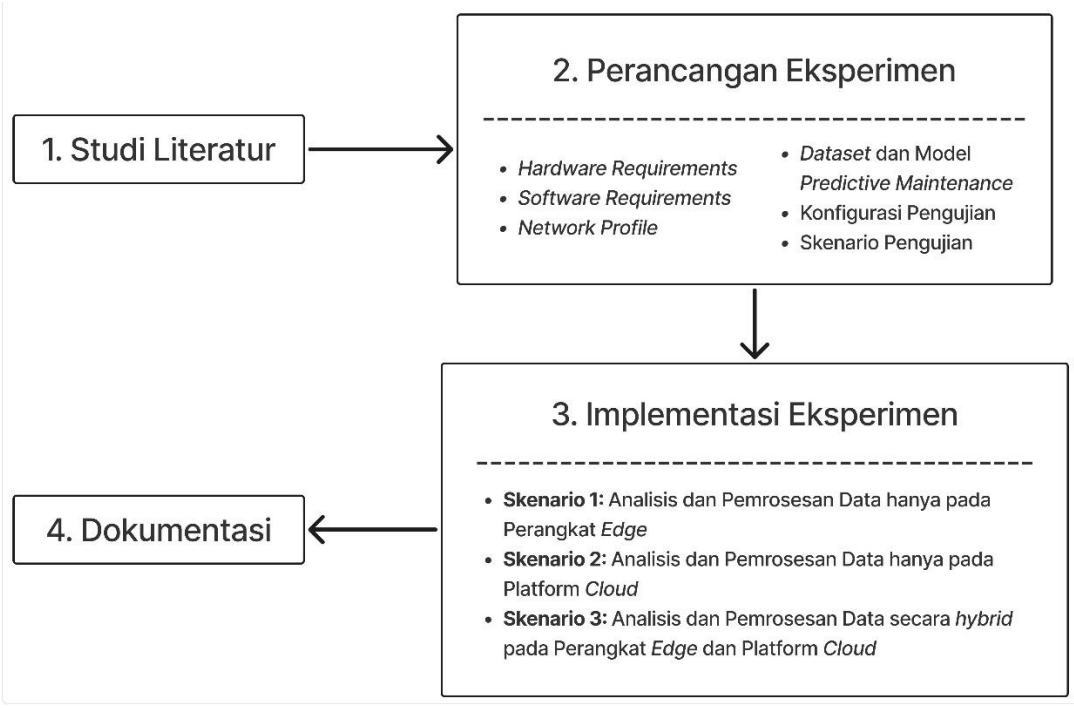
BAB III
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

Metodologi penelitian yang digunakan yaitu metodologi eksperimental berbasis simulasi, dengan tujuan untuk membandingkan pemrosesan data pada *Edge Computing*, *Cloud Computing*, dan arsitektur *Hybrid* dan mengamati perbedaan kinerja dari ketiganya. Penelitian ini termasuk pada penelitian dengan pendekatan kuantitatif karena berfokus pada eksperimen berbasis data numerik dan analisis statistik untuk mengukur dan membandingkan kinerja *Edge Computing* dan *Cloud Computing* serta arsitektur *Hybrid* pada pemrosesan data.

3.2 Tahapan Penelitian



Gambar III.1 Tahapan Penelitian

Penelitian ini dilakukan dalam 4 tahapan: Studi Literatur, Perancangan Eksperimen, Implementasi Eksperimen, dan Dokumentasi. Keempat tahapan tersebut diilustrasikan pada diagram di Gambar III.1. Tiap tahapan akan dijelaskan lebih rinci di sub-bab selanjutnya.

3.2.1 Studi Literatur

Studi literatur pada penelitian ini dilakukan dengan cara membaca dan menganalisis penelitian-penelitian terdahulu untuk memahami konsep-konsep mendasar yang relevan dengan topik, seperti *Edge Computing* (Kubiak dkk., 2022b), *Cloud Computing* (Taleb & Mohamed, 2020), dan perannya dalam Industri 4.0. Studi literatur ini bertujuan untuk mendapatkan wawasan tentang pendekatan-pendekatan yang telah digunakan dalam penelitian sebelumnya, serta untuk mengidentifikasi celah penelitian yang akan diisi oleh penelitian ini.

3.2.2 Perancangan Eksperimen

Tahap perancangan eksperimen untuk penelitian ini dilakukan dengan cara mengidentifikasi kebutuhan dari skenario-skenario yang akan diuji. Kebutuhan tersebut antara lain:

1. *Hardware Requirement*
2. *Software Requirement*
3. *Network Profile*
4. *Dataset* dan Model *Predictive Maintenance*
5. Konfigurasi Pengujian
6. Skenario Pengujian

3.2.2.1 *Hardware Requirement* (Kebutuhan Perangkat Keras)

Perangkat keras yang digunakan pada penelitian ini adalah *Raspberry Pi* 4 Model B sebagai perangkat *Edge* dan *EC2 t2.micro instance* dari *AWS* sebagai platform komputasi di *Cloud*. Spesifikasi detail dari kedua perangkat keras tersebut tercantum pada Tabel III.1 di bawah ini:

Tabel III.1 Hardware *Requirement*

Komponen	***Raspberry Pi* 4 Model B (Perangkat *Edge*)**	***AWS EC2* t3.micro (Platform *Cloud*)**
CPU	*Broadcom BCM2711, Quad core Cortex-A72 (ARM v8)* 64-bit *SoC* @ 1.8GHz	*Intel Xeon Family*, 1v*CPU* @ 3.3GHz
RAM	4 GB LPDDR4-3200	1 GB
Penyimpanan	*MicroSD*	*Elastic Block Storage*
Network	2.4 GHz dan 5.0 GHz IEEE 802.11ac *wireless*	Jaringan internal infrastruktur *AWS*

Pemilihan kedua *platform* dengan spesifikasi yang berbeda ini merupakan keputusan metodologis yang disengaja. Tujuannya adalah untuk menganalisis bagaimana sebuah beban kerja analitik yang identik berdampak pada dua filosofi desain perangkat keras yang berbeda, yang masing-masing merepresentasikan paradigmanya. Penelitian sebelumnya menegaskan bahwa perangkat *Edge* tipikalnya adalah sistem berdaya rendah dengan banyak inti (*core*), sementara instans *Cloud* seringkali berupa *Virtual CPU* (*vCPU)* tunggal dengan kecepatan *clock* yang tinggi (Capra dkk., 2019).

Perbedaan fundamental inilah yang secara langsung memengaruhi interpretasi dari metrik kekuatan komputasi. Penelitian lain menegaskan bahwa utilisasi *CPU* pada perangkat *Edge* memiliki banyak *core* berbasis *Advanced RISC (Reduced Instruction Set Computing) Machine* (*ARM)* untuk menangani tugas

paralel, di mana akan merefleksikan seberapa besar "usaha" yang dikeluarkan oleh sistem berdaya rendah tersebut untuk menjalankan *workload*, sedangkan utilisasi pada *vCPU Cloud* tidak hanya mencerminkan beban kerja analitik itu sendiri, tetapi juga mencakup *overhead* dari lapisan virtualisasi dan proses jaringan yang berjalan di latar belakang infrastruktur *cloud* (Yoo, 2018). Perbandingan metrik kekuatan komputasi dalam penelitian ini tidak bertujuan untuk menyimpulkan "prosesor mana yang lebih kuat", melainkan untuk memahami karakteristik respons beban dari setiap paradigma dalam konfigurasi implementasi yang realistis.

3.2.2.2 *Software Requirement* (Kebutuhan Perangkat Lunak)

Tabel III.2 Software *Requirement*

Kategori	***Edge Computing***	***Cloud Computing***
Sistem Operasi	*Raspberry Pi* OS	*Amazon* Linux 2023
Bahasa Pemrograman	*Python*	*Python*
Library Analitik/ML	*Pandas, NumPy, scikit-learn, Pickle, psutil, pytz, Paho*	*Pandas, NumPy, scikit-learn, Pickle, psutil, pytz, Paho*
Data Collection	*Mosquitto*	*AWS IoT* Core
Model Deployment	*Pickle*	*Pickle*
Penyimpanan Data	*SQLite*	*Amazon RDS* dengan *MySQL Engine*

Tabel III.2 menampilkan detail *Software Requirement* untuk perangkat *Edge* (*Raspberry Pi* 4 Model B) dan platform *Cloud* (*EC2* di *AWS* dengan memanfaatkan model pembayaran *free-tier*). Dapat dilihat bahwa *Edge* dan *Cloud* menggunakan *tools* serupa, namun berbeda dalam infrastruktur. *Edge* bersifat lokal dan ringan, sedangkan *Cloud* terpusat dan bisa diskalakan (*scalable*).

3.2.2.3 *Network Profile* (Profil Jaringan)

Network Profile merupakan deskripsi teknis mengenai kondisi jaringan yang digunakan selama proses pengujian berlangsung. Profil ini penting untuk memastikan konsistensi hasil pengujian, khususnya pada skenario yang melibatkan komunikasi lintas jaringan seperti *Cloud* dan *Hybrid*. Rincian lengkap spesifikasi jaringan disajikan pada Tabel III.3 berikut ini.

Tabel III.3 Network Profile

Parameter	**Nilai**
Frekuensi	2442 *MHz* (2.4*GHz band*)
Kekuatan Sinyal	-48 *dBm*
RX/TX Rate	72.2 *Mbit/s*
TX Power	31 *dBm*
Wi-Fi Standard	*IEEE* 802.11

3.2.2.4 *Dataset* dan Model *Predictive Maintenance*

1. *Dataset*

Dataset yang digunakan adalah "AI4I 2020 *Predictive Maintenance Dataset*". *Dataset* sintetis ini dibuat untuk mencerminkan penggunaan *predictive maintenance* di dunia nyata dalam lingkungan industri. *Dataset* ini dirancang untuk menawarkan representasi praktis karena sulitnya memperoleh dan berbagi *dataset* di dunia nyata, khususnya untuk *predictive maintenance* (Matzka, 2020).

Dataset ini berisi 10.000 titik data, yang masing-masing memiliki 14 fitur. Fitur-fitur ini mencakup data operasional seperti suhu, torsi, dan keausan alat, yang secara langsung memengaruhi kinerja dan kerusakan mesin. *Dataset* ini juga menyertakan label kerusakan mesin yang menunjukkan apakah kerusakan telah terjadi karena salah satu dari lima mode kerusakan: *Tool Wear Failure (TWF), Heat*

Dissipation Failure (HDF), Power Failure (PWF), Overstrain Failure (OSF), dan *Random Failures (RNF)*. *Dataset* ditujukan untuk menyimulasikan skenario realistis di mana kerusakan dapat dipicu oleh berbagai faktor selama pengoperasian mesin industri.

2. *Exploratory Data Analysis* dan Praproses (*Preprocessing*)

Exploratory Data Analysis (*EDA*) dilakukan untuk memahami distribusi *dataset* dan mengidentifikasi potensi anomali atau pola yang dapat memengaruhi kinerja model. Konversi tipe data diterapkan untuk memastikan bahwa semua fitur diformat dengan benar. Teknik penanganan *outlier* juga diterapkan untuk menghilangkan nilai ekstrem yang dapat mendistorsi kinerja model. Transformasi dan visualisasi dibuat untuk mengeksplorasi hubungan antara fitur dan variabel target.

3. Pelatihan Model *Predictive Maintenance*

Dataset dibagi menjadi *training set* (70%) dan *testing set* (30%). Untuk mengatasi ketidakseimbangan kelas dalam *dataset*, di mana kondisi kerusakan jauh lebih jarang daripada operasi normal, *oversampling* diterapkan pada data pelatihan untuk menyeimbangkan kelas. Beberapa algoritma pembelajaran mesin diuji, termasuk *Decision Tree, k-Nearest Neighbors (k-NN), Random Forest, Gradient Boosting, Gaussian Naive Bayes,* dan *Multi-layer Perceptron (MLP)*. Hasil dari tiap algoritma dapat dilihat pada Tabel III.4.

Tabel III.4 Hasil Uji tiap Algoritma

Model	***Accuracy***	***Precision***	***Recall***	***F1-Score***	***Training Time (s)***	***Prediction Time (s)***
Decision Tree	0.9638	0.9765	0.9638	0.9688	0.1149	0.0030
k-NN	0.9745	0.9738	0.9745	0.9741	0.0156	0.1679
Random Forest	0.9794	0.9797	0.9794	0.9796	1.6238	0.0381
Gradient Boosting	0.9670	0.9781	0.9670	0.9713	4.0398	0.0057
Gaussian Naive Bayes	0.8291	0.9713	0.8291	0.8859	0.0103	0.0026
MLP	0.9748	0.9804	0.9748	0.9770	30.0276	0.0141

Model *Decision Tree* dipilih sebagai model *predictive maintenance* dalam penelitian ini karena kombinasi antara efisiensi komputasi, kinerja prediktif yang kuat, dan interpretabilitas tinggi, yang sangat selaras dengan kebutuhan aplikasi industri *real-time*. Model ini menunjukkan kemampuan yang sangat baik dari sisi performa, dengan akurasi 96,38% dan *F1-Score* 96,88%. Skor F1 yang tinggi ini sangat krusial karena menandakan adanya keseimbangan yang kuat antara presisi dan *recall*, di mana kesalahan prediksi, baik *false positive* maupun *false negative*, dapat menyebabkan kerugian operasional yang signifikan.

Keunggulan utama *Decision Tree* terletak pada efisiensinya. Dengan waktu pelatihan hanya 0,1149 detik dan yang terpenting, waktu prediksi sangat cepat yaitu 0,0030 detik, model ini sangat ideal untuk lingkungan Industri 4.0 yang menuntut respons nyaris instan. Meskipun model lain seperti *Random Forest* menawarkan akurasi yang sedikit lebih tinggi (97,94%), pilihan tersebut menjadi tidak praktis jika dihadapkan pada tuntutan kecepatan. Peningkatan akurasi sebesar ~1,6% dari *Random Forest* harus dibayar dengan waktu prediksi yang 12,7 kali lebih lambat

(0,0381 detik) dibandingkan *Decision Tree*. Penundaan seperti ini tidak dapat diterima dan dapat menghilangkan nilai dari prediksi itu sendiri jika dilihat dalam konteks industri di mana keputusan harus dibuat dalam hitungan milidetik. Keuntungan tambahannya adalah *Decision Tree* menawarkan interpretabilitas atau kemudahan untuk dipahami. Pertimbangan keseimbangan terbaik antara akurasi yang solid, kecepatan prediksi yang superior, dan kemudahan interpretasi, *Decision Tree* merupakan pilihan yang paling logis dan tepat untuk studi kasus dalam penelitian ini.

3.2.2.5 Konfigurasi Pengujian

Pengujian untuk setiap skenario dilakukan sebanyak 3 kali iterasi, masing-masing berjalan selama 3 jam. Simulasi transmisi data dilakukan dari PC lokal dengan laju normal 20 pesan/detik dan target pengiriman 216.450 pesan via protokol MQTT. Disimulasikan juga 5 kali lonjakan beban (*burst*) menjadi 30 pesan/detik selama 3 detik untuk menguji ketahanan sistem. Dampak dari beban lonjakan ini tercermin pada metrik performa keseluruhan seperti latensi tertinggi, dan tidak dianalisis sebagai kejadian terpisah. Detail konfigurasi pengujian disajikan pada Tabel III.5.

Tabel III.5 Konfigurasi Pengujian

Parameter	**Nilai**
Jumlah Iterasi	3
Durasi	3 Jam
Kadar Pesan per Detik	20 pesan
Lonjakan Pesan (*Burst*)	30 pesan/detik selama 3 detik (5 kali)
Target Pesan	216.450 pesan
Protokol	MQTT (*QoS* 1, sesi non-persisten)

3.2.2.6 Skenario Pengujian

Studi ini mengevaluasi kinerja tiga skenario arsitektur *Edge*, *Cloud*, dan *Hybrid*. Sebelum merinci alur kerja spesifik pada setiap arsitektur, penting untuk memahami tahapan pemrosesan data umum yang diterapkan secara konsisten setelah pesan *MQTT* diterima. Alur ini terdiri dari lima tahap utama:

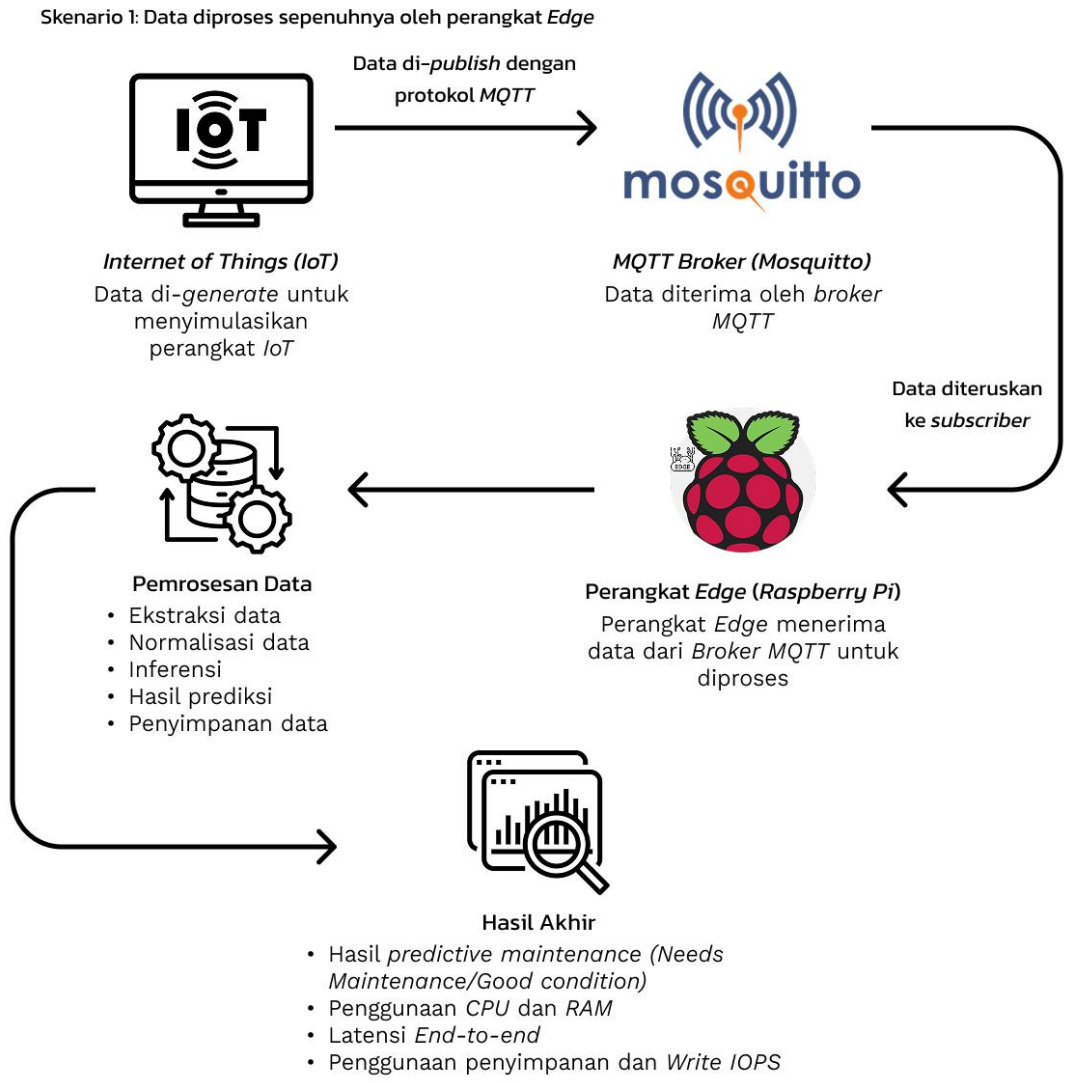
1) Ekstraksi Data: Data sensor dari perangkat simulasi diterima melalui protokol MQTT dalam format pesan terstruktur, diasumsikan sebagai JSON. Tahap pertama adalah melakukan penguraian (*parsing*) terhadap struktur pesan tersebut untuk mengekstrak nilai dari setiap kunci (*key*) fitur yang relevan. Fitur-fitur kunci yang diekstraksi mencakup data operasional mesin seperti '*Air temperature [K]*', '*Process temperature [K]*', dan lainnya, sesuai dengan kolom pada *dataset* AI4I 2020 *Predictive Maintenance*. Proses ekstraksi ini mengubah data mentah dari pesan menjadi sebuah vektor fitur numerik yang siap untuk tahap selanjutnya.
2) Normalisasi Data (Z-score): Setelah vektor fitur berhasil diekstraksi, tahap selanjutnya adalah normalisasi data. Tahap ini sangat krusial karena fitur-fitur yang ada memiliki skala dan rentang nilai yang sangat berbeda (misalnya, suhu dalam Kelvin vs. torsi dalam Nm). Model *machine learning* cenderung bekerja lebih baik dengan data yang memiliki skala seragam. Metode normalisasi yang digunakan dalam penelitian ini adalah *Z-score normalization*, yang mengubah setiap nilai fitur (x) menjadi nilai Z-score (z) dengan menggunakan rumus:

$$\mathrm{Z} = \frac{\chi - \mu}{\sigma}$$

μ adalah nilai rata-rata (*mean*) dan σ adalah standar deviasi dari fitur tersebut. Poin metodologis yang sangat penting dalam implementasi ini adalah bahwa nilai μ dan σ yang digunakan untuk normalisasi bukanlah dihitung dari data yang sedang masuk, melainkan telah dihitung sebelumnya dari *dataset* pelatihan saat tahap pembuatan model. Pendekatan ini merupakan praktik standar untuk mencegah kebocoran informasi (*data leakage*) dan memastikan proses inferensi secara akurat menyimulasikan bagaimana model bereaksi terhadap data baru. Setelah semua fitur dinormalisasi, data siap untuk tahap inferensi.

3) Inferensi Model: Vektor fitur yang telah dinormalisasi kemudian dimasukkan ke dalam model *Decision Tree* untuk proses inferensi. Model ini sebelumnya telah dilatih dan disimpan sebagai objek ter-serialisasi dalam format file .pkl. Objek model ini dimuat kembali ke dalam memori pada saat eksekusi menggunakan *library pickle*. Metode prediksi dari model dipanggil dengan vektor fitur yang sudah siap sebagai inputnya. Proses ini menghasilkan output prediksi awal berupa nilai biner (misalnya, 0 untuk kondisi normal dan 1 untuk potensi kegagalan).
4) Klasifikasi Status Mesin: Hasil prediksi biner dari model kemudian diklasifikasikan ke dalam label yang lebih mudah dipahami, yaitu "Kondisi Baik" atau "Memerlukan Perawatan".
5) Penyimpanan Hasil: Hasil akhir prediksi beserta *timestamp* dan data relevan lainnya kemudian disimpan ke dalam *database* yang sesuai dengan skenarionya (*SQLite* untuk *Edge*, *RDS MySQL* untuk *Cloud*).

1. Skenario 1: Data diproses sepenuhnya oleh perangkat *Edge*



Gambar III.2 Skema Skenario 1

a) Alur Data

Alur data dirancang untuk mencerminkan skema pemrosesan terdesentralisasi yang sepenuhnya dilakukan di sisi *Edge*. Proses dimulai dari komputer lokal yang menyimulasikan data sensor industri seperti ditunjukkan pada Gambar III.2. Data ini dikirimkan melalui protokol *Message Queuing Telemetry Transport (MQTT)*, yang dikenal efisien dan ringan untuk komunikasi *IoT*, ke *Mosquitto MQTT Broker*

yang dijalankan langsung di *Raspberry Pi* 4. *Raspberry Pi* bertindak sebagai broker dan *subscriber*, yang secara aktif menerima setiap pesan dari topik yang telah ditentukan.

b) Tahapan Pemrosesan

   i. Ekstraksi parameter sensor

   ii. Normalisasi data berbasis *z-score*

   iii. Inferensi menggunakan model *predictive maintenance* (*pickle*)

   iv. Klasifikasi status mesin: sehat atau perlu perbaikan

   v. Penyimpanan hasil prediksi ke *database* lokal (*SQLite*)

c) Parameter yang Dipantau

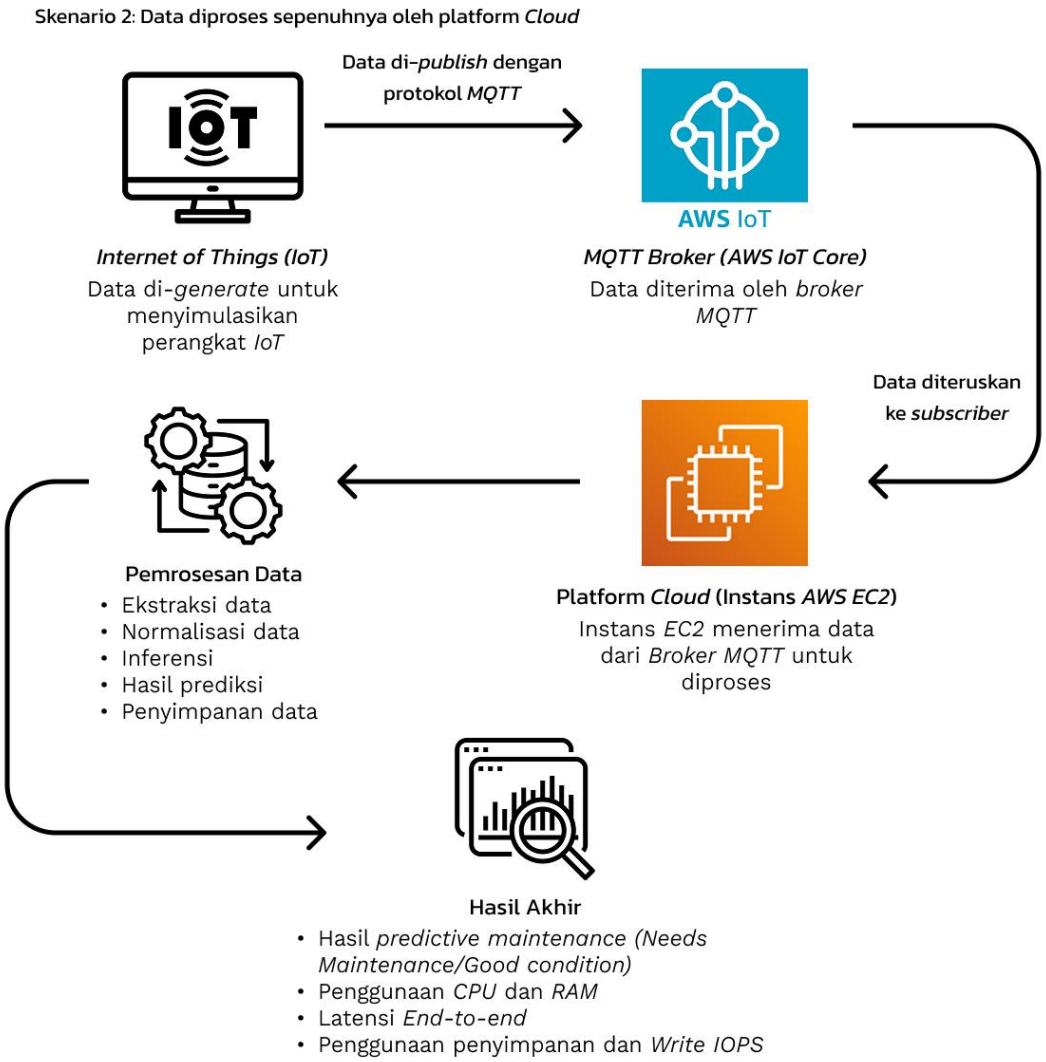
   i. Komputasi: *CPU* & *RAM* (*monitoring* via *psutil*)

   ii. Penyimpanan: *Disk usage* (MB) dan *Write IOPS* (jumlah operasi tulis/5 detik)

   iii. Latensi: *End-to-end latency* dari waktu *publish* ke waktu inferensi

   iv. Semua metrik dicatat otomatis per menit melalui log sistem lokal.

2. Skenario 2: Data diproses sepenuhnya oleh platform *Cloud*



Gambar III.3 Skema Skenario 2

a) Alur Data

Alur data di skenario ini dirancang untuk mencerminkan skema pemrosesan terpusat yang sepenuhnya dilakukan di sisi *Cloud*. Proses dimulai dari komputer lokal yang menyimulasikan data sensor industri secara kontinu seperti ditunjukkan pada Gambar III.3. Data ini dikirimkan melalui protokol *MQTT* ke *AWS IoT Core*, yang berfungsi sebagai *MQTT* broker berbasis *cloud*. *AWS IoT Core* bertugas

menerima, mengautentikasi, dan meneruskan data secara aman menuju instans *Amazon EC2* sebagai *subscriber*. Seluruh tahapan inferensi dilakukan di EC2 tanpa keterlibatan pemrosesan lokal, mencerminkan pendekatan *cloud-centric* yang mengandalkan skalabilitas dan fleksibilitas infrastruktur komputasi awan.

b) Tahapan Pemrosesan

   i. Ekstraksi data

   ii. Normalisasi data dengan *z-score*

   iii. Inferensi menggunakan model *predictive maintenance* yang sama

   iv. Klasifikasi kondisi mesin

   v. Penyimpanan ke *database AWS RDS* (*MySQL*)

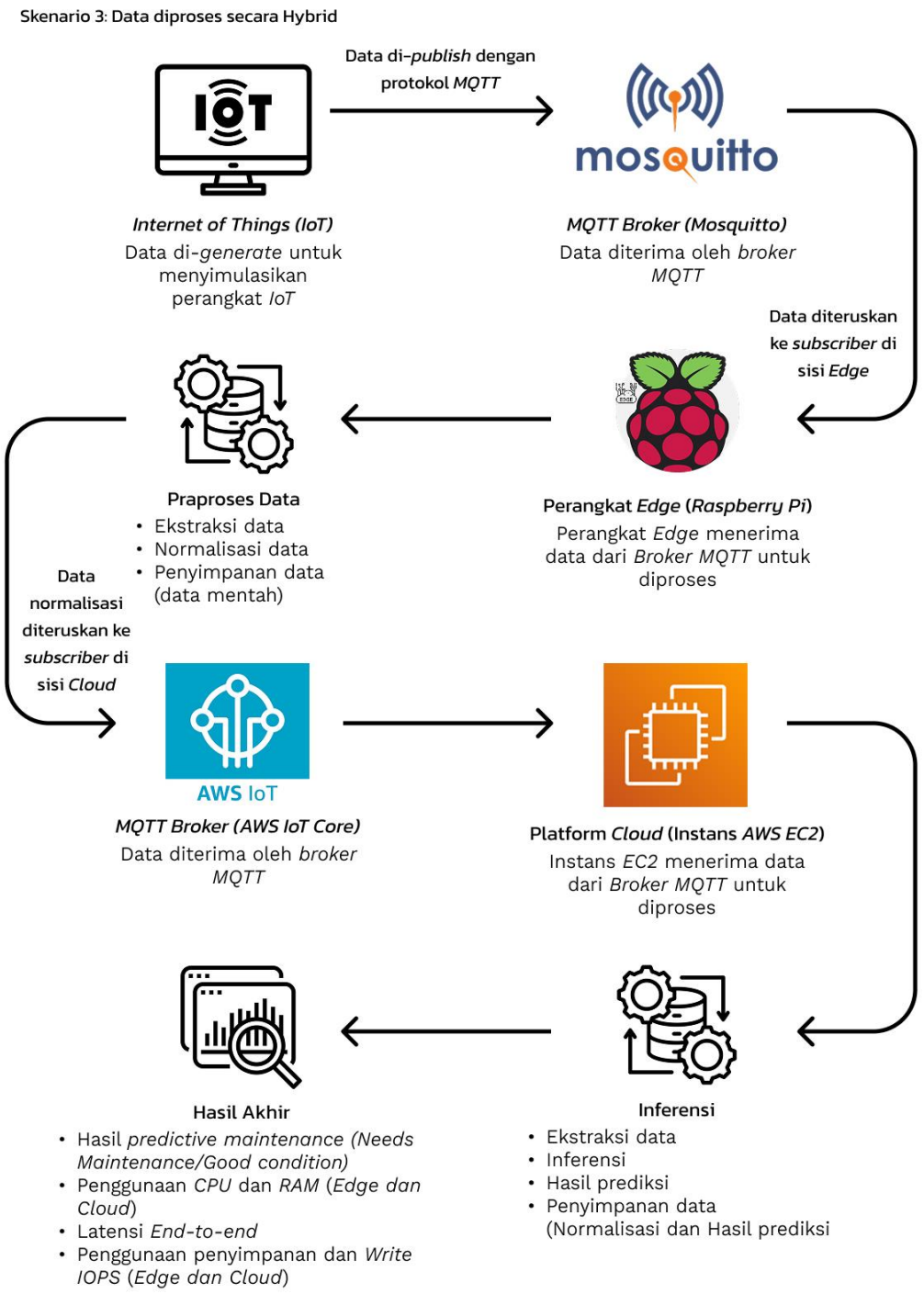
c) Parameter yang Dipantau

   i. Komputasi: *CPU* (via *AWS Cloud*Watch: *CPU*Utilization), *RAM* (*psutil*)

   ii. Penyimpanan: *Disk usage* (MB) dan *Write IOPS* (via *CloudWatch* Write*IOPS*)

   iii. Latensi: Pengukuran waktu *end-to-end* dari *timestamp* data di-*publish* hingga hasil inferensi tercatat

3. Skenario 3: Data diproses secara *Hybrid* oleh perangkat *Edge* dan platform *Cloud*



Gambar III.4 Skema Skenario 3

a) Alur Data

Skenario ini mencerminkan pemrosesan dua tahap, di mana beban awal ditangani oleh *Edge* dan inferensi dilakukan oleh *Cloud* untuk mengoptimalkan kecepatan dan skalabilitas. Data dari perangkat simulasi dikirim ke *Mosquitto broker* di *Raspberry Pi*. *Edge* melakukan *preprocessing* awal, kemudian meneruskan data ke *AWS IoT Core* untuk pemrosesan lanjutan. Skema skenario 3 dapat dilihat pada Gambar III.4.

b) Tahapan Pemrosesan

Pemrosesan terjadi dalam dua tahap, yaitu *preprocessing* di sisi *Edge* dan inferensi menggunakan model *Predictive Maintenance* di sisi *Cloud*. Hal ini sesuai dengan yang dijelaskan di bagian Alur Data sebelumnya. Detail dari proses kedua tahap tersebut dapat dilihat pada Tabel III.6.

Tabel III.6 Tahap Pemrosesan di Sisi *Edge* dan *Cloud*

Di sisi *Edge*	**Di sisi *Cloud***
Normalisasi data	Inferensi menggunakan model
Logging ke *SQLite* lokal	Klasifikasi kondisi mesin
Forward ke *AWS IoT Core*	Penyimpanan hasil ke *RDS MySQL*

c) Parameter yang Dipantau

i. Komputasi: *CPU* dan *RAM* (*Edge* via *psutil*, *Cloud* via *CloudWatch+psutil*)

ii. Penyimpanan: *Disk usage* (MB) dan *Write IOPS*, dipantau terpisah untuk *Edge* dan *Cloud*

iii. Latensi: *End-to-end latency* dihitung dari waktu *publish* awal hingga hasil akhir tersedia

3.2.3 Implementasi Eksperimen

Tahap implementasi eksperimen dilakukan untuk memungkinkan pemodelan sistem yang kompleks secara akurat, mengidentifikasi potensi masalah, serta menguji berbagai strategi dalam lingkungan terkontrol. Pendekatan ini mengurangi risiko dan biaya dengan memastikan solusi terbaik berdasarkan data yang mendetail (Alrabghi dkk., 2017). Validasi dan verifikasi model melalui simulasi memastikan keakuratan hasil penelitian, meningkatkan pemahaman sistem, serta membantu mendapatkan solusi optimal sebelum diterapkan di dunia nyata.

1. Implementasi Skenario 1: Data dari *IoT* diproses oleh perangkat *Edge* saja

a) Deployment

Model *predictive maintenance* di-*deploy* ke perangkat *Edge* (*Raspberry Pi*), menggunakan format *Pickle* untuk efisiensi *loading*. *Database SQLite* digunakan untuk menyimpan hasil prediksi secara lokal.

b) Arsitektur dan Proses

PC lokal sebagai *publisher* menyimulasikan dan menghasilkan data kemudian mengirim data tersebut ke *broker* MQTT *Mosquitto* pada *Raspberry Pi*. *Raspberry Pi* menjalankan *subscriber* MQTT yang membaca data, menjalankan *preprocessing* (normalisasi *z-score*), dan menjalankan model. Hasil prediksi disimpan ke *SQLite*.

c) Monitoring dan *Logging*

CPU dan *RAM* dimonitor menggunakan *psutil* dan dicatat ke file log setiap 60 detik. *Write IOPS* dicatat menggunakan psutil dan disimpan ke log. Latensi

dihitung dari *timestamp* waktu *publish* (*published_timestamp*) dan waktu terima (*time*.*time*()), lalu dicatat ke latency.log.

d) Pseudocode

```
loop:
    data = mqtt_receive()
    normalized = z_score_normalization(data)
    prediction = model.predict(normalized)
    store_to_sqlite(prediction)
    log_cpu_ram()
    log_latency(published_timestamp, received_time)
```

2. Implementasi Skenario 2: Data dari *IoT* diproses oleh platform *Cloud* saja

a) Deployment

Model di-host pada *EC2* instance dengan Python dan *MySQL* connector. *Database AWS RDS* dengan *MySQL engine* digunakan sebagai penyimpanan backend.

b) Arsitektur dan Proses

PC lokal sebagai *Publisher* mengirim data ke *AWS IoT* Core menggunakan sertifikat TLS (QoS 1). *EC2* sebagai *subscriber* membaca data dari *AWS*, melakukan *preprocessing* dan inferensi. Hasil prediksi disimpan ke *AWS RDS* dengan *MySQL engine*.

c) Monitoring dan *Logging*

CPU dan *Write IOPS* dimonitor menggunakan *AWS CloudWatch* (*CPU*Utilization, Write*IOPS*). *RAM* dimonitor secara lokal menggunakan *psutil* dan dicatat ke log. Latensi dihitung sama seperti pada *Edge*, disimpan ke file *latency.log*.

d) Pseudocode

```
loop:
   data = mqtt_receive()
    normalized = preprocess(data)
    prediction = model.predict(normalized)
    store_to_rds(prediction)
    log_cloudwatch_metrics()
    log_latency(published_timestamp, received_time)
```

3. Implementasi Skenario 3: Data dari *IoT* diproses oleh perangkat *Edge* dan platform *Cloud* (*Hybrid*)

a) Deployment

Raspberry Pi melakukan *preprocessing*, menyimpan raw data di *SQLite*. Data hasil *preprocessing* dikirim ke *AWS IoT* Core. *EC2* instance menerima data dari *AWS IoT* Core dan melakukan inferensi serta menyimpan hasilnya di *AWS RDS* dengan *MySQL* Engine.

b) Arsitektur dan Proses

PC lokal menyimulasikan data *IoT* dan mem-*publish* ke *broker* MQTT *Mosquitto*. *Raspberry Pi* sebagai *subscriber* pertama kemudian melakukan proses *preprocessing* untuk menormalisasikan data. Data yang telah dinormalisasi kemudian diteruskan ke *broker* MQTT *AWS IoT* Core. *EC2* sebagai *subscriber* kedua mengambil data yang diteruskan kemudian melakukan proses inferensi akhir untuk mendapatkan hasil prediksi. Data prediksi disimpan di *AWS RDS* dengan *MySQL engine*.

c) Monitoring dan *Logging*

Pada sisi *Edge*, *CPU* dan *RAM* dimonitor menggunakan *psutil* dan dicatat ke file log setiap 60 detik. *Write IOPS* dicatat menggunakan psutil dan disimpan ke

log. Latensi dihitung dari *timestamp* waktu *publish* (*published_timestamp*) dan waktu terima (*time*.*time*()), lalu dicatat ke *latency*.*log*.

CPU dan *Write IOPS* dimonitor menggunakan *AWS Cloud*Watch (*CPUUtilization*, *WriteIOPS*) pada sisi *Cloud*. *RAM* dimonitor secara lokal menggunakan *psutil* dan dicatat ke log. Latensi dihitung sama seperti pada *Edge*, disimpan ke file *latency*.*log*.

Latensi *end-to-end* dicatat dari waktu awal *publish* hingga penyimpanan hasil inferensi.

d) Pseudocode

```
# Edge Preprocessing
loop_edge:
    data = mqtt_receive_local()
   normalized = z_score_normalization(data)
    mqtt_publish_aws(normalized)
    store_to_sqlite_raw(data)
    log_edge_metrics()

# Cloud Inferencing
loop_cloud:
    data = mqtt_receive_aws()
    prediction = model.predict(data)
    store_to_rds(prediction)
    log_cloud_metrics()
    log_latency(published_timestamp, received_time)
```

3.2.4 Dokumentasi

Keluaran yang diharapkan dari tahap Dokumentasi ini adalah laporan hasil penelitian yang berisi jawaban atas rumusan masalah yang telah ditentukan sebelumnya. Laporan ini mencakup hasil analisis dan evaluasi terhadap penyimpanan data, kekuatan komputasi, dan latensi dalam setiap skenario pengujian. Tahap ini juga menghasilkan dokumentasi teknis terkait implementasi eksperimen serta visualisasi hasil dalam bentuk tabel, grafik, dan laporan kondisi mesin.