BAB IV
PEMBUATAN ALAT

Dalam penyusunan tugas akhir proses implementasi dan realisasi fisik dari sistem yang telah dirancang. Pembuatan alat mencakup tiga aspek utama, yaitu pembuatan perangkat keras, pembuatan perangkat lunak, dan pembuatan algoritma sistem deteksi dini penyakit.

4.1 Pembuatan Perangkat Keras

4.1.1 Pembuatan Komponen Mekanik

4.1.1.1 Alat dan Bahan

Sebelum proses fabrikasi dimulai, seluruh komponen dan peralatan yang dibutuhkan dikumpulkan dan diverifikasi kesesuaiannya dengan spesifikasi perancangan. Alat dan bahan yang digunakan dalam proses pembuatan prototipe dirangkum dalam Tabel 4-1 dan Tabel 4-2 berikut.

Tabel 4. 1 Alat yang Digunakan dalam Pembuatan Komponen Mekanik

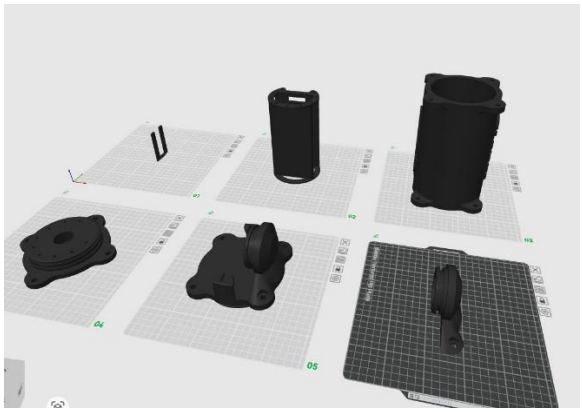
No.	Nama Alat	Jumlah
1.	3D printer	1 buah
2.	Laptop	1 buah
3.	Tang potong	1 buah
4.	Kunci pas	1 buah
5.	Kuas cat	1 buah
6.	Bor	1 buah
7.	Gelas takar	1 buah

Tabel 4. 2 Bahan yang Digunakan
dalam Pembuatan Komponen Mekanik

No.	Nama Alat	Jumlah
1.	Filamen PLA	985 gr
2.	Amplas	1 buah
3.	O-ring Seal	1 buah
4.	Seal tape	1 buah
5.	Resin epoxy	750 gr
6.	Kuas cat	1 buah
7.	Lem dextone	1 buah
8.	Dome	1 buah
9.	Mur	4 buah

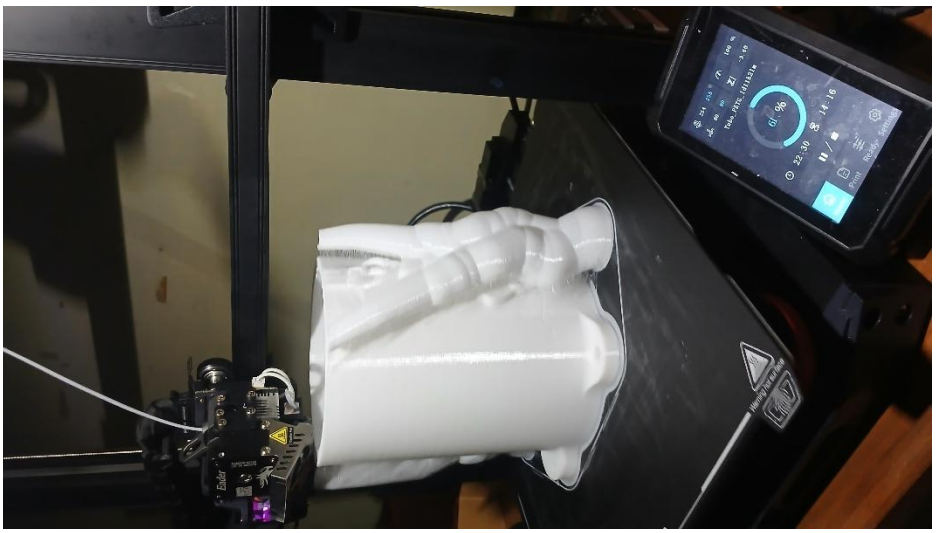
4.1.1.2 Proses Fabrikasi Komponen Utama

Proses fabrikasi fisik komponen housing prototipe dilakukan dengan menggunakan teknologi 3D Printing. Bahan dasar yang digunakan dalam proses pencetakan ini adalah filamen *PolyLactic Acid* (PLA) dengan total estimasi massa penyerapan material sebesar 985 gram. Proses pencetakan dilakukan secara bertahap untuk setiap bagian struktur, meliputi komponen *main housing*, *end cap*, dan *lamp holder*. Proses fabrikasi pada bagian-bagian mekanik ini dimulai dari tahap persiapan model 3D hasil perancangan yang akan diekstraksi ke dalam mesin cetak. Sebelum memasuki tahap produksi fisik, wujud rancangan geometri dari masing-masing komponen ditunjukkan pada Gambar 4.1 dan 4.2.



Gambar 4. 1 Proses Cetak 3D

Setelah visualisasi perancangan diverifikasi, model tersebut diproses menggunakan perangkat lunak slicing untuk mengatur parameter cetak, seperti ketebalan dinding dan kerapatan struktur bagian dalam. Proses pencetakan secara riil menggunakan mesin 3D Printer untuk merealisasikan komponen fisik ditunjukkan pada Gambar 4.3.



Gambar 4. 2 Proses Fabrikasi

Secara keseluruhan, durasi total yang diperlukan untuk memfabrikasi seluruh elemen *housing* meliputi *main housing*, *end cap*, dan *lamp holder* menggunakan mesin 3D Printer adalah sekitar 48 jam.

4.1.1.3 Proses Pascapemrosesan

Setelah seluruh komponen utama selesai dicetak, langkah berikutnya adalah fase pascapemrosesan untuk memperbaiki kualitas permukaan dan meningkatkan

kekuatan struktur mekanik. Karakteristik hasil cetak 3D printing umumnya memiliki celah mikroskopis antar lapisan, yang berpotensi memicu kebocoran air. Oleh karena itu, dilakukan proses pengamplasan permukaan luar housing terlebih dahulu menggunakan kertas amplas secara bertahap dari tekstur kasar hingga halus untuk meratakan permukaan.

Langkah berikutnya adalah pelapisan menggunakan cairan resin epoxy dengan total massa 750 gram. Cairan resin diaduk di dalam gelas takar dengan komposisi perbandingan yang sesuai antara resin dan *hardener*, kemudian diaplikasikan ke seluruh permukaan luar dan dalam komponen PLA menggunakan kuas cat. Proses pelapisan resin ini berfungsi ganda: pertama, sebagai zat pengikat penutup celah antar-lapisan struktur guna menjamin karakteristik kedap air kedua, untuk menambah rigiditas dan kekokohan struktural komponen mekanik agar tahan terhadap tekanan hidrostatik di dalam air tambak.

Gambar 4. 3 Proses Pelapisan Resin Epoxy

Berdasarkan dokumentasi pada Gambar 4.3, pengaplikasian resin epoxy dilakukan secara merata ke setiap sudut kritis housing, terutama pada area sambungan dan lekukan geometri yang memiliki risiko keretakan atau kebocoran tertinggi. Setelah proses pelapisan selesai, komponen memasuki fase pengeringan pada suhu ruang selama sekitar 36 jam hingga cairan resin terpolimerisasi sepenuhnya dan membentuk lapisan pelindung yang keras, mengkilap, dan solid.

4.1.1.4 Perakitan Mekanik

Struktur kubah transparan disatukan pada bagian depan *housing* menggunakan lem Dextone untuk memastikan sambungan yang permanen dan kuat. Komponen internal, termasuk elemen dudukan perangkat keras, dimasukkan ke dalam kompartemen utama sebelum akhirnya ditutup menggunakan komponen *end cap* pada sisi belakang. Sistem penguncian akhir menggunakan kombinasi 4 buah mur dan baut yang dikencangkan secara silang menggunakan kunci pas untuk meratakan tekanan mekanis pada permukaan perimeter sirkular. Guna mencapai spesifikasi kedap air yang andal, komponen *o-ring seal* dipasang secara presisi pada celah sambungan utama dan diperkuat dengan lilitan *seal tape* pada bagian ulir mekanis. Proses perakitan elemen-elemen mekanik ini ditunjukkan pada gambar 4.4, sedangkan wujud utuh komparasi fisik keseluruhan housing setelah selesai dirakit diperlihatkan pada gambar 4.5.

Gambar 4. 4 Proses Perakitan Elemen Mekanik dan Sistem Penguncian

Gambar 4. 5 Hasil Akhir Perakitan Housing Prototipe

4.1.2 Pembuatan Komponen Elektrik

4.1.2.1 Alat dan Bahan

Proses perakitan komponen elektrik melibatkan integrasi seluruh perangkat keras ke dalam housing prototipe yang telah selesai diproduksi. Implementasi rangkaian elektrik ini dilakukan berdasarkan perancangan sebelumnya, dengan menempatkan Raspberry Pi 5 sebagai unit komputasi pusat, yang mengoordinasikan seluruh subsistem.

Daftar alat dan bahan yang digunakan dalam proses pembuatan serta perakitan komponen elektrik ini masing-masing dirinci pada tabel 4.3 dan tabel 4.4.

Tabel 4. 3 Alat yang Digunakan dalam Pembuatan Komponen Elektrik

No.	Nama Alat	Jumlah
1.	Laptop	1 buah
2.	Obeng	1 buah
3.	Solder	1 buah
4.	Tang potong	1 buah
5.	Multimeter	1 buah
6.	Tenol	1 buah

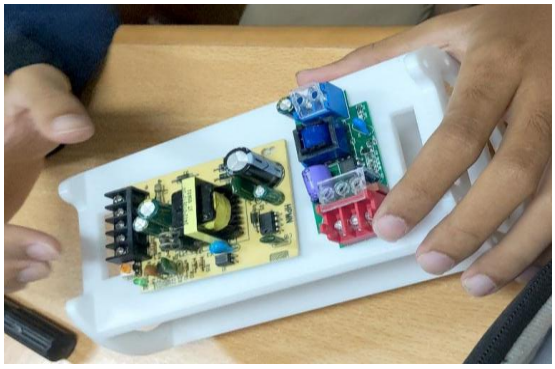
Tabel 4. 4 Bahan yang Digunakan dalam Pembuatan Komponen Elektrik

No.	Nama Alat	Jumlah
1.	Raspberry Pi 5	1 buah
2.	Hailo 8L NPU	1 buah
3.	HQ Camera IMX477	1 buah
4.	Lensa CS-Mount 5 MP	1 buah

No.	Nama Alat	Jumlah
5.	Power Supply 5V	1 buah
6.	Power Supply 12 V	1 buah
7.	LED Eagle Eye	1 buah
8.	Sambungan kabel	1 buah
9.	Kabel jumper	1 buah
10.	Kabel CSI	1 buah

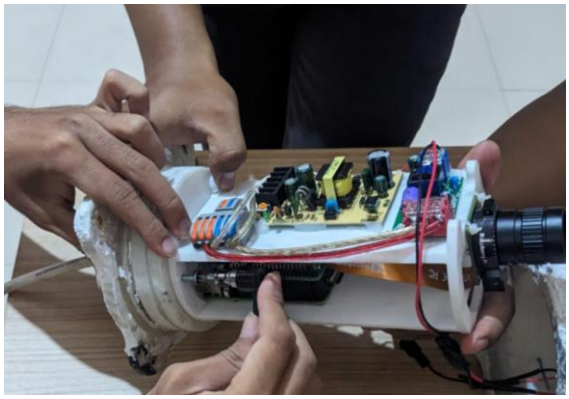
3.1.2.2 Perakitan Perangkat Elektrik

Pembuatan perangkat elektrik meliputi penyambungan dan pengujian komponen elektronik sesuai konfigurasi pada *wiring diagram* yang telah dirancang. Sebelum tahap pengabelan, catu daya dan modul konverter dipasang pada dudukan internal prototipe sesuai tata letak yang telah ditentukan. Penempatan komponen regulator daya ditunjukkan pada Gambar 4.7.



Gambar 4. 6 Penempatan Komponen Catu Daya pada *Housing* Prototipe

Perakitan sistem diawali dengan integrasi catu daya sebagai sumber tegangan DC. Keluaran catu daya didistribusikan ke jalur 12 V untuk lampu LED dan jalur 5 V untuk Raspberry Pi 5. Modul Hailo-8L diintegrasikan melalui antarmuka M.2 HAT+, sedangkan kamera HQ IMX477 dihubungkan melalui antarmuka MIPI CSI. Hasil perakitan dan integrasi seluruh komponen ditunjukkan pada Gambar 4.8.



Gambar 4. 7 Hasil Perakitan Fisik Komponen Elektrik Prototipe

4.2 Pembuatan Perangkat Lunak

4.2.1 Pembuatan Sistem Notifikasi

4.2.1.1 Arsitektur Notifikasi Firebase

Arsitektur notifikasi mengadopsi model *server-initiated push*, yaitu model di mana server berperan sebagai producer yang secara aktif menginisiasi pengiriman notifikasi ketika kondisi tertentu terpenuhi. *Back-end* yang berjalan pada raspberry pi 5 bertindak sebagai *producer* notifikasi yang mengirimkan pesan ketika model deteksi berhasil mengidentifikasi adanya penyakit pada sampel udang. Gambaran arsitektur lengkap sistem notifikasi disajikan pada Tabel 4.10 berikut

Tabel 4. 5 Komponen Arsitektur Notifikasi Firebase

Komponen	Peran	Keterangan
Back-end FastAPI (RPi 5)	*Producer* notifikasi	Menginisiasi pengiriman pesan FCM saat penyakit terdeteksi
Firebase Admin SDK	Lapisan abstraksi FCM	Memformat dan mengirimkan pesan ke server FCM Google
Server FCM Google	Infrastruktur pengiriman	Meneruskan notifikasi ke perangkat target berdasarkan token FCM
Firebase SDK	Penerimaan notifikasi	Menerima dan memproses notifikasi di sisi klien flutter
Firebase	Penyimpanan token	Menyimpan token FCM pengguna

Realtime DB	FCM	dan data deteksi secara persisten
Android / iOS OS	Tampilan notifikasi	Menampilkan notifikasi di status bar dan *notification drawer*

Berdasarkan Tabel 4.5, sistem notifikasi melibatkan enam komponen utama yang bekerja secara sinergis. Setiap komponen memiliki tanggung jawab yang spesifik dan terdefinisi dengan jelas dalam arsitektur sistem, sehingga kegagalan pada salah satu komponen dapat diidentifikasi dan ditangani secara terisolasi tanpa mengganggu komponen lainnya.

4.2.1.2 Alur Pengiriman Notifikasi End-to-End

Proses pengiriman notifikasi *push* dari sistem kepada *smartphone* melibatkan sembilan langkah yang berjalan secara berurutan dan melibatkan beberapa komponen yang berbeda. berikut ini menyajikan penjelasan rinci dari setiap langkah dalam alur pengiriman notifikasi

Tabel 4. 6 Alur End-to-End Pengiriman *Push Notification* FCM

NO	Proses	Protokol
1.	Aplikasi Flutter terinstal pada gawai pembudidaya dan meminta izin notifikasi dari pengguna	*Firebase* SDK
2.	Firebase SDK secara otomatis menghasilkan token FCM unik yang merepresentasikan perangkat tersebut	*Firebase Registration*
3.	Token FCM dikirimkan ke *back-end* melalui *endpoint* POST /register-token dan disimpan ke *firebase realtime database*	HTTPS REST API
4.	Prototipe mendeteksi adanya penyakit dan *back-end* memanggil fungsi send_fcm_notification()	Internal function call

NO	Proses	Protokol
5.	Firebase Admin SDK memformat pesan FCM dengan *payload* notifikasi visual dan data payload terstruktur	Firebase Admin SDK
6.	Pesan FCM yang telah diformat dikirimkan ke server FCM Google melalui koneksi HTTPS terenkripsi	HTTPS / TLS 1.3
7.	Server FCM Google meneruskan notifikasi ke perangkat target berdasarkan token FCM yang terdaftar	WebSocket / QUIC
8.	Sistem operasi Android atau iOS pada gawai pembudidaya menerima dan menampilkan notifikasi *push*	FCM SDK / APNs
9.	Aplikasi flutter memproses data *payload* yang diterima dan menavigasi pengguna ke layar	Flutter FirebaseMessaging

Pada Tabel 4.6, proses registrasi token pada langkah pertama hingga ketiga hanya dilakukan sekali saat aplikasi pertama kali dijalankan atau ketika token FCM mengalami pembaruan otomatis oleh *firebase* SDK. Token yang tersimpan di firebase *realtime database* kemudian digunakan secara berulang untuk setiap notifikasi berikutnya tanpa perlu melakukan registrasi ulang. Proses pengiriman notifikasi pada langkah keempat hingga kedelapan berjalan secara asinkron di latar belakang sebagai *background task* FastAPI, sehingga tidak menghambat pengiriman respon deteksi ke aplikasi flutter.

Waktu latensi rata-rata dari saat penyakit terdeteksi hingga notifikasi muncul di smartphone pembudidaya adalah sekitar 0,8 detik, yang dianggap cukup cepat untuk kebutuhan pemantauan kondisi tambak secara real-time. Latensi ini terdiri atas waktu pemrosesan inferensi (23–30 milidetik), waktu komunikasi HTTPS antara Raspberry Pi 5 dan server FCM Google (200–400 milidetik), waktu

penerusan notifikasi oleh infrastruktur FCM Google ke perangkat Android/iOS (100–300 milidetik), serta waktu rendering notifikasi oleh sistem operasi (50–100 milidetik). Implementasi fungsi send_fcm_notification() pada berkas firebase_service.py ditampilkan dalam blok kode berikut.

```
# firebase_service.py — Fungsi pengiriman notifikasi FCM
async def send_fcm_notification(fcm_token: str, record: dict):
    try:
        disease = record.get('disease', 'Tidak Diketahui')
        confidence = record.get('confidence', 0.0)
        pond_id = record.get('pond_id', 'Kolam-01')
        farm_name = record.get('farm_name', 'Tambak')
        mitigation = record.get('mitigation', [])
        # Bangun objek pesan FCM
        message = messaging.Message(
            notification=messaging.Notification(
                title=f'\u26a0\ufe0f PERINGATAN: {disease}
Terdeteksi!',
                body=f'{farm_name} > {pond_id} | {confidence*100:.1f}%
kepercayaan',
            ),
            data={
                'disease': disease,
                'confidence': str(confidence),
                'pond_id': pond_id,
                'farm_name': farm_name,
                'timestamp': record.get('timestamp',''),
                'is_sick': '1' if record.get('detected') else '0',
                'mitigation': '|'.join(mitigation),
            },
            token=fcm_token,
        )
        response = messaging.send(message)
        logger.info(f'FCM terkirim: {response}')
    except Exception as e:
        logger.error(f'Gagal kirim FCM: {e}')
```

Gambar 4. 8 Implementasi Fungsi send_fcm_notification() pada firebase_service.py

Pada Gambar 4.9, ditampilkan implementasi lengkap fungsi pengiriman notifikasi FCM. Blok *try-except* digunakan untuk memastikan bahwa kegagalan dalam pengiriman notifikasi tidak menyebabkan crash pada keseluruhan proses deteksi. Objek messaging terdiri atas dua bagian utama yakni *notification* yang berisi judul dan isi notifikasi yang akan ditampilkan secara visual di status bar gawai, serta kamus data yang berisi informasi terstruktur yang akan diteruskan ke aplikasi flutter untuk pemrosesan lebih lanjut. Bidang token menyimpan token

FCM perangkat tujuan yang telah diregistrasikan sebelumnya. Fungsi messaging.send() mengirimkan pesan ke server FCM google dan mengembalikan string ID pesan apabila pengiriman berhasil.

4.2.1.3 Format Payload Notifikasi FCM

Setiap notifikasi yang dikirimkan oleh sistem melalui FCM terdiri dari dua komponen *payload* yang memiliki fungsi berbeda namun saling melengkapi. Komponen pertama adalah *notification payload* yang bertanggung jawab untuk menampilkan notifikasi secara visual kepada pengguna melalui system notification drawer pada smartphone. Komponen kedua adalah data *payload* yang berisi informasi terstruktur yang diproses secara programatik oleh aplikasi Flutter untuk menentukan tindakan navigasi yang tepat.

Notification payload dirancang agar dapat langsung dipahami oleh pembudidaya tanpa perlu membuka aplikasi terlebih dahulu. Judul notifikasi memuat nama penyakit yang terdeteksi beserta simbol peringatan, sementara isi notifikasi memuat informasi singkat mengenai lokasi kolam, tingkat kepercayaan model, dan tindakan yang perlu segera dilakukan. Berikut adalah format payload notifikasi yang dikirimkan oleh firebase admin SDK python ke server FCM

```
// notification payload — ditampilkan sebagai notifikasi visual di
status bar
"notification": {
  "title": "\u26a0\ufe0f PERINGATAN: WSSV Terdeteksi!",
  "body":  "Tambak Utama > Kolam-01 | 87.4% kepercayaan | Segera panen
dini!"
}

// data payload — diproses secara programatik oleh NotificationService
Flutter
"data": {
  "disease":    "WSSV",
  "confidence": "0.8741",
  "pond_id":    "Kolam-01",
  "farm_name":  "Tambak Utama",
  "timestamp":  "2026-05-25T14:32:01",
  "is_sick":    "1",
  "mitigation": "KRITIS...|Pasang jaring...|Desinfeksi..."
}
```

Gambar 4. 9 Format Payload Notifikasi FCM

Pada Gambar 4.10, ditampilkan struktur lengkap payload notifikasi FCM yang dikirimkan oleh sistem. *Notification payload* terdiri atas dua bidang yakni *title* yang memuat judul notifikasi berupa nama penyakit dengan ikon peringatan, dan body yang memuat isi notifikasi berupa ringkasan informasi penting yang dapat langsung dibaca oleh pembudidaya tanpa membuka aplikasi.

Data *payload* memuat bidang informasi yang bersifat lebih teknis dan digunakan oleh kode aplikasi flutter untuk pengambilan keputusan. Bidang *disease* dan *confidence* memungkinkan aplikasi flutter untuk menentukan warna latar belakang kartu notifikasi dan ikon yang ditampilkan. Bidang pond_id dan farm_name digunakan untuk menavigasi pengguna secara otomatis ke layar pemantauan kolam yang sesuai saat notifikasi disentuh. Bidang is_sick bernilai '1'

apabila penyakit terdeteksi dan '0' apabila udang dalam kondisi sehat, digunakan sebagai penanda cepat tanpa perlu membandingkan nama penyakit. Bidang mitigation memuat daftar rekomendasi tindakan yang dipisahkan oleh karakter pipa (|) untuk kemudahan parsing di sisi klien menggunakan metode split().

4.2.1.4 Struktur *Firebase Realtime Database*

Firebase realtime database digunakan sebagai media penyimpanan data deteksi yang dapat diakses secara *real-time* dari berbagai perangkat. Struktur basis data dirancang secara hierarkis dengan tiga *node* utama, yaitu *detections* yang menyimpan riwayat deteksi per kolam, Pengguna yang menyimpan informasi dan token FCM pengguna terdaftar, serta *system* yang menyimpan status operasional sistem secara keseluruhan.


```
"detections": {
    "Kolam-01": {
      "-NxABC123def": {
        "disease":            "WSSV",
        "confidence":          0.8741,
        "timestamp":          "2026-05-25T14:32:01",
        "processing_time_ms": 23.4,
        "farm_name":          "Tambak Utama",
        "pond_id":            "Kolam-01",
        "server_timestamp":   1748176321000
      }
    }
  },
  "users": {
    "user_001": {
      "fcm_token":  "dWxxx...token...",
      "pond_ids":   ["Kolam-01", "Kolam-02"],
      "platform":   "android",
      "registered": "2026-05-25T10:00:00"
    }
  },
  "system": {
    "status": {
      "hailo_available": true,
      "camera_active":   true,
      "last_updated":    "2026-05-25T14:32:05"
    }
  }
}
```


Gambar 4. 10 Struktur Hierarkis *Firebase Realtime Database*

Pada gambar 4.11, ditampilkan struktur lengkap *firebase realtime database* yang digunakan oleh sistem. *Node detections* diorganisasikan berdasarkan

identifikasi kolam (pond_id) sebagai kunci tingkat pertama. Struktur hierarkis ini memungkinkan query data deteksi untuk kolam tertentu dengan efisien tanpa perlu memuat seluruh isi basis data. Kunci tingkat kedua berupa string yang dimulai dengan tanda minus, misalnya '-NxABC123def', merupakan kunci otomatis yang dihasilkan oleh Firebase secara kronologis dan unik secara global.

Setiap rekord deteksi menyimpan tujuh bidang data yang mencakup nama penyakit, *confidence*, waktu deteksi dalam format ISO 8601, durasi pemrosesan inferensi, nama tambak, identifikasi kolam , dan stempel waktu server dalam format *unix timestamp* milidetik. Bidang server_timestamp disimpan dalam format numerik milidetik unix untuk memudahkan operasi pengurutan dan pemfilteran berbasis rentang waktu dari sisi klien.

Node users menyimpan relasi antara identifikasi pengguna dengan token FCM dan daftar kolam yang dipantau, sehingga memungkinkan sistem untuk mengirimkan notifikasi hanya kepada pengguna yang memiliki kepentingan terhadap kolam yang bersangkutan. Kolom pond_ids merupakan larik yang memuat daftar identifikasi kolam yang dipantau oleh pengguna tersebut, sehingga sistem dapat melakukan pengiriman notifikasi yang tertarget hanya kepada pengguna yang terdaftar sebagai pemilik atau pengelola kolam yang sedang dipantau. *Node system* diperbarui secara berkala oleh server untuk mencerminkan kondisi terkini dari komponen-komponen *hardware* yang kritikal, yaitu akselerator hailo, modul kamera, dan koneksi *firebase*.

4.2.2 Pembuatan Sistem *Front End*

4.2.2.1 Struktur Sistem *Front End*

Struktur sistem *front end* diorganisasikan dalam direktori pola arsitektur *service layer* yang diadopsi. Setiap kategori berkas ditempatkan dalam direktori yang sesuai dengan perannya dalam sistem, sehingga memudahkan navigasi dan pemeliharaan kode. Hierarki lengkap dari struktur ditampilkan dalam blok kode berikut:

```
sdides/lib/
├── main.dart                          # Entry point +
├── firebase_options.dart              # Konfigurasi Firebase
├── screens/
│   ├── monitor_screen.dart           # Live kamera + deteksi
│   ├── penyakit_screen.dart          # Database informasi penyakit
│   ├── riwayat_screen.dart           # Riwayat deteksi + filter
│   ├── settings_screen.dart          # Konfigurasi URL server
│   └── disease_detail_screen.dart    # Halaman detail informasi
├── services/
│   ├── api_service.dart              # HTTP client + model data
│   └── notification_service.dart     # FCM handler + local
└── widgets/
    ├── app_header.dart               # Widget header aplikasi
    └── stat_card.dart                # Widget kartu statistik
```

Gambar 4. 11 Struktur Direktori Front End

Pada gambar 4.12, direktori terdiri atas empat lapisan utama. Lapisan pertama adalah berkas-berkas di akar direktori lib/, yaitu main.dart dan firebase_options.dart. Berkas main.dart merupakan *entry point* aplikasi yang menginisialisasi seluruh layanan yang diperlukan, termasuk *firebase* dan *notificationservice*, sebelum aplikasi mulai merender antarmuka pengguna. Berkas firebase_options.dart merupakan berkas yang dihasilkan secara otomatis oleh alat flutterfire CLI dan berisi konfigurasi spesifik *platform* (Android dan iOS) untuk proyek *firebase* yang terhubung.

Lapisan kedua adalah direktori screens/ yang memuat lima berkas layar yang masing-masing merepresentasikan satu halaman dalam navigasi aplikasi. Berkas monitor_screen.dart merupakan layar utama yang menampilkan aliran video langsung dan menyediakan fungsi deteksi penyakit. Berkas penyakit_screen.dart menampilkan basis data informasi tentang berbagai jenis penyakit udang yang dapat diakses secara offline. Berkas riwayat_screen.dart menampilkan riwayat seluruh hasil deteksi dengan fitur filter dan statistik. Berkas settings_screen.dart menyediakan antarmuka konfigurasi koneksi server dan pengaturan kolam. Berkas disease_detail_screen.dart menampilkan informasi lengkap dan panduan penanganan untuk setiap jenis penyakit.

Lapisan ketiga adalah direktori services/ yang memuat dua kelas service yang merupakan inti dari pola arsitektur *service layer*. Berkas api_service.dart mengimplementasikan seluruh komunikasi HTTP dengan server fastAPI, termasuk pengiriman gambar untuk deteksi dan pengambilan riwayat deteksi. Berkas notification_service.dart mengimplementasikan penanganan notifikasi FCM untuk tiga skenario yang berbeda, yaitu saat aplikasi berjalan di latar depan, latar belakang, dan saat aplikasi tidak berjalan. Lapisan keempat adalah direktori widgets/ yang memuat dua widget yang dapat digunakan ulang di berbagai layar untuk menjaga konsistensi tampilan.

Tabel 4.7 berikut menyajikan penjelasan lengkap mengenai fungsi dan tanggung jawab dari setiap berkas

Tabel 4. 7 Penjelasan Berkas Front End

Berkas	Kategori	Fungsi Utama
main.dart	Entry point	Inisialisasi Firebase, NotificationService; konfigurasi tema Material 3; penentuan rute navigasi
firebase_options.dart	Konfigurasi	Menyimpan API keys, project ID, app ID, messaging sender ID per platform
monitor_screen.dart	Screen	Live MJPEG stream; tombol deteksi; dialog hasil deteksi; statistik hari ini
penyakit_screen.dart	Screen	Daftar penyakit udang; navigasi ke detail penyakit; data statis offline
riwayat_screen.dart	Screen	Tabel riwayat deteksi; filter penyakit/waktu; auto-refresh via StreamBuilder

Berkas	Kategori	Fungsi Utama
settings_screen.dart	Screen	Input URL server; ping koneksi; simpan pond_id dan farm_name ke SharedPreferences
disease_detail_screen.dart	Screen	Detail penyakit: deskripsi, gejala, penyebab, panduan darurat
api_service.dart	Service	HTTP client; model DetectionResult; StreamController broadcast; fetchHistory()
notification_service.dart	Service	Inisialisasi FCM; penanganan foreground/background/terminated; navigasi otomatis
app_header.dart	Widget	Header bar dengan logo, judul, dan indikator status koneksi server
stat_card.dart	Widget	Kartu statistik dengan label, nilai, dan warna yang dapat dikonfigurasi

4.2.2.2 Alur Komunikasi Data

Alur komunikasi data antara aplikasi dan server menggambarkan proses pertukaran data mulai dari permintaan deteksi oleh pengguna hingga pembaruan informasi pada antarmuka aplikasi. Proses komunikasi melibatkan beberapa tahapan yang mencakup pengiriman data, pemrosesan hasil deteksi, penyimpanan data, hingga pengiriman informasi kembali ke aplikasi. Rincian tahapan komunikasi data ditunjukkan pada tabel 4.8.

Tabel 4. 8 Alur Komunikasi Data

No	Aksi	Kode
1.	Pengguna menekan tombol 'Start Detection' pada MonitorScreen	GestureDetector.onTap
2.	Kamera mengambil satu frame gambar melalui CameraController	_cameraController.takePicture()
3.	ApiService membungkus gambar dalam permintaan HTTP multipart	ApiService.detectDisease()
4.	Permintaan multipart dikirimkan ke endpoint POST /detect	http.MultipartRequest.send()
5.	Server FastAPI menjalankan inferensi model dan mengembalikan JSON	FastAPI /detect endpoint
6.	Aplikasi Flutter melakukan parsing respons JSON ke model Dart	DetectionResult.fromJson()
7.	Antarmuka pengguna diperbarui: dialog hasil dan badge status	setState() + showDialog()
8.	StreamController menyiarkan data terbaru ke RiwayatScreen	ApiService.detectionStream
9.	Notifikasi FCM muncul di status bar (apabila penyakit terdeteksi)	NotificationService (Firebase)

Berdasarkan tabel 4.8, langkah pertama hingga kedelapan berlangsung secara berurutan mulai dari proses permintaan deteksi hingga pembaruan informasi pada antarmuka aplikasi. Penggunaan *streamcontroller broadcast* memungkinkan data hasil deteksi diterima oleh beberapa komponen aplikasi melalui satu aliran data. Proses pengiriman notifikasi dilakukan secara terpisah dari alur utama

sehingga tidak mempengaruhi proses pembaruan tampilan aplikasi.

4.2.3 Pembuatan Sistem *BackEnd*

4.2.3.1 Struktur Modul *BackEnd*

Struktur direktori *backend* sistem diimplementasikan secara modular dengan memisahkan setiap komponen berdasarkan fungsi masing-masing. Pemisahan struktur ini dilakukan untuk mengelompokkan modul konfigurasi, layanan API, model data, dan penyimpanan sistem sehingga proses pengembangan dan pengelolaan kode lebih terorganisasi. Hierarki direktori *backend* beserta fungsi setiap berkas ditunjukkan sebagai berikut.

```
backend/
├── main.py                  # Entry point FastAPI
├── firebase_service.py      # Firebase Admin SDK wrapper
├── requirements.txt         # Daftar dependensi Python
├── models/                  # Direktori file model AI
│   └── sdides_yolov11.hef   # Model YOLOv11 terformat Hailo
└── firebase/
    └── serviceAccountKey.json  # Kredensial Firebase Admin SDK
```

Gambar 4. 12 Struktur Direktori Back-End

Pada gambar 4.13, ditampilkan hierarki direktori backend yang terdiri atas beberapa komponen utama dalam pengelolaan sistem. Berkas *main.py* digunakan sebagai modul utama aplikasi fastAPI yang menangani inisialisasi sistem, konfigurasi layanan, serta pengaturan proses utama pada *backend*. Berkas *firebase_service.py* digunakan sebagai modul penghubung antara *backend* dengan layanan *firebase*. Modul menangani proses komunikasi dengan *firebase admin* SDK yang meliputi penyimpanan data ke *firebase realtime database* dan pengiriman notifikasi melalui *firebase cloud messaging* (FCM). Selanjutnya, berkas *requirements.txt* berisi daftar dependensi python yang digunakan untuk menjalankan seluruh layanan *backend*. Direktori *models* digunakan untuk menyimpan berkas model YOLOv11 dalam format *hailo executable format* (.hef) yang telah dikompilasi untuk dijalankan pada akselerator hailo-8L. Sementara itu, direktori *firebase* menyimpan berkas konfigurasi autentikasi yang digunakan untuk menghubungkan backend dengan layanan *firebase*.

4.2.3.2 Daftar *Endpoint* Back End

Sistem *endpoint* pada back end mengimplementasikan tujuh *endpoint* REST API yang digunakan untuk mendukung proses pertukaran data antara backend dan aplikasi. Setiap *endpoint* memiliki fungsi dan parameter yang disesuaikan dengan kebutuhan sistem, mulai dari pengiriman hasil deteksi, pengambilan data, hingga pengelolaan informasi pada aplikasi. Seluruh *endpoint* didefinisikan pada berkas *main.py* dan berjalan menggunakan protokol HTTP. Rincian implementasi setiap endpoint beserta fungsi dan parameter yang digunakan ditunjukkan pada Tabel 4.7.

Tabel 4. 9 Daftar Endpoint REST API

Endpoint	Metode	Fungsi	Parameter
POST /detect	POST	Deteksi penyakit dari gambar — endpoint utama sistem	image (file), fcm_token?, pond_id?, farm_name?
GET /stream	GET	MJPEG live stream kamera + overlay bounding box real-time	— (tidak ada)
GET /history	GET	Riwayat deteksi dengan filter waktu dan jenis penyakit	limit, hours, disease_filter?
GET /status	GET	Status sistem: hardware (Hailo, Firebase, kamera) + statistik	— (tidak ada)
POST /notify	POST	Kirim push notifikasi FCM manual (testing/debugging)	fcm_token, disease, confidence, pond_id, farm_name
POST /register-token	POST	Daftarkan token FCM pengguna ke database Firebase	user_id, token, pond_ids[], platform

GET /docs	GET	Swagger UI dokumentasi API otomatis	— (browser)

Berdasarkan tabel 4.9, *endpoint* POST */detect* digunakan sebagai bagian utama dalam proses deteksi objek pada sistem. *Endpoint* menerima data citra dari aplikasi flutter, menjalankan proses inferensi menggunakan model YOLOv11, kemudian mengembalikan hasil deteksi berupa informasi kelas, nilai *confidence*, dan rekomendasi tindakan dalam format JSON.

Endpoint GET */stream* digunakan untuk menyediakan tampilan aliran video secara langsung dari kamera raspberry pi 5 dengan penambahan visualisasi *bounding box* pada objek hasil deteksi. Selanjutnya, endpoint GET */history* digunakan untuk mengambil data riwayat deteksi yang tersimpan pada *firebase realtime database* dengan dukungan parameter penyaringan sesuai kebutuhan aplikasi. *Endpoint* GET */status* digunakan untuk menampilkan informasi kondisi sistem yang mencakup status kamera, koneksi *firebase*, dan ketersediaan akselerator Hailo-8L. *Endpoint* POST */notify* digunakan untuk melakukan pengujian pengiriman notifikasi FCM secara manual, sedangkan endpoint POST */register-token* digunakan untuk menyimpan token perangkat pengguna pada *firebase realtime database*. *Endpoint* GET */docs* menyediakan dokumentasi API berbasis swagger UI yang dibuat secara otomatis oleh FastAPI.

Implementasi *endpoint* POST */detect* sebagai salah satu layanan utama backend ditampilkan pada blok kode berikut. Kode tersebut menunjukkan struktur pendefinisian *endpoint* FastAPI beserta parameter masukan dan proses pengolahan data yang dilakukan.

4.2.3.3 *Format Request-Response Endpoint*

Endpoint /detect menerima permintaan HTTP POST dengan tipe konten *multipart/form-data*. Pengiriman data dilakukan dalam bentuk berkas gambar beserta data tambahan dari aplikasi flutter dalam satu permintaan menuju *backend*. Citra yang diterima selanjutnya diproses sebagai masukan pada tahap inferensi model YOLOv11 untuk menghasilkan informasi deteksi. Struktur permintaan yang dikirimkan oleh aplikasi flutter ke server ditampilkan sebagai berikut.

```
POST /detect HTTP/1.1
Host: 192.168.1.100:5000
Content-Type: multipart/form-data; boundary=boundary123

--boundary123
Content-Disposition: form-data; name="image"; filename="udang.jpg"
Content-Type: image/jpeg
[binary JPEG data]

--boundary123
Content-Disposition: form-data; name="fcm_token"
dWxxxxxxxxxxx_FCM_TOKEN_DISINI_xxxxxxxxxx
```

Gambar 4. 13 Format HTTP Request Multipart ke Endpoint

Pada gambar 4.14, ditampilkan format permintaan HTTP POST yang dikirimkan oleh aplikasi flutter ke endpoint */detect*. Permintaan tersebut memuat data citra yang akan digunakan sebagai masukan pada proses inferensi, serta informasi tambahan berupa *FCM token* dan identitas kolam (*pond_id*). Data citra digunakan untuk proses deteksi objek, sedangkan *FCM token* dan *pond_id* digunakan untuk mendukung proses pengiriman notifikasi dan penyimpanan hasil deteksi pada sistem.

Setelah proses inferensi selesai dilakukan, *backend* mengembalikan respon dalam format JSON yang memuat informasi hasil deteksi. Data yang dikembalikan mencakup hasil klasifikasi, nilai *confidence*, serta informasi lain yang diperlukan untuk ditampilkan pada aplikasi Flutter. Respon yang diterima aplikasi ditunjukkan sebagai berikut.

```
{
  "detection_id":  "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "timestamp":     "2026-05-25T14:32:01.123456",
  "detected":      true,
  "disease":       "WSSV",
  "confidence":    0.8741,
  "all_detections": [
    { "class_id":3, "class_name":"WSSV",
      "confidence":0.8741, "bbox":[120, 80, 200, 160] }
  ],
  "mitigation": [
    "KRITIS: Segera lakukan panen dini jika berat udang memungkinkan",
    "Pasang jaring penutup untuk mencegah masuknya vektor penular",
    "Lakukan desinfeksi air dengan klorin 30 ppm sebelum dibuang"
  ],
  "processing_time_ms": 23.4
}
```


Gambar 4. 14 Format *JSON Respons Endpoint*

Pada gambar 4.14, ditampilkan struktur respons JSON yang dikembalikan oleh server fastAPI setelah proses inferensi selesai dilakukan. Respons tersebut memuat informasi hasil deteksi yang selanjutnya digunakan oleh aplikasi flutter untuk menampilkan hasil klasifikasi beserta informasi pendukung pada antarmuka.

Data hasil deteksi diawali dengan bidang *detection_id* yang digunakan sebagai identitas pada setiap rekaman deteksi yang tersimpan dalam sistem. Bidang *timestamp* mencatat waktu proses deteksi dilakukan, sedangkan bidang *detected* menunjukkan status keberadaan objek penyakit pada citra yang dianalisis. Hasil klasifikasi model disimpan pada bidang *disease* berupa kelas penyakit yang terdeteksi atau kondisi *Sehat*, dengan tingkat keyakinan model terhadap hasil tersebut disimpan pada bidang *confidence*.

Selain hasil klasifikasi utama, respons juga memuat bidang *all_detections* yang berisi seluruh objek hasil deteksi pada citra, meliputi informasi kelas, nilai kepercayaan, serta koordinat *bounding box*. Data tersebut digunakan oleh aplikasi untuk menampilkan posisi objek terdeteksi pada tampilan citra. Bidang *mitigation* berisi rekomendasi tindakan berdasarkan hasil klasifikasi penyakit, sedangkan

bidang *processing_time_ms* mencatat waktu yang diperlukan sistem sejak citra diterima hingga hasil deteksi dikirimkan kembali.

4.3 Pembuatan Dataset dan Model CNN

4.3.1 Proses Anotasi dengan Roboflow

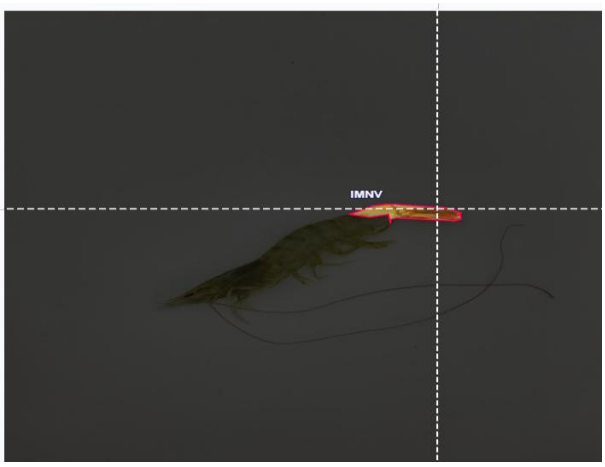
Proses anotasi dilakukan secara manual dengan berpedoman pada karakteristik klinis spesifik setiap kelas penyakit yang telah dikonfirmasi berdasarkan literatur ilmiah terkait patologi udang. Dataset citra yang digunakan pada tahap anotasi diperoleh dari dua sumber utama. Sumber pertama berasal dari proses akuisisi langsung menggunakan prototipe yang ditempatkan pada akuarium laboratorium dengan objek udang vaname berusia 40 hari. Proses pengambilan citra dilakukan menggunakan kamera HQ IMX477 dengan resolusi 1920×1080 piksel.

Sumber kedua diperoleh dari dokumentasi citra gejala klinis penyakit udang vaname pada literatur ilmiah yang telah melalui proses validasi diagnostik. Penggabungan kedua sumber data tersebut menghasilkan total 3.000 citra yang kemudian dikelompokkan ke dalam empat kelas berdasarkan kondisi udang yang diamati. Distribusi jumlah citra pada setiap kelas ditampilkan pada Tabel 4.10.

Tabel 4. 10 Distribusi Dataset Citra per Kelas

No.	Kelas / Kondisi Udang	Jumlah Citra	Persentase (%)
1	Sehat	870 citra	29,0%
2	IMNV	780 citra	26,0%
3	WSSV	750 citra	25,0%
4	TSV	600 citra	20,0%
Total		3.000 citra	100%

Penentuan area anotasi dilakukan berdasarkan karakteristik visual yang membedakan masing-masing kelas sehingga label yang dihasilkan dapat merepresentasikan kondisi objek pada citra. Pada kelas IMNV, anotasi dilakukan berdasarkan perubahan warna abnormal pada jaringan otot abdomen berupa area putih opak hingga kemerahan. Area tersebut digunakan sebagai acuan dalam menentukan batas *bounding box* pada citra. Dokumentasi hasil anotasi kelas IMNV ditunjukkan pada Gambar 4.16.



Gambar 4. 15 Area Anotasi pada IMNV

Berdasarkan Gambar 4.16, area *bounding box* ditempatkan pada bagian tubuh yang menunjukkan gejala nekrosis secara visual. Apabila perubahan hanya terlihat pada bagian tertentu, batas anotasi disesuaikan dengan area gejala yang tampak untuk menjaga ketepatan label dataset.

Pada kelas WSSV, anotasi dilakukan berdasarkan keberadaan bintik putih pada permukaan tubuh udang, terutama pada bagian karapas. Dokumentasi hasil anotasi kelas WSSV ditunjukkan pada Gambar 4.17.

Gambar 4. 16 Area Anotasi pada WSSV

Berdasarkan hasil anotasi pada Gambar 4.17, area *bounding box* mencakup bagian tubuh udang yang menunjukkan karakteristik gejala penyakit. Penentuan batas anotasi dilakukan berdasarkan area objek yang terlihat jelas pada citra karena gejala WSSV dapat muncul pada beberapa bagian permukaan tubuh. Kondisi TSV,

penentuan anotasi mengacu pada perubahan warna kemerahan pada tubuh udang, terutama bagian ekor dan anggota gerak. Dokumentasi hasil anotasi kelas TSV ditunjukkan pada gambar 4.18.

Gambar 4. 17 Area Anotasi pada TSV

Berdasarkan gambar 4.18, *bounding box* ditempatkan pada area tubuh yang mengalami perubahan warna maupun gejala nekrosis. Penyesuaian area anotasi dilakukan berdasarkan bagian yang menunjukkan perubahan visual akibat infeksi untuk mempertahankan konsistensi label pada dataset. Pada kelas sehat, proses anotasi dilakukan berdasarkan kondisi visual udang tanpa menunjukkan adanya gejala klinis penyakit. Area *bounding box* ditempatkan pada seluruh bagian tubuh udang sebagai representasi objek normal yang digunakan sebagai pembanding terhadap kelas yang mengalami infeksi. Dokumentasi hasil anotasi kelas Sehat ditunjukkan pada gambar 4.19.

Gambar 4. 18 Area Anotasi pada Kategori Sehat

Berdasarkan gambar 4.19, kelas sehat mencakup keseluruhan tubuh udang

dengan memastikan objek terlihat secara utuh pada citra. Data keleas sehat digunakan untuk membantu model mengenali karakteristik visual udang normal sehingga dapat membedakan kondisi sehat dengan udang yang menunjukkan gejala penyakit. Hasil akhir proses anotasi menghasilkan total 4.553 *bounding box* dari 3.000 citra dataset. Distribusi jumlah *bounding box* dan rata-rata objek pada setiap kelas ditampilkan pada tabel 4.11.

Tabel 4. 11 Distribusi Anotasi Bounding Box per Kelas

No.	Kelas Penyakit	Jumlah Bounding Box	Rata-rata Objek per Citra
1	Sehat	812	1,04
2	IMNV	896	1,24
3	TSV	1.052	1,40
4	WSSV	1.793	2,39
	Total	4.553	1,52

Berdasarkan tabel 4.11, kelas WSSV memiliki jumlah *bounding box* tertinggi, yaitu sebanyak 1.793 anotasi dengan rata-rata 2,39 objek per citra. Sementara itu, kelas Sehat memiliki jumlah anotasi terendah, yaitu sebanyak 812 *bounding box*. Perbedaan jumlah anotasi pada setiap kelas dipengaruhi oleh variasi jumlah objek yang muncul dalam satu citra serta karakteristik visual masing-masing kondisi udang.

Dataset yang telah melalui proses anotasi selanjutnya dipersiapkan untuk tahap pelatihan model melalui penerapan teknik augmentasi citra. Teknik augmentasi diterapkan menggunakan *pipeline* augmentasi roboflow yang dieksekusi secara otomatis pada saat ekspor dataset. Setelah seluruh augmentasi diterapkan, jumlah data *training* meningkat dari 2.100 citra menjadi 11.430 citra. Sementara itu, subset validasi dan pengujian tidak mengalami proses augmentasi sehingga tetap mempertahankan karakteristik data asli. Distribusi dataset setelah proses pembagian dan augmentasi ditampilkan pada tabel 4.12.

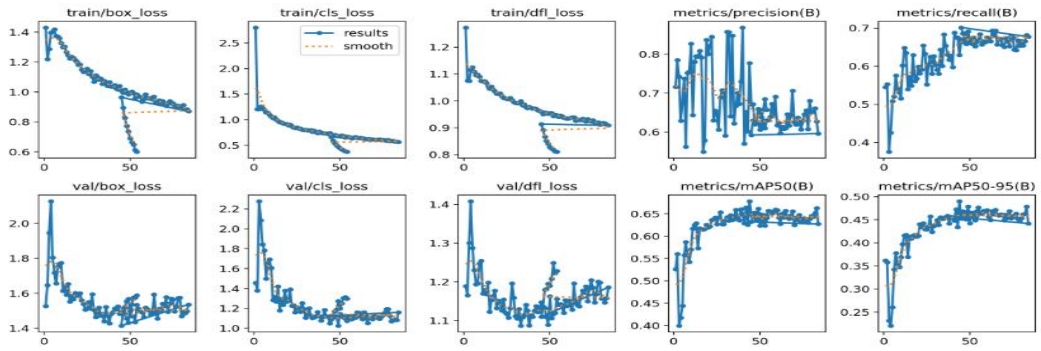
Tabel 4. 12 Pembagian Dataset Setelah Augmentasi

No.	Subset	Jumlah Citra Asli	Jumlah Citra Pasca-Augmentasi	Proporsi (%)
1	Training Set	2.100 citra	11.430 citra	75,5%
2	Validation Set	678 citra	678 citra tidak diaugmentasi	22,6%
3	Testing Set	222 citra	222 citra tidak diaugmentasi	7,4%
	Total	3.000 citra asli	12.330 citra total	100%

Berdasarkan tabel 4.12, proses augmentasi menyebabkan jumlah data *training* meningkat secara signifikan dari 2.100 citra menjadi 11.430 citra. Sementara itu, data validasi dan pengujian tidak mengalami augmentasi sehingga tetap merepresentasikan kondisi data asli. Pembagian dataset tersebut digunakan sebagai dasar pada proses pelatihan, validasi, dan pengujian model.

4.3.2 Pelatihan Model CNN

Selama proses pelatihan, sistem secara otomatis mencatat perkembangan nilai *loss* dan metrik evaluasi pada setiap *epoch*. Parameter yang diamati meliputi *train box loss*, *train classification loss*, *train distribution focal loss (DFL)*, *validation loss*, *precision*, *recall*, *mAP@50*, dan *mAP@50-95*.Hasil pemantauan selama proses pelatihan ditampilkan pada gambar 4.20.



Gambar 4. 19 Kurva Loss dan Metrik Evaluasi

Berdasarkan gambar 4.20, nilai *train box loss*, *train classification loss*, dan *train DFL loss* mengalami penurunan secara bertahap seiring bertambahnya *epoch* pelatihan. Penurunan pada *validation loss* menunjukkan model mampu

menyesuaikan parameter terhadap karakteristik dataset yang digunakan. Nilai *precision*, *recall*, *mAP@50*, dan *mAP@50-95* mengalami peningkatan hingga mencapai kondisi yang relatif stabil pada *epoch* akhir. Nilai *mAP@50* mencapai sekitar 0,65, sedangkan *mAP@50-95* mencapai sekitar 0,45 pada akhir pelatihan. Hasil evaluasi model ditampilkan pada gambar 4.21

Gambar 4. 20 Hasil Evaluasi Model

Berdasarkan gambar 4.21, model memperoleh nilai *mAP@50* sebesar 67,8%, *precision* sebesar 77,6%, *recall* sebesar 63,0%, dan *F1-score* sebesar 69,5%. Hasil tersebut menunjukkan model telah mampu melakukan deteksi dan klasifikasi penyakit udang pada dataset yang digunakan.

4.3.3 Konversi Model ke Format Hailo

Tahap awal konversi dilakukan dengan mengekspor model *best.pt* hasil pelatihan ke format ONNX menggunakan utilitas ekspor dari Ultralytics YOLOv11. Proses ekspor dilakukan dengan mengaktifkan parameter *simplify=True* untuk menyederhanakan grafik komputasi model sehingga lebih kompatibel dengan proses kompilasi pada Hailo DFC. Hasil ekspor berupa berkas model dengan format *.onnx* yang selanjutnya digunakan sebagai masukan pada tahap optimasi dan kompilasi model ke format *.hef*. Implementasi proses ekspor model YOLOv11n ke format ONNX ditunjukkan pada Gambar 4.22.

```
# ============================================
# EXPORT MODEL TO ONNX
# ============================================
print("\n" + "="*70)
print("📦 EXPORTING MODEL TO ONNX FORMAT")
print("="*70 + "\n")

try:
    onnx_path = best_model.export(format='onnx', simplify=True)
    print(f"☑ ONNX export successful: {onnx_path}\n")
except Exception as e:
    print(f"⚠  ONNX export failed: {e}\n")
    onnx_path = None
```

Gambar 4. 21 Koversi Fotmat ONNX\

Proses selanjutnya adalah kompilasi menggunakan *Hailo Dataflow Compiler* (DFC). Proses kompilasi membutuhkan data kalibrasi yang diambil dari subset dataset pelatihan. Data tersebut digunakan untuk menentukan parameter kuantisasi sehingga perubahan representasi model dari FP32 menjadi INT8 dapat dilakukan dengan tetap mempertahankan performa deteksi. Jumlah kelas pada proses kompilasi disesuaikan dengan dataset yang terdiri dari empat kelas, yaitu Sehat, IMNV, TSV, dan WSSV. Implementasi proses konversi model ONNX menjadi format HEF ditunjukkan pada gambar 4.23.

```
hailomz compile yolov11n \
--ckpt=best.onnx \
--hw-arch hailo8l \
--calib-path train/images \
--classes 4 \
--performance
```

Gambar 4. 22 Konversi ONNX ke hef

Berdasarkan gambar 4.23, proses kompilasi dilakukan dengan memasukkan model *best.onnx* sebagai input dan menentukan perangkat target menggunakan parameter *hailo8l*. Parameter *--calib-path* menunjuk lokasi citra kalibrasi yang digunakan selama proses optimasi, sedangkan parameter *--classes* disesuaikan dengan jumlah kelas deteksi pada model. Hasil akhir proses kompilasi berupa berkas *sdides_yolov11.hef* yang selanjutnya digunakan sebagai model inferensi pada raspberry pi 5