

BAB II

TINJAUAN PUSTAKA

2.1. Penelitian Terkait

Kajian terhadap sejumlah penelitian terdahulu yang berkaitan dengan sistem pemantauan rantai dingin, analisis sinyal getaran, serta penerapan *TinyML* pada perangkat berbasis IoT menunjukkan adanya beragam pendekatan metodologis dan teknologi yang telah dikembangkan. Berbagai temuan tersebut menjadi landasan konseptual dan teknis dalam perancangan sistem pada penelitian ini. Untuk memberikan gambaran yang lebih terstruktur, perbandingan fokus penelitian, metode yang digunakan, serta kontribusi utama masing-masing studi dirangkum secara sistematis dalam Tabel 2.1.

Tabel 2.1 Penelitian Terkait

No.	Judul, Penulis & Tahun	Dataset / Sumber Data	Metode & Instrumen	Hasil Evaluasi	Research Gap
1	<i>IoT Device for Detecting Abnormal Vibrations in Motors Using TinyML</i> , Arcinie	Data rekaman sensor MPU6050 pada kipas motor (<i>fan mockup</i>) di laboratorium, dikumpulkan via	Sensor MPU6050, mikrokontroler XIAO ESP32S3, FFT (<i>window</i> 16 sampel, <i>overlap</i> 50%), DNN (2 <i>hidden layer</i> : 20 & 10 neuron), K-Means	<i>Accuracy</i> : 96.5%, <i>Precision</i> : 96.19%, <i>Recall</i> : 97.11%, <i>F1-Score</i> : 96.64%, <i>Inference time</i> : 25 ms, <i>Latency end-to-</i>	Seluruh alur (<i>preProcessing</i> , ekstraksi fitur, <i>deployment</i>) bergantung pada platform <i>Edge Impulse</i> sehingga sulit direplikasi tanpa platform tersebut. Target klasifikasi berupa kondisi kecepatan motor

No.	Judul, Penulis & Tahun	Dataset / Sumber Data	Metode & Instrumen	Hasil Evaluasi	Research Gap
	gas et al., 2025 [38]	platform Edge Impulse.	Clustering (32 cluster), Quantization INT8, pruning via matrix factorization, deployment via Edge Impulse.	end: 300 ms	(Stopped, Average Speed, High Speed) bukan jenis kerusakan mekanis, dan pengujian terbatas pada kondisi laboratorium.
2	<i>An Adaptable and Unsupervised TinyML Anomaly Detection System for Extreme Industrial Environments</i> , Antoni et al., 2023 [39]	Data rekaman sensor IMU ICM-20948 (akselerometer + giroskop 9-DoF) dan sensor suhu, dipasang pada pompa air bawah laut (submersible pump) di instalasi pengolahan air limbah.	ESP32 (MicroPython), sensor IMU ICM-20948, komunikasi PLC (ST7540), algoritma Isolation Forest (unsupervised, 10–50 trees), TinyML pipeline, Blockchain Ethereum untuk pencatatan anomali.	Inference time: 6,9–20,33 ms (50 trees), Training time: 1,2–6,4 s, Memory footprint: 70–84 KB (Inference mode, 50 trees), tidak melaporkan metrik akurasi klasifikasi.	Menggunakan Isolation Forest (unsupervised) yang hanya membedakan Normal vs. Anomali secara umum tanpa mengidentifikasi jenis kerusakan spesifik. Membutuhkan RAM hingga 84 KB dan waktu pelatihan on-device hingga 6,4 detik. Tidak menerapkan transformasi FFT untuk ekstraksi fitur spektral.
3	<i>Vibration-Based</i>	Dataset publik MAFAU	Ekstraksi 11 fitur statistik	DNN+FFT: Accuracy	Implementasi menggunakan sistem akuisisi

No.	Judul, Penulis & Tahun	Dataset / Sumber Data	Metode & Instrumen	Hasil Evaluasi	Research Gap
	<i>Anomaly Detection for Induction Motors Using Machine Learning</i> , Ullah et al., 2025 [40]	LDA, dikumpulkan menggunakan sensor akselerometer industri dan modul akuisisi NI-9234 pada <i>machine fault simulator Spectra Quest</i> .	domain waktu, FFT-based autocorrelation untuk DNN, tiga algoritma: DNN, SVM (RBF kernel), KNN; oversampling dan optimasi hyperparameter.	99.7%, Precision 99.9%, Recall 99.8%, SVM: Accuracy 95.4%, KNN: Accuracy 92.8%, AUC-ROC SVM: 0.98, KNN: 0.96.	data laboratorium berbiaya tinggi (NI-9234, sensor industri IMI) dan tidak diujicobakan pada perangkat edge berdaya terbatas untuk inferensi <i>real-time</i> tanpa ketergantungan <i>cloud</i> .
4	<i>LightWeight Signal Processing and Edge AI for Real-time Anomaly Detection in IoT Sensor Networks</i> ,	Dataset sintesis yang dibuat sendiri mencakup tiga modalitas: sinyal getaran (500 Hz), akustik (8 kHz), dan lingkungan (1 Hz), dengan injeksi	Ekstraksi fitur FFT dan Wavelet Transform (DWT), tiga model ringan: Shallow MLP (SNN, 1 hidden layer 32 neuron), Decision Tree, dan Quantized TinyML INT8; dijalankan pada	SNN: F1 $\approx 0,94$, Accuracy 0,95, Latency 1,40 ms, Memory 220 KB; TinyML INT8: F1 $\approx 0,92$, Latency 0,90 ms, Memory 60 KB, Energy 0,22 mJ/sampel (hemat	Evaluasi hanya menggunakan dataset sintesis dan tidak divalidasi pada skenario industri nyata dengan sensor fisik terintegrasi, sehingga hasilnya belum tentu merepresentasikan kondisi operasional sebenarnya.

No.	Judul, Penulis & Tahun	Dataset / Sumber Data	Metode & Instrumen	Hasil Evaluasi	Research Gap
	Reis, 2025 [41]	anomali berupa <i>point spike</i> dan <i>contextual drift</i> .	<i>Raspberry Pi 4</i> dan <i>STM32 Nucleo</i> .	60% vs SNN).	
5	<i>An Edge-Deployable TinyML Approach Enhanced by Transfer Learning for Efficient Bearing Fault Diagnosis</i> , Gao et al., 2025 [42]	<i>Dataset proprietary Bearing</i> (4 kondisi: normal, inner race, ball, outer race fault; ~3000 sampel/kondisi) menggunakan ADXL345 + ESP8684 ; <i>dataset publik SEU gearbox and Bearing</i> untuk pre-Training .	Res-CNN-LSTM (DscConv + residual block + LSTM 16 unit), transfer learning, int8 <i>Quantization</i> + <i>structured pruning</i> 30%, <i>TensorFlow Lite Micro</i> , deployment via <i>Arduino IDE</i> , <i>cloud</i> OneNet via MQTT.	<i>Accuracy</i> : 88.28%, <i>Recall</i> : 88.28%, <i>F1-Score</i> : 88.28%, <i>Inference time on MCU</i> : 44.9 ms, <i>RAM usage</i> : 74.2 KB, <i>Flash usage</i> : 57.6 KB, <i>Energy per Inference</i> : 17.7 mJ, <i>Model size</i> (int8): 73 KB.	<i>Dataset</i> terbatas (~192 sampel/kondisi untuk <i>Training</i>) pada satu jenis mesin statis. Model belum diuji pada mesin bergerak atau kondisi beban dinamis. Sistem <i>monitoring</i> bergantung konektivitas Wi-Fi. Adaptabilitas transfer <i>learning</i> terhadap jenis mesin berbeda belum terbukti.

2.2. Landasan Teori

2.2.1. *Internet of Things (IoT)*

Konsep IoT merupakan jaringan global yang menghubungkan perangkat komputasi dan sensor sehingga dapat saling bertukar data melalui berbagai protokol komunikasi [43]. Dalam sistem IoT, perangkat biasanya dilengkapi dengan sensor untuk mengumpulkan data dari lingkungan sekitar, kemudian data tersebut dikirimkan melalui jaringan komunikasi menuju sistem pemrosesan atau platform *monitoring*. Dengan adanya teknologi ini, proses pemantauan suatu sistem dapat dilakukan secara lebih efisien karena data dapat diperoleh secara *real-time* tanpa harus melakukan pengamatan secara manual. Penerapan IoT banyak digunakan dalam berbagai bidang seperti sistem pemantauan lingkungan, sistem industri, serta *monitoring* berbasis sensor yang memanfaatkan mikrokontroler seperti ESP32 untuk mengirimkan data hasil pengukuran secara otomatis ke sistem *monitoring* [44], [45].

2.2.2. *Edge computing*

Konsep *edge computing* merupakan paradigma komputasi terdistribusi yang menempatkan pemrosesan data dan kapasitas penyimpanan lebih dekat ke sumber data, yaitu di tepi jaringan (*network edge*), alih-alih mengandalkan pusat data terpusat yang jauh [46]. Paradigma ini muncul sebagai respons atas keterbatasan arsitektur *cloud computing* konvensional. Pada arsitektur tersebut,

seluruh data IoT harus dikirim ke *cloud* untuk dianalisis, sehingga menimbulkan beban jaringan yang besar dan latensi yang tinggi bagi aplikasi yang membutuhkan respons waktu nyata [47]. Secara arsitektural, *edge computing* menempatkan sumber daya komputasi dan penyimpanan di antara perangkat *Terminal* dan server *cloud*, sehingga pemrosesan data dapat dilakukan secara lokal tanpa bergantung penuh pada konektivitas internet, sekaligus meningkatkan kualitas layanan untuk aplikasi yang sensitif terhadap latensi [48]. Keunggulan ini menjadikan *edge computing* sebagai teknologi kunci di berbagai sektor, mulai dari pemantauan industri hingga sistem *real-time*. Selain itu, karena data diproses secara lokal, risiko kebocoran informasi dapat ditekan dan kepatuhan terhadap regulasi seperti *General Data Protection Regulation* (GDPR) lebih mudah dipenuhi [49].

2.2.3. *Tiny Machine Learning (TinyML)*

Konsep *Tiny Machine Learning (TinyML)* merupakan cabang *machine learning* yang berfokus pada penerapan model inferensi secara langsung pada perangkat tertanam berdaya sangat rendah dan memori sangat terbatas, seperti *Microcontroller Unit* (MCU), sensor cerdas, dan perangkat IoT. Perangkat yang dimaksud umumnya beroperasi dengan konsumsi daya di bawah 1 miliwatt, memori *Flash* berkisar 32 KB hingga 2 MB, dan RAM antara 2 KB hingga 512 KB [50]. Paradigma ini muncul sebagai respons atas keterbatasan

arsitektur *cloud-centric* yang menghadapi masalah latensi tinggi, konsumsi *bandwidth* besar, risiko privasi akibat transmisi data sensitif melalui jaringan, serta ketergantungan pada konektivitas internet yang tidak selalu tersedia [51]. Dengan pendekatan ini, pengambilan keputusan berbasis kecerdasan buatan dapat dilakukan secara lokal tanpa bergantung pada komputasi *cloud*, sehingga memperluas penerapan *machine learning* ke berbagai skenario dunia nyata yang sebelumnya tidak terjangkau [52].

Untuk dapat beroperasi pada perangkat sekecil itu, *TinyML* memanfaatkan teknik optimasi model seperti *Quantization* yang mengonversi bobot model dari *floating-point* 32-bit menjadi bilangan bulat 8-bit, serta *pruning* yang menghilangkan parameter jaringan saraf yang memiliki kontribusi rendah, sehingga menghasilkan model berukuran kecil namun tetap memiliki performa inferensi yang memadai untuk dijalankan langsung pada MCU [53]. Teknologi ini telah diterapkan secara luas di berbagai domain seperti kesehatan, pertanian cerdas, keamanan jaringan IoT, *smart city*, dan UAV, dengan dukungan *framework* seperti *TensorFlow Lite Micro* dan *Edge Impulse*. Meskipun demikian, *TinyML* masih menghadapi tantangan berupa kompromi antara akurasi dan efisiensi sumber daya, heterogenitas perangkat keras, serta isu keamanan pada perangkat *edge* yang menjadi fokus penelitian aktif saat ini [54].

2.2.4. Pra-pemrosesan Data

Pra-pemrosesan data adalah tahap awal yang penting dalam *machine learning* dan analisis data, bertujuan mengubah data mentah menjadi bentuk yang bersih dan siap digunakan model. Salah satu teknik yang diterapkan adalah *windowing*, yaitu membagi sinyal menjadi segmen-segmen kecil sebelum transformasi FFT dilakukan, guna mengurangi efek *spectral leakage* dan menghasilkan fitur yang lebih representatif [55]. Tahapan ini juga mencakup teknik *filtering* untuk menghilangkan gangguan seperti *baseline wander*, *power line interference*, dan *muscle artifacts* dari sinyal mentah, sehingga mempermudah ekstraksi fitur relevan dan mendukung performa model *deep learning* yang lebih akurat [56]. Normalisasi bertujuan untuk menskalakan data menggunakan metode seperti *z-Score* dan *Min-Max* agar setiap fitur memiliki kontribusi yang sebanding, dan direkomendasikan diterapkan secara adaptif berbasis *sliding window* di sisi *edge* sehingga model ML dapat belajar lebih efisien dari data yang bersih dan bebas *outlier* [57].

2.2.5. Fast Fourier Transform (FFT)

Algoritma FFT adalah algoritma komputasi efisien untuk menghitung *Discrete Fourier Transform* (DFT) yang mengubah sinyal dari domain waktu ke domain frekuensi dengan kompleksitas komputasi $O(N \log N)$, jauh lebih rendah dibandingkan perhitungan DFT langsung yang membutuhkan $O(N^2)$, sehingga menjadikannya

metode fundamental dalam pengolahan sinyal digital pada berbagai bidang seperti biomedis, telekomunikasi, dan sistem IoT [58]. Secara matematis, DFT untuk sinyal diskrit $x[n]$ sepanjang N sampel didefinisikan sebagai:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\left(\frac{2\pi}{N}\right)nk}, k = 0, 1, \dots, N-1$$

di mana $X[k]$ adalah nilai spektrum pada frekuensi ke- k , j merupakan unit imajiner, serta n dan k masing-masing adalah indeks waktu dan frekuensi. Dalam konteks data *time-series* IoT dan sensor *networks*, FFT digunakan sebagai teknik pra-pemrosesan inovatif untuk mengubah data dari domain waktu ke domain frekuensi sehingga memperkaya representasi data dengan fitur-fitur frekuensi tambahan yang meningkatkan akurasi prediksi model *deep learning*, termasuk arsitektur *transformer*, dalam menangani tantangan seperti dependensi jangka panjang, *noise*, dan anomali pada data sensor [59]. Lebih lanjut, efisiensi komputasi FFT juga mendukung implementasinya pada perangkat *edge computing* dan mikrokontroler dengan sumber daya terbatas, di mana teknik *in-place* FFT berbasis *block floating point* (BFP) terbukti mampu mengurangi kebutuhan memori hingga setengahnya sekaligus meningkatkan *signal-to-noise ratio* (SNR) minimal 3 dB dibandingkan representasi *floating point* konvensional, sehingga mempercepat pengambilan keputusan lokal pada sistem IoT yang membutuhkan respons *real-time* [60].

2.2.6. Ekstraksi Fitur

Ekstraksi fitur merupakan tahap penting dalam pengolahan sinyal getaran yang bertujuan untuk memperoleh karakteristik utama dari data mentah hasil transformasi FFT sehingga dapat direpresentasikan dalam bentuk parameter yang lebih ringkas dan informatif sebagai masukan bagi model klasifikasi. Proses ini berperan krusial dalam menentukan kemampuan sistem mengenali pola kondisi mesin, karena kualitas fitur yang diekstrak secara langsung mempengaruhi akurasi deteksi anomali [61]. Dalam domain frekuensi, fitur spektral yang umum digunakan mencakup *Root Mean Square* (RMS) yang merepresentasikan intensitas energi getaran secara keseluruhan, *Peak Amplitude* yang mengidentifikasi komponen frekuensi dominan sebagai penanda khas suatu kondisi mesin, serta *Spectral Energy* yang mencerminkan distribusi energi di seluruh rentang spektrum frekuensi [62]. Ketiga fitur tersebut diekstrak secara independen dari masing-masing sumbu akselerasi (x, y, z) sensor MPU6050 sehingga menghasilkan vektor fitur berdimensi 9 yang merepresentasikan karakteristik getaran mesin secara multidimensional, karena akselerometer tiga sumbu memungkinkan sistem menangkap informasi getaran dari seluruh arah gerak mekanis secara komprehensif sehingga meningkatkan kemampuan deteksi kerusakan dibandingkan sensor sumbu tunggal [63]. Pendekatan ekstraksi fitur spektral berbasis FFT terbukti efektif dalam

membedakan ketiga kondisi mesin. Kondisi *Imbalance* ditandai oleh dominansi amplitudo pada frekuensi putaran ($1 \times \text{RPM}$), sedangkan kondisi *Bearing* menghasilkan komponen frekuensi non-sinkron yang tersebar pada rentang frekuensi. Perbedaan pola ini menjadikan fitur RMS, *Peak Amplitude*, dan *Spectral Energy* memiliki daya diskriminatif yang baik sebagai masukan model MLP [64].

2.2.7. *Multi-Layer Perceptron* (MLP)

Model *Multi-Layer Perceptron* (MLP) merupakan arsitektur jaringan saraf tiruan *feedforward* yang tersusun atas *Input layer*, *hidden layer*, dan *Output layer* yang saling terhubung melalui bobot (*Weights*)[65]. MLP dilatih menggunakan algoritma *backpropagation* untuk memperbarui bobot dan bias jaringan berdasarkan *gradient descent*, sehingga mampu mempelajari pola non-linier dari data secara bertahap [66]. Arsitektur ini telah banyak diterapkan sebagai modul klasifikasi pada sistem diagnosis berbasis sinyal sensor, seperti pada model WDCAE-LKA untuk diagnosis kerusakan *Bearing* tanpa label [67] dan pada arsitektur fusi sinyal suara-getaran berbasis mekanisme atensi untuk diagnosis kerusakan mesin [68]. Kemampuan MLP dalam mengklasifikasikan kondisi operasi mesin berdasarkan fitur hasil ekstraksi sinyal menjadikannya metode yang relevan digunakan dalam penelitian ini untuk mengklasifikasikan kondisi kompresor berdasarkan fitur spektral getaran.

2.2.8. *Dataset* MAFAULDA

Dataset MAFAULDA (*Machinery Fault Database*) merupakan *dataset* publik rekaman sinyal getaran mesin rotasi yang dikembangkan oleh *Universidade Federal do Rio de Janeiro* (UFRJ), Brasil, dan tersedia secara terbuka untuk keperluan penelitian di bidang *condition monitoring* mesin [69]. *Dataset* ini berisi rekaman sinyal akselerasi tiga sumbu yang direkam menggunakan sensor akselerometer pada berbagai kondisi operasional mesin, meliputi kondisi Normal, berbagai tingkat ketidakseimbangan massa (*Imbalance*), dan berbagai jenis kerusakan *Bearing* seperti kerusakan pada *inner race*, *outer race*, dan *cage*, yang direkam pada frekuensi *sampling* tinggi sehingga memungkinkan analisis spektral dalam rentang frekuensi yang luas [70]. *Dataset* MAFAULDA telah terbukti menjadi *benchmark* yang terstandar secara ilmiah dalam penelitian diagnosis kerusakan mesin berbasis *machine learning*, sebagaimana dibuktikan oleh Ribeiro dkk, yang menggunakannya untuk mengembangkan model berbasis kemiripan (*similarity-based*) yang ditingkatkan untuk klasifikasi kegagalan mesin rotasi [71], serta Souza dkk. yang memanfaatkan *dataset* ini dalam pendekatan *deep learning* untuk diagnosis dan klasifikasi kerusakan mesin industri rotasi dengan hasil yang menunjukkan kemampuan generalisasi model yang baik terhadap berbagai kondisi kerusakan [72]. Dalam penelitian ini, *dataset* MAFAULDA digunakan sebagai sumber data

kondisi *Imbalance*, *Bearing*, dan sebagian data kondisi Normal yang dikombinasikan dengan data rekaman langsung dari CV. Mutiara Samudera Indonesia, guna membangun model *TinyML* yang mampu mengenali kondisi operasional kompresor secara akurat.

2.2.9. Optimasi Model (*Pruning* dan *Quantization*)

Optimasi model bertujuan meningkatkan efisiensi model *machine learning* agar dapat dijalankan pada perangkat dengan keterbatasan memori, daya, dan kapasitas pemrosesan, mengingat MCU umumnya hanya memiliki *Flash Memory* 1 MB atau kurang dan RAM di bawah 512 KB [73]. Salah satu teknik yang umum digunakan adalah *pruning*, yaitu menghilangkan bobot atau koneksi jaringan saraf yang berkontribusi kecil sehingga ukuran model mengecil dan komputasi menjadi lebih cepat tanpa menurunkan akurasi secara signifikan [74]. Metode *differentiable pruning* dikembangkan untuk mengoptimalkan ukuran model secara langsung terhadap batasan memori dan latensi mikrokontroler, menghasilkan model lebih kompak dibanding *pruning* konvensional tanpa degradasi akurasi yang berarti [75]. Sejalan dengan hal tersebut, pendekatan *pruning* berbasis *Morphological Neural Network* (MNN) juga telah diterapkan pada perangkat IoT seperti ESP32, menghasilkan peningkatan efisiensi energi dan kecepatan inferensi yang signifikan tanpa kompromi akurasi [76].

Teknik lain yang umum digunakan adalah *Quantization*, yaitu mengurangi presisi representasi bobot dan aktivasi dari *floating-point* 32-bit menjadi 16-bit atau 8-bit integer. Penelitian tersebut membuktikan bahwa *Quantization* 8-bit pada mikrokontroler ARM Cortex-M4 menghasilkan penghematan memori yang signifikan dengan penurunan akurasi minimal pada berbagai *dataset* sensor [77]. Sebagai contoh penerapan lain, *Post-Training Quantization* dan *Quantization-Aware Training* pada perangkat IoT berbasis NVIDIA Jetson menunjukkan bahwa *Quantization* INT8 mampu mencapai pengurangan latensi hingga 90% dibanding model *baseline Float32*, meskipun efektivitasnya bervariasi tergantung arsitektur jaringan yang digunakan [78]. Namun, efisiensi *Quantization* bergantung pada karakteristik *hardware* target, di mana mikrokontroler dengan FPU terintegrasi seperti ESP32-S3 (Xtensa LX7) mampu mengeksekusi operasi *floating-point* 32-bit secara *native*, sehingga *overhead deQuantization* pada *runtime TFLite Micro* dapat mengeliminasi keunggulan latensi INT8, bahkan menghasilkan inferensi lebih lambat dibanding varian *Float32* [79]. Dalam penelitian ini, keempat varian model (V1–V4) dievaluasi dengan kombinasi *pruning* dan *Quantization* yang berbeda untuk menentukan konfigurasi optimal yang menyeimbangkan akurasi, ukuran model, dan latensi inferensi pada ESP32-S3.

2.2.10. *Majority voting*

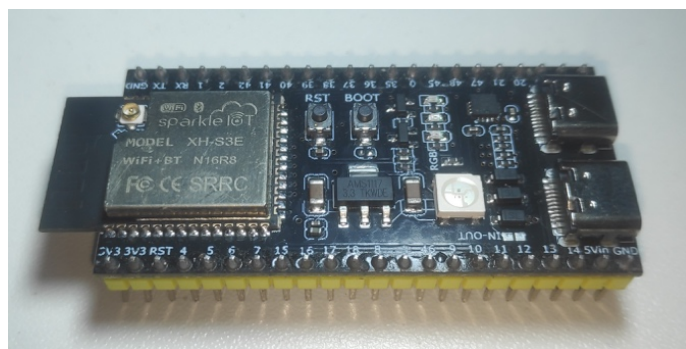
Teknik *majority voting* merupakan teknik pengambilan keputusan kolektif dalam *ensemble learning* yang menentukan label akhir berdasarkan kelas yang paling sering diprediksi dari sejumlah model atau segmen prediksi independen, diformulasikan sebagai $\hat{y} = \text{mode}(y_1, y_2, \dots, y_n)$ di mana kelas dengan jumlah suara terbanyak dipilih sebagai hasil akhir [80]. Dalam konteks pemrosesan sinyal berbasis *window*, teknik ini diterapkan dengan mengumpulkan prediksi dari beberapa *window* sinyal berturut-turut kemudian memilih kelas yang muncul paling sering sebagai label klasifikasi akhir, sehingga efektif meredam pengaruh prediksi yang salah secara individual akibat *noise* sesaat atau variasi *transien* pada sinyal. Sejumlah penelitian membuktikan keunggulan *majority voting* dibandingkan *classifier* tunggal, di antaranya *Weighted majority voting* terbukti mengungguli CNN tunggal maupun *ensemble* tanpa pembobotan terutama pada data sinyal dengan *noise* tinggi dan distribusi kelas tidak seimbang [81], serta pendekatan *ensemble learning* berbasis agregasi prediksi untuk diagnosis multi-kerusakan pada komponen mesin berputar terbukti menghasilkan performa lebih unggul dibandingkan model individual pada kondisi kecepatan dan beban operasional yang bervariasi [82]. Dalam penelitian ini, *majority voting* diterapkan terhadap 7 *window* prediksi berturut-turut dari model MLP, di mana label yang muncul paling sering ditetapkan

sebagai keputusan klasifikasi kondisi kompresor (Normal, *Imbalance*, atau *Bearing*) yang kemudian dikirimkan ke platform *monitoring* melalui protokol MQTT.

2.2.11. Mikrokontroler ESP32-S3

Mikrokontroler ESP32-S3 merupakan mikrokontroler yang dikembangkan oleh *Espressif Systems* dan dirancang untuk aplikasi IoT serta sistem komputasi pada perangkat *edge*. Mikrokontroler ini menggunakan prosesor Xtensa LX7 *dual-core* dengan frekuensi hingga 240 MHz, serta dilengkapi konektivitas Wi-Fi 2.4 GHz dan *Bluetooth Low Energy* (BLE) yang memungkinkan perangkat terhubung dengan jaringan secara nirkabel. Selain kemampuan pemrosesan tersebut, ESP32-S3 juga memiliki memori internal berupa 512 KB SRAM serta dukungan memori eksternal PSRAM hingga 8 MB dan *Flash* hingga 16 MB, yang memungkinkan perangkat menjalankan aplikasi *embedded* yang lebih kompleks serta mendukung pemrosesan data lokal pada sistem IoT [83]. Selain itu, ESP32-S3 menyediakan berbagai antarmuka komunikasi seperti SPI, I2C, UART, ADC, dan GPIO yang memungkinkan integrasi dengan berbagai sensor dan perangkat eksternal, serta didukung instruksi SIMD (*Single Instruction Multiple Data*) pada inti LX7 yang memungkinkan pemrosesan data secara paralel, dan FPU (*Floating Point Unit*) terintegrasi yang memungkinkan eksekusi operasi *floating-point* 32-bit secara natif pada level *hardware*, sehingga

meningkatkan efisiensi komputasi dan efisiensi daya saat menjalankan model *machine learning* [84]. Dalam sistem *monitoring* berbasis IoT, ESP32-S3 berfungsi sebagai unit pengendali utama yang melakukan akuisisi data dari sensor, pemrosesan awal data, serta transmisi data ke sistem pemantauan secara *real-time* melalui jaringan internet [85]. Bentuk fisik perangkat mikrokontroler ESP32-S3 ditampilkan pada Gambar 2.1.

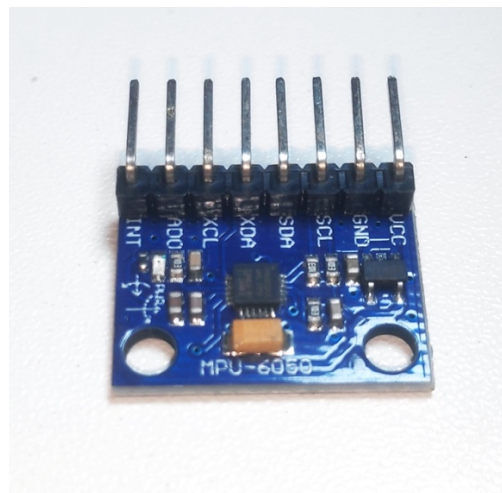


Gambar 2.1 ESP32-S3

2.2.12. Sensor Inersia (MPU6050)

Sensor MPU6050 (atau sering dikenal sebagai GY-521) adalah sebuah sensor pintar serbaguna yang dirancang khusus untuk mendeteksi gerakan dan orientasi suatu objek di ruang tiga dimensi. Di dalam satu cip berukuran kecil ini, terdapat dua indra utama yaitu akselerometer tiga sumbu yang bertugas mengukur kemiringan dan percepatan (pada sumbu x, y, dan z), serta giroskop yang berfungsi untuk memantau gerakan berputar atau rotasi. Sensor ini memiliki resolusi sampling 16-bit sehingga mampu mendeteksi perubahan sudut dan gerakan yang sangat kecil dengan tingkat akurasi yang tinggi. Untuk melengkapi fiturnya, MPU6050 juga dibekali dengan

sensor suhu internal untuk memantau kondisi lingkungan sekitarnya dan menggunakan antarmuka komunikasi I2C agar bisa berkomunikasi dengan mudah ke otak mikrokontroler seperti ESP32-S3 [86]. Dalam konteks pemantauan kondisi mesin, MPU6050 banyak digunakan sebagai sensor getaran berbiaya rendah untuk mendeteksi pola getaran pada mesin berputar, di mana perubahan pola getaran dapat mengindikasikan adanya kerusakan pada komponen mesin [87]. Selain itu, kemampuan MPU6050 dalam melakukan pengukuran inersia secara akurat telah divalidasi melalui berbagai penelitian yang menggunakan filter komplementer maupun filter Kalman untuk meningkatkan presisi pembacaan data akselerasi dan giroskop secara bersamaan [88]. Bentuk fisik sensor MPU6050 ditampilkan pada Gambar 2.2.

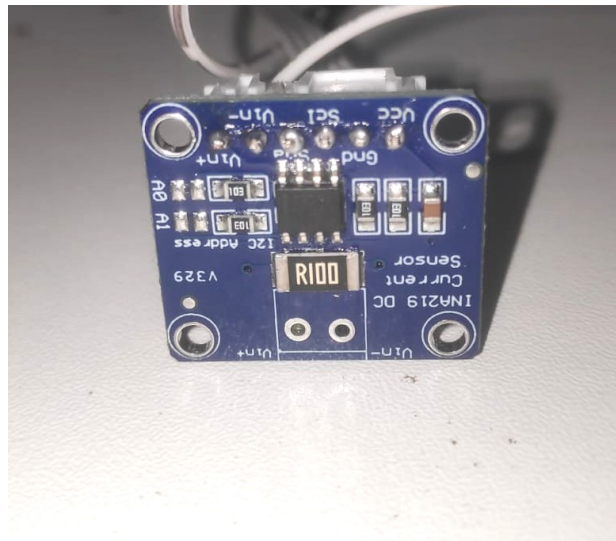


Gambar 2.2 Sensor Inersia MPU6050

2.2.13. INA219

Sensor INA219 adalah modul sensor pengukur daya berbasis IC dari *Texas Instruments* yang mampu mengukur tegangan, arus, dan

daya secara *real-time* melalui antarmuka I2C dengan prinsip pengukuran tegangan *shunt* pada resistor internal $0,1\ \Omega$ yang terpasang seri pada jalur daya, sehingga arus dihitung secara presisi menggunakan hukum Ohm. Sensor ini mendukung tegangan bus hingga 26V dan resolusi pengukuran arus hingga 0,8 mA dengan konversi melalui ADC 12-bit internal, menjadikannya solusi pengukuran daya yang andal untuk *profiling* konsumsi energi pada *board* mikrokontroler *embedded* berdaya rendah, sebagaimana dibuktikan oleh Lambert dkk. yang menggunakan INA219 sebagai *high-side current sense amplifier* untuk mengukur daya, arus, dan tegangan secara *real-time* pada sistem berbasis mikrokontroler dengan hasil pengukuran yang menunjukkan perbedaan statistik signifikan antar kondisi operasi [89]. Validasi penggunaan INA219 pada platform IoT berbasis ESP32 juga telah dikonfirmasi oleh Teso-Fz-Betoño dkk. dalam implementasi kontrol energi *edge* adaptif berbasis kecerdasan buatan, di mana sensor ini memantau konsumsi arus dan tegangan *node* secara langsung di sisi perangkat untuk mengevaluasi efisiensi strategi penghematan energi [90]. Dalam penelitian ini, INA219 digunakan untuk mengukur konsumsi daya ESP32-S3 selama inferensi berlangsung, sehingga efisiensi energi keempat varian model dapat dibandingkan secara objektif. Bentuk fisik sensor INA219 ditampilkan pada Gambar 2.3.



Gambar 2.3 Sensor INA219

2.2.14. *Visual Studio Code*

Perangkat Lunak *Visual Studio Code* merupakan editor kode sumber yang dikembangkan oleh Microsoft dan dirancang untuk mendukung pengembangan aplikasi *desktop*, *web*, maupun *embedded system* dengan performa yang ringan dan antarmuka yang sederhana. *Visual Studio Code* mendukung berbagai bahasa pemrograman dan menyediakan fitur seperti *syntax highlighting*, *debugging*, *IntelliSense*, *Git integration*, serta *extension marketplace* yang membantu proses pengembangan perangkat lunak [91]. Popularitas *Visual Studio Code* telah dikonfirmasi secara empiris melalui *Stack Overflow Developer Survey 2024* yang melibatkan lebih dari 65.000 pengembang dari 185 negara, di mana *Visual Studio Code* digunakan oleh lebih dari dua kali lipat jumlah pengembang dibandingkan alternatif terdekatnya, menjadikannya IDE paling dominan digunakan komunitas pengembang secara global [92]. Untuk keperluan

pengembangan *embedded system*, *Visual Studio Code* dapat dikombinasikan dengan ekstensi PlatformIO IDE yang menyediakan lingkungan pengembangan terintegrasi mencakup manajemen *library*, konfigurasi *board*, *debugging*, dan *Serial Monitor*, sehingga mendukung pengembangan pada mikrokontroler seperti ESP32 dan *Arduino* [93]. Selain itu, *Visual Studio Code* juga mendukung penggunaan *Jupyter Notebook* melalui ekstensi *Jupyter* resmi, yang memungkinkan penulisan dan eksekusi file dengan ekstensi *.ipynb* secara langsung di dalam editor tanpa memerlukan browser terpisah, sehingga memudahkan proses eksplorasi data, visualisasi, dan pengembangan model *machine learning* secara interaktif [94]. Dalam penelitian ini, *Visual Studio Code* digunakan sebagai editor utama untuk proses pengembangan program pada implementasi *TinyML* dan analisis FFT, meliputi penulisan kode program, pengelolaan *library*, *debugging*, serta integrasi dengan *TensorFlow Lite Micro* dan berbagai pustaka *Python* pendukung pengolahan data sensor serta klasifikasi kondisi mesin. Selain itu, eksekusi *notebook* interaktif berbasis file *.ipynb* juga dilakukan di dalam *Visual Studio Code* untuk keperluan analisis dan visualisasi data.

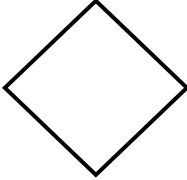
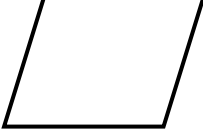
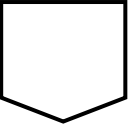
2.2.15. Diagram Alir

Diagram alir atau *Flowchart* merupakan representasi grafis yang menggambarkan langkah-langkah, urutan, dan logika suatu

proses atau algoritma menggunakan simbol-simbol standar yang saling dihubungkan oleh garis aliran. Diagram alir berfungsi tidak hanya sebagai sarana komunikasi, tetapi juga sebagai panduan bagi pemrogram dalam merancang dan mengimplementasikan algoritma, serta terbukti efektif membantu pemrogram pemula dalam mengembangkan kemampuan pemecahan masalah secara terstruktur [95]. Setiap langkah dalam proses direpresentasikan oleh simbol dengan makna tertentu, di mana simbol-simbol tersebut dihubungkan menggunakan anak panah yang menunjukkan aliran dan urutan proses secara berurutan, dengan orientasi penyusunan dari atas ke bawah dan dari kiri ke kanan, serta dimulai dari satu titik awal dan diakhiri oleh satu atau lebih titik *Terminal* [96].

Diagram alir memiliki aplikasi yang luas dalam berbagai bidang, termasuk pengembangan perangkat lunak, desain rekayasa, dan eksperimen ilmiah, karena kemampuannya menyederhanakan proses yang kompleks menjadi lebih mudah dipahami secara visual [97]. Dalam penelitian ini, diagram alir digunakan untuk menggambarkan alur kerja sistem, mulai dari tahap akuisisi data sensor, pemrosesan sinyal FFT, ekstraksi fitur, hingga proses inferensi model *TinyML* pada perangkat ESP32-S3, dengan simbol-simbol yang disajikan pada Tabel 2.2.

Tabel 2.2 Simbol-Simbol Diagram Alir / *Flowchart*

No.	Gambar	Simbol	Fungsi
1		<i>Process</i>	Proses operasional atau komputasi yang dilakukan oleh komputer.
2		<i>Decision</i>	Menunjukkan kondisi tertentu yang menghasilkan dua kemungkinan percabangan jalur: Ya / Tidak (<i>Yes / No</i>).
3		<i>Output/Input</i>	Proses memasukkan data (<i>Input</i>) atau menampilkan data (<i>Output</i>), terlepas dari jenis perangkat keras yang digunakan.
4		<i>Terminal</i>	Menunjukkan awal (<i>start</i>) atau akhir (<i>end</i>) dari suatu program (Prosedur).
5		<i>Flow</i>	Menyatakan arah atau jalurnya alur proses dari satu simbol ke simbol berikutnya.
6		<i>Offline Connector</i>	Menghubungkan satu langkah proses ke proses lainnya pada halaman yang berbeda.
7		<i>Preparation</i>	Menunjukkan proses persiapan atau <i>inisialisasi</i> nilai awal (misalnya nilai awal variabel/ <i>counter</i>) sebelum proses utama dilakukan.

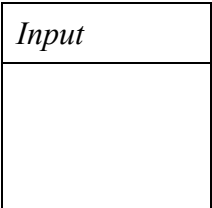
2.2.16. Diagram Blok

Diagram blok adalah diagram suatu sistem di mana bagian-bagian atau fungsi-fungsi utamanya direpresentasikan oleh blok yang dihubungkan oleh garis yang menunjukkan hubungan antar blok

tersebut. Diagram blok digunakan secara luas dalam perancangan sistem berbasis *embedded* dan IoT untuk menggambarkan arsitektur sistem beserta hubungan antar komponen, mencakup hubungan antar komponen perangkat keras, jalur data, dan komunikasi antar subsistem [98], [99].

Dalam penelitian ini, diagram blok digunakan untuk menggambarkan seluruh subsistem, mulai dari akuisisi data sensor (MPU6050, PT100, GPS NEO-8M, RTC DS3231), pemrosesan *on-device* berbasis *pipeline TinyML* pada ESP32-S3, hingga komunikasi data ke platform *monitoring* melalui protokol MQTT. Komponen-komponen utama diagram blok yang digunakan dalam penelitian ini disajikan pada Tabel 2.3.

Tabel 2.3 Komponen Utama Diagram Blok

No.	Simbol	Nama Komponen	Keterangan
1		Blok	Merepresentasikan sebuah subsistem, modul, fungsi, atau komponen dalam sistem. Diberi label nama fungsi/komponen di dalamnya.
2		Garis Penghubung	Menunjukkan arah aliran sinyal, data, atau energi antara dua blok. Arah panah menunjukkan arah transmisi informasi.
3		<i>Input Block</i>	Blok yang menerima sinyal atau data dari luar sistem (misal: sensor MPU6050, PT100, GPS). Biasanya terletak di sisi paling kiri diagram.

No.	Simbol	Nama Komponen	Keterangan
4	Proses	<i>Processing Block</i>	Blok yang melakukan operasi atau transformasi terhadap data (misal: ESP32-S3, FFT, model <i>TinyML</i>). Berada di tengah alur data sistem.
5	<i>Output</i>	<i>Output Block</i>	Blok yang menghasilkan keluaran akhir sistem (misal: hasil klasifikasi kondisi mesin, data MQTT ke platform <i>monitoring</i>). Terletak di sisi paling kanan diagram.