IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 15, Issue 12, December 2025, ||Series -1|| PP 64-82

Multi-Sensor Fusion of Vibration, Stator Current and Thermal Measurements for Fault Diagnosis and Prognostics of Induction Motors: A Simulation Study

Rahul Nijhawan
Research Scholar, Invictus International.

Mr. Jagdip Singh
Assistant Professor
Department of C.S and I.T
Baring Union Christian College, Batala

Abstract
Condition monitoring of induction motors usually relies on a single measurement channel, most often vibration or stator current, and each channel is sensitive to only part of the fault spectrum. This paper quantifies how much is gained by fusing an accelerometer, a stator current sensor and winding temperature measurements, and where that gain comes from. A physically motivated signal model was used to generate 1,200 two-second records covering healthy operation, outer-race bearing defects, broken rotor bars and stator inter-turn short circuits at severities from incipient to advanced and loads from 25 % to 100 %. Eighteen features were extracted and combined through feature-level and decision-level fusion with random forest and support vector machine classifiers. Under repeated five-fold cross-validation, single sensors reached 41–69 % accuracy, while three-sensor feature-level fusion reached 92.3 % (random forest) and product-rule decision fusion 91.6 % (support vector machine). Fusion raised accuracy on incipient faults from at most 56 % to 80 %. Decision-level fusion was less accurate on clean data but degraded more gracefully when the vibration channel was corrupted or lost, retaining 69.7 % accuracy against 50.8 % for feature-level fusion. For bearing prognostics, a Mahalanobis health index built from fused features matched, but did not improve on, a vibration-only index, because the secondary sensors carry weak bearing information. The results indicate that fusion pays off mainly for diagnosis across fault types and for resilience to sensor failure, and the full simulation code is provided for reproduction.
Keywords: Sensor fusion; predictive maintenance; induction motor; fault diagnosis; motor current signature analysis; vibration analysis; remaining useful life

I. Introduction

Induction motors drive most pumps, fans, compressors and conveyors in industry, and an unplanned motor outage often stops an entire process line. Predictive maintenance aims to detect deterioration early enough that repairs can be scheduled rather than forced [1]. Large field surveys of industrial motors attribute roughly 40 % of failures to bearings, between a quarter and a third to stator windings and around 10 % to the rotor [2, 3]. A useful monitoring system therefore has to cover at least these three fault families.

Each family leaves its clearest trace in a different physical quantity. Localised bearing defects produce periodic impacts that are best observed with an accelerometer and envelope analysis [4]. Broken rotor bars create slip-dependent sidebands around the supply frequency in the stator current, which is the basis of motor current signature analysis (MCSA) [5, 6]. Stator inter-turn faults alter current harmonics and raise local winding temperature [7]. A system built on one sensor is consequently strong for some faults and blind to others, a point that has been made repeatedly in review articles [6, 8] but is less often quantified on a common data set.

**Research Scholar, Invictus International.*

Multi-sensor data fusion offers a way to combine these complementary views [9, 10]. Fusion can take place at the level of raw data, extracted features or classifier decisions, and each choice involves trade-offs in accuracy, complexity and robustness that are rarely compared directly for electric motors. Two questions are of practical interest. First, how much accuracy is gained, and for which faults and severities? Second, how does each fusion architecture behave when one sensor becomes noisy or fails, a common event in plant environments?

This paper addresses both questions with a controlled computational study. Simulation was chosen deliberately: it allows fault severity, load and sensor noise to be varied independently, and it produces exact ground truth for degradation trajectories, which is costly to obtain experimentally. The contributions are (i) a reproducible three-sensor signal model for four motor conditions, (ii) a like-for-like comparison of single-sensor, feature-level

and decision-level schemes with two classifier families, (iii) robustness tests under vibration noise and sensor dropout, and (iv) an evaluation of fused health indices for remaining useful life (RUL) estimation. All code is listed in the Appendix.

II. Related Work

Vibration-based bearing diagnosis is mature. Envelope analysis after band-pass filtering around a structural resonance remains the reference technique, and spectral kurtosis is often used to select the band [4, 11]. Benchmark studies on public data, notably the Case Western Reserve University set, have shown that many published high accuracies reflect easy test conditions rather than robust methods [12].

MCSA is attractive because current transformers are often already installed in motor control centres. Rotor bar breakage appears at $(1 \pm 2ks)f_s$, and its detection through current is well established [5, 8]. Bearing defects also modulate the air gap and produce current components at $|f_s \pm m f_o|$ [13], but these are weak and easily masked, which is why bearing data sets that record current and vibration together were created [14]. Stator inter-turn faults are typically detected from negative-sequence quantities, harmonic changes and temperature [7].

Fusion theory classifies methods by the level of abstraction at which information is merged [9, 10]. At the decision level, fixed combination rules such as the sum and product rules are simple and surprisingly effective [15]. For prognostics, reviews emphasise that health indicator quality, measured by monotonicity, trendability and prognosability, largely determines achievable RUL accuracy [16, 17], and stochastic degradation models such as exponential trends are commonly used [18, 19]. What is less available is a single study that evaluates diagnostic accuracy, robustness and prognostic value of the same fused feature set, which motivates the present work.

III. Methodology

3.1 Motor and sensor model

A four-pole, 50 Hz induction motor with a 6205-type deep-groove ball bearing was modelled. For each record, load L was drawn uniformly from 0.25 to 1.0 per unit, and slip was set to $s = (0.006 + 0.038L)$ with ±8 % random variation. Shaft frequency and the outer-race defect frequency follow from the bearing geometry:

$$f_r = f_s(1 - s)/p, \quad f_o = (N_b/2)\, f_r\, (1 - d/D) \tag{1}$$

where $p = 2$ is the number of pole pairs, $N_b = 9$ balls, $d = 7.94$ mm and $D = 39.04$ mm, giving $f_o \approx 3.585\, f_r$ (about 87 Hz at full load). Vibration and current were sampled at 10 kHz for 2 s, which gives 0.5 Hz spectral resolution; this is needed because rotor bar sidebands lie only 0.5–4.8 Hz from the supply line.

The healthy vibration signal contains components at f_r, $2f_r$ and $2f_s$ with randomised amplitudes, plus Gaussian noise ($\sigma = 0.30$ m/s²). The healthy current contains the load-dependent fundamental, a variable third harmonic representing supply unbalance and saturation, fifth and seventh harmonics, small inherent rotor-asymmetry sidebands and noise at 2 % of the fundamental. Winding temperature rise above ambient follows a quadratic load law with ±3 °C measurement scatter, and ambient temperature varies from 22 to 40 °C. Faults were injected with a severity σ_f drawn uniformly from 0.12 to 1.0, as summarised in Table 1. The bearing model uses an impulse train at f_o with 1.2 % period jitter, exciting a 3 kHz resonance with 0.8 ms decay. Rotor bar faults add current sidebands at

$$f_{brb} = (1 \pm 2s)\, f_s \tag{2}$$

and a small modulation of the shaft-rate vibration at $2sf_s$. Bearing faults also add weak current components at $|f_s \pm f_o|$ [13], and stator shorts add third-harmonic current, $2f_s$ vibration and extra heating. Signatures were scaled so that faults at the low end of the severity range are hard to separate from healthy variability, which is the regime that matters for early warning.

Table 1. Simulated fault signatures (σ_f: fault severity, 0.12–1.0).

Condition	Vibration	Stator current	Winding temperature
Healthy	f_r, $2f_r$, $2f_s$ lines; noise σ = 0.30 m/s²	Fundamental; 3rd, 5th, 7th harmonics; inherent $(1 \pm 2s)f_s$ sidebands ≈ −58 dB	Rise = $48L^2$ + 6 °C; ±3 °C scatter
Outer-race bearing	Impacts at f_o exciting 3 kHz resonance; amplitude ∝ σ_f	$\lvert f_s \pm f_o \rvert$ components, $7\times10^{-4}\sigma_f$ of fundamental	$+4\sigma_f$ °C

Condition	Vibration	Stator current	Winding temperature
Broken rotor bar	Shaft-rate line modulated at $2sf_s$	$(1 \pm 2s)f_s$ sidebands, $0.025\sigma_f$ of fundamental	$+3\sigma_f$ °C
Stator inter-turn short	$2f_s$ line increased by $0.30\sigma_f$ m/s²	3rd harmonic increased by $0.013\sigma_f$	$+14\sigma_f$ °C

3.2 Feature extraction

Eighteen features were computed for each record (Table 2). For vibration, kurtosis and crest factor are defined as

$$K = \mathrm{E}[(x-\mu)^4]/\sigma^4 - 3, \quad \mathrm{CF} = \max|x| / x_{\mathrm{rms}} \tag{3}$$

The envelope was obtained by band-pass filtering the vibration between 2 and 4 kHz (fourth-order Butterworth, zero phase) and taking the magnitude of the analytic signal, $e(t) = |x_{\mathrm{bp}}(t) + j\mathcal{H}\{x_{\mathrm{bp}}(t)\}|$. Peak envelope amplitudes in ±2 % bands around f_o and $2f_o$ were divided by the median envelope spectrum level to make them insensitive to overall vibration level. Shaft speed, measured with 0.15 % error, was used to locate f_o and the slip-dependent current sidebands. Current sidebands and harmonics are expressed relative to the fundamental, and the residual current after removing the fundamental captures broadband changes. Temperature rise was normalised by the expected rise at the load inferred from current RMS, so that a hot motor at full load is not confused with a faulty motor at part load.

Table 2. Extracted features (18 in total).

Sensor	Features
Vibration (9)	RMS, kurtosis, crest factor, skewness, amplitude at f_r, amplitude at $2f_s$, normalised envelope peak at f_o and $2f_o$, energy ratio in 2–4 kHz band
Current (7)	RMS; lower and upper $(1 \pm 2s)f_s$ sidebands (dB); 3rd and 5th harmonic ratios; mean $\lvert f_s \pm f_o \rvert$ sideband ratio; residual RMS after removing fundamental
Temperature (2)	Winding-minus-ambient rise; rise normalised by expected rise at estimated load

3.3 Fusion architectures

Three configurations were compared. In single-sensor models, a classifier is trained on one feature group. In feature-level fusion (FLF), the 18 features are concatenated into one vector before classification. In decision-level fusion (DLF), one classifier is trained per sensor and their class-probability outputs $P_j(c|\mathbf{x}_j)$ are combined by either a weighted vote or the product rule [15]:

$$P(c) = \Sigma_j\, w_j\, P_j(c|\mathbf{x}_j) \,/\, \Sigma_j\, w_j, \quad \hat{c} = \arg\max_c \Pi_j\, P_j(c|\mathbf{x}_j) \tag{4}$$

The weights w_j are the inner three-fold cross-validated accuracies of each sensor on the training fold only, so no test information leaks into the weights. A practical advantage of DLF is that a failed sensor can be dropped at run time by removing its term from (4), whereas FLF needs a value for every input; in the dropout tests, FLF inputs from the lost sensor were replaced by their training means. Two classifiers were used: a random forest (RF) with 300 trees [20] and a support vector machine (SVM) with radial basis kernel, $C = 10$ and standardised inputs [21], both implemented in scikit-learn [22]. The overall framework is shown in Fig. 1.

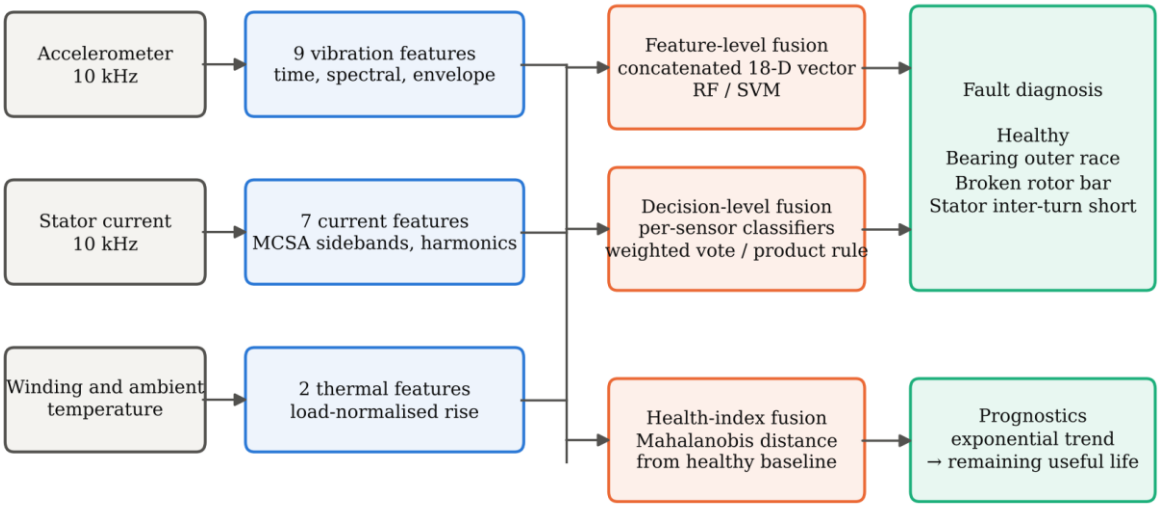


Fig. 1. Proposed multi-sensor framework for diagnosis and prognostics.

3.4 Health index and remaining useful life

For prognostics, twelve run-to-failure trajectories with outer-race degradation were simulated. Each run had a life T between 80 and 140 h (actual range 83–137 h), one record per hour and a random load between 0.4 and 1.0 per unit. Severity grew exponentially from 0.03 to 1.0 with 12 % multiplicative scatter. A health index (HI) was defined as the Mahalanobis distance of a feature vector $\mathbf{z}$ from the healthy population:

$$\mathrm{HI} = \sqrt{[(\mathbf{z}-\boldsymbol{\mu})^{\mathrm{T}}\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{z}-\boldsymbol{\mu})]} \tag{5}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ were estimated from the 300 healthy records of the diagnostic data set using Ledoit–Wolf shrinkage [23], so that the baseline is independent of the trajectories. A vibration HI used six vibration features; the fused HI added the current bearing sideband and normalised temperature rise. A five-point moving average was applied. HI quality was assessed with monotonicity $M = |\#(\Delta\mathrm{HI}>0) - \#(\Delta\mathrm{HI}<0)|/(n-1)$, trendability (absolute Spearman correlation with time) and prognosability [17]. To estimate RUL at prediction time t_{p}, a line was fitted to ln HI over the preceding 30 h, and the time at which it crosses a failure threshold θ was found:

$$t_{\mathrm{f}} = (\ln\theta - \beta_0)/\beta_1, \quad \mathrm{RUL} = t_{\mathrm{f}} - t_{\mathrm{p}} \tag{6}$$

The threshold θ for each run was the median end-of-life HI of the other eleven runs (leave-one-run-out). Predictions were made at 70 %, 80 % and 90 % of life and capped at 300 h.

3.5 Evaluation protocol

Diagnostic models were assessed with stratified five-fold cross-validation repeated three times, reporting accuracy and macro-averaged F1 score as mean ±standard deviation. Robustness tests used a fixed 70/30 split (840 training, 360 test records). For the noise test, the test records were regenerated from the same random seeds with additional white noise of standard deviation 0.3 to 2.0 m/s² on the vibration channel only, while models remained trained on clean data. Every number reported below was produced by the script in the Appendix with fixed random seeds.

IV. Results

4.1 Diagnostic accuracy

Fig. 2 illustrates the signatures on which the features rely: periodic impacts and clear envelope peaks at multiples of f_{o} for the bearing fault, sidebands around 50 Hz rising from about −60 dB to about −36 dB relative to the fundamental for the rotor fault, and a raised third harmonic for the stator fault.

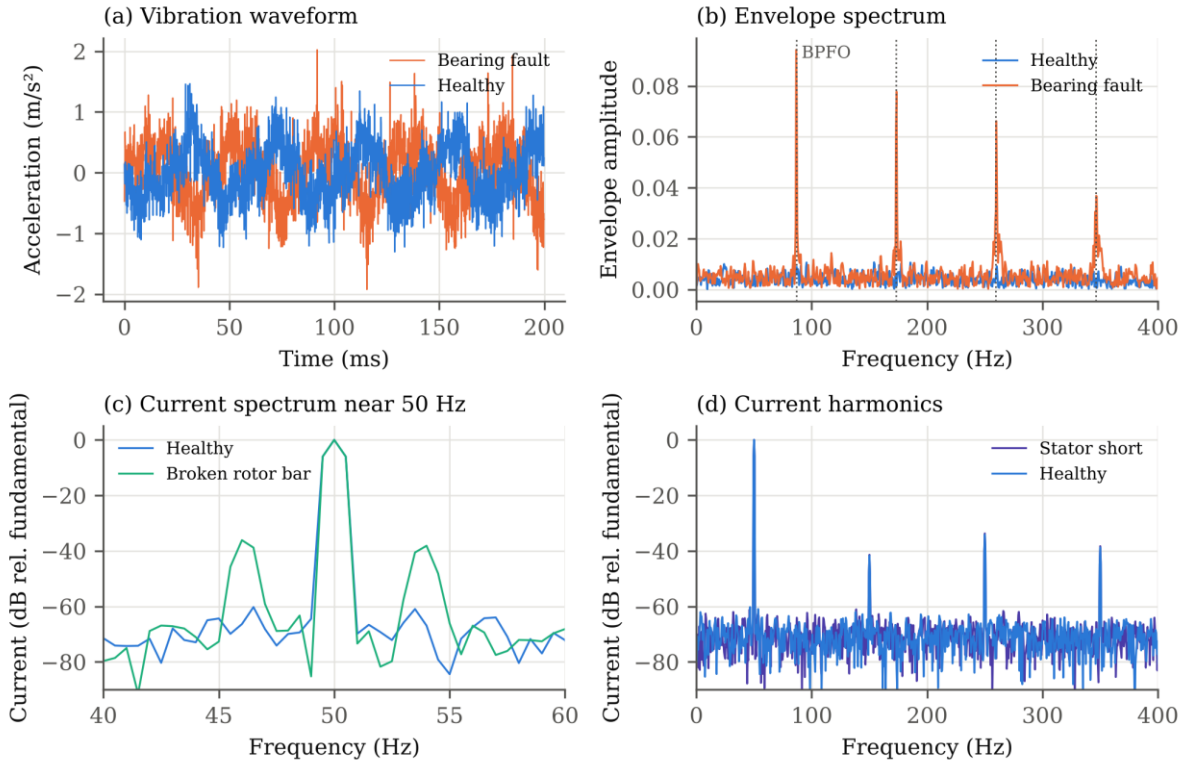


Fig. 2. Example records at 80 % load: (a) vibration waveform, (b) envelope spectrum, (c) current spectrum near the supply frequency, (d) current harmonics.

Table 3 and Fig. 3 summarise cross-validated performance. Vibration alone reached 66.5 % (RF) and 68.7 % (SVM), current alone 65.0 % and 66.4 %, and temperature alone 41.2 % and 45.4 %. Combining vibration and current at feature level raised accuracy to 89.6 % (RF), and adding temperature gave the best overall result of 92.3 ±0.3 % with a macro-F1 of 92.4 %. The two DLF rules reached 88.1–91.6 %. The product rule outperformed weighted voting for both classifiers, because it lets a confident single sensor veto classes that the others cannot exclude.

Table 3. Diagnostic performance, 3 × 5-fold cross-validation (mean ±std, %).

Configuration	RF accuracy	RF macro-F1	SVM accuracy	SVM macro-F1
Vibration only	66.5 ± 0.3	67.2 ± 0.3	68.7 ± 0.2	69.3 ± 0.2
Current only	65.0 ± 0.5	65.3 ± 0.5	66.4 ± 0.8	66.7 ± 0.7
Temperature only	41.2 ± 1.0	41.2 ± 0.9	45.4 ± 0.5	45.0 ± 0.7
FLF: vibration + current	89.6 ± 0.2	89.6 ± 0.2	88.7 ± 0.7	88.7 ± 0.7
FLF: all three sensors	92.3 ± 0.3	92.4 ± 0.3	90.8 ± 0.1	90.9 ± 0.1
DLF: weighted vote	88.1 ± 0.4	88.0 ± 0.4	89.4 ± 0.3	89.4 ± 0.3
DLF: product rule	90.5 ± 0.3	90.6 ± 0.3	91.6 ± 0.1	91.6 ± 0.1

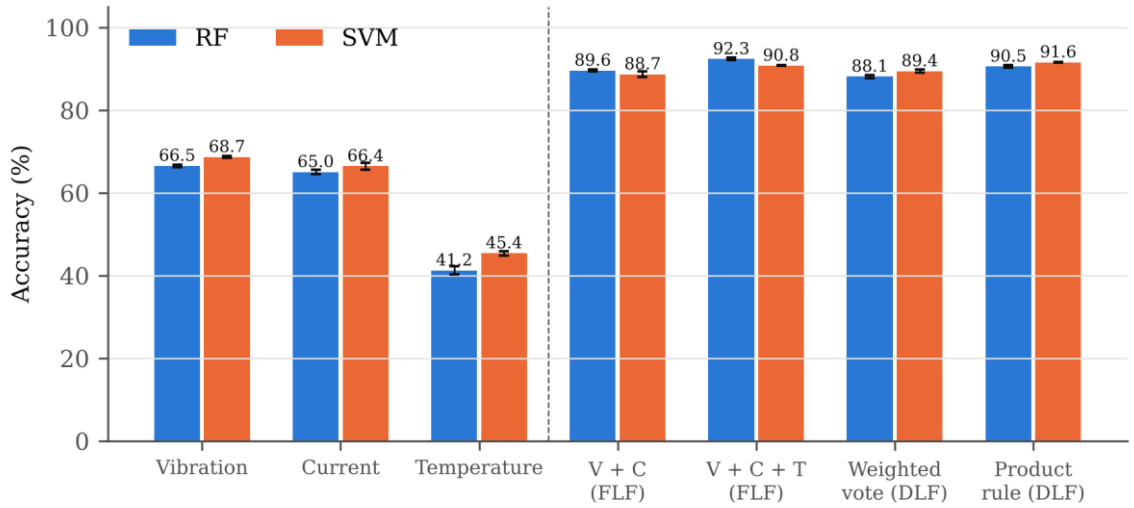


Fig. 3. Mean cross-validated accuracy (error bars: ±1 standard deviation over three repetitions). FLF: feature-level fusion; DLF: decision-level fusion.

The confusion matrices in Fig. 4 explain the gain. Vibration identifies 287 of 300 bearing faults but cannot reliably separate rotor faults from healthy motors (117 rotor records predicted as healthy). Current identifies all 300 rotor faults but confuses bearing faults with healthy operation in 106 cases. These blind spots are complementary, so the fused model inherits the strengths of both. Remaining errors in the fused model are concentrated between incipient stator faults and healthy motors (51 stator records predicted as healthy, 23 healthy records as stator), where the third-harmonic increase is comparable to normal supply variation. Random forest importances confirm that the model draws on all three sensors: the two rotor sideband features (0.162, 0.143), the two envelope features (0.128, 0.108) and the normalised temperature rise (0.092) rank highest.

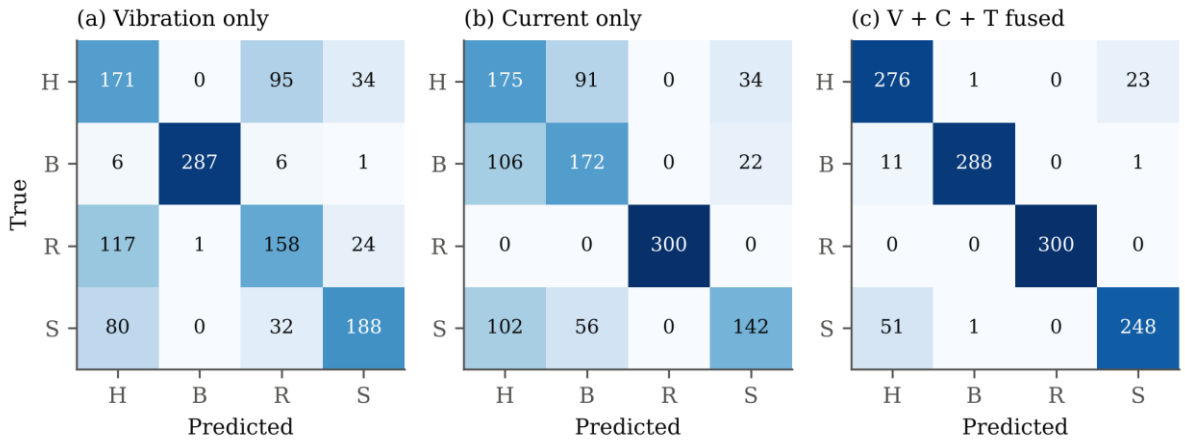


Fig. 4. Confusion matrices (RF, first cross-validation repetition). H: healthy, B: bearing, R: rotor bar, S: stator.

4.2 Effect of fault severity

Table 4 splits the fault records by severity. Fusion matters most where it is hardest to diagnose. For incipient faults ($\sigma_f < 0.4$), the best single sensor reached 56.3 % and FLF 79.5 %, an improvement of 23 percentage points. For severe faults, FLF classified every record correctly, whereas single sensors still reached only 82–84 %, because each one remains blind to at least one fault type regardless of severity.

Table 4. Fault-class accuracy (%) by severity band: incipient 0.12–0.40, moderate 0.40–0.70, severe 0.70–1.0 (RF, first repetition).

Model	Incipient (n = 288)	Moderate (n = 299)	Severe (n = 313)
Vibration only	51.7	75.9	82.1
Current only	56.3	63.2	84.0
Temperature only	25.0	39.8	50.5
FLF (all sensors)	79.5	98.3	100.0
DLF (weighted vote)	77.8	94.0	99.4

4.3 Robustness to noise and sensor loss

Fig. 5 shows test accuracy as noise is added to the vibration channel. The vibration-only model falls from 67.2 % to 38.6 % at $\sigma = 2.0$ m/s² (SNR ≈ −13 dB relative to healthy vibration). FLF remains the most accurate model up to $\sigma = 1.0$ m/s² but declines steadily to 68.1 %, because its trees have learned to rely on vibration features whose values are now shifted. DLF starts lower (86.9 %) but flattens out at about 70 %, slightly overtaking FLF at the highest noise level. Its current and temperature classifiers are unaffected by the noise, so they stabilise the combined decision.

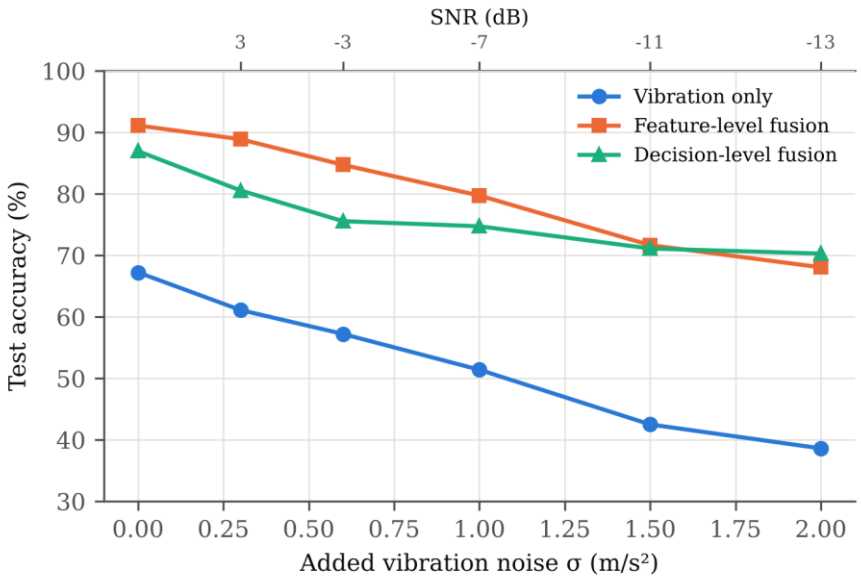


Fig. 5. Test accuracy versus added vibration noise (RF); the upper axis gives the signal-to-noise ratio relative to healthy vibration.

Complete sensor loss produces a sharper contrast (Table 5). When the accelerometer is lost, mean-imputed FLF drops to 50.8 %, whereas DLF, which simply drops the missing classifier, retains 69.7 %. When the current sensor is lost, the figures are 63.3 % and 68.3 %. Losing the temperature sensor has little effect on either scheme. For an installation where sensor availability cannot be guaranteed, these figures argue for DLF, or for a hybrid that switches to DLF when a channel fails self-checks.

Table 5. Test accuracy (%) with one sensor unavailable (RF, 360 test records).

Unavailable sensor	FLF (mean-imputed)	DLF (sensor removed)
None	91.1	86.9
Vibration	50.8	69.7

Unavailable sensor	FLF (mean-imputed)	DLF (sensor removed)
Current	63.3	68.3
Temperature	87.5	85.0

4.4 Prognostic performance

Fig. 6 plots the health indices for all twelve runs against normalised life. With nominal sensors, both indices remain flat at about 2–3 for the first 40 % of life, then rise by almost two orders of magnitude. The vibration and fused indices are practically indistinguishable, and their quality metrics in Table 6 are within 0.01 of each other (trendability 0.94, prognosability 0.91). Both outperform raw kurtosis, whose trendability is only 0.75, and match the envelope amplitude at f_o, which is itself the dominant feature in both indices.

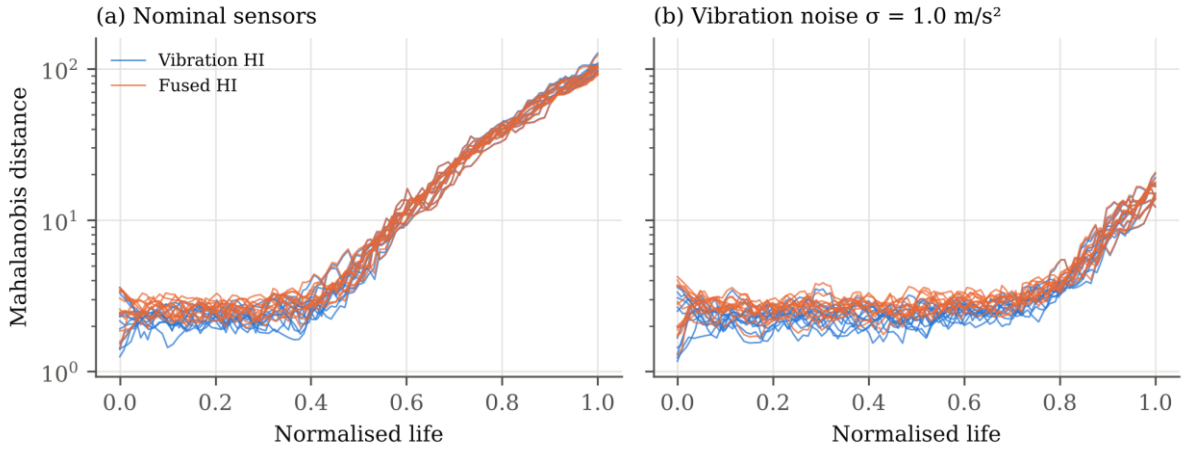


Fig. 6. Mahalanobis health indices of the twelve run-to-failure trajectories (log scale): (a) nominal sensors; (b) vibration noise σ = 1.0 m/s².

RUL errors follow the same pattern. With nominal sensors, the RMSE at 70 %, 80 % and 90 % of life was 10.4, 8.4 and 4.7 h for the vibration index, and 9.7, 8.3 and 4.7 h for the fused index. The bias was negative in all cases (−4 to −9 h), meaning that failure was predicted early, which is the safe direction for maintenance planning (Fig. 7). With vibration noise of σ = 1.0 m/s², the indices stay flat until about 75 % of life. Extrapolation at 70 % and 80 % then fails for both indices, giving RMSE above 140 h because the fitted slope is close to zero. At 90 % of life, the RMSE is 21.0 h for the vibration index and 23.5 h for the fused index.

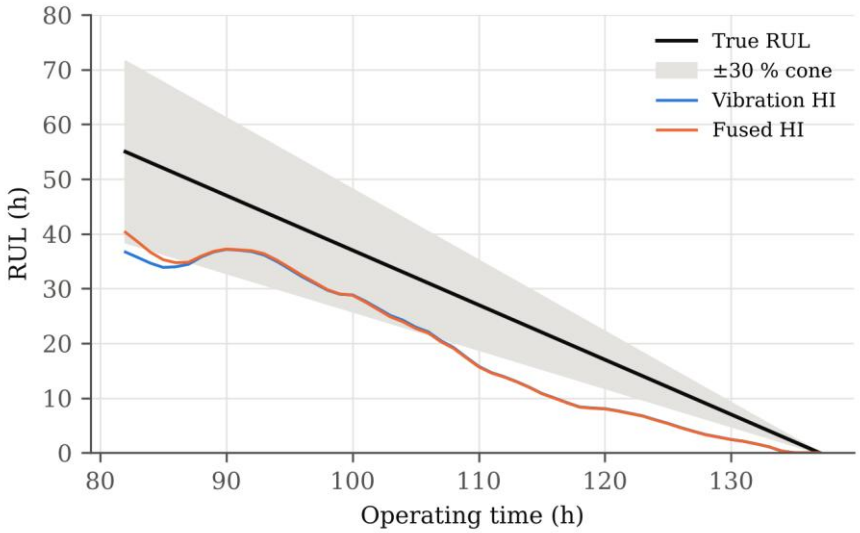


Fig. 7. RUL predictions for one trajectory (life 137 h) with nominal sensors; the shaded region is a ±30 % accuracy cone.

Table 6. Health-index quality and RUL error (12 trajectories).

Scenario	Indicator	Monotonicity	Trendability	Prognosability	RUL RMSE (h) at 70 / 80 / 90 % life
Nominal	Kurtosis	0.173	0.750	0.814	–
Nominal	Envelope peak at f_o	0.357	0.935	0.932	–
Nominal	Vibration HI	0.349	0.941	0.913	10.4 / 8.4 / 4.7
Nominal	Fused HI	0.352	0.935	0.915	9.7 / 8.3 / 4.7
Noisy vibration	Kurtosis	0.088	0.388	0.780	–
Noisy vibration	Envelope peak at f_o	0.108	0.597	0.855	–
Noisy vibration	Vibration HI	0.168	0.679	0.847	235.7 / 166.7 / 21.0
Noisy vibration	Fused HI	0.144	0.676	0.842	243.6 / 193.4 / 23.5

This is a negative result for prognostic fusion in this setting. The secondary channels carry only weak information about bearing wear: the current sideband at $|f_s \pm f_o|$ is at least 60 dB below the fundamental, and the bearing contribution to temperature rise is small compared with the thermal effect of load. Adding these features therefore neither sharpens the trend nor protects it when vibration is corrupted.

V. Discussion

The results support a nuanced view of sensor fusion for motor maintenance. Its main benefit is coverage. No single sensor in this study exceeded 69 % accuracy, because each is structurally blind to at least one fault family, while fusion of vibration and current alone exceeded 88 %. Temperature contributed a further 2–3 points, mainly by separating stator faults from healthy motors under varying load, which shows that cheap, slow sensors can still add value when they are normalised for operating condition. The size of the incipient-fault gain (23 points) is relevant in practice because early detection is the purpose of predictive maintenance.

The comparison of fusion architectures shows a clear trade-off. FLF learns interactions between sensors and is the most accurate when every channel is healthy. DLF is more modular, and its accuracy degrades gracefully when a channel is noisy or missing. A deployed system could use FLF by default and switch to DLF when sensor self-diagnostics flag a fault.

For prognostics, fusion is not automatically beneficial. A fused index helps only if the added sensors observe the same degradation mechanism with useful sensitivity. For bearing wear, vibration dominates. Fusion would be expected to help for degradation processes with multi-physical signatures, such as insulation ageing, which affects both temperature and current, or for combining redundant accelerometers. Investigating these cases is a natural next step.

Several limitations should be stated. The data are synthetic, and although the fault signatures follow established physics [4, 5, 7, 13], real machines add effects that were not modelled: transmission path variability, inverter-supply harmonics, coupled faults, and load torque oscillations that mimic rotor faults in the current. The absolute accuracies should therefore not be compared directly with experimental studies; the relative comparisons between sensors and fusion schemes are the main finding. Validation on public multi-sensor data such as the Paderborn bearing set [14], and on laboratory tests with seeded rotor and stator faults, is required before these conclusions are applied in practice.

VI. Conclusion

This study compared single-sensor, feature-level and decision-level fusion of vibration, stator current and temperature for diagnosis and prognostics of induction motors on a controlled simulated data set. Fusion raised four-class diagnostic accuracy from at most 69 % to 92.3 % and improved incipient fault recognition by 23 percentage points. Decision-level fusion was slightly less accurate than feature-level fusion under nominal conditions but retained substantially more accuracy when the vibration channel was lost (69.7 % versus 50.8 %). For bearing prognostics, a fused Mahalanobis health index offered no measurable advantage over a vibration-only index, which shows that the value of fusion depends on whether the added sensors observe the degradation mechanism. Future work will validate the framework on experimental data and extend the prognostic analysis to stator insulation degradation.

Declarations

Use of AI-assisted tools. [Edit to match the target journal's policy.] The authors used an AI assistant (Claude, Anthropic) to help draft the manuscript text and the simulation code. The authors reviewed, executed and verified all code and results, edited the text, and take full responsibility for the content of this article.

Data and code availability. All data were generated by the code listed in the Appendix, which reproduces every reported result with fixed random seeds (Python 3, NumPy, SciPy, scikit-learn, Matplotlib).

Funding. [State funding sources, or "This research received no external funding."]

Conflict of interest. The authors declare no conflict of interest.

References

[1] R. K. Mobley, *An Introduction to Predictive Maintenance*, 2nd ed. Amsterdam: Butterworth-Heinemann, 2002.

[2] IEEE Motor Reliability Working Group, "Report of large motor reliability survey of industrial and commercial installations, Part I," *IEEE Trans. Ind. Appl.*, vol. IA-21, no. 4, pp. 853–864, 1985.

[3] O. V. Thorsen and M. Dalva, "A survey of faults on induction motors in offshore oil industry, petrochemical industry, gas terminals, and oil refineries," *IEEE Trans. Ind. Appl.*, vol. 31, no. 5, pp. 1186–1196, 1995.

[4] R. B. Randall and J. Antoni, "Rolling element bearing diagnostics—A tutorial," *Mech. Syst. Signal Process.*, vol. 25, no. 2, pp. 485–520, 2011.

[5] M. E. H. Benbouzid, "A review of induction motors signature analysis as a medium for faults detection," *IEEE Trans. Ind. Electron.*, vol. 47, no. 5, pp. 984–993, 2000.

[6] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors—A review," *IEEE Trans. Energy Convers.*, vol. 20, no. 4, pp. 719–729, 2005.

[7] A. Gandhi, T. Corrigan, and L. Parsa, "Recent advances in modeling and online detection of stator interturn faults in electrical motors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 5, pp. 1564–1575, 2011.

[8] A. Bellini, F. Filippetti, C. Tassoni, and G.-A. Capolino, "Advances in diagnostic techniques for induction machines," *IEEE Trans. Ind. Electron.*, vol. 55, no. 12, pp. 4109–4126, 2008.

[9] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proc. IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[10] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Inf. Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[11] J. Antoni and R. B. Randall, "The spectral kurtosis: application to the vibratory surveillance and diagnostics of rotating machines," *Mech. Syst. Signal Process.*, vol. 20, no. 2, pp. 308–331, 2006.

[12] W. A. Smith and R. B. Randall, "Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study," *Mech. Syst. Signal Process.*, vols. 64–65, pp. 100–131, 2015.

[13] R. R. Schoen, T. G. Habetler, F. Kamran, and R. G. Bartheld, "Motor bearing damage detection using stator current monitoring," *IEEE Trans. Ind. Appl.*, vol. 31, no. 6, pp. 1274–1279, 1995.

[14] C. Lessmeier, J. K. Kimotho, D. Zimmer, and W. Sextro, "Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification," in *Proc. Eur. Conf. Prognostics Health Manag. Soc.*, 2016.

[15] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 3, pp. 226–239, 1998.

[16] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, "Machinery health prognostics: A systematic review from data acquisition to RUL prediction," *Mech. Syst. Signal Process.*, vol. 104, pp. 799–834, 2018.

[17] J. Coble and J. W. Hines, "Identifying optimal prognostic parameters from data: A genetic algorithms approach," in *Proc. Annu. Conf. Prognostics Health Manag. Soc.*, 2009.

[18] N. Z. Gebraeel, M. A. Lawley, R. Li, and J. K. Ryan, "Residual-life distributions from component degradation signals: A Bayesian approach," *IIE Trans.*, vol. 37, no. 6, pp. 543–557, 2005.

[19] A. K. S. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mech. Syst. Signal Process.*, vol. 20, no. 7, pp. 1483–1510, 2006.
[20] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[21] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[22] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[23] O. Ledoit and M. Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *J. Multivar. Anal.*, vol. 88, no. 2, pp. 365–411, 2004.

Appendix A. Simulation and analysis code

The listing below (motor_fusion.py) generates the data sets, extracts features, trains all models and writes every number reported in Section 4 to results.json. The plotting script used for Figs. 1–7 (figures.py) is supplied as supplementary material.

```
"""
Sensor fusion for predictive maintenance of induction motors.
Synthetic multi-sensor data generation, feature extraction, fault diagnosis
with single-sensor and fused models, robustness tests, and degradation
prognostics.

Requirements: numpy, scipy, scikit-learn, matplotlib
Run: python motor_fusion.py   (writes results.json and figures/*.png)
"""
import json
import warnings
import os
import numpy as np
from scipy import signal, stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.covariance import LedoitWolf

warnings.filterwarnings("ignore", category=FutureWarning)

# ---------------------------------------------------------------
constants
FS = 10_000            # sampling rate of vibration and current channels
(Hz)
T_WIN = 2.0            # window length (s)
N = int(FS * T_WIN)
F_SUP = 50.0           # supply frequency (Hz)
POLES = 4
N_BALLS, D_BALL, D_PITCH = 9, 7.94, 39.04          # 6205-type bearing (mm)
BPFO_K = N_BALLS / 2 * (1 - D_BALL / D_PITCH)       # BPFO / shaft
frequency
F_RES = 3000.0         # structural resonance excited by bearing impacts
(Hz)
CLASSES = ["Healthy", "Bearing", "Rotor bar", "Stator"]
T_AXIS = np.arange(N) / FS
HANN = np.hanning(N)
FREQS = np.fft.rfftfreq(N, 1 / FS)
BP_SOS = signal.butter(4, [2000, 4000], btype="band", fs=FS, output="sos")

OUT = os.path.dirname(os.path.abspath(__file__))
FIG = os.path.join(OUT, "figures")
os.makedirs(FIG, exist_ok=True)
```

```
# ------------------------------------------------------------ signal model
def simulate_window(seed, cls, sev, load, vib_noise_extra=0.0):
    """Return raw vibration, current, speed reading, winding and ambient
temp."""
    rng = np.random.default_rng(seed)
    t = T_AXIS
    slip = (0.006 + 0.038 * load) * rng.uniform(0.92, 1.08)
    fr = F_SUP * (1 - slip) / (POLES / 2)
    bpfo = BPFO_K * fr
    ph = rng.uniform(0, 2 * np.pi, 8)

    # ---- vibration (m/s^2)
    vib = (0.45 * (0.7 + 0.3 * load) * rng.uniform(0.8, 1.2) * np.sin(2 *
np.pi * fr * t + ph[0])
           + 0.18 * rng.uniform(0.7, 1.3) * np.sin(2 * np.pi * 2 * fr * t +
ph[1])
           + 0.12 * (0.5 + load) * rng.uniform(0.7, 1.3) * np.sin(2 * np.pi
* 2 * F_SUP * t + ph[2]))
    vib += rng.normal(0, 0.30, N)

    # ---- stator current (A)
    i0 = 10.0 * (0.35 + 0.65 * load)
    h3 = 0.004 * rng.uniform(0.3, 2.6)       # supply unbalance /
saturation
    asym = 0.0012 * rng.uniform(0.4, 1.6)                  # inherent rotor
asymmetry
    cur = i0 * (np.sin(2 * np.pi * F_SUP * t)
                + h3 * np.sin(2 * np.pi * 3 * F_SUP * t + ph[3])
                + 0.020 * np.sin(2 * np.pi * 5 * F_SUP * t + ph[4])
                + 0.012 * np.sin(2 * np.pi * 7 * F_SUP * t + ph[5])
                + asym * np.sin(2 * np.pi * (1 - 2 * slip) * F_SUP * t +
ph[6])
                + asym * 0.8 * np.sin(2 * np.pi * (1 + 2 * slip) * F_SUP *
t + ph[7]))
    cur += rng.normal(0, 0.02 * i0, N)

    # ---- temperatures (deg C)
    t_amb = rng.uniform(22, 40)
    rise = 48 * load ** 2 + 6
    if cls == 1:      # localized outer-race bearing defect
        period = 1 / bpfo
        n_imp = int(T_WIN / period) + 2
        times = np.cumsum(period * (1 + rng.normal(0, 0.012, n_imp))) +
rng.uniform(0, period)
        imp = np.zeros(N)
        idx = (times * FS).astype(int)
        idx = idx[idx < N]
        imp[idx] = 2.2 * sev * rng.uniform(0.75, 1.25, idx.size)
        tk = np.arange(int(0.006 * FS)) / FS
        ir = np.exp(-tk / 0.0008) * np.sin(2 * np.pi * F_RES * tk)
        vib += signal.fftconvolve(imp, ir)[:N]
        for k, a in ((1, 1.0), (-1, 1.0)):
            cur += i0 * 0.0007 * sev * a * np.sin(2 * np.pi * (F_SUP + k *
bpfo) * t + rng.uniform(0, 6.3))
        rise += 4 * sev
    elif cls == 2:    # broken rotor bar(s)
        m = 0.6 * sev
```

```
        vib += 0.45 * m * np.sin(2 * np.pi * fr * t) * np.cos(2 * np.pi * 2
* slip * F_SUP * t)
        a = 0.025 * sev
        cur += i0 * a * np.sin(2 * np.pi * (1 - 2 * slip) * F_SUP * t +
rng.uniform(0, 6.3))
        cur += i0 * 0.8 * a * np.sin(2 * np.pi * (1 + 2 * slip) * F_SUP * t
+ rng.uniform(0, 6.3))
        rise += 3 * sev
    elif cls == 3:    # stator inter-turn short circuit
        cur += i0 * 0.013 * sev * np.sin(2 * np.pi * 3 * F_SUP * t +
rng.uniform(0, 6.3))
        vib += 0.30 * sev * np.sin(2 * np.pi * 2 * F_SUP * t +
rng.uniform(0, 6.3))
        rise += 14 * sev

    if vib_noise_extra > 0:
        vib += np.random.default_rng(seed + 10**7).normal(0,
vib_noise_extra, N)
    t_wind = t_amb + rise + rng.normal(0, 3.0)
    t_amb_meas = t_amb + rng.normal(0, 0.5)
    speed = fr * (1 + rng.normal(0, 0.0015))
    return vib, cur, speed, t_wind, t_amb_meas


# ------------------------------------------------------- feature
extraction
def amp_spectrum(x):
    return 2 * np.abs(np.fft.rfft(x * HANN)) / HANN.sum()


def band_peak(spec, f0, half_width):
    m = (FREQS >= f0 - half_width) & (FREQS <= f0 + half_width)
    return spec[m].max()


VIB_FEATS = ["v_rms", "v_kurt", "v_crest", "v_skew", "v_A_fr", "v_A_2fs",
             "v_env_bpfo", "v_env_2bpfo", "v_hf_ratio"]
CUR_FEATS = ["c_rms", "c_lsb_db", "c_usb_db", "c_h3", "c_h5", "c_bpfo_sb",
"c_resid"]
TMP_FEATS = ["t_rise", "t_rise_norm"]


def extract(vib, cur, speed, t_wind, t_amb):
    fr = speed
    slip = 1 - fr * (POLES / 2) / F_SUP
    bpfo = BPFO_K * fr
    # vibration
    rms = np.sqrt(np.mean(vib ** 2))
    vs = amp_spectrum(vib)
    env = np.abs(signal.hilbert(signal.sosfiltfilt(BP_SOS, vib)))
    es = amp_spectrum(env - env.mean())
    floor = np.median(es[(FREQS > 10) & (FREQS < 500)])
    total = np.sum(vs ** 2)
    hf = np.sum(vs[(FREQS > 2000) & (FREQS < 4000)] ** 2)
    v = [rms, stats.kurtosis(vib), np.max(np.abs(vib)) / rms,
stats.skew(vib),
         band_peak(vs, fr, 0.6), band_peak(vs, 2 * F_SUP, 0.6),
         band_peak(es, bpfo, 0.02 * bpfo) / floor, band_peak(es, 2 * bpfo,
0.02 * bpfo) / floor,
         hf / total]
```

```
    # current
    cs = amp_spectrum(cur)
    a1 = band_peak(cs, F_SUP, 0.6)
    lsb = band_peak(cs, (1 - 2 * slip) * F_SUP, 0.5) / a1
    usb = band_peak(cs, (1 + 2 * slip) * F_SUP, 0.5) / a1
    h3 = band_peak(cs, 3 * F_SUP, 0.6) / a1
    h5 = band_peak(cs, 5 * F_SUP, 0.6) / a1
    sb = (band_peak(cs, abs(F_SUP - bpfo), 0.015 * bpfo) + band_peak(cs,
F_SUP + bpfo, 0.015 * bpfo)) / (2 * a1)
    resid = cur - a1 * np.sin(2 * np.pi * F_SUP * T_AXIS +
np.angle(np.fft.rfft(cur * HANN)[np.argmin(abs(FREQS - F_SUP))]) + np.pi /
2)
    c = [np.sqrt(np.mean(cur ** 2)), 20 * np.log10(lsb), 20 *
np.log10(usb), h3, h5, sb,
         np.sqrt(np.mean(resid ** 2)) / a1]
    # temperature (current RMS used as load proxy)
    load_proxy = (c[0] * np.sqrt(2) / 10.0 - 0.35) / 0.65
    trise = t_wind - t_amb
    tm = [trise, trise / (48 * max(load_proxy, 0.2) ** 2 + 6)]
    return v + c + tm


def build_dataset(n_per_class=300, seed0=0, vib_noise_extra=0.0,
seeds=None, meta=None):
    rows, y, sevs, loads, seed_list = [], [], [], [], []
    rng = np.random.default_rng(seed0)
    if seeds is None:
        meta = []
        for cls in range(4):
            for _ in range(n_per_class):
                sev = 0.0 if cls == 0 else rng.uniform(0.12, 1.0)
                meta.append((int(rng.integers(1, 2**31)), cls, sev,
rng.uniform(0.25, 1.0)))
    for s, cls, sev, load in meta:
        rows.append(extract(*simulate_window(s, cls, sev, load,
vib_noise_extra)))
        y.append(cls); sevs.append(sev); loads.append(load)
    return np.array(rows), np.array(y), np.array(sevs), np.array(loads),
meta


# ---------------------------------------------------------- classifiers
def make_model(kind):
    if kind == "RF":
        return RandomForestClassifier(n_estimators=300, random_state=0,
n_jobs=-1)
    return make_pipeline(StandardScaler(), SVC(C=10, gamma="scale",
probability=True, random_state=0))


IDX = {"V": list(range(9)), "C": list(range(9, 16)), "T": list(range(16,
18))}


def decision_fusion(Xtr, ytr, Xte, kind, sensors=("V", "C", "T"),
available=None):
    """Weighted soft voting and product rule over per-sensor classifiers.
    Weights = inner cross-validated accuracy of each sensor on training
data."""
    available = available or sensors
```

```
    probs, weights = [], []
    for s in sensors:
        if s not in available:
            continue
        m = make_model(kind)
        inner = cross_val_predict(make_model(kind), Xtr[:, IDX[s]], ytr,
                                  cv=StratifiedKFold(3, shuffle=True,
random_state=1))
        w = accuracy_score(ytr, inner)
        m.fit(Xtr[:, IDX[s]], ytr)
        probs.append(m.predict_proba(Xte[:, IDX[s]])); weights.append(w)
    P = np.array(probs); W = np.array(weights)
    weighted = np.tensordot(W / W.sum(), P, axes=1)
    product = np.exp(np.sum(np.log(np.clip(P, 1e-4, 1)), axis=0))
    return weighted.argmax(1), product.argmax(1)


CONFIGS = {"Vibration only": ["V"], "Current only": ["C"], "Temperature
only": ["T"],
           "Vib + Current (FLF)": ["V", "C"], "Vib + Cur + Temp (FLF)":
["V", "C", "T"]}


def cv_experiment(X, y, sev, n_rep=3):
    res = {}
    preds_store = {}
    for kind in ("RF", "SVM"):
        for name in list(CONFIGS) + ["Weighted voting (DLF)", "Product rule
(DLF)"]:
            res[(kind, name)] = {"acc": [], "f1": []}
        for rep in range(n_rep):
            skf = StratifiedKFold(5, shuffle=True, random_state=rep)
            fold_preds = {k: np.zeros_like(y) for k in list(CONFIGS) +
["Weighted voting (DLF)", "Product rule (DLF)"]}
            for tr, te in skf.split(X, y):
                for name, sens in CONFIGS.items():
                    cols = sum((IDX[s] for s in sens), [])
                    m = make_model(kind).fit(X[tr][:, cols], y[tr])
                    p = m.predict(X[te][:, cols])
                    fold_preds[name][te] = p
                wv, pr = decision_fusion(X[tr], y[tr], X[te], kind)
                fold_preds["Weighted voting (DLF)"][te] = wv
                fold_preds["Product rule (DLF)"][te] = pr
            for name, p in fold_preds.items():
                res[(kind, name)]["acc"].append(accuracy_score(y, p))
                res[(kind, name)]["f1"].append(f1_score(y, p,
average="macro"))
            if rep == 0:
                preds_store[kind] = fold_preds
    summary = {f"{k}|{n}": {"acc_mean": float(np.mean(v["acc"])),
"acc_std": float(np.std(v["acc"])),
                            "f1_mean": float(np.mean(v["f1"])), "f1_std":
float(np.std(v["f1"]))}
               for (k, n), v in res.items()}
    return summary, preds_store


# ------------------------------------------------------------ prognostics
HI_V = ["v_rms", "v_kurt", "v_crest", "v_env_bpfo", "v_env_2bpfo",
"v_hf_ratio"]
```

```
HI_F = HI_V + ["c_bpfo_sb", "t_rise_norm"]
ALL = VIB_FEATS + CUR_FEATS + TMP_FEATS


def run_to_failure(n_runs=12, seed=7, vib_noise_extra=0.0):
    rng = np.random.default_rng(seed)
    runs = []
    for r in range(n_runs):
        life = int(rng.integers(80, 141))
        b = np.log(1.0 / 0.03) / life
        feats = []
        for h in range(life + 1):
            sev = min(0.03 * np.exp(b * h) * rng.lognormal(0, 0.12), 1.2)
            load = rng.uniform(0.4, 1.0)
            s = int(rng.integers(1, 2**31))
            feats.append(extract(*simulate_window(s, 1, sev, load,
vib_noise_extra)))
        runs.append({"life": life, "X": np.array(feats)})
    return runs


def mahalanobis_hi(Xh, X, cols):
    sc = StandardScaler().fit(Xh[:, cols])
    lw = LedoitWolf().fit(sc.transform(Xh[:, cols]))
    Z = sc.transform(X[:, cols]) - lw.location_
    return np.sqrt(np.einsum("ij,jk,ik->i", Z, lw.precision_, Z))


def smooth(x, w=5):
    return np.convolve(np.pad(x, (w - 1, 0), mode="edge"), np.ones(w) / w,
mode="valid")


def hi_metrics(his):
    mono = np.mean([abs(np.sum(np.diff(h) > 0) - np.sum(np.diff(h) < 0)) /
(len(h) - 1) for h in his])
    trend = np.mean([abs(stats.spearmanr(np.arange(len(h)), h)[0]) for h in
his])
    finals = np.array([h[-1] for h in his]); starts = np.array([h[0] for h
in his])
    prog = np.exp(-np.std(finals) / np.mean(np.abs(finals - starts)))
    return float(mono), float(trend), float(prog)


def rul_predict(hi, tp, thr, win=30):
    t = np.arange(max(0, tp - win), tp + 1)
    yv = np.log(np.maximum(hi[t], 1e-3))
    b1, b0 = np.polyfit(t, yv, 1)
    if b1 <= 1e-6:
        return 300.0
    t_fail = (np.log(thr) - b0) / b1
    return float(np.clip(t_fail - tp, 0, 300))


# ---------------------------------------------------------------- main
if __name__ == "__main__":
    out = {}
    print("Building diagnosis dataset ...")
    X, y, sev, load, meta = build_dataset(300, seed0=2026)
```

```
    np.savez(os.path.join(OUT, "dataset.npz"), X=X, y=y, sev=sev,
load=load)
    out["n_samples"] = int(len(y)); out["n_features"] = int(X.shape[1])
    out["bpfo_factor"] = float(BPFO_K)

    print("Cross-validated comparison ...")
    summary, preds = cv_experiment(X, y, sev)
    out["cv"] = summary
    for k, v in summary.items():
        print(f"{k:40s} acc {v['acc_mean']*100:6.2f} ±
{v['acc_std']*100:4.2f}  F1 {v['f1_mean']*100:6.2f}")

    # confusion matrices and accuracy by severity (RF, repetition 0)
    rf = preds["RF"]
    out["cm_vib"] = confusion_matrix(y, rf["Vibration only"]).tolist()
    out["cm_cur"] = confusion_matrix(y, rf["Current only"]).tolist()
    out["cm_fused"] = confusion_matrix(y, rf["Vib + Cur + Temp
(FLF)"]).tolist()
    bins = [(0.12, 0.40, "Incipient"), (0.40, 0.70, "Moderate"), (0.70,
1.01, "Severe")]
    sev_tab = {}
    for lo, hi, lab in bins:
        m = (y > 0) & (sev >= lo) & (sev < hi)
        sev_tab[lab] = {n: float(np.mean(rf[n][m] == y[m])) for n in
                        ["Vibration only", "Current only", "Temperature
only", "Vib + Cur + Temp (FLF)", "Weighted voting (DLF)"]}
        sev_tab[lab]["n"] = int(m.sum())
    out["severity"] = sev_tab
    # feature importance of fused RF
    rf_full = make_model("RF").fit(X, y)
    imp = sorted(zip(ALL, rf_full.feature_importances_), key=lambda z: -
z[1])
    out["importance"] = [(a, float(b)) for a, b in imp]

    print("Noise robustness ...")
    rng = np.random.default_rng(5)
    perm = rng.permutation(len(y)); ntr = int(0.7 * len(y))
    tr, te = perm[:ntr], perm[ntr:]
    meta_te = [meta[i] for i in te]
    models = {n: make_model("RF").fit(X[tr][:, sum((IDX[s] for s in sens),
[])], y[tr]) for n, sens in CONFIGS.items()}
    noise_levels = [0.0, 0.3, 0.6, 1.0, 1.5, 2.0]
    noise_res = {"levels": noise_levels, "Vibration only": [], "Vib + Cur +
Temp (FLF)": [], "Weighted voting (DLF)": [], "snr_db": []}
    for nl in noise_levels:
        Xn, yn, *_ = build_dataset(seeds=True, meta=meta_te,
vib_noise_extra=nl)
        for n in ("Vibration only", "Vib + Cur + Temp (FLF)"):
            cols = sum((IDX[s] for s in CONFIGS[n]), [])
            noise_res[n].append(float(accuracy_score(yn,
models[n].predict(Xn[:, cols]))))
        wv, _ = decision_fusion(X[tr], y[tr], Xn, "RF")
        noise_res["Weighted voting (DLF)"].append(float(accuracy_score(yn,
wv)))
        sig_p = np.mean(X[y == 0][:, 0] ** 2)
        noise_res["snr_db"].append(float(10 * np.log10(sig_p / nl**2)) if
nl > 0 else None)
    out["noise"] = noise_res
    print(noise_res)
```

```
    print("Sensor dropout ...")
    drop = {}
    mu = X[tr].mean(0)
    full_cols = sum((IDX[s] for s in ("V", "C", "T")), [])
    for lost in ("V", "C", "T"):
        Xd = X[te].copy(); Xd[:, IDX[lost]] = mu[IDX[lost]]
        acc_flf = accuracy_score(y[te], models["Vib + Cur + Temp
(FLF)"].predict(Xd[:, full_cols]))
        wv, _ = decision_fusion(X[tr], y[tr], X[te], "RF", available=[s for
s in "VCT" if s != lost])
        drop[lost] = {"FLF_mean_imputed": float(acc_flf), "DLF_reweighted":
float(accuracy_score(y[te], wv))}
    wv, _ = decision_fusion(X[tr], y[tr], X[te], "RF")
    drop["none"] = {"FLF_mean_imputed": float(accuracy_score(y[te],
models["Vib + Cur + Temp (FLF)"].predict(X[te][:, full_cols]))),
                    "DLF_reweighted": float(accuracy_score(y[te], wv))}
    out["dropout"] = drop
    print(drop)

    print("Run-to-failure prognostics ...")
    cv_ = [ALL.index(c) for c in HI_V]; cf = [ALL.index(c) for c in HI_F]
    healthy_meta = [mm for mm in meta if mm[1] == 0]
    out["prog"] = {}
    for scen, nl in (("clean", 0.0), ("noisy_vibration", 1.0)):
        runs = run_to_failure(vib_noise_extra=nl)
        Xh = X[y == 0] if nl == 0 else build_dataset(seeds=True,
meta=healthy_meta, vib_noise_extra=nl)[0]
        his = {"Kurtosis": [], "Envelope BPFO": [], "Vibration HI (MD)":
[], "Fused HI (MD)": []}
        for r in runs:
            his["Kurtosis"].append(smooth(r["X"][:, ALL.index("v_kurt")]))
            his["Envelope BPFO"].append(smooth(r["X"][:,
ALL.index("v_env_bpfo")]))
            his["Vibration HI (MD)"].append(smooth(mahalanobis_hi(Xh,
r["X"], cv_)))
            his["Fused HI (MD)"].append(smooth(mahalanobis_hi(Xh, r["X"],
cf)))
        res = {"hi_metrics": {k: dict(zip(["monotonicity", "trendability",
"prognosability"], hi_metrics(v)))
                              for k, v in his.items()}, "rul": {},
"rul_trace": {}}
        for name in ("Vibration HI (MD)", "Fused HI (MD)"):
            errs = {0.7: [], 0.8: [], 0.9: []}
            for i, r in enumerate(runs):
                thr = np.median([his[name][j][-1] for j in range(len(runs))
if j != i])
                for frac in errs:
                    tp = int(frac * r["life"])
                    errs[frac].append(rul_predict(his[name][i], tp, thr) -
(r["life"] - tp))
                if i == 0:
                    tps = np.arange(int(0.6 * r["life"]), r["life"] + 1)
                    res["rul_trace"][name] = {"t": tps.tolist(),
                                              "pred":
[rul_predict(his[name][i], int(tp), thr) for tp in tps],
                                              "true": (r["life"] -
tps).tolist()}
            res["rul"][name] = {str(k): {"rmse":
float(np.sqrt(np.mean(np.square(v)))),
```

```
                                        "mae": float(np.mean(np.abs(v))),
"bias": float(np.mean(v))}
                                for k, v in errs.items()}
        res["runs_life"] = [r["life"] for r in runs]
        out["prog"][scen] = res
        np.save(os.path.join(OUT, f"hi_curves_{scen}.npy"), np.array(his,
dtype=object), allow_pickle=True)
        print(scen, json.dumps(res["hi_metrics"], indent=0),
json.dumps(res["rul"], indent=0))
    with open(os.path.join(OUT, "results.json"), "w") as f:
        json.dump(out, f, indent=1)
    print("done")
```