



SmartCNC AI: An Industry 4.0 Machine Digital Twin Framework for CNC Condition Monitoring and AI-Based Predictive Maintenance

Aryan Bhole¹, Suraj Gond², Deepak Naik³, Prof. Satish Kuchiwale⁴

Department of Computer Engineering, Smt. Indira Gandhi College of Engineering (SIGCE), Ghansoli, Navi Mumbai, Maharashtra, India Mentor,
Department of Computer Engineering, Smt. Indira Gandhi College of Engineering (SIGCE), Ghansoli, Navi Mumbai, Maharashtra, India

ABSTRACT :

Computer numerical control (CNC) machines are central to modern manufacturing, and unplanned downtime, tool breakage, and spindle or bearing failure are costly. Industry 4.0 technologies allow machine condition to be sensed, transmitted, and analysed continuously. A stream of sensor readings, or an alarm triggered by a threshold, is not by itself a machine digital twin, and a prototype that has not been validated must never be used to make safety-critical decisions. This paper describes SmartCNC AI, a proposed framework based on the Industrial Internet of Things (IIoT) that connects timestamped vibration, current, temperature, and machine-state observations to a structured virtual machine state, keeps recent history together with data-quality metadata, and supports review of anomalies and maintenance recommendations through transparent baselines and candidate machine-learning methods. The planned prototype pairs an ESP32-class edge controller with a MEMS accelerometer, a spindle-current sensor, and a temperature sensor, while further channels such as acoustic sensing and controller data via MTConnect or OPC UA remain optional and subject to verification. The study plan keeps offline analysis of public datasets distinct from later hardware testing. The candidate datasets, the NASA milling tool-wear dataset and the AI4I 2020 predictive maintenance dataset, carry labels, machines, and operating conditions that do not represent every real shop floor. The paper sets out an architecture, a data schema, an evaluation protocol, implementation milestones, and safety and data-governance safeguards. At this stage no physical prototype, trained model, or experimental result is claimed. The intended contribution is a reproducible evaluation framework for testing whether a machine-specific, context-aware baseline offers value beyond fixed-threshold condition monitoring on a clearly defined task.

Keywords: Industry 4.0, Industrial IoT, CNC Machine, Predictive Maintenance, Condition Monitoring, Anomaly Detection, Digital Twin, Tool Wear

1. Introduction

Condition monitoring relies on sensors, communication networks, and information systems to gather observations about machine health during operation. On a CNC machine, vibration, spindle and axis motor current, temperature, and acoustic signals change as tools wear, bearings degrade, and process conditions drift. Networked services can store, display, and analyse these observations. Artificial intelligence (AI) can reveal patterns that no single rule would capture, yet how useful it is depends on the quality of the data, the representativeness of the evaluation, and the care taken in reading its output (Zhang et al., 2019; Carvalho et al., 2019). A screen that merely shows live readings is a dashboard. Calling a system a digital twin is a stronger claim: it implies that a structured model of one machine is held in software and revised each time new observations come in.

Industry 4.0 and cyber-physical systems (CPS) architectures describe how physical machines can be connected to analytics and decision support (Lee et al., 2015). Maintenance strategies have moved from reactive repair, to fixed-interval preventive maintenance, to condition-based and predictive maintenance (Jardine et al., 2006). The label "predictive" should not be applied casually. Here, the machine twin means a software-held state made up of the most recent valid sensor readings, their timestamps, short-term history, signal-quality indicators, data freshness, machine operating context (for example spindle speed, feed rate, and program or tool in use), and baseline summaries once enough data have accumulated. This representation is intentionally limited and open to audit, and it does not attempt to simulate the physics of cutting or of the machine structure.

SmartCNC AI is put forward as a student-level engineering framework that ties CNC sensor observations to a timestamped machine state and to a workflow in which a maintenance engineer reviews unusual patterns. The smallest hardware set consists of an ESP32 development board, a MEMS accelerometer mounted near the spindle housing, and a non-invasive current sensor on the spindle motor supply. A temperature sensor, an acoustic sensor, a display, and a beacon could be added later, and reading data from the machine controller could be added where access is permitted, but only once their purpose, compatibility, and safety consequences have been checked. The design leaves room for such extensions, and the paper keeps planned features clearly apart from those already built.

The main research question is this: on a clearly specified public machining or equipment dataset, does a machine-specific or context-aware baseline reveal anything useful that a transparent fixed-threshold baseline does not? The question is deliberately narrower than whether the system can predict failures on a real production line. Public datasets allow offline study of algorithms, but they cannot on their own validate the proposed hardware, demonstrate economic benefit, or show real-time behaviour from sensor to screen.

Four contributions are planned: (1) a modular architecture that keeps sensing, validation, communication, twin-state management, analytics, and presentation separate; (2) a defined policy for updating state and handling stale or missing data; (3) a repeatable protocol for comparing a fixed-

threshold baseline with a candidate statistical or machine-learning method; and (4) reporting guidelines that stop anomaly scores, health indices, or remaining-useful-life estimates from being passed off as confirmed failure predictions.

2. Related Work

2.1. Industry 4.0, Cyber-Physical Systems, and Digital Twins

Lee et al. (2015) proposed a layered CPS architecture for Industry 4.0 manufacturing, running from connection and conversion of data to cognition and configuration. Digital twins in manufacturing range from simple data-linked models to detailed physics-based simulations, and the terminology is used inconsistently. This motivates a tightly defined scope for SmartCNC AI. The first version of the twin is a synchronized, structured record of machine state, not a mechanistic model of the machine, which would demand validated dynamic models and calibration data beyond what this project has.

2.2. Machine Learning for Predictive Maintenance

Carvalho et al. (2019) systematically reviewed machine-learning methods applied to predictive maintenance and noted the diversity of algorithms, the scarcity of run-to-failure data, and the difficulty of comparing studies. Zhang et al. (2019) surveyed data-driven methods for maintenance of industrial equipment, covering fault detection, diagnosis, and prognosis. Jardine et al. (2006) distinguished diagnostics (what is wrong) from prognostics (how long until failure), a distinction that this paper keeps, because anomaly detection, fault classification, and remaining-useful-life (RUL) estimation need different labels and different evidence.

2.3. IoT Architecture and Industrial Protocols

For a small prototype, Wi-Fi combined with MQTT or HTTPS is a sensible starting point. Industrial deployments often use OPC UA (IEC 62541) or MTConnect to expose machine-controller data in a standard form. The choice of protocol should be driven by the needs of the actual deployment and is not in itself a research contribution. Message identity, timestamps, units, reconnection behaviour, network segmentation, and access control matter at least as much as the transport used.

2.4. Anomaly Detection and Prognostics

Liu, Ting, and Zhou (2008) introduced Isolation Forest, which separates observations by random partitioning and yields an anomaly score. That score measures how unusual a sample is compared with the training data; it says nothing by itself about whether a component is failing. Statistical process control, rolling baselines, spectral features, autoencoders, and rule-based approaches are alternatives. Whichever is chosen should be justified by the features available, whether reference labels exist, and how easily false alarms can be explained.

2.5. Public Datasets

The NASA milling dataset (Agogino & Goebel, 2007) holds vibration, acoustic emission, and motor-current recordings from milling experiments under varied cutting conditions, with measured flank wear on the cutting tool for many runs. It suits tool-wear analysis but covers a limited number of tools and conditions. The AI4I 2020 dataset (Matzka, 2020) is a synthetic predictive maintenance dataset with process variables such as air and process temperature, rotational speed, torque, and tool wear, together with failure labels. Because it is synthetic, it suits method development and not claims about real machines. Other candidates include the Case Western Reserve University bearing data (Smith & Randall, 2015) for bearing fault diagnosis and the NASA C-MAPSS dataset (Saxena et al., 2008) for RUL estimation, although neither is CNC-specific. Whether a dataset is suitable therefore turns on the exact question asked and on what its labels actually mean.

Digital twins, IIoT sensing, and anomaly detection are not new in themselves, so the opportunity lies elsewhere. SmartCNC AI proposes to bring together data freshness, explicit state handling, operating-context-aware baselines, and a transparent maintenance-review workflow within a single testable engineering pipeline. Whether this combination actually improves monitoring can only be settled by building it and evaluating it repeatably.

3. Problem Formulation and Objectives

A monitoring system can keep showing numbers without saying whether they are current, trustworthy, or out of the ordinary for that machine and that operation. Fixed thresholds are easy to explain, but they are only as good as the signal, the operating context, and the threshold itself. A spindle vibrating at a given level may be normal during heavy roughing and abnormal during a finishing pass. Adaptive methods can reflect machine-specific patterns, yet they may overfit, raise too many alerts, or read a change in cutting conditions as a fault. Missing or outdated readings add another hazard, because a dashboard can display a believable value even after the sensor has stopped sending data.

Let $x(t)$ be a feature vector computed from sensor observations at time t , $c(t)$ the operating context (spindle speed, feed, tool, program state), $q(t)$ the associated data-quality metadata, and $z(t)$ the virtual machine state being maintained. The state update is written $z(t) = U(z(t-1), x(t), c(t), q(t))$, where U takes in a new observation only if its timestamp and quality status pass defined checks. An anomaly module then produces a score $a(t) = A(x(t), c(t), z(t), \theta)$, in which θ stands for parameters learned from training data. A prognostic module, if later justified by suitable run-to-failure data, could produce a health index or an RUL estimate. These expressions mark out the design boundary and do not imply that any such function has been implemented.

The objectives are to: (1) specify a modular architecture running from sensor to dashboard; (2) define a state schema that holds values, timestamps, operating context, history, freshness, and quality flags; (3) audit a public dataset before settling on a task and a method; (4) compare a transparent baseline with a candidate context-aware or statistical method using the same held-out data; (5) record false positives, false negatives, data loss, and reproducibility; and (6) identify the safety, security, privacy, and validation requirements that must be met before any shop-floor use.

Scope and assumptions: at the time of writing, the project exists only as a concept. Firmware, a physical prototype, a database, a dashboard, a trained model, and experimental results have not been built or verified. The architecture and evaluation described here are proposed work. The system is intended as a read-only monitor: it does not send commands to the machine controller, does not replace the machine's own protection and emergency-stop systems, and no claim about failure-prediction performance is made.

4. Proposed System Architecture

The architecture comprises six logical layers: sensor acquisition; edge validation and timestamping; communication and storage; management of machine-twin state; anomaly and health analysis; and visualization with alerts that can be reviewed. Splitting the system this way makes each stage easier to test and prevents a sensor driver from being tied directly to a model or an interface. Fig. 1 outlines the proposed flow.

Fig. 1 – Proposed SmartCNC AI architecture (not yet implemented)

Sensors	→	Edge device (ESP32)	→	Data layer	→	Machine twin
Accelerometer, spindle current, temperature, optional controller data		Timestamping, quality checks		Wi-Fi / MQTT, storage		Current state, history, freshness, context

Machine twin → Anomaly and health analysis (threshold baseline, context-aware method) → Dashboard (trends, alerts, maintenance review).

4.1. Sensor Acquisition

A MEMS accelerometer mounted on or near the spindle housing supplies vibration data from which features such as RMS, peak value, kurtosis, and band energies can be derived. Low-cost MEMS sensors have limited bandwidth, noise, and mounting sensitivity, so the sensor, its range, sampling rate, and mounting method must be documented, and results from a low-cost sensor must not be assumed to match those from an industrial IEPE accelerometer. A non-invasive current sensor on the spindle motor supply provides a load indicator; installation on mains-connected equipment must be carried out by a qualified person and in line with electrical safety rules. A temperature sensor can monitor the spindle housing or coolant; the sensor and the measurement site must be named and justified, because surface temperature is not the same as internal bearing temperature. Detecting tool wear or tool breakage from these signals would still need a defined algorithm and validation against a suitable reference such as measured flank wear.

4.2. Edge Device

An ESP32-class board is proposed to read the sensors, stamp each reading with a time, compute basic features or buffered windows, run basic validity checks, and transmit records over Wi-Fi. High-rate vibration sampling may exceed what a small microcontroller can stream, so the design may compute features at the edge and transmit those instead of raw waveforms; this trade-off must be documented. The sampling rate and sensor settings must be recorded for every channel. The device should tell a missing observation apart from a genuine zero, keep sensor error states, and never present an old value as if it were current. If the connection drops briefly, local buffering is an option.

4.3. Communication and Storage

Structured messages can be sent to a backend using MQTT or HTTPS; the implementation should pick one route and record why. Every message should carry a machine or session identifier, a timestamp, the signal name, a value or feature vector, a unit, a quality status, and a software or schema version. Raw observations should be stored apart from derived features and model scores so that an auditor can retrace exactly how any alert came about. The monitoring network should be separated from the machine control network, and the device should be unable to write to the controller.

4.4. Machine Twin State

The state manager holds, for each signal, the latest valid observation, recent history, sensor-status metadata, the operating context, and baseline summaries wherever enough data exist. An incoming observation that fails quality checks must not replace a valid state. The state should openly show how old the data are. When the newest valid reading exceeds a configured age limit, the interface should display "stale" or "unavailable" and avoid suggesting that all is normal.

4.5. Analytics and Presentation

A transparent threshold-based baseline should be built first, after which a candidate statistical or machine-learning method can be assessed on a defined

task. The interface should show measured values, operating context, data freshness, quality warnings, the anomaly score or flag, and a short explanation of which signal or feature prompted the review. Alerts should read "unusual pattern: inspection recommended" and never "failure confirmed" or "stop the machine." Responsibility for maintenance decisions stays with a human, and the machine's own safety systems remain the only protection against hazardous conditions.

Because the design is modular, the dataset-analysis pipeline can be developed before any hardware exists, and the same feature and state definitions can later be tried on hardware observations. Offline data experiments and hardware integration tests nevertheless address different questions, so they must be reported separately.

5. Materials and Methods

5.1. Proposed Materials

The smallest prototype consists of an ESP32 development board, a MEMS accelerometer module, and a non-invasive spindle-current sensor. Possible additions are a temperature sensor, a microphone or acoustic-emission sensor, an OLED display, a status beacon, and a gateway computer, each to be included only when its role and safety implications are understood. Reading controller data (spindle load, feed override, program state) through MTConnect or OPC UA depends on what the machine supports and on permission from the machine owner. The software may be built with Python, pandas, NumPy, SciPy, scikit-learn, a lightweight time-series database, and a dashboard framework. These are intended implementation choices and do not show that a system has been completed.

5.2. Observation Schema

Each record should contain a machine or session identifier, a timestamp, the sensor or channel name, a value or feature vector, a unit, the source device, a quality flag, and sampling and configuration metadata. Operating context records should include the spindle speed, feed rate, tool identifier, and program or cycle state where available. Derived records should also state the feature-window length and the algorithm version. Time should be represented consistently, with the handling of time zones documented in the logs. The schema should separate, at a minimum, the states valid, missing, stale, out-of-range, and sensor-error.

5.3. Data-Quality Processing

Preprocessing starts with an audit of the dataset: its data dictionary, licence, run and session structure, units, sampling rates, and labels. Duplicate timestamps, implausible values, gaps in channels, and sensor errors should be counted. Any filtering or interpolation needs a justification for each signal, and interpolation must not span long gaps or hide a sensor failure. Whenever records are dropped, both the number removed and the reason should be reported.

Fig. 2 – Proposed offline experimental workflow

1. Acquire and timestamp readings
2. Validate signal quality and missing data
3. Preprocess into documented windows
4. Update twin state and baseline
5. Score with baseline / candidate model
6. Log anomaly flag for human review

5.4. Dataset Selection

The main candidate is the NASA milling dataset (Agogino & Goebel, 2007), provided its signals and wear measurements are confirmed to support the chosen task, such as separating low-wear from high-wear cutting states. The AI4I 2020 dataset (Matzka, 2020) may serve as a secondary dataset for method development on tabular process variables, with the caveat that it is synthetic. A dataset should not be picked just because it contains sensor measurements; the task must fit what its labels describe. Where no valid ground truth for faults exists, the study should examine descriptive deviations and refrain from reporting fault-detection accuracy.

5.5. Windowing and Features

For time-series work, continuous channels can be cut into fixed-length windows with a stated overlap, ideally aligned with a cutting pass or cycle. Candidate features include the mean, standard deviation, RMS, peak-to-peak, kurtosis, crest factor, spectral band energies, and current-based load summaries, depending on which channels are present. Window length should strike a balance between responsiveness and noise reduction and should be chosen on training and validation data. Feature scaling must use statistics from the training set alone, and nothing from held-out runs, tools, or machines may influence preprocessing or model selection.

5.6. Data Splitting and Leakage Control

When the aim is to generalize to tools, runs, or machines not seen before, splitting by tool, run, or machine is preferable. Dividing neighbouring windows from one run at random can overstate performance, since near-identical temporal patterns end up in both the training and test sets. For wear progression, the split must also respect time order. The split method, random seed, exclusions, feature definitions, library versions, and model parameters should all be recorded. Under cross-validation, every learned preprocessing step must be fitted inside each training fold.

5.7. Baseline and Candidate Method

The baseline should be easy to understand and suited to the defined task. It can be a documented rule, such as a fixed limit on vibration RMS or spindle current, or a statistical boundary derived from training data, but thresholds must not be made up or borrowed from unrelated machines. The candidate can use rolling machine-specific statistics conditioned on operating context, or Isolation Forest (Liu et al., 2008), provided the feature space and dataset allow it. All baseline parameters must be chosen without looking at held-out test results. If a machine-specific baseline is assessed, the least amount of earlier data needed to initialize it must be stated.

5.8. Evaluation Metrics

Where trustworthy labels exist for the task, precision, recall, F1-score, and a confusion matrix should be reported, along with class imbalance and, where suitable, variation between tools, runs, or machines. Precision is the share of flagged samples that agree with the reference label, and recall is the share of reference-labelled samples that were found. F1 combines the two but ignores the cost of a missed failure compared with the cost of an unnecessary inspection. For prognostic tasks, error in estimated RUL and how early a warning arrives before the labelled event should be reported with uncertainty. If the task lacks meaningful labels, the report should present score distributions, case studies, and limitations instead of classification accuracy.

5.9. System Metrics

Once the hardware prototype is built, the following should be measured under a documented test schedule: delay from timestamp to dashboard, message delivery success, rate of missing readings, reconnection time, delay in updating state, and uptime. Controlled bench tests such as disconnecting a sensor, interrupting the network, repeating messages, and delaying packets should be used. An offline dataset cannot supply these metrics, and they must not be inferred from how well an algorithm performs.

6. Machine-Twin State and Alert Logic

A good state representation keeps both the measured value and the context needed to interpret it. A conceptual state record might include: (1) the current value; (2) the timestamp of the last valid sample; (3) recent history or a summary window; (4) a quality and availability flag; (5) the source sensor and unit; (6) the operating context; (7) a baseline summary; and (8) the latest anomaly score together with the model version. With this structure, "no unusual value was seen" can be told apart from "no trustworthy value is available."

State updates should follow a fixed order. First, the structure and source of the incoming message are validated. Second, timestamp order, duplicate identity, range, and sensor status are checked. Third, the raw observation and its quality status are stored. Fourth, the current state is updated only if the sample meets the acceptance policy. Fifth, derived features and baseline summaries are updated under documented rules. Last, the anomaly output is computed and logged with a reference to the input window and model version. Rejected records should stay visible in diagnostic logs and not vanish without trace.

Alert handling should distinguish three situations: an unusual but valid observation, a data-quality problem, and a loss of connectivity. Each calls for different follow-up and so needs its own label. Repeated anomalies may be merged into one event to limit alert fatigue, but any such grouping rule should be documented and tested. Thresholds for stale data and for repeated alerts are engineering settings and not manufacturer limits.

A machine-specific baseline needs safeguards as well. If it adapts continuously, gradual wear could slowly be absorbed into what counts as normal, which is the very drift a predictive maintenance system is meant to expose. A cautious design could update baselines only during accepted periods, such as after a tool change or a reviewed maintenance event, or through a recalibration process that has been reviewed. Operating context should be used to avoid reading a change of cutting parameters as a fault. For this reason the paper treats baseline personalization as an experiment and does not assume it to be safer or better.

7. Implementation Plan and Reproducibility

The work is organized in stages so that each component can be shown and tested on its own. Stage 1 audits the dataset and defines the offline task. Stage 2 produces a reproducible preprocessing script that outputs versioned feature tables. Stage 3 implements a simple baseline and one candidate method. Stage 4 evaluates both on held-out tools, runs, or machines and analyses errors. Stage 5 brings up the sensors and timestamped logging on the ESP32, first on a bench rig or a non-critical machine. Stage 6 adds backend storage and a minimal state manager. Stage 7 integrates the dashboard and runs controlled network and sensor-failure tests.

The smallest demonstrable prototype should display a live or replayed observation, its timestamp, a quality and freshness indicator, a recent trend, the

operating context, and an anomaly flag that can be audited. Each alert should be traceable to the observation window, feature definition, and model version behind it. A replay mode helps with demonstrations because it makes scenarios repeatable, without suggesting that a public dataset is being streamed from a live machine.

For reproducibility, the following should be kept: dataset version, access date, licence, preprocessing configuration, feature-window definition, the train, validation, and test split, random seed, software environment, model parameters, and how the metrics were computed. A README should explain how to repeat the analysis. If code or models change after an experiment, the paper should name the new version and avoid mixing results across configurations.

No evidence of an implementation was present in the project materials reviewed for this paper, so the stages above are planned milestones. A later version should fill in the actual hardware revisions, sensor mounting, wiring, firmware, backend, database schema, model configuration, and test conditions, but only after these have been verified.

8. Proposed Experimental Evaluation

8.1. Experiment 1 – Offline Data Study

The first experiment should address a single bounded question using one dataset and one clearly defined task. One example is testing whether a context-aware statistical method separates low-wear from high-wear tool states more consistently than a fixed threshold, on condition that the wear labels justify that reading. A tool-wear label must not be recast as a prediction of catastrophic failure. If the dataset has no suitable reference labels, the experiment should be recast as descriptive anomaly scoring, and supervised classification metrics should not be reported.

The protocol should specify the dataset and its version, the channels, runs, and tools used, unit conversions, window size and overlap, the feature set, exclusions, the split strategy, baseline and candidate parameters, and the definitions of the metrics. Both methods must be tested on the same held-out records. The results table should list the task, method, test split, metric, and variability. Where possible, error analysis should examine false positives and false negatives by cutting condition, signal quality, and tool or run.

8.2. Experiment 2 – State Synchronization and Communication

Once the prototype is assembled, observations should be sent at a known interval and the messages expected compared with those the backend actually received. Controlled interruptions, delayed packets, duplicates, sensor disconnections, and reconnections should be introduced. The measurements are missing messages, update latency, detection of stale state, and recovery behaviour. Test duration, network conditions, number of trials, and the definition of each metric should be reported. These tests gauge engineering reliability and not predictive accuracy.

8.3. Experiment 3 – Sensor and Data Agreement

Where feasible and safe, the prototype's sensor output should be compared with a suitable reference instrument, such as a calibrated industrial accelerometer or a clamp power meter, under a documented protocol, with the reference instrument, time alignment, machine conditions, and agreement metric all stated. Correlation by itself does not establish agreement. Any trial on a production machine must comply with the owner's safety, access, and data-sharing rules.

The comparison should be reported even when the context-aware method does no better than the baseline. A neutral or negative outcome is still valuable because it shows the limits of the proposed integration. Claims such as "real-time accuracy," "zero downtime," or "reduced maintenance cost" should be avoided unless suitable evidence directly supports them.

9. Results and Discussion

This is a proposal-stage paper, so there are no experimental results. No model scores, dataset findings, sensor-agreement values, hardware latency figures, or maintenance outcomes are reported. This is intentional, since filling the section with plausible-looking numbers that no experiment produced would mislead readers. The final experimental paper should replace this section with real tables, figures, uncertainty estimates where fitting, and a discussion based on those results.

The central issue will be whether the chosen context-aware method adds value over the fixed-threshold baseline on the defined task. A cleaner separation of anomaly scores does not automatically mean better maintenance decisions, as differences may stem from changes in cutting parameters, tool or material variation, sensor mounting, noise, or conditions specific to the dataset. False positives can erode trust in alerts and cause needless stoppages, while false negatives can hide developing faults, and both should be examined in context.

Even strong offline performance would have limited external validity, since the laboratory sensors and machines in public datasets differ from the planned low-cost ESP32-based setup. Sampling frequency, sensor bandwidth, mounting, machine type, tooling, and preprocessing can all alter model behaviour. The offline study and the hardware prototype should therefore be described as distinct evaluation stages, joined by a carefully reasoned bridge.

A realistic outcome of this work may be an auditable engineering demonstration and not a validated failure-prediction system. The benefit of a structured twin state is that it makes freshness, missing data, operating context, historical trend, and model provenance visible. Whether that visibility helps maintenance teams make better decisions is a separate question, requiring a defined user study and long-term data from real machines.

10. Limitations, Safety, Security, and Ethics

The most immediate limitation is that SmartCNC AI has not yet been built. Public datasets may come from laboratory experiments or from synthetic generators and not from machines running real production. Wear and failure labels describe the design of the original studies and cannot be assumed to transfer to other tools, materials, or machines. Moreover, the evidence at hand cannot show that low-cost sensors will give reliable readings once mounted on a working CNC machine.

Safety comes first. The prototype is a monitoring aid and must not be relied on to protect operators or equipment. It must not interfere with the machine's controller, interlocks, or emergency stop, and any electrical installation, sensor mounting near rotating parts, and access to the machine must follow the workshop's safety procedures and be carried out by qualified personnel. Alerts should be framed as prompts for human inspection and not as stop commands.

Industrial networks are a security concern. A deployed system should use network segmentation, authenticated and encrypted connections where supported, role-based access, a read-only connection to controller data, secure update practice for firmware, and rules for log retention. Machine data can also reveal proprietary information, such as part programs, production volumes, and process parameters, so data ownership, access, and sharing must be agreed with the machine owner, and dataset licences and usage conditions must be respected. Where operator information is recorded, it should be minimized and pseudonymized.

Bias and generalizability also need attention. A model trained on one machine, tool type, or material may not carry over to another, and a baseline learned on a new machine reflects that machine's present condition, which may already include wear. Future evaluations should describe the composition of their datasets and avoid broad claims about machines and conditions that were not represented.

11. Conclusion and Future Work

This paper puts forward SmartCNC AI as a modular Industry 4.0 framework for CNC machine monitoring that connects timestamped sensor observations to a structured virtual machine state and to a limited anomaly-review workflow for predictive maintenance. The design stresses explicit data freshness, quality metadata, operating context, traceable state updates, and a clear separation between an anomaly score and a maintenance conclusion. It also lays out an evaluation plan that compares a transparent baseline with a candidate context-aware or statistical method on suitable public data.

Until implementation and evaluation are finished, the contribution remains a proposal. The next steps are to examine the selected dataset and define a defensible task, build reproducible preprocessing and baseline methods, assemble and validate the sensor prototype on a bench rig, create a minimal state manager and dashboard, and run controlled communication tests. Any later paper should report real measurements, include error analysis, and keep dataset findings distinct from hardware performance. Claims about failure prediction or cost savings should stay out of scope unless backed by appropriate data from real machines, domain-expert collaboration, and formal validation.

REFERENCES

- Agogino, A., & Goebel, K. (2007). Milling data set. BEST Lab, UC Berkeley. NASA Ames Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA.
- Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., Francisco, R. da P., Basto, J. P., & Alcalá, S. G. S. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137, 106024.
- Jardine, A. K. S., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7), 1483–1510.
- Lee, J., Bagheri, B., & Kao, H.-A. (2015). A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3, 18–23.
- Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining* (pp. 413–422). <https://doi.org/10.1109/ICDM.2008.17>
- Matzka, S. (2020). Explainable artificial intelligence for predictive maintenance applications. In *2020 Third International Conference on Artificial Intelligence for Industries (AI4I)* (pp. 69–74). IEEE. (AI4I 2020 Predictive Maintenance Dataset, UCI Machine Learning Repository.)
- Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *Proceedings of the International Conference on Prognostics and Health Management (PHM08)*. IEEE.
- Smith, W. A., & Randall, R. B. (2015). Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study. *Mechanical Systems and Signal Processing*, 64–65, 100–131.
- Zhang, W., Yang, D., & Wang, H. (2019). Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Systems Journal*, 13(3), 2213–2227.
- *Note: Verify all references, dataset versions, and sensor datasheets against the original sources before submission, and add datasheet citations for the specific sensor modules you select.*