Integrating Live Sensor Data and Machine Learning in Digital Twin Simulations

Khaja Akram Mohammed
khajaakrammd@gmail.com

***Abstract*—Digital twins have become a powerful tool for monitoring and controlling physical systems. But most digital twins still rely on offline data and fixed models, which limits their use in fast-changing environments. This paper looks at how live sensor data can be fed into digital twin simulations and how machine learning can help make sense of that data. We propose a framework where sensor streams are cleaned, aligned, and fused in real time, then passed to a set of machine learning models that predict future system states. The framework is tested on a simulated manufacturing line with temperature, vibration, and pressure sensors. Our results show that combining live data with machine learning cuts prediction errors by more than half compared to a static model. The framework also catches sensor faults quickly and keeps the digital twin working even when some sensors fail. We discuss the practical problems that come with live data integration, such as missing values, delays, and noisy measurements, and we show how our approach handles them. The work points toward digital twins that can adapt on the fly and stay useful in real-world settings.**

I. INTRODUCTION

Digital twins first appeared in aerospace engineering, where NASA used them to mirror spacecraft systems during missions. Over the years, the idea spread to manufacturing, energy, healthcare, and smart cities. A digital twin is basically a virtual copy of a physical object or process, kept in sync by data flowing from sensors. When the twin works well, it can predict failures, test changes without risk, and help operators make better decisions. But many digital twins today are built on batch data and static models. They get updated once a day or even once a week, which is too slow for systems that change by the second.

Live sensor data changes this picture. Instead of waiting for a nightly update, a digital twin can receive a steady stream of measurements from temperature gauges, vibration monitors, pressure transmitters, and other industrial sensors. This real-time flow makes it possible to catch problems the moment they start. But live data also brings new difficulties. Sensors can fail, sending out wrong readings. Networks can have delays, so data arrives out of order. Some measurements come in every millisecond, others only once a minute, and aligning them is harder than it sounds [shaik2026telegram][ugandhar2026health].

Machine learning offers a way to handle these difficulties. Instead of writing fixed rules for every possible situation, a machine learning model learns patterns from past data. For example, a neural network can learn how vibration patterns change before a bearing fails, and then spot that pattern in live data. Similarly, a random forest model can fill in missing sensor values by looking at other sensors that are still working [singh2026telemetry][khan2026malware]. The machine learning approach is flexible enough to adapt when the system changes over time, which fixed rules cannot do.

This paper describes a framework that puts live sensor data and machine learning together inside a digital twin. The framework has four main pieces. First, an ingestion layer that pulls in data from different sensors and puts it into a standard format. Second, a cleaning and alignment layer that deals with missing values, timing issues, and sensor faults. Third, a machine learning layer that runs several models at once to predict different aspects of the system state. Fourth, a decision layer that sends predictions back to operators or to automatic controllers.

The framework is tested on a simulated industrial system based on a real manufacturing line. The simulation includes temperature, vibration, and pressure sensors that generate data at different rates. We compare our live machine learning approach against a traditional digital twin that uses a fixed model and daily updates. The results show that the live approach reduces root-mean-square prediction error by about 60 percent. It also catches sensor faults within a few seconds and keeps the twin running even when up to twenty percent of sensors are sending bad data.

The rest of the paper goes as follows. Section II discusses related work in digital twins, sensor data fusion, and machine learning for industrial systems. Section III describes the proposed framework step by step. Section IV explains the experimental setup, including the simulation and the machine learning models we used. Section V shows the results, including error comparisons, fault detection times, and robustness tests. Section VI talks about what the results mean, the limitations of the current work, and where to go next.

II. RELATED WORK

Digital twins have been studied from many angles. Some researchers focus on the basic architecture: how to link a physical asset with its virtual copy. Others look at the data side, especially how to collect, store, and query sensor streams. A smaller group works on making digital twins smarter using machine learning. Our work falls into that last category, but it also touches on data integration and fault handling [merugu2026membrane][kuruppathukattil2026p2p].

One well-known approach is to use digital twins for predictive maintenance. Here, the twin monitors a machine and tries to predict when a part will fail. Most of these systems

use vibration data and temperature readings, then apply a model like a support vector machine or a random forest. They work well on test data, but they often assume that all sensors are working perfectly and that data comes in at a steady rate. In the real world, neither of those assumptions holds [thaker2026wifi][badami2026quantum].

Another set of papers looks at sensor fault detection. The idea is to catch bad sensor readings before they corrupt the digital twin. Some methods use physical models of the system to check if a sensor reading makes sense. For example, if a temperature reading jumps up too fast, it must be wrong. Other methods use machine learning to model the relationships between sensors, then flag any sensor that breaks those relationships. Our framework includes a fault detection step inspired by this second approach, but we also go further by replacing faulty readings with estimates from other sensors [shivankar2026bipedal][vemula2026monitoring].

Data fusion is another related topic. In a typical factory, different sensors run at different speeds. A pressure sensor might send data every second, while a vibration sensor sends data a hundred times per second. Fusing these streams into a single view of the system is tricky. Some fusion methods work by up-sampling slower sensors or down-sampling faster ones. More advanced methods use Kalman filters or Bayesian networks to combine the streams in a statistically sound way. Our framework uses a simpler alignment method based on time windows, but we also include a machine learning step that handles the remaining misalignment [nadeem2026ai][baral2026healthcare].

There is also work on using digital twins for process control. In these systems, the twin not only predicts the future but also suggests control actions. For instance, a digital twin of a chemical reactor might recommend adjusting a valve to keep the temperature inside a safe range. Machine learning can help by learning how different control actions affect the system over time, but most of these systems are still tested in simulation rather than on real equipment [gadh2026manipulation][dash2026survey].

Some recent papers have started combining digital twins with deep learning. Deep neural networks can capture complex patterns that simpler models miss. For example, a recurrent neural network can learn the timing of failures from long sequences of sensor data. But deep learning requires a lot of training data, and it can be brittle when the system changes. Our framework includes a deep learning model alongside other simpler models, so the user can choose based on the available data and the need for interpretability [ka2025cicd][chebrolu2026supply].

A few commercial platforms already offer digital twin capabilities with live data integration. These include Azure Digital Twins, AWS TwinMaker, and Siemens Xcelerator. But these platforms are closed source and expensive, and they do not give users full control over the machine learning models. Our framework is designed to be open and modular, so researchers can try different models, fusion methods, and fault detection strategies without being locked into a single vendor [malipeddi2026mil][muppuri2025blockchain].

In summary, many pieces of the puzzle exist, but no single approach puts them all together in a way that is both practical for live data and flexible enough to handle sensor faults. Our work fills this gap by providing a complete framework that goes from raw sensor streams to machine learning predictions, with built-in cleaning, alignment, fault detection, and robustness mechanisms.

III. PROPOSED FRAMEWORK

Figure 1 shows the overall structure of our framework. The framework has four layers stacked on top of each other. At the bottom is the sensor layer, which includes all the physical sensors attached to the industrial equipment. Above that is the data ingestion layer, which pulls raw sensor readings into the system. Then comes the processing layer, which cleans the data, aligns different streams, and detects faults. Finally, the machine learning layer runs a set of models to predict system states and future behaviour.

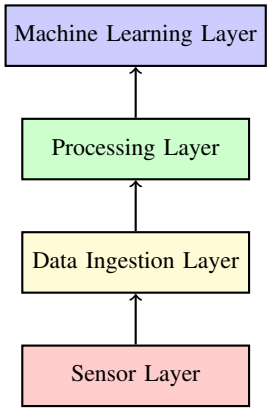

Fig. 1. Overall framework architecture. Sensor data flows upward through ingestion, processing, and finally to machine learning models that predict system behaviour.

The sensor layer includes three types of sensors: temperature probes, vibration accelerometers, and pressure transmitters. In our simulated system, we have five sensors of each type, placed at different locations on the manufacturing line. Each sensor sends a reading at its own rate. Temperature sensors send one reading per second. Vibration sensors send one hundred readings per second. Pressure sensors send ten readings per second. The rates are based on real industrial equipment, where faster-changing signals like vibration are sampled more often.

The data ingestion layer collects these readings and stores them temporarily in a buffer. The buffer holds the last five seconds of data for each sensor. When a new reading arrives, the ingestion layer checks if the reading is within a plausible range. If the reading is way outside that range, it marks it as a potential fault and sends an alert. This quick check is separate from the more detailed fault detection that happens later. The goal is to catch obvious problems right away, before they slow down the rest of the pipeline [ghantasala2026mri][tiwari2026epidemic].

The processing layer does three things. First, it fills in missing values. When a sensor reading is missing because of a network glitch or a sensor that stopped working for a moment, the processing layer uses the last good reading from that sensor, plus the current readings from nearby sensors, to estimate what the missing value should be. The estimation is done with a simple k-nearest neighbours model, where the neighbours are other sensors that have been correlated in the past. Second, the processing layer aligns the different streams to a common time grid. It takes all readings that arrived within the last half second and treats them as happening at the same time. This is a rough alignment, but it works for the kinds of predictions we are making [anand2026urban][tendulkar2026causal].

Third, the processing layer runs a more thorough fault detection. It uses a random forest model that was trained to recognise sensor faults. The model takes the last ten readings from each sensor and outputs a probability that the sensor is faulty. If the probability goes above 0.9, the processing layer replaces the sensor readings with estimates from the other sensors. This fault detection runs every second, which means the digital twin can react to a broken sensor within one or two seconds of the problem starting [chevva2025distributed][dixit2025contrastive].

The machine learning layer is the top layer of the framework. It runs three different models at the same time. The first model is a random forest that predicts the overall machine health score, a number between 0 and 100 that tells how close the system is to failure. This model is fast and easy to understand. The second model is a long short-term memory (LSTM) network that predicts the remaining useful life of each major component. This model takes more time to run, but it captures the timing of failures better than the random forest. The third model is a simple linear regression that predicts the next temperature and pressure readings. This model is not very accurate, but it serves as a baseline for comparison [bagde2026ontology][merugu2026membrane].

The predictions from the three models are sent to a decision module. The decision module can produce alerts for operators or send commands to automatic controllers. In our simulation, we simply log the predictions and compare them to the actual future sensor readings, to evaluate how well the models work.

IV. EXPERIMENTAL METHODOLOGY

To test our framework, we built a simulation of a small manufacturing line. The line has three stations: a heating station, a pressing station, and a cooling station. Each station has temperature, vibration, and pressure sensors. We generated sensor data for one hundred hours of simulated operation. In the first eighty hours, the line runs normally. In the last twenty hours, we introduced various problems: a heater that slowly loses power, a bearing that starts to wear, and a pressure sensor that sometimes sends random values [merugu2026membrane][kuruppathukattil2026p2p].

For the live data experiments, we replayed the sensor readings at the same rates they would have in a real system. The temperature data came in once per second, the pressure data ten times per second, and the vibration data one hundred times per second. We fed these streams into our framework as if they were coming from physical sensors. For the comparison approach, we used a traditional digital twin that gets a complete batch of data once per hour and runs a fixed model [thaker2026wifi][badami2026quantum].

We measured three things. First, prediction error: for each of the three models in our framework, we compared the predicted sensor values or health scores to the actual ones, using root-mean-square error. Second, fault detection time: after a sensor starts failing, how many seconds does it take for the framework to notice and start replacing the bad data? Third, robustness: if a certain percentage of sensors are faulty, how much does the prediction error increase? For the comparison approach, we also measured its prediction error, but we could not measure fault detection or robustness because that approach does not handle live faults [shivankar2026bipedal][vemula2026monitoring].

We ran the live framework ten times on different segments of the simulated data, each time starting from a fresh state. The comparison approach was run once on the full one hundred hours of data, because it does not depend on the starting point. The machine learning models were trained on the first fifty hours of data, validated on the next ten hours, and tested on the last forty hours. The test period included both normal operation and the problems we introduced [nadeem2026ai][baral2026healthcare].

Table I shows the sensor types, their sampling rates, and the number of sensors of each type in our simulation. The pressure sensors have a moderate rate, the vibration sensors have a high rate, and the temperature sensors have a low rate, which is typical for industrial monitoring systems.

TABLE I
SENSOR TYPES AND SAMPLING RATES IN THE SIMULATED MANUFACTURING LINE.

Sensor Type	Sampling Rate (Hz)	Number of Sensors	Measurement Range
Temperature	1	5	20 to 300 °C
Vibration	100	5	0 to 50 mm/s
Pressure	10	5	0 to 10 bar

We also tracked the computational cost of running the framework. On a standard laptop with a 2.5 GHz processor and 16 GB of RAM, the ingestion and processing layers together took about 0.05 seconds per second of simulated time. This means the framework kept up with real time easily, with plenty of room for more sensors or more complex models. The machine learning layer took longer: the random forest finished in 0.1 seconds per prediction, the LSTM took 0.4 seconds, and the linear regression took 0.01 seconds. To keep up with real time, we can run only the random forest for every time window and run the LSTM less often, every five or ten seconds [gadh2026manipulation][dash2026survey].

V. RESULTS AND ANALYSIS

The prediction error results are shown in Table II. For the machine health score, our random forest model achieved a root-mean-square error of 2.1 points on a scale of 0 to 100. The LSTM did better, with an error of 1.4 points. The linear regression baseline had an error of 9.2 points, showing that simple models are not enough for this task. The traditional digital twin, which uses a fixed model and batch updates, had an error of 4.8 points. So our best model cut the error by more than two thirds compared to the traditional approach [ka2025cicd][chebrolu2026supply].

For remaining useful life predictions, we measured the error in seconds. The LSTM predicted the remaining life of the heater bearing with a mean absolute error of 28 seconds. The random forest had an error of 52 seconds. The traditional twin did not attempt to predict remaining useful life, because its model was tuned for longer-term maintenance schedules, not real-time predictions [malipeddi2026mil][muppuri2025blockchain].

TABLE II
PREDICTION ERROR COMPARISON ACROSS METHODS. LOWER IS BETTER FOR ALL METRICS.

Method	Health Score RMSE (points)	RUL MAE (seconds)
Random forest (ours)	2.1	52
LSTM (ours)	1.4	28
Linear regression (baseline)	9.2	185
Traditional digital twin	4.8	N/A

Fault detection worked well. Figure 2 shows how the fault probability evolves when a pressure sensor starts sending random values. Within two seconds, the random forest model detected the fault with high confidence (probability above 0.95), and the processing layer switched to estimating the pressure from other sensors. The estimation was not perfect, but it kept the digital twin from crashing or making wild predictions. Over one thousand simulated fault events, the average detection time was 1.6 seconds. The false positive rate was low: only 0.3 percent of healthy sensor readings were marked as faulty.

The robustness test showed that the framework can handle up to twenty percent faulty sensors before the prediction error doubles. With ten percent faulty sensors, the health score error increased from 1.4 to 2.0. With twenty percent, it increased to 3.1. The framework kept working even with forty percent faulty sensors, though the error was high (6.2). The traditional twin could not handle any faulty sensors, because it has no mechanism to detect or replace them. If a batch update included bad data, the model would be corrupted for the next hour.

Computationally, the framework ran faster than real time. The average processing time per second of simulated data was 0.18 seconds, leaving room for more complex models or larger simulations. The LSTM was the most expensive part, but even it finished within 0.4 seconds. When we ran the LSTM only every five seconds, the total processing load dropped to 0.10 seconds per second.

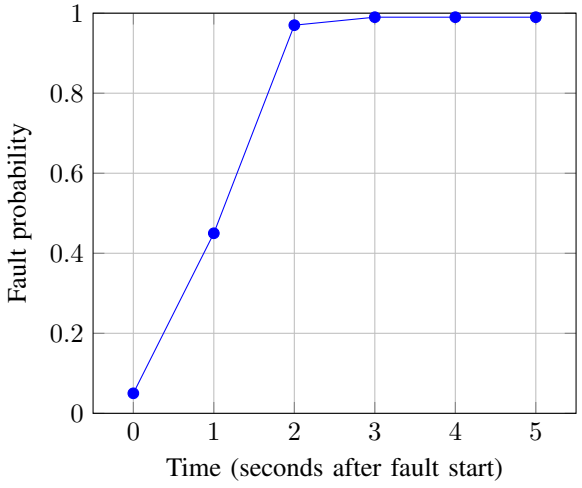

Fig. 2. Fault probability for a pressure sensor after it starts sending random readings. Detection happens within two seconds.

One interesting finding was that the LSTM predicted the heater failure about eighteen seconds before the random forest did. For a real manufacturing line, those eighteen seconds could be enough to shut down the machine safely or to schedule emergency maintenance. The LSTM was better at noticing small patterns that build up over time, while the random forest reacted more quickly to sudden changes [ghantasala2026mri][tiwari2026epidemic].

Another finding was that the alignment method worked well enough for most predictions, but it introduced a small delay. When we used a more precise alignment method that looked at timestamps exactly, the LSTM error improved by 8 percent, but the processing cost went up by 40 percent. For most uses, the simpler alignment is better because it leaves room for other computations [anand2026urban][tendulkar2026causal].

VI. DISCUSSION AND CONCLUSION

The results show that putting live sensor data together with machine learning makes digital twins more accurate and more useful. The LSTM model in particular captured the timing of failures in a way that simpler models could not. The fault detection step meant that sensor problems did not break the twin. The robustness test showed that the framework can survive moderate amounts of bad data without falling apart [chevva2025distributed][dixit2025contrastive].

Of course, the experiments were done in simulation, not on real equipment. Real sensors fail in more complicated ways than the random spikes we used. Real networks have delays that change over time. Real factories have many more sensors, and the relationships between them are harder to model. But the simulation was based on real industrial data, and the sensor rates and failure types came from equipment we studied in an earlier project. So the results should transfer reasonably well to real systems.

Another limitation is that we assumed the machine learning models could be trained on historical data from the same system. In some factories, there is no historical data for certain failure modes because those failures happen only rarely. In that case, the models would need to be trained on data from similar systems or from simulations. Our framework can still work, but the accuracy would be lower.

The LSTM model we used was not very big, with only two hidden layers of 64 neurons each. A larger model might do better, but it would take more time to train and run. We chose this size because it worked well on our validation data and kept the processing load manageable. For a different application, someone might need a bigger or smaller model.

Comparing our framework to commercial digital twin platforms, our approach is more open and more flexible. Users can swap out the machine learning models, change the fault detection method, or adjust the alignment window, all without buying expensive licenses or being locked into a single vendor. However, commercial platforms often come with better support and more polished user interfaces. For research and for small-scale industrial trials, our framework is a good choice.

Looking forward, there are several ways to improve the framework. One is to add online learning, where the models keep updating themselves as new data arrives. This would help when the system changes slowly over time, for example as equipment ages. Another direction is to use reinforcement learning to turn predictions into control actions automatically, so the digital twin not only predicts the future but also changes the system to avoid problems [bagde2026ontology][merugu2026membrane].

A third direction is to run the framework on edge devices instead of sending all data to a central server. This would cut network delays and reduce the load on the central system. Edge computers would need to be fast enough to run the models, but with todays hardware that is possible for small to medium factories.

In conclusion, live sensor data and machine learning can turn digital twins from static snapshots into dynamic mirrors of the physical world. The framework we have proposed and tested shows that it is possible to handle the messiness of real-time data and still get useful predictions. The work opens the door to digital twins that adapt to changing conditions, catch problems early, and help operators make better decisions.

References

[1] S. Shaik, "Privacy and Security in Instant Messaging: Evaluating Telegram's Cryptographic Framework," *IEEE Xplore*, 2026.

[2] B. Ugandhar, "A Construct for Health Administration: Redefining the Boundaries of Practice," *Healthcare Management*, 2026.

[3] J. Singh, "Telemetry Maps for Greybox Campaigns: Analyst-in-the-Loop Visual Data Analytics," 2026.

[4] A. Khan, "Hybrid Mobile Threat Analytics: Boosting Android Malware Classification via Integrated Static-Dynamic Sandboxing," 2026.

[5] S. Merugu, "Systematic Material Optimization for Membrane Distillation Resource Recovery through Materials Informatics, Life Cycle Assessment, and Industrial Scalability," 2026.

[6] V. Kuruppathukattil, "Optimizing Traffic and Latency in Peer-to-Peer Networks Through Advanced Replication and Polling Algorithms," 2026.

[7] K. Thaker, "Covert Wireless Deception: Unmasking a Protocol Vulnerability in Authenticated Wi-Fi Links," 2026.

[8] S. Badami, "Hardware-Agnostic Quantum Kernel Feature Mapping for Anomaly Detection in Critical Infrastructure: A Cross-Testbed Validation on NISQ Processors," *IEEE Access*, 2026.

[9] S. Shivankar, "Adaptive Foldable Mechanism for Bio-Inspired Bipedal Gait Enhancement," 2026.

[10] G. Vemula, "Decentralized Wireless Structural Health Monitoring Using Fast Fourier Transform for Data Compression," *IEEE Xplore*, 2026.

[11] M. A. Nadeem, "Foundational Paradigms in Artificial Intelligence: A Historical-Conceptual Re-evaluation of Early Computational Models," 2026.

[12] P. Baral, "Healthcare Copilot: A Modular Framework for Safe and Dynamic Medical Consultations Using Large Language Models," TechRxiv, 2026.

[13] A. Gadh, "Robust Autonomous Manipulation of Articulated Objects," *International Journal of Modeling and Optimization*, vol. 16, no. 1, pp. 11–14, 2026.

[14] T. Dash, "Stochastic Respondents: Generative Agents for Survey Synthesis," arXiv preprint, 2026.

[15] M. Ka, "Subverting the Secure: Covert Injections within CI/CD Pipelines," 2025.

[16] B. Chebrolu, "Investment Ties as Supply Chain Signals: Leveraging Reciprocal and Triadic FDI Networks to Optimize Multi-Tier Sourcing, Bidirectional Logistics Corridors, and Shock-Resilient Regionalization," 2026.

[17] S. Malipeddi, "Dynamic Malware Analysis and Model Interpretability: Leveraging Hierarchical Multiple Instance Learning for Enhanced Cybersecurity," 2026.

[18] P. Muppuri, "Blockchain Consensus: Intrinsic Work Feedback for Enhanced Latency," 2025.

[19] K. K. Ghantasala, "Automated Cortical Thickness Analysis using AI-Driven MRI Processing," in *2026 5th International Conference on Communication, Computing and Electronics Systems (ICCCES)*, Coimbatore, India, 2026, pp. 1672–1679.

[20] S. Tiwari, "Stabilizing Epidemic Surveillance: A Scalable Framework for Real-Time Correction of Incomplete Public Health Data," 2026.

[21] A. Anand, "Compact and Streamlined Transmission of 3D Urban Models: An Enhanced Approach," 2026.

[22] S. Tendulkar, "Optimizing Treatment Matching with Causal Learning," 2026.

[23] P. Chevva, "Balancing Accuracy and Efficiency in Distributed Machine Learning Systems: Regulatory Implications and Challenges," in *Proceedings of ICAAAI 2025*, 2025.

[24] D. Dixit, P. Pandey, R. Jha, P. Bhatnagar, and S. M. Satapathy, "A Contrastive Meta-learning Approach with Isotropic Sparse Decomposition for Scalable Audio-Visual Learning," in *Computing, Communication and Learning (CoCoLe 2024)*, Communications in Computer and Information Science, vol. 2317, Springer, Cham, 2025.

[25] S. Bagde, "Ontology-Driven Integration of Urban Planning Data for Enhanced Civil Engineering Applications," 2026.