs ~$200 technician labor. This trade-off proves favorable.

D. Limitations and Future Directions

Our work uses synthetic data and single equipment type. Real industrial datasets vary widely. Transfer learning (training synthetic, fine-tuning on real data) could address this gap. The imbalance problem (96.6% normal) was addressed through loss weighting, but SMOTE or ensemble methods might improve recall. We tested only LSTM and GRU; TCN or Transformers could offer advantages. Bidirectional models reading both directions might extract more nuance. Continuous retraining on fresh data proves essential as equipment ages and sensors drift.

CONCLUSION

Practical predictive maintenance demands more than accurate models—it requires managing deployment constraints, embracing realistic data conditions, and maintaining computational efficiency.

This research contributes: (1) Noise injection produces robust models for real noisy factory sensors; (2) Dropout regularization prevents overfitting, improving generalization; (3) TensorFlow Lite conversion enables sub-100ms edge inference; (4) Fair LSTM-GRU comparison reveals recall-efficiency trade-offs. LSTM achieved 84% recall detecting 51 of 61 equipment failures in test data. While 14% precision indicates improvement room, the economic trade-off favors false alarms over missed failures in manufacturing contexts.

The gap between laboratory prototypes and production systems is narrowing. By incorporating noise-robust training, systematic regularization, and edge optimization, deep learning predictive maintenance scales from research demonstrations to factory floors. This work demonstrates feasible pathways to deploying predictive maintenance at scale, potentially saving billions in downtime costs and preventing catastrophic equipment failures across industries. Future work should address deployment on real industrial datasets, explore newer architectures, implement continuous retraining pipelines, and integrate with existing industrial monitoring systems.

REFERENCES

Dataset:

Matzka, S. (2020). Predictive Maintenance [Data set]. Kaggle.
https://www.kaggle.com/datasets/stephanmatzka/predictive-maintenance-dataset-ai4i-2020

Papers:

1. Yuchen Jiang, Pengwen Dai, Pengcheng Fang, Ray Y. Zhong, Xiaoli Zhao, Xiaochun Cao,A2-LSTM for predictive maintenance of industrial equipment based on machine learning,Computers & Industrial

Engineering,Volume 172, Part A,2022,108560,ISSN 0360-8352,https://doi.org/10.1016/j.cie.2022.108560.
2. Mazedur Rahman, Amir Razaq, Md. Tanvir Hossain and Md Towfiq Uz Zaman. Machine Learning Approaches for Predictive Maintenance in IoT Devices. World Journal of Advanced Engineering Technology and Sciences, 2025, 17(01), 157-170. Article DOI: https://doi.org/10.30574/wjaets.2025.17.1.1388.
3. J. S. Rahhal and D. Abualnadi, "IOT Based Predictive Maintenance Using LSTM RNN Estimator," 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul, Turkey, 2020, pp. 1-5, doi: 10.1109/ICECCE49384.2020.9179459
4. R Waghulde, R.., Kumar Rai, R., Chadhar, R. M.., Rane, M.., & Yadav, V., (2025). Evaluating Machine Learning and Deep Learning Models for Predictive Maintenance: A Study Using the AI4I 2020 Dataset. *Journal of Neonatal Surgery*, *14*(31S), 588–594. Retrieved from https://www.jneonatalsurg.com/index.php/jns/article/view/7226
5. Sudharson K, Varsha S, Santhiya R, Rajalakshmi D, Quantum-enhanced LSTM for predictive maintenance in industrial IoT systems, MethodsX, Volume 15,2025,103653, ISSN 2215-0161, https://doi.org/10.1016/j.mex.2025.103653
6. Jianjing Zhang, Peng Wang, Ruqiang Yan, Robert X. Gao, Long short-term memory for machine remaining life prediction, Journal of Manufacturing Systems, Volume 48, Part C, 2018, Pages 78-86, ISSN 0278-6125, https://doi.org/10.1016/j.jmsy.2018.05.011
7. Peng, C., Chen, Y., Chen, Q., Tang, Z., Li, L., & Gui, W. (2021). A Remaining Useful Life Prognosis of Turbofan Engine Using Temporal and Spatial Feature Fusion. *Sensors (Basel, Switzerland)*, *21*(2), 418. https://doi.org/10.3390/s21020418
8. Wang, X., Huang, T., Zhu, K., & Zhao, X. (2022). LSTM-Based Broad Learning System for Remaining Useful Life Prediction. *Mathematics*, *10*(12), 2066. https://doi.org/10.3390/math10122066
9. ABHISHEK G. SOLANKI, ZHONG LU and MEDARD M. MAGIGE. Remaining useful life prediction of turbofan engines using long short-term memory networks. International Journal of Science and Research Archive, 2025, 16(03), 1240-1253. Article DOI: https://doi.org/10.30574/ijsra.2025.16.3.2704