EVALUASI KINERJA EMPAT GENERASI YOLO DALAM DETEKSI *ACID-FAST BACILLI* PADA CITRA APUSAN MIKROSKOPIS *TUBERCULOSIS*



Disusun Oleh:

Nama : Raisha Alma Sahara

NIM : 22523110

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2026

HALAMAN PENGESAHAN DOSEN PEMBIMBING

EVALUASI KINERJA EMPAT GENERASI YOLO DALAM DETEKSI *ACID-FAST BACILLI* PADA CITRA APUSAN MIKROSKOPIS *TUBERCULOSIS*

TUGAS AKHIR

Disusun Oleh:

Nama : Raisha Alma Sahara

NIM : 22523110

Yogyakarta, 27 Juli 2026

Pembimbing,

(Arrie Kurniawardhani, S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

EVALUASI KINERJA EMPAT GENERASI YOLO DALAM DETEKSI *ACID-FAST BACILLI* PADA CITRA APUSAN MIKROSKOPIS *TUBERCULOSIS*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 27 Agustus 2026

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom. _____

Anggota 1

Dr. Hendrik, S.T., M.Eng. _____

Anggota 2

Rahadian Kurniawan, S.Kom., M.Kom. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Ir. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Raisha Alma Sahara

NIM : 22523110

Tugas akhir dengan judul:

EVALUASI KINERJA EMPAT GENERASI YOLO DALAM DETEKSI *ACID-FAST BACILLI* PADA CITRA APUSAN MIKROSKOPIS *TUBERCULOSIS*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 4 Juli 2026



(Raisha Alma Sahara)

HALAMAN PERSEMBAHAN

Bismillah,

Begitu besar rasa syukur yang saya panjatkan kepada Allah SWT atas rahmat dan karunia-Nya berupa kesehatan, kemudahan, dan kelancaran, sehingga penulis dapat menyelesaikan tugas akhir ini.

Terima kasih yang sebesar-besarnya saya ucapkan untuk diri saya sendiri atas kerja keras dan semangat yang cukup pasang surut dalam menyusun tugas akhir ini. Penulis juga sangat berterima kasih kepada semua pihak yang telah membantu, mendoakan, dan mendukung penulis selama proses ini berlangsung. Semoga Allah SWT membalas segala kebaikan yang telah diberikan kepada penulis dengan pahala yang berlipat ganda.

HALAMAN MOTO

“Jika bukan karena Allah yang memampukan, aku mungkin sudah lama menyerah.”

“*Long story short, I survived.*”
-Taylor Swift

KATA PENGANTAR

Puji dan syukur penulis panjatkan ke hadirat Allah SWT, karena atas rahmat, karunia, dan hidayah-Nya, laporan tugas akhir ini dapat diselesaikan dengan baik. Tugas akhir ini disusun untuk memenuhi salah satu syarat dalam memperoleh gelar Sarjana pada Program Studi Informatika.

Penulis menyadari bahwa dalam proses pelaksanaan penelitian hingga penyusunan laporan ini, penulis tidak luput dari berbagai hambatan dan kesulitan non-ilmiah, baik dari keterbatasan waktu, tenaga, maupun penyesuaian kondisi di lainnya. Namun, berkat bantuan, bimbingan, petunjuk, dan dukungan dari berbagai pihak, segala kendala tersebut dapat diatasi dengan baik.

Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa hormat dan terima kasih yang sebesar-besarnya kepada:

1. Prof. Dr. Ir. Hari Purnomo, M.T., IPU., ASEAN Eng., selaku Rektor Universitas Islam Indonesia.
2. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D., selaku Ketua Program Studi Program Sarjana Informatika serta seluruh dosen dan staf pengajar yang telah memberikan bekal ilmu pengetahuan dan bantuan administratif selama penulis menempuh masa perkuliahan.
3. Ibu Arrie Kurniawardhani, S.Si., M.Kom., selaku Dosen Pembimbing yang telah dengan sabar memberikan bimbingan, arahan, waktu, dan ilmu yang sangat berharga selama proses penyusunan tugas akhir ini.
4. Ibu Izzati Muhimmah S.T., M.Sc., Ph.D., selaku Dosen Pembimbing Akademik yang senantiasa memberikan arahan, nasihat, serta dukungan kepada penulis dari awal masa perkuliahan hingga masa penyelesaian studi ini.
5. Almarhum Ayah tercinta, meskipun Ayah telah lebih dahulu berpulang, kenangan, kasih sayang, serta segala pengorbanan yang diberikan semasa hidup akan selalu menjadi inspirasi dan kekuatan bagi penulis dalam menghadapi setiap proses kehidupan, termasuk dalam menyelesaikan skripsi ini.
6. Mama tercinta dan adik-adik, yang senantiasa menjadi sumber kekuatan, memberikan doa yang tiada putus, kasih sayang, dan dukungan yang luar biasa kepada penulis.
7. Tante Pia yang telah tulus membantu membiayai studi penulis, sehingga penulis dapat menyelesaikan pendidikan hingga tahap ini.
8. Keluarga besar yang selalu memberikan semangat dan dukungan dalam setiap langkah perjalanan akademik penulis.

9. Sahabat terdekat, Ailsa, Cahya, dan Mayta, atas kebersamaan, hiburan, dan tempat ternyaman yang selalu bersedia mendengarkan cerita-cerita penulis sejak kecil.
10. Ibel selaku kucing kesayangan yang selalu setia menemani penulis di meja belajar, memberikan hiburan, dan menjadi pereda stres yang efektif selama malam-malam panjang penyusunan laporan ini.
11. Grup Bogel (Dela, Ipeh, Kina, Wawa, Keca, Keye, Wiwi) dan Info Labris (Mifta, Nisya, Belina, Diena, Fath, Kasyiful) beberapa teman penulis yang selalu ada untuk mendampingi, memberikan dukungan, dan berbagi suka duka dari awal masa perkuliahan hingga akhir perjuangan ini.
12. Seseorang yang spesial, terima kasih secara khusus penulis sampaikan atas kehadiran, perhatian, kesabaran, dan dukungan emosional yang luar biasa. Terima kasih telah menjadi sandaran, pendengar yang baik, dan pemberi semangat di kala penulis menempuh perkuliahan dari awal hingga akhir.
13. Teman-teman seperjuangan Prodi Informatika, terima kasih atas solidaritas, diskusi, dan memori berharga yang tercipta selama masa studi kita.

Penulis menyadari bahwa laporan tugas akhir ini masih jauh dari kesempurnaan. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan. Akhir kata, penulis berharap semoga tugas akhir ini dapat memberikan manfaat bagi perkembangan ilmu pengetahuan, khususnya di bidang informatika, serta bagi pembaca sekalian.

Yogyakarta, 8 Juni 2026



(Raisha Alma Sahara)

SARI

Tuberkulosis (TB) merupakan salah satu penyakit menular yang masih menjadi permasalahan kesehatan global, termasuk di Indonesia. Diagnosis TB secara mikroskopis menggunakan cairan sputum masih banyak digunakan karena relatif sederhana dan ekonomis. Namun, proses identifikasi bakteri *Mycobacterium tuberculosis* secara manual membutuhkan ketelitian tinggi, waktu yang cukup lama, serta rentan terhadap subjektivitas pengamat. Perkembangan teknologi kecerdasan buatan, khususnya pada bidang *computer vision*, membuka peluang untuk mengotomatisasi proses deteksi bakteri TB melalui pemanfaatan algoritma deteksi objek berbasis *deep learning*. Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk membandingkan performa model YOLO dalam mendeteksi bakteri *Mycobacterium tuberculosis* pada citra mikroskopis apusan sputum serta menentukan model yang paling optimal untuk implementasi pada lingkungan *edge computing*.

Dataset yang digunakan merupakan Tuberculosis Image Dataset yang diperoleh dari Kaggle yang terdiri atas 1.265 citra dengan resolusi asli 1.632 x 1.224 piksel. Penelitian ini menggunakan delapan model yang merepresentasikan varian *nano* dan *small* dari masing-masing arsitektur, yaitu YOLOv8n, YOLOv8s, YOLOv10n, YOLOv10s, YOLO11n, YOLO11s, YOLO26n, dan YOLO26s. Seluruh model dilatih menggunakan platform Google Colab dengan konfigurasi pelatihan yang sama untuk menjamin keadilan proses komparasi. Evaluasi performa dilakukan menggunakan metrik mAP@50, mAP@50-95, *Precision*, *Recall*, dan *F1-Score*. Selain itu, model terbaik dikonversi ke format TensorFlow Lite (TFLite) dengan kuantisasi INT8 untuk mengevaluasi efisiensi komputasi berdasarkan ukuran model dan waktu inferensi.

Hasil penelitian menunjukkan bahwa seluruh model mampu mendeteksi bakteri *Mycobacterium tuberculosis* dengan tingkat performa yang baik. YOLO11n memperoleh performa terbaik dengan mAP@50 sebesar 0,834, mAP@50-95 sebesar 0,407, *Precision* sebesar 0,781, *Recall* sebesar 0,771, dan *F1-Score* sebesar 0,776 setelah konversi ke format TFLite INT8. Dari sisi efisiensi, YOLO11n memiliki ukuran model sebesar 2,85 MB dan waktu inferensi rata-rata sebesar 119,05 ms, sehingga tetap kompetitif dibandingkan model lainnya. Berdasarkan kombinasi performa deteksi dan efisiensi komputasi, pada penelitian ini YOLO11n ditetapkan sebagai model yang paling optimal untuk implementasi sistem deteksi bakteri *Mycobacterium tuberculosis* pada perangkat *edge computing* dengan sumber daya terbatas.

Kata kunci: YOLO, Mycobacterium tuberculosis, Deteksi Objek, TensorFlow Lite, Edge Computing.

GLOSARIUM

Anotasi	pemberian label pada objek dalam citra untuk menghasilkan data yang dapat digunakan dalam pelatihan model deteksi objek.
Augmentasi	proses untuk menambah atau meningkatkan variasi data pelatihan dengan tujuan untuk memperbanyak data latih sehingga memperkuat
Batch size	ukuran banyaknya data yang dimasukkan ke dalam model saat proses pelatihan.
Bounding box	kotak pembatas yang digunakan untuk menunjukkan lokasi objek pada citra.
Computer Vision	cabang kecerdasan buatan yang memungkinkan komputer memperoleh, memproses, dan memahami informasi dari citra atau video.
Deep Learning	metode pembelajaran mesin yang menggunakan jaringan saraf tiruan berlapis banyak untuk mempelajari pola dari data.
Edge Computing	komputasi yang memproses data pada perangkat lokal atau dekat sumber data sehingga mengurangi ketergantungan pada server pusat.
Epoch	siklus pelatihan ketika selutuh data pelatihan telah diproses oleh model.
Hiperparameter	parameter yang diatur sebelum proses pelatihan model dan akan berdampak pada kinerja proses pembelajaran model.
Ground truth	data label atau anotasi yang dianggap sebagai kebenaran acuan, digunakan untuk mengevaluasi hasil prediksi model dengan membandingkan antara output model dan kondisi aktual yang telah diketahui.
Tuberkulosis (TB)	penyakit infeksi yang disebabkan oleh bakteri *Mycobacterium tuberculosis*.

DAFTAR ISI

HALAMAN JUDUL i
HALAMAN PENGESAHAN DOSEN PEMBIMBING ii
HALAMAN PENGESAHAN DOSEN PENGUJI iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR iv
HALAMAN PERSEMBAHAN v
HALAMAN MOTO vi
KATA PENGANTAR vii
SARI ix
GLOSARIUM x
DAFTAR ISI xi
DAFTAR TABEL xiii
DAFTAR GAMBAR xiv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 3
1.4 Tujuan Penelitian 3
1.5 Manfaat Penelitian 3
1.6 Metode Penelitian 4
1.7 Sistematika Penelitian 4
BAB II LANDASAN TEORI 6
2.1 Mycobacterium Tuberculosis 6
2.2 Kajian Literatur Penelitian Sebelumnya 7
2.3 Pendekatan Deteksi Objek Sebelum Arsitektur YOLO 11
2.4 Evolusi Arsitektur YOLO 13
2.5 Tantangan Deteksi Objek Kecil dan Implikasi terhadap Pemilihan Arsitektur YOLO 17
2.6 Kuantisasi Model dan TensorFlow Lite 18
2.7 Metrik Evaluasi 20
2.7.1 *Confusion Matrix* dan Intersection over Union 21
2.7.2 *Precision*, *Recall*, dan *F1-Score* 22
2.7.3 Average *Precision* dan Mean Average *Precision* 23
2.7.4 Metrik Efisiensi Komputasi dan Relevansinya 24
BAB III METODOLOGI PENELITIAN 26
3.1 Persiapan *Dataset* 27
3.2 Konversi Anotasi *Dataset* 27
3.3 Pembagian *Dataset* 28
3.4 Konfigurasi Augmentasi dan Hiperparameter 29
3.4.1 Konfigurasi Augmentasi 29
3.4.2 Konfigurasi Hiperparameter 30
3.5 Konfigurasi Pelatihan dan Validasi Model 31
3.6 Pengujian Model 32
3.7 Optimasi Model ke Format TensorFlow Lite 32
3.8 Pengujian Model TFLite pada *Dataset* Uji 33
3.9 Metode Perhitungan Waktu Inferensi 34
3.10 Metode Evaluasi dan Analisis Komparatif 35
BAB IV HASIL DAN PEMBAHASAN 37

4.1 Hasil Persiapan *Dataset* 38
4.2 Hasil Konversi Anotasi *Dataset* 39
4.3 Hasil Pembagian *Dataset* 41
4.4 Hasil Augmentasi dan Pengaturan Hiperparameter 43
4.5 Hasil Pelatihan dan Validasi Model 44
4.6 Hasil Pengujian Model 51
4.7 Hasil Optimasi Model ke Format TensorFlow Lite 58
4.8 Hasil Pengujian Model TensorFlow Lite pada *Dataset* Uji 60
4.9 Hasil Perhitungan Waktu Inferensi 63
4.10 Hasil Evaluasi dan Analisis Komparatif 67
BAB V KESIMPULAN DAN SARAN 75
5.1 Kesimpulan 75
5.2 Saran 76
DAFTAR PUSTAKA 78
LAMPIRAN 82

DAFTAR TABEL

Tabel 2.1 Skala IUALTD.....7
Tabel 2.2 Penelitian Mengenai Deteksi Bakteri TBC Citra Sputum. 10
Tabel 2.3 Perbandingan Arsitektur YOLO yang Digunakan dalam Penelitian.....17
Tabel 3.1 Metrik Evaluasi Performa dan Efisiensi Model.....36
Tabel 4.1 Sumber Berkas *Pretrained Weights* Model YOLO 45
Tabel 4.2 Hasil Pelatihan dan Evaluasi Kinerja Model Berdasarkan Variasi *Random Seed*...48
Tabel 4.3 Perbandingan Hasil *Box loss* dan *Epoch* Best Model.....49
Tabel 4.4 Perbandingan Hasil Pengujian.....52
Tabel 4.5 Hasil Ukuran Model.....59
Tabel 4.6 Perbandingan Hasil Pengujian Setelah Konversi.....61
Tabel 4.7 Selisih Performa Sebelum dan Sesudah Konversi.....62
Tabel 4.8 Hasil Pengukuran Waktu Inferensi.....65
Tabel 4.9 Perbandingan Hasil Metrik Model TFLite.....67

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi perhitungan Intersection over Union (IoU) 21
Gambar 2.2 *Confusion Matrix* 22
Gambar 3.1 Flowchart Rancangan Penelitian 26
Gambar 3.2 Contoh Isi File Anotasi Format YOLO 28
Gambar 4.1 Contoh Citra Mikroskopis Sputum 38
Gambar 4.2 Struktur Anotasi Format XML 40
Gambar 4.3 Struktur Anotasi Format TXT 40
Gambar 4.4 Kode Program untuk Konversi Format Label 41
Gambar 4.5 Kode Program Pembagian *Dataset* 42
Gambar 4.6 Struktur Folder Dataset 43
Gambar 4.7 Jumlah File pada Folder Dataset 43
Gambar 4.8 Kode Program Konfigurasi Hiperparameter 44
Gambar 4.9 Kode Program Pelatihan Model 44
Gambar 4.10 Hasil Grafik Loss Pelatihan Awal 46
Gambar 4.11 Grafik Training dan Validation Box Loss 48
Gambar 4.12 *Confusion Matrix* Tiga Model YOLO 54
Gambar 4.13 Hasil Prediksi pada Data Testing Pertama 56
Gambar 4.14 Hasil Prediksi pada Data Testing Kedua 57
Gambar 4.15 Kode Program untuk Optimasi ke Format TFLite 59
Gambar 4.16 Kode Program untuk Pengukuran Waktu Inferensi 64
Gambar 4.17 Visualisasi *Ground Truth* dan Jumlah Anotasi Objek pada Tiga Sampel Citra 70
Gambar 4.18 Perbandingan Kesesuaian *Bounding Box* Prediksi terhadap *Ground Truth* 71
Gambar 4.19 Perbandingan Hasil Deteksi YOLO11n & YOLO26n pada Tiga Sampel Citra 71
Gambar 4.20 Distribusi *Confidence Score* Hasil Deteksi 72

BAB I
PENDAHULUAN

1.1 Latar Belakang

Tuberkulosis (TBC) merupakan penyakit infeksi kronis yang disebabkan oleh bakteri *Mycobacterium tuberculosis* dan hingga kini masih menjadi permasalahan kesehatan masyarakat global yang serius dan dapat menyebar melalui udara ketika penderita mengeluarkan *droplet*, misalnya saat batuk (Davidson et al., 2024). Berdasarkan laporan World Health Organization (WHO), beban penyakit tuberkulosis (TBC) berdasarkan data tahun 2024, diperkirakan terdapat 10,7 juta kasus insiden TBC di seluruh dunia, dengan tingkat fatalitas kasus mencapai 11,5%. Indonesia berada di peringkat kedua negara dengan beban TBC tertinggi di dunia setelah India dengan kontribusi sebesar 10% terhadap total kasus global (Tuberculosis Global Report, 2025), tingginya kasus TBC di Indonesia mencerminkan tantangan klinis dalam hal diagnosis yang menjadikan upaya deteksi dini sebagai prioritas utama dalam strategi penanggulangan penyakit ini.

Metode konvensional pemeriksaan mikroskopis sputum masih menjadi standar yang banyak digunakan karena biaya yang relatif rendah dan kemudahan implementasi (de Carvalho Brito et al., 2026). Namun, pendekatan ini sangat bergantung pada keahlian tenaga analis laboratorium dan rentan terhadap variasi interpretasi, terutama pada citra dengan kualitas rendah atau keberadaan *debris* yang tinggi. Meskipun metode pembacaan sampel mikroskopis sputum telah lama digunakan dan terbukti efektif, proses pembacaan slide secara manual memerlukan keahlian seorang analis laboratorium berpengalaman, membutuhkan waktu yang relatif lama, dan rentan terhadap *human error*.

Dalam beberapa tahun terakhir, perkembangan teknologi kecerdasan buatan khususnya pada bidang *computer vision* telah membuka peluang untuk mengotomatisasi proses deteksi bakteri tuberkulosis pada citra mikroskopis. Pendekatan berbasis *deep learning*, terutama model deteksi objek seperti YOLO, mampu mengidentifikasi dan melokalisasi objek secara langsung dalam satu tahap proses dengan efisiensi komputasi yang tinggi (Sigit, Yuniarti, Karlita, Kusumawati, & Anwar, 2025). Keunggulan ini menjadikan YOLO sebagai kandidat yang relevan untuk diterapkan pada sistem diagnosis berbasis citra, termasuk dalam konteks deteksi bakteri *Mycobacterium tuberculosis* yang memiliki karakteristik objek kecil dan sering kali tertutup oleh *noise* berupa *debris*.

Meskipun demikian, pemilihan arsitektur model yang optimal masih menjadi tantangan utama, khususnya ketika mempertimbangkan implementasi pada perangkat dengan sumber

daya terbatas pada aspek memori dan daya komputasi, sehingga diperlukan model yang tidak hanya memiliki performa tinggi tetapi juga efisien dalam proses inferensi. Namun demikian, penelitian-penelitian terdahulu terkait deteksi bakteri TB pada citra mikroskopis umumnya masih terbatas pada evaluasi satu arsitektur YOLO secara tunggal, tanpa membandingkannya secara langsung dengan versi arsitektur lain dalam kondisi eksperimen yang setara. Keterbatasan ini menyebabkan klaim keunggulan suatu arsitektur sulit diverifikasi secara objektif, karena perbedaan performa antar penelitian berpotensi dipengaruhi oleh variasi dataset, konfigurasi pelatihan, maupun prosedur evaluasi yang digunakan, bukan semata-mata oleh perbedaan arsitektur itu sendiri.

Hingga saat ini, belum ditemukan penelitian yang mengevaluasi YOLOv8, YOLOv10, YOLO11, dăn YOLO26 secara simultan pada domain deteksi bakteri TB menggunakan kondisi eksperimen yang identik, meliputi dataset yang sama, konfigurasi pelatihan yang seragam, prosedur optimasi yang konsisten, serta metrik evaluasi yang sama, sehingga perbedaan performa yang diperoleh dapat dikaitkan dengan perbedaan arsitektur, bukan pada variasi proses eksperimen. Oleh karena itu, penelitian ini berfokus pada komparasi keempat arsitektur tersebut untuk mengevaluasi performa masing-masing model dalam mendeteksi bakteri TB pada citra mikroskopis sputum, dengan mempertimbangkan metrik performa seperti *Precision*, *Recall*, mAP, dan F1-Score, serta metrik efisiensi komputasi melalui ukuran model dan waktu inferensi.

Penelitian ini difokuskan pada tahap evaluasi performa dan efisiensi komputasi model melalui simulasi konversi TensorFlow Lite INT8, sebagai dasar justifikasi kelayakan sebelum implementasi lebih lanjut. Hasil penelitian ini diharapkan dapat menjadi acuan awal untuk pengembangan sistem deteksi tuberkulosis yang diimplementasikan secara langsung pada perangkat *edge computing*, seperti Raspberry Pi, guna mendukung diagnosis TBC yang portabel dan dapat diakses secara luas pada fasilitas kesehatan dengan sumber daya terbatas.

1.2 Rumusan Masalah

Rumusan masalah dalam penelitian ini disusun untuk mengidentifikasi permasalahan utama yang akan dianalisis, sebagai berikut:

a. Bagaimana tingkat performa masing-masing model YOLO dalam mendeteksi bakteri *Mycobacterium tuberculosis* pada citra mikroskopis?
b. Bagaimana perbandingan performa masing-masing model berdasarkan mAP, *Precision*, *Recall*, dan F1-score?

c. Bagaimana efisiensi komputasi masing-masing model ditinjau dari ukuran model, waktu inferensi, dan dampak konversi ke format TensorFlow Lite INT8?
d. Model YOLO mana yang paling optimal setelah dikonversi TensorFlow Lite dengan kuantisasi INT8?

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini ditetapkan agar pembahasan tetap terarah dan tidak menyimpang dari tujuan penelitian, yaitu sebagai berikut:

a. *Dataset* menggunakan citra mikroskopis sputum dari Kaggle sebanyak 1.265 citra.
b. Varian model meliputi *nano* dan *small*.
c. Pengujian performa inferensi model, termasuk hasil optimasi kuantisasi INT8, dilakukan pada platform Google Colab dengan runtime CPU sebagai simulasi lingkungan komputasi terbatas, bukan pada perangkat *edge computing* fisik seperti Raspberry Pi atau perangkat *embedded* lainnya.
d. Setiap arsitektur model hanya dilatih dan dievaluasi dalam satu kali proses pelatihan menggunakan konfigurasi hiperparameter yang sama.
e. Penelitian ini tidak melakukan validasi hasil deteksi model oleh tenaga medis atau ahli mikrobiologi.

1.4 Tujuan Penelitian

a. Mengukur tingkat performa model YOLO dalam mendeteksi bakteri *Mycobacterium tuberculosis* pada citra mikroskopis.
b. Membandingkan performa model YOLOv8, YOLOv10, YOLO11, dan YOLO26 berdasarkan metrik evaluasi mAP, *Precision*, *Recall*, dan F1-score.
c. Mengevaluasi efisiensi komputasi masing-masing model berdasarkan ukuran model dan waktu inferensi.
d. Menentukan model YOLO yang paling optimal setelah dikonversi ke TensorFlow Lite dengan kuantisasi INT8 berdasarkan pertimbangan akurasi dan efisiensi komputasi.

1.5 Manfaat Penelitian

a. Memberikan kontribusi dalam pengembangan deteksi objek pada citra medis.
b. Menjadi referensi penelitian terkait YOLO untuk deteksi bakteri.
c. Mendukung pengembangan sistem diagnosis berbasis kecerdasan buatan.
d. Memberikan solusi untuk implementasi pada perangkat dengan sumber daya terbatas.

1.6 Metode Penelitian

Metodologi penelitian ini disusun sebagai tahapan sistematis yang digunakan untuk mencapai tujuan penelitian secara terarah dan terukur. Setiap tahapan dirancang untuk memastikan proses penelitian berjalan secara logis, mulai dari pengolahan data hingga analisis hasil yang diperoleh. Adapun tahapan dalam penelitian ini adalah sebagai berikut:

a. Persiapan *Dataset*, yaitu pengumpulan dan pemeriksaan *dataset* citra mikroskopis sputum yang bersumber dari platform Kaggle.
b. Konversi Anotasi *Dataset*, di mana format anotasi bounding box dari *dataset* diubah ke format yang kompatibel dengan framework Ultralytics YOLO.
c. Pembagian *Dataset*, yaitu tahap pembagian *dataset* ke dalam partisi pelatihan, validasi, dan pengujian dengan rasio 70:20:10.
d. Konfigurasi Augmentasi dan Hiperparameter, meliputi penentuan strategi augmentasi data dan penetapan nilai hiperparameter pelatihan yang seragam untuk seluruh model.
e. Konfigurasi Pelatihan dan Validasi, yaitu proses pelatihan kedelapan varian model YOLO secara individual menggunakan konfigurasi yang telah ditetapkan.
f. Pengujian Model, di mana seluruh model dievaluasi pada data pengujian menggunakan metrik mAP@50, mAP@50-95, *Precision*, Recall, dan F1-Score.
g. Optimasi Model ke Format TensorFlow Lite, yaitu dengan kuantisasi INT8 untuk mempersiapkan model pada skenario *deployment edge computing*.
h. Pengujian Model TFLite, dilakukan pengujian model TFLite pada *dataset* uji untuk mengukur dampak konversi terhadap performa deteksi.
i. Perhitungan Waktu Inferensi, yaitu perhitungan waktu inferensi pada kondisi sebelum dan sesudah konversi sebagai proksi kelayakan *deployment* pada perangkat *edge*.
j. Mengevaluasi dan Menganalisis Komparatif, tahap ini untuk menentukan model yang paling optimal berdasarkan kombinasi performa deteksi dan efisiensi komputasi.

1.7 Sistematika Penelitian

Dalam penyusunan tugas akhir ini, sistematika penulisan dibagi menjadi beberapa bab sebagai berikut:

a. Bab I Pendahuluan

Berisi latar belakang masalah yang menjadi dasar penelitian mengenai deteksi bakteri TB menggunakan arsitektur YOLO, yang selanjutnya dikembangkan menjadi rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan metodologi penelitian.

b. Bab II Landasan Teori

Berisi teori dan konsep yang berkaitan dengan penelitian, meliputi penyakit tuberculosis, evolusi model YOLO, tantangan deteksi objek berukuran kecil, kuantisasi model, serta penjelasan metrik evaluasi yang digunakan. Teori dan konsep penelitian ini bersumber dari jurnal, buku, dan artikel ilmiah yang relevan.

c. Bab III Metodologi

Berisi tahapan yang dilakukan dalam penelitian ini, mulai dari persiapan dataset hingga pengujian, evaluasi, dan analisis hasil dari masing-masing model.

d. Bab IV Hasil dan Pembahasan

Berisi hasil dan pembahasan dari setiap tahapan yang telah dijelaskan pada BAB III, hasil evaluasi, serta hasil analisis komparatif antar model.

e. Bab V Kesimpulan dan Saran

Berisi Kesimpulan dari hasil penelitian yang telah diperoleh, serta saran yang dapat digunakan sebagai acuan untuk pengembangan penelitian selanjutnya.

BAB II
LANDASAN TEORI

2.1 Mycobacterium Tuberculosis

Mycobacterium tuberculosis (MTB) adalah suatu basil Gram-positif aerobik yang tergolong dalam famili *Mycobacteriaceae* dan merupakan bakteri yang menyebabkan penyakit tuberkulosis (Davidson et al., 2024). Karakteristik khas bakteri ini adalah dinding sel yang kaya mikolat, sebuah komponen lipid lilin yang memberikan sifat Bakteri Tahan Asam (BTA). Dari perspektif morfologi, MTB berbentuk batang lurus atau sedikit melengkung dengan ukuran panjang 1-4 μm dan diameter 0,3-0,6 μm, tidak berspora, tidak berkapsul, dan tidak motil (Yuniarti et al., 2024). Metode diagnostik yang paling umum digunakan untuk mendeteksi TB di fasilitas kesehatan primer adalah pemeriksaan apusan sputum. Meskipun teknik ini relatif terjangkau, proses analisisnya bersifat manual, sangat bergantung pada keahlian analis laboratorium, dan memerlukan waktu antara 40 menit hingga 3 jam per sampel untuk dianalis (Yuniarti et al., 2024).

Penularan tuberkulosis (TB) terutama terjadi melalui inhalasi *droplet* di udara yang mengandung *Mycobacterium tuberculosis*. *Droplet* ini umumnya dilepaskan oleh individu yang terinfeksi saat batuk, bersin, atau berbicara. Mekanisme penularan ini menegaskan pentingnya kontak jarak dekat dan penggunaan ruang bersama dalam meningkatkan risiko penyebaran penyakit (Davidson et al., 2024). Gejala klinis pada penderita tuberkulosis umumnya ditandai dengan gangguan respirasi kronis, terutama batuk berdahak yang menetap selama lebih dari dua minggu. Dalam kondisi yang lebih parah, batuk tersebut sering kali disertai dengan ekspektorasi darah (hemoptisis) serta sesak napas yang signifikan (Davidson et al., 2024). Selain gejala lokal pada sistem pernapasan, penderita juga mengalami gejala sistemik yang meliputi keletihan tubuh (malaise), penurunan drastis pada nafsu makan yang berujung pada penyusutan berat badan, serta hiperhidrosis atau keringat berlebih pada malam hari meski tanpa aktivitas fisik. Kondisi ini sering kali dibarengi dengan demam dan rasa meriang (Davidson et al., 2024).

Tingkat keparahan infeksi Tuberkulosis diidentifikasi melalui metode kuantitatif dengan menghitung kepadatan bakteri dalam sampel dahak. Penentuan klasifikasi ini mengacu pada standar observasi mikroskopis terhadap jumlah Bakteri Tahan Asam (BTA) yang ditemukan. Jumlah BTA dibaca berdasarkan skala International Union Against Tuberculosis and Lung Disease (IUATLD) seperti ditunjukkan pada Tabel 2.1 (Aulia et al., 2024).

Tabel 2.1 Skala IUALTD

Pembacaan di bawah mikroskop	Pelaporan hasil
BTA tidak ditemukan dalam 100 lapang pandang	Negatif
1–9 BTA dalam 100 lapang pandang	Scanty
10–99 BTA dalam 100 lapang pandang	+1 (positif 1)
1–10 BTA dalam 1 lapang pandang	+2 (positif 2)
>10 BTA dalam 1 lapang pandang	+3 (positif 3)

2.2 Kajian Literatur Penelitian Sebelumnya

Berbagai pendekatan untuk mendeteksi bakteri TB telah dieksplorasi, antara lain konversi ruang warna dari RGB ke HSV (*Hue, Saturation, Value*) untuk memanfaatkan karakteristik warna khas bakteri TB, teknik *thresholding* untuk segmentasi bakteri dari latar belakang, operasi morfologi untuk memperhalus hasil segmentasi, serta metode deteksi tepi Canny dan *connected component labeling* untuk mengidentifikasi dan menghitung objek individual, sebagaimana dilakukan oleh (Yuniarti et al., 2024) Penelitian tersebut memberikan bukti kuantitatif yang komprehensif mengenai batas kemampuan *classical image processing.* Pengujian pada 10 citra sputum dari pasien TB di rumah sakit menunjukkan bahwa metode *classical image processing* hanya mencapai akurasi penghitungan rata-rata 59,59%.

Analisis per-citra mengungkapkan bahwa metode *classical image processing* yang dilakukan oleh (Yuniarti et al., 2024) bekerja cukup baik hanya pada kondisi bakteri yang terpisah dan jarang, namun gagal secara signifikan pada kondisi bakteri yang saling tumpang tindih atau beragregasi. Pada kasus terparah (Bacteria 09), *error* mencapai 125% karena sistem menghitung kelompok bakteri yang melebur sebagai objek yang jauh lebih sedikit dari jumlah aktual. Keterbatasan struktural ini berakar pada fakta bahwa metode *classical image processing* tidak dapat beradaptasi terhadap variasi kontekstual yang inheren pada citra sputum klinis, termasuk variasi intensitas pewarnaan antarpereparat, keberadaan *debris* yang menyerupai bakteri, dan kerapatan bakteri. Kenyataan inilah yang mendorong pergeseran menuju pendekatan berbasis pembelajaran mesin yang mampu memahami konteks visual secara adaptif.

Selain menggunakan metode klasik, penelitian (Yuniarti et al., 2024) juga menggunakan YOLOv5s mencapai akurasi rata-rata 86,09%. Namun, analisis yang lebih dalam terhadap studi ini mengungkap beberapa batasan metodologis yang perlu diakui yaitu *dataset* yang sangat kecil (hanya 10 citra) membuat hasil ini rentan terhadap variasi sampling, dan metrik yang digunakan adalah perhitungan akurasi rata-rata sampel dan tidak langsung dengan nilai mAP yang digunakan sebagian besar studi lain. Meskipun demikian, studi ini memberikan kontribusi

penting dengan mendokumentasikan secara kuantitatif batas kemampuan pendekatan berbasis aturan, menegaskan argumen untuk adopsi *deep learning*.

Penelitian (Y. Wang et al., 2024) mengembangkan YOLO-FMS, sebuah varian YOLOv5 yang dimodifikasi secara ekstensif dengan penambahan modul GSConv, C3-B-CBAM, dan *detection head* tambahan, mencapai mAP 87,6% pada *dataset* tuberculosis-phonecamera. Studi ini merepresentasikan satu pendekatan yang dominan dalam literatur, yaitu modifikasi arsitektur yang ada untuk mengadaptasinya terhadap karakteristik domain spesifik. Keunggulan pendekatan ini adalah kontribusi teknis yang jelas dan terukur. Namun, keterbatasannya adalah bahwa setiap modifikasi menciptakan model yang semakin jauh dari baseline standar, sehingga sulit untuk mengisolasi kontribusi arsitektur dasar dari kontribusi modifikasi yang ditambahkan.

Penelitian yang dilakukan oleh (Chen et al., 2024) mengikuti pola serupa dengan YOLOv8s yang dimodifikasi melalui penambahan *detection head*, modul multi-scale *feature fusion*, dan Coordinate Attention, mencapai mAP 85,7% pada *dataset* publik MTB. Sedangkan (Saurina et al., 2025) mengembangkan pendekatan multimodal yang mengintegrasikan citra dari tiga ruang warna (RGB, HSV, CIELAB) secara bersamaan, menghasilkan akurasi tertinggi yang dilaporkan dalam literatur sebesar 92,24%.

Penelitian oleh (Sigit et al., 2025) adalah studi yang paling dekat dengan konteks *deployment* klinis dalam literatur yang ditinjau. Implementasi sistem deteksi *real-time* menggunakan YOLOv8 pada dua konfigurasi kamera (Hayear dan Optilab) mengungkap kesenjangan mAP yang cukup signifikan (0,873 dan 0,678) hanya akibat perbedaan kamera. Temuan ini menunjukkan bahwa performa model deteksi objek sangat dipengaruhi oleh karakteristik proses akuisisi citra, seperti jenis kamera yang digunakan. Oleh karena itu, model yang menunjukkan performa tinggi pada satu kondisi pengambilan citra belum tentu menghasilkan performa yang sama ketika diterapkan pada kondisi atau perangkat yang berbeda.

Penelitian (Rodrick et al., 2025) mengevaluasi YOLOv8 pada citra beresolusi rendah yang diambil menggunakan kamera smartphone, mencapai mAP@50 sebesar 82,7%. Meskipun konteks yang dipilih sangat relevan untuk fasilitas kesehatan terbatas, studi ini tidak mengintegrasikan evaluasi efisiensi komputasi sebagai dimensi eksplisit, sehingga tidak dapat menjawab pertanyaan tentang kelayakan *deployment* pada perangkat *edge* secara terukur.

Penelitian oleh (Wulandari et al., 2026) mengusulkan Mod-YOLOv10, yaitu pengembangan arsitektur YOLOv10 dengan menambahkan modul pemodelan fitur berbasis RepVGG pada *detection head* untuk mendeteksi bakteri *Mycobacterium tuberculosis* pada citra mikroskopis sputum. Penelitian ini menggunakan dataset ID-SPUT30K yang terdiri atas 313 preparat sputum dan 31.300 citra. Hasil pengujian menunjukkan bahwa Mod-YOLOv10 mencapai akurasi sebesar 97,5% dan spesifisitas sebesar 97,9%. Sementara itu, varian nano dan small juga menunjukkan performa yang kompetitif dengan kompleksitas model yang lebih rendah. Penelitian ini menunjukkan bahwa modifikasi arsitektur YOLOv10 yang memiliki modifikasi mampu meningkatkan performa deteksi bakteri TB, namun evaluasi masih difokuskan pada satu arsitektur sehingga belum memberikan gambaran mengenai perbandingan performa dengan generasi YOLO lainnya.

Berbeda dengan penelitian sebelumnya yang berfokus pada pengembangan model tertentu, (Kamaluddin et al., 2026) mengombinasikan prapemrosesan LAB-CLAHE untuk meningkatkan visibilitas bakteri dengan strategi *bacilli-aware loss weighting*, yaitu *Weighted Cross-Entropy* dan *Focal Loss*, guna mengatasi ketidakseimbangan jumlah sampel antar kelas. Untuk mendeteksi TB pada citra biopsi jaringan paru-paru. Pada tahap deteksi objek, penelitian membandingkan YOLOv8n dan YOLO11n, di mana YOLO11n menunjukkan performa yang lebih baik dibandingkan YOLOv8n pada seluruh metrik evaluasi, dengan *precision* sebesar 0,5, *recall* sebesar 0,479, mAP@50 sebesar 0,439, dan mAP@50–95 sebesar 0,14. Penulis menyatakan bahwa hasil tersebut dipengaruhi oleh penggunaan modul C3k2 dan C2PSA yang mampu meningkatkan representasi fitur spasial pada objek berukuran sangat kecil, sehingga YOLO11n dinilai lebih sesuai untuk aplikasi skrining tuberkulosis secara otomatis.

Berdasarkan berbagai penelitian terdahulu, peningkatan performa deteksi objek umumnya dicapai melalui modifikasi arsitektur YOLO, seperti penambahan modul *attention*, maupun komponen jaringan lainnya. Meskipun demikian, sebagian besar penelitian hanya membandingkan model hasil modifikasi dengan model dasar (*baseline*) dari generasi yang sama. Akibatnya, perbandingan performa antar generasi arsitektur YOLO standar masih terbatas sehingga perkembangan performa setiap generasi belum dapat dievaluasi secara komprehensif. Selain itu, berdasarkan literatur yang ditinjau dalam penelitian ini, belum ditemukan penelitian yang menerapkan arsitektur YOLO26 untuk deteksi bakteri TB pada citra mikroskopis. Kondisi tersebut menunjukkan bahwa masih terdapat kebutuhan akan penelitian yang mengevaluasi beberapa generasi arsitektur YOLO standar secara langsung menggunakan kondisi eksperimen yang seragam.

Lebih lanjut, hingga saat ini belum ditemukan penelitian yang mengevaluasi YOLOv8, YOLOv10, YOLO11, dan YOLO26 secara simultan menggunakan kondisi eksperimen yang identik pada domain deteksi bakter TB. Kondisi identik tersebut meliputi penggunaan dataset yang sama, konfigurasi pelatihan yang seragam, prosedur optimasi yang konsisten, serta metrik evaluasi yang sama, sehingga perbedaan performa yang diperoleh dapat diatribusikan secara lebih valid pada perbedaan arsitektur, bukan pada variasi proses eksperimen. Oleh karena itu, penelitian ini bertujuan menyajikan perbandingan yang lebih adil dan sistematis terhadap perkembangan arsitektur YOLO dalam konteks deteksi objek medis berukuran kecil. Ringkasan penelitian terdahulu disajikan pada Tabel 2.2.

Tabel 2.2 Penelitian Mengenai Deteksi Bakteri TBC Citra Sputum.

Studi	Model	Tujuan Penelitian	Dataset	Hasil Utama
(Yuniarti et al., 2024)	YOLOv5s dan pengolahan citra klasik	Membandingkan pendeteksian dan penghitungan gambar mikroskop menggunakan pemrosesan gambar klasik dan YOLOv5s	10 citra ZN	Accuracy YOLOv5s = 86,09%, Klasik = 59,59%
(Y. Wang et al., 2024)	YOLO-FMS (berbasis YOLOv5)	Mengusulkan YOLO-FMS, yaitu model yang ringan dan efisien berbasis YOLOv5 untuk deteksi apusan mikroskopis medis	1,265 citra sputum	mAP@50= 87,6%
(Chen et al., 2024)	YOLOv8s (dimodifikasi)	Mengusulkan model YOLOv8s yang telah ditambahkan satu *detection head* tambahan untuk memfokuskan pembelajaran pada informasi objek berukuran kecil.	1,265 citra sputum	mAP = 85,7%
(Saurina et al., 2025)	YOLOv8	Memperkenalkan model *deep learning* multimodal baru yang mengintegrasikan segmentasi citra sputum pada berbagai ruang warna, yaitu RGB, *hue*, *saturation*, *value* (HSV), serta CIELAB, dengan memanfaatkan model YOLOv8 untuk deteksi dan segmentasi secara *real-time*.	1,265 citra sputum dari Universitas Airlangga	Accuracy = 92,24%
(Sigit et al., 2025)	YOLOv8	Pemanfaatan YOLOv8 untuk mengotomatisasi deteksi basil tahan asam (*acid-fast bacilli*).	138 citra ZN (Hayear); 75 citra (Optilab)	mAP Hayear = 0,873; mAP Optilab = 0,678

(Rodrick et al., 2025)	YOLOv8	Mengevaluasi kinerja model YOLOv8 dalam mendeteksi bakteri	1,224 citra sputum dari Roboflow	*Precision* =72,7%, *Recall* = 78,7%, (mAP) = 82,7%, mAP@95=41,5%.
(Wulandari et al., 2026)	Mod-YOLOv10 (Modifikasi dari YOLOv10)	Mengusulkan Mod-YOLOv10, sebuah arsitektur YOLOv10 yang dimodifikasi dengan mengintegrasikan RepVGG pada bagian deteksi untuk mendeteksi Bacillus Acid-Fast secara andal serta melakukan penilaian dahak berdasarkan IUATLD.	31,300 citra sputum dari 313 sampel sputum	*Accuracy = 97,5%, Specificity = 97,5%*
(Kamaluddin et al., 2026)	YOLOv8n dan YOLO11n	Mengembangkan sistem deteksi otomatis tuberkulosis pada citra biopsi jaringan menggunakan *deep learning* guna meningkatkan akurasi dan efisiensi proses diagnosis dibandingkan pemeriksaan mikroskopis manual.	1,331 gambar biosi jaringan paru-paru	YOLOv8n (Precison = 0,456, Recall = 0,449, mAP@50 = 0,403, mAP@50-95 = 0,13) YOLO11n (Precision = 0,5, Recall = 0,479, mAP@50 = 0,439, mAP@50-95 = 0,14)

2.3 Pendekatan Deteksi Objek Sebelum Arsitektur YOLO

Sebelum diperkenalkannya YOLO, deteksi objek didominasi oleh pendekatan dua tahap (*two-stage detectors*), di mana *Region Proposal Network* (RPN) terlebih dahulu menghasilkan kandidat wilayah objek, kemudian setiap proposal diproses secara terpisah oleh *Convolutional Neural Network* (CNN) untuk diklasifikasikan. Paradigma ini, sebagaimana diimplementasikan pada R-CNN, Fast R-CNN, dan Faster R-CNN, mampu mencapai tingkat akurasi yang tinggi, namun memiliki keterbatasan pada kecepatan inferensi sehingga kurang optimal untuk aplikasi *real-time* (Redmon et al., 2016). YOLO (*You Only Look Once*), yang diperkenalkan oleh Joseph Redmon pada tahun 2016, menghadirkan perubahan konseptual dengan merumuskan deteksi objek sebagai permasalahan regresi tunggal secara *end-to-end*, yang secara langsung memetakan piksel citra ke koordinat *bounding box* dan probabilitas kelas.

YOLO (*You Only Look Once*) dikembangkan sebagai pendekatan *one-stage object detector* yang menyederhanakan proses deteksi objek dengan menggabungkan lokalisasi dan

klasifikasi ke dalam satu jaringan yang dioptimalkan secara bersamaan (*jointly optimized*) (Redmon et al., 2016). Berbeda dengan keluarga R-CNN, termasuk Fast R-CNN dan Faster R-CNN, yang menggunakan pendekatan *two-stage* melalui proses pengusulan *region proposal* sebelum klasifikasi objek, YOLO melakukan deteksi secara langsung pada seluruh citra dalam satu kali proses inferensi. Selain itu, YOLO hanya menghasilkan sejumlah kecil kandidat *bounding box* dibandingkan ribuan *region proposal* yang dihasilkan oleh metode seperti *Selective Search*, sehingga mampu mengurangi kompleksitas komputasi dan mempercepat proses inferensi. Karakteristik tersebut menjadikan YOLO lebih sesuai untuk aplikasi deteksi objek secara *real-time* tanpa mengorbankan akurasi secara signifikan (Redmon et al., 2016).

Secara fundamental, YOLO membagi citra input menjadi grid berukuran $S \times S$, di mana setiap sel grid bertanggung jawab untuk memprediksi sejumlah B *bounding box* beserta nilai *confidence* dan probabilitas kelas sebanyak C. Pendekatan arsitektural ini memungkinkan pemrosesan seluruh citra dalam satu kali *forward pass*, sehingga menghasilkan kecepatan inferensi yang jauh lebih tinggi, yaitu sekitar 45 FPS pada YOLOv1, dibandingkan dengan Faster R-CNN yang berada pada kisaran 7 FPS (Redmon et al., 2016). Meskipun pada awalnya terdapat *trade-off* berupa penurunan akurasi, perbedaan tersebut terus diminimalkan pada pengembangan versi YOLO berikutnya.

Pendekatan lain yang cukup signifikan adalah *Single-Shot MultiBox Detector* (SSD), yang serupa dengan YOLO dalam hal penekanan pada kecepatan dan efisiensi melalui penghilangan tahap *region proposal* yang terpisah (Terven & Cordova-Esparza, 2024). Selain itu, metode seperti Mask R-CNN memperluas kemampuan deteksi objek ke tugas *instance segmentation*, sehingga memungkinkan lokalisasi objek yang lebih presisi hingga tingkat piksel (Terven & Cordova-Esparza, 2024).

Perkembangan lain, seperti RetinaNet dan EfficientDet, turut memperkaya lanskap algoritma deteksi objek. Masing-masing metode menawarkan karakteristik dan keunggulan tersendiri, khususnya dalam hal *trade-off* antara kecepatan, akurasi, dan kompleksitas komputasi, sehingga dapat disesuaikan dengan kebutuhan aplikasi serta keterbatasan sumber daya yang tersedia (Terven & Cordova-Esparza, 2024).

Pada penelitian yang dilakukan oleh (Tan et al., 2021) hasil menunjukkan nilai *mean Average Precision* (mAP) pada RetinaNet mencapai 82,89%, namun kecepatan pemrosesannya (*frames per second* / FPS) hanya sekitar sepertiga dari YOLOv3, sehingga sulit untuk memenuhi kebutuhan deteksi secara *real-time*. Sementara itu, SSD menunjukkan performa yang lebih baik sedikit pada metrik mAP (82,7%). Meskipun mAP YOLOv3 sedikit lebih

rendah dibandingkan model lain (80,69%), model ini memiliki keunggulan signifikan dalam hal kecepatan deteksi. Selain itu, YOLOv3 juga menunjukkan kinerja yang lebih baik dalam mendeteksi sampel sulit (*hard samples*), sehingga lebih sesuai untuk diimplementasikan pada perangkat medis di lingkungan rumah sakit (Tan et al., 2021)

Hasil penelitian tersebut menunjukkan bahwa terdapat *trade-off* yang jelas antara akurasi dan kecepatan pada model deteksi objek. Meskipun RetinaNet memiliki nilai *mean Average Precision* (mAP) yang lebih tinggi, keterbatasan pada kecepatan inferensi menjadikannya kurang ideal untuk aplikasi *real-time*. Di sisi lain, YOLOv3, dengan sedikit kompromi pada akurasi, justru menawarkan keunggulan signifikan dalam hal kecepatan serta kemampuan yang lebih baik dalam menangani *hard samples*, sehingga lebih sesuai untuk implementasi pada perangkat medis.

Temuan ini menjadi landasan penting bagi penelitian ini, khususnya dalam konteks deteksi bakteri TB pada citra mikroskopis yang menuntut keseimbangan antara akurasi dan efisiensi komputasi. Dalam skenario diagnostik berbasis *edge computing*, seperti penggunaan pada perangkat dengan sumber daya terbatas, kecepatan inferensi menjadi faktor krusial selain akurasi. Berdasarkan pertimbangan tersebut, YOLO dipilih sebagai metode utama dalam penelitian ini dibandingkan SSD dan Mask R-CNN. Dibandingkan SSD, YOLO generasi terbaru menawarkan keseimbangan akurasi-kecepatan yang lebih matang serta dukungan ekosistem *deployment* ke perangkat *edge* yang lebih luas. Sementara itu, dibandingkan Mask R-CNN, YOLO lebih efisien untuk deteksi objek tanpa memerlukan segmentasi tingkat piksel, sehingga lebih sesuai dengan tujuan implementasi pada sistem skrining TB berbasis *edge computing* dengan sumber daya terbatas.

2.4 Evolusi Arsitektur YOLO

Seperti yang telah dijelaskan pada Subbab 2.3, YOLO (*You Only Look Once*) merupakan salah satu algoritma deteksi objek yang menerapkan proses lokalisasi dan klasifikasi objek dalam satu kali inferensi. Secara umum, arsitektur YOLO terdiri atas tiga tahapan utama, yaitu *input*, ekstraksi fitur (*feature extraction*), dan prediksi (Yeerjiang et al., 2024). Pada tahap *input*, citra dimasukkan ke dalam model dan dapat melalui proses augmentasi data untuk meningkatkan kemampuan generalisasi model terhadap berbagai variasi data. Selanjutnya, pada tahap ekstraksi fitur, jaringan *Convolutional Neural Network* (CNN) digunakan sebagai *backbone* untuk mengekstraksi karakteristik penting dari citra (Yeerjiang et al., 2024). Hasil ekstraksi fitur kemudian diproses pada tahap prediksi untuk menghasilkan *bounding box* beserta kelas objek yang terdeteksi. Kombinasi dari ketiga tahapan tersebut memungkinkan

YOLO menghasilkan performa deteksi objek yang baik dengan tetap mempertahankan kecepatan inferensi yang tinggi, sehingga sesuai untuk aplikasi *real-time* (Yeerjiang et al., 2024).

Sejak diperkenalkan pertama kali oleh Redmon pada tahun 2016, arsitektur YOLO mengalami serangkaian pengembangan yang secara konsisten berfokus pada peningkatan akurasi tanpa mengorbankan kecepatan inferensi *real-time*. YOLOv1 menerapkan pendekatan regresi satu tahap (*one-shot regression*) terhadap seluruh citra sehingga mencapai kecepatan inferensi tinggi, namun memiliki keterbatasan pada akurasi lokalisasi serta kemampuan mendeteksi objek kecil dan berdekatan (Terven & Cordova-Esparza, 2024). Keterbatasan tersebut diperbaiki secara bertahap pada generasi berikutnya, meliputi penerapan *anchor box* dan *batch normalization* pada YOLOv2, mekanisme *multi-scale detection* dengan *backbone* Darknet-53 pada YOLOv3, kombinasi teknik *bag-of-freebies* dan *bag-of-specials* pada YOLOv4, serta implementasi berbasis PyTorch dengan fitur *AutoAnchor* dan augmentasi data lanjutan pada YOLOv5 (Terven & Cordova-Esparza, 2024) Kelima generasi awal ini menjadi fondasi arsitektural bagi pengembangan YOLO versi terbaru, khususnya empat arsitektur yang digunakan dalam penelitian ini, yaitu YOLOv8, YOLOv10, YOLO11, dan YOLO26.

YOLOv8 diperkenalkan pada Januari 2023 oleh Ultralytics sebagai pengembangan lanjutan dari YOLOv5. Model ini tersedia dalam lima varian skala sama seperti YOLOv5, serta mendukung berbagai tugas visi komputer seperti deteksi objek, segmentasi, estimasi pose, pelacakan, dan klasifikasi. Secara arsitektural, YOLOv8 menggunakan *backbone* yang serupa dengan YOLOv5, namun dengan pembaruan pada modul CSP (*Cross Stage Partial*) yang digantikan oleh modul C2f (CSP *bottleneck with two convolutions*) untuk meningkatkan integrasi fitur dan akurasi deteksi. Model ini mengadopsi pendekatan *anchor-free* dengan *decoupled head*, sehingga proses prediksi *objectness*, klasifikasi, dan regresi dilakukan secara terpisah untuk meningkatkan kinerja keseluruhan. Dalam hal fungsi *loss*, YOLOv8 memanfaatkan kombinasi CIoU dan Distribution Focal Loss (DFL) untuk regresi *bounding box*, serta *binary cross-entropy* untuk klasifikasi, yang terbukti efektif terutama dalam mendeteksi objek berukuran kecil (Terven & Cordova-Esparza, 2024). Dari sisi implementasi, YOLOv8 mendukung penggunaan melalui *command line interface* (CLI) maupun sebagai paket Python (PIP), serta terintegrasi dengan berbagai alat untuk anotasi, pelatihan, dan *deployment* (Terven & Cordova-Esparza, 2024).

YOLOv10 merupakan salah satu arsitektur yang bukan dikembangkan oleh Ultralytics, namun YOLOv10 menghadirkan sejumlah peningkatan kemampuan model untuk

meningkatkan kinerja deteksi secara keseluruhan. Salah satu peningkatan utama adalah kemampuan deteksi objek berukuran kecil yang lebih baik dibandingkan dengan generasi sebelumnya. Hal ini menjadi penting terutama pada aplikasi seperti pengawasan dan pencitraan medis, di mana objek target sering kali berukuran kecil (Hussain, 2024). YOLOv10 memperkenalkan strategi *dual label assignment* yang menggabungkan pendekatan *one-to-many* pada tahap pelatihan untuk meningkatkan *recall* dan *one-to-one* pada tahap inferensi untuk mempertahankan *precision*. Strategi ini memungkinkan YOLOv10 menghilangkan kebutuhan *Non-Maximum Suppression* (NMS), sehingga proses inferensi menjadi lebih sederhana, latensi berkurang, dan model lebih sesuai untuk aplikasi *real-time* (A. Wang et al., 2024). Dari sisi fleksibilitas, YOLOv10 menyediakan berbagai skala model (n, s, m, l, x) yang dapat disesuaikan dengan kebutuhan komputasi dan skenario penggunaan. Pendekatan ini memungkinkan implementasi pada berbagai platform, mulai dari perangkat *edge* dengan sumber daya terbatas hingga sistem komputasi berperforma tinggi.

YOLO11 merupakan pengembangan lanjutan dari arsitektur YOLOv8 yang diperkenalkan oleh Ultralytics. YOLO11 dikembangkan dengan fokus pada keseimbangan antara akurasi, stabilitas, dan efisiensi perangkat, melalui penyempurnaan pada aspek representasi fitur dan mekanisme supervisi. Pada sisi representasi, digunakan varian *compact* CSP seperti C3/C3k2 serta modul *lightweight attention* untuk meningkatkan interaksi antarkanal tanpa menambah latensi secara signifikan. Pada bagian *neck*, desain tetap menekankan fusi fitur multi-skala dengan harmonisasi *stride* yang lebih baik untuk mengurangi efek *aliasing* dan meningkatkan akurasi lokalisasi objek kecil. Dari sisi supervisi, YOLO11 mengadopsi strategi *hybrid task-aware assignment* (Sapkota & Karkee, 2026), di mana proses penentuan label dan pembobotan *loss* dioptimalkan secara simultan untuk tugas klasifikasi, lokalisasi, serta tugas tambahan seperti segmentasi. Pendekatan ini mampu mengurangi kesalahan pada skenario padat (*crowded scenes*) serta meningkatkan *Recall* pada tingkat *Precision* tertentu dengan menjaga konsistensi optimasi fitur selama proses pelatihan. Selain itu, strategi ini dirancang agar tetap *robust* terhadap variasi ukuran *batch*, yang umum terjadi pada skenario pelatihan di perangkat dengan sumber daya terbatas (Sapkota & Karkee, 2026).

Dari perspektif sistem, YOLO11 memperkuat dukungan multi-tugas dalam ekosistem Ultralytics, meliputi deteksi objek, segmentasi, klasifikasi, estimasi pose, serta Oriented Bounding Boxes (OBB), tanpa mengubah mekanisme ekspor model dari versi sebelumnya. Meskipun masih menggunakan Non-Maximum Suppression (NMS) pada tahap inferensi, beban *post-processing* berhasil dikurangi melalui prediksi yang lebih presisi dan minim

redundansi. Secara arsitektural, YOLO11 menggantikan modul C2f dengan blok C3k2 yang lebih ringkas, mempertahankan SPPF untuk ekstraksi fitur multiskala, serta menambahkan modul C2PSA untuk meningkatkan *spatial attention*. Kombinasi ini menghasilkan arsitektur *backbone* dan *neck* yang lebih ringan dengan selektivitas fitur yang lebih baik. Secara empiris, YOLO11 menunjukkan jumlah parameter yang lebih sedikit dibandingkan varian YOLOv8 yang sebanding (2,5 juta dan 3 juta) untuk varian *small*, serta peningkatan kecepatan inferensi dibandingkan YOLOv10, dengan tetap mendukung berbagai tugas visi komputer. Dengan demikian, YOLO11 dapat diposisikan sebagai generasi menengah yang lebih akurat dan stabil, khususnya pada skenario deteksi objek kecil dan lingkungan dengan tingkat kompleksitas sedang, baik pada platform *edge* maupun GPU (Sapkota & Karkee, 2026).

YOLO26 yang dirilis pada 14 Januari 2026 merupakan perancangan ulang arsitektur YOLO yang berorientasi pada *edge computing*, dengan penekanan pada kesederhanaan *end-to-end* dan kemudahan *deployment*. Dua perubahan utama menjadi fondasi pendekatan ini. Pertama, mekanisme regresi *bounding box* tidak lagi menggunakan *Distribution Focal Loss* (DFL), melainkan digantikan dengan *bounding box regression* yang lebih ringan dan ramah perangkat keras. Penghapusan DFL menyederhanakan kompleksitas komputasi serta meningkatkan kompatibilitas lintas *runtime* dan proses kuantisasi. Kedua, jalur inferensi dirancang tanpa *Non-Maximum Suppression* (NMS), sehingga model mampu menghasilkan prediksi yang ringkas dan tidak redundan secara langsung tanpa tahap *post-processing*, yang selama ini menjadi sumber latensi tambahan. Dari sisi pelatihan, YOLO26 mengintegrasikan strategi *Progressive Loss Balancing* (ProgLoss) dan *Small-Target-Aware Label Assignment* (STAL) (Sapkota & Karkee, 2026). ProgLoss mengatur bobot *loss* secara adaptif sepanjang pelatihan untuk menjaga keseimbangan antara klasifikasi, lokalisasi, dan komponen tambahan, sehingga meningkatkan stabilitas konvergensi. Sementara itu, STAL dirancang untuk memperkuat sinyal supervisi pada objek berukuran kecil, teroklusi, atau berkontras rendah, yang sangat relevan pada citra mikroskopis. Optimasi model dilakukan menggunakan MuSGD (Jocher & Raimbekovm, 2025), yaitu varian *optimizer* yang mengombinasikan keunggulan SGD dengan mekanisme adaptif berbasis kurvatur dan momentum, sehingga mempercepat konvergensi dan mengurangi fluktuasi pada *epoch* akhir (Sapkota & Karkee, 2026).

Dari perspektif sistem, YOLO26 menekankan portabilitas dan efisiensi *deployment*. Dengan menghilangkan DFL dan NMS, model yang diekspor menjadi lebih sederhana dan kompatibel dengan berbagai platform seperti ONNX, TensorRT, CoreML, dan TFLite, serta mendukung kuantisasi INT8 dan FP16 dengan degradasi performa yang minimal. Keuntungan

latensi terutama terlihat pada eksekusi berbasis CPU dan perangkat *embedded*, khususnya pada skenario *batch size* kecil (Sapkota & Karkee, 2026). Dalam konteks penelitian ini, karakteristik YOLO26 menjadi sangat relevan. Kemampuan STAL dalam meningkatkan sensitivitas terhadap objek berukuran kecil berpotensi meningkatkan *Recall* terhadap basil MTB yang memiliki ukuran mikroskopis. Selain itu, desain *end-to-end* tanpa NMS dan struktur yang ringan menjadikan YOLO26 kandidat yang menjanjikan untuk implementasi pada perangkat dengan sumber daya terbatas (Sapkota & Karkee, 2026). Perbandingan karakteristik utama dari keempat arsitektur yang diuji dalam penelitian ini, yaitu YOLOv8, YOLOv10, YOLO11, dan YOLO26, disajikan pada Tabel 2.3

Tabel 2.3 Perbandingan Arsitektur YOLO yang Digunakan dalam Penelitian

Aspek	**YOLOv8**	**YOLOv10**	**YOLO11**	**YOLO26**
Pengembang	Ultralytics	Tsinghua University	Ultralytics	Ultralytics
Tahun Rilis	2023	2024	2024	2026
Pendekatan Deteksi	Anchor-free, C2f, *decoupled head*	Anchor-free, *dual label assignment*	Anchor-free, modul C3k2 + C2PSA	Anchor-free, tanpa DFL
Non-Maximum Suppression (NMS)	Digunakan	Tidak digunakan	Digunakan	Tidak digunakan
Fungsi Loss	CIoU + DFL	Dual label assignment (one-to-many & one-to-one)	CIoU + DFL, hybrid task-aware assignment	ProgLoss + STAL, optimizer MuSGD

2.5 Tantangan Deteksi Objek Kecil dan Implikasi terhadap Pemilihan Arsitektur YOLO

Karakteristik deteksi bakteri TB pada citra mikroskop yang melibatkan objek berukuran sangat kecil, distribusi tidak merata, serta adanya potensi klaster yang padat menuntut arsitektur deteksi yang tidak hanya mampu menangkap informasi multiskala, tetapi juga adaptif terhadap keterbatasan representasi fitur objek kecil. Meskipun integrasi *Feature Pyramid Network* (FPN) sejak generasi awal YOLO (misalnya YOLOv3) secara teoritis meningkatkan kemampuan deteksi multiskala, berbagai studi menunjukkan bahwa performa terhadap objek dengan ukuran sangat kecil (<32×32 piksel) masih mengalami degradasi yang signifikan (Terven & Cordova-Esparza, 2024). Selain itu, pendekatan *anchor-based* memiliki keterbatasan dalam menangani variasi ukuran objek yang tinggi akibat ketergantungan pada distribusi *anchor box*, sedangkan pendekatan *anchor-free* menawarkan fleksibilitas yang lebih

baik meskipun berpotensi menghadapi tantangan dalam presisi lokalisasi pada objek yang kecil dan berdekatan (Hussain, 2024).

Berdasarkan tantangan tersebut, arsitektur YOLOv8, YOLOv10, YOLO11, dan YOLO26 didasarkan pada evolusi pendekatan yang secara eksplisit berupaya mengatasi keterbatasan tersebut. Sebagaimana yang telah dijelaskan pada Subbab 2.4, YOLOv8 sebagai generasi awal *anchor-free* modern menawarkan penyederhanaan deteksi dan peningkatan efisiensi fitur melalui modul C2f. YOLOv10 memperkenalkan mekanisme *dual-assignment* yang meningkatkan kualitas supervisi tanpa ketergantungan pada NMS saat inferensi. YOLO11 melanjutkan pengembangan dengan peningkatan representasi fitur melalui modul yang lebih efisien dan strategi pelatihan yang lebih stabil, yang berpotensi meningkatkan generalisasi pada objek kecil dalam lingkungan kompleks. Sementara itu, YOLO26 dirancang secara khusus untuk skenario *edge computing* dengan pendekatan *end-to-end* tanpa NMS serta mekanisme *small-target–aware label assignment* (STAL) yang secara langsung menargetkan peningkatan sensitivitas terhadap objek berukuran kecil.

Sebagai penegasan, model YOLOv8, YOLOv10, YOLO11, dan YOLO26 didasarkan pada representasi evolusi arsitektur deteksi objek yang tidak hanya berfokus pada peningkatan performa, tetapi juga pada efisiensi komputasi dan kesiapan *deployment*. Secara khusus, karakteristik YOLO26 yang mengusung pendekatan *end-to-end* tanpa NMS, didukung oleh mekanisme *small-target–aware label assignment* (STAL) dan desain yang ringan, menjadikannya sangat relevan untuk skenario deteksi objek berukuran mikroskopis seperti bakteri TB.

Secara umum, *deployment* model YOLO standar pada perangkat *edge* menghadapi tantangan yang signifikan karena karakteristik lingkungan *edge* yang secara inheren memiliki keterbatasan sumber daya. Selain itu, jejak memori (*memory footprint*) yang tinggi pada model YOLO kerap melampaui kapasitas memori perangkat *edge* yang terbatas, sehingga *deployment* model tanpa modifikasi signifikan menjadi tidak realistis untuk dilakukan (Altaie et al., 2025).

2.6 Kuantisasi Model dan TensorFlow Lite

Salah satu pertimbangan yang mendasari arah optimasi model dalam penelitian ini adalah skenario *deployment* pada lingkungan *edge computing*, yaitu pemrosesan citra yang dilakukan sedekat mungkin dengan sumber data tanpa bergantung pada koneksi internet ke server cloud. Skenario ini relevan mengingat fasilitas kesehatan primer di berbagai wilayah, khususnya

daerah dengan keterbatasan infrastruktur konektivitas, berpotensi mengalami kendala jika bergantung pada solusi berbasis *cloud*.

Perangkat *edge* seperti Raspberry Pi umumnya memiliki daya komputasi yang relatif rendah dibandingkan CPU desktop modern, dan sebagian besar model Raspberry Pi tidak dilengkapi GPU khusus, sementara GPU terintegrasi yang tersedia tidak cukup kuat untuk mengakselerasi tugas *deep learning* secara efektif. Kondisi ini berimplikasi langsung pada performa inferensi, mengingat perbandingan performa CPU dan GPU yang dilakukan oleh Boukili et al. (2026) membuktikan bahwa akselerasi GPU memberikan percepatan signifikan terhadap komputasi konvolusi pada CNN, dengan GPU tercatat empat kali lebih cepat dibandingkan CPU dalam pemrosesan citra per detik, disertai penurunan latensi sebesar 75 persen dan peningkatan *throughput* yang signifikan. Oleh karena itu, keterbatasan akselerasi GPU pada perangkat *edge* seperti Raspberry Pi menjadi salah satu faktor yang perlu dipertimbangkan dalam desain model yang tetap efisien meskipun dijalankan sepenuhnya pada CPU.

Kuantisasi model merupakan salah satu teknik paling efektif untuk mengurangi biaya komputasi dan memori, sekaligus memberikan *trade-off* praktis antara akurasi dan efisiensi (Altaie et al., 2025). Dalam konteks *deployment* pada perangkat *edge* dengan keterbatasan sumber daya sebagaimana telah disinggung pada Subbab 2.5, kuantisasi menjadi strategi optimasi yang esensial untuk menjembatani kesenjangan antara kapasitas model YOLO standar dan keterbatasan perangkat target.

Untuk mendukung analisis efisiensi model pada skenario *edge computing*, seluruh model dikonversi ke format TensorFlow Lite (TFLite). Format TFLite dipilih karena dirancang khusus untuk *deployment* model *deep learning* pada perangkat seperti *smartphone*, Raspberry Pi, *embedded system*, dan perangkat IoT yang memiliki keterbatasan memori, kapasitas penyimpanan, serta daya komputasi dibandingkan dengan komputer desktop atau GPU server (Jocher, 2024).

Proses optimasi dilakukan dengan menerapkan teknik Quantization INT8 melalui *Ultralytics export API*. Teknik ini bertujuan untuk mentransformasi representasi bobot model dari presisi *floating-point* 32-bit (FP32) menjadi *integer* 8-bit (INT8) (Sapkota & Karkee, 2026). Secara konseptual, kuantisasi bekerja dengan memetakan rentang nilai kontinu pada representasi *floating-point* 32-bit (FP32) ke rentang nilai diskrit yang lebih sempit pada representasi integer 8-bit (INT8). Proses ini umumnya melibatkan penentuan parameter *scale* dan *zero-point* yang mendefinisikan bagaimana nilai *floating-point* dipetakan ke nilai integer

terdekat. Pemilihan teknik ini didasarkan pada prinsip bahwa transformasi representasi numerik dari *floating-point* ke *integer* presisi rendah dapat mengurangi jejak memori dan latensi komputasi tanpa mengorbankan akurasi secara signifikan (Gholami et al., 2021). Secara lebih spesifik pada konteks implementasi, dokumentasi resmi Ultralytics menyatakan bahwa kuantisasi INT8 dapat mereduksi ukuran berkas model hingga sekitar 4x (75%) dibandingkan representasi FP32, sekaligus mempercepat waktu inferensi melalui optimalisasi operasi aritmatika integer pada tingkat perangkat keras (Ultralytics, 2026). Hal ini menjadikan teknik kuantisasi INT8 dapat diterapkan pada perangkat dengan sumber daya terbatas seperti Raspberry Pi atau perangkat berbasis ARM lainnya.

Namun, penentuan parameter *scale* dan *zero-point* tidak dapat dilakukan secara langsung, melainkan memerlukan proses kalibrasi terlebih dahulu. Pada skema *Post-Training Quantization* (PTQ), model yang telah selesai dilatih dikalibrasi menggunakan *representative dataset*, yaitu sejumlah kecil data yang mewakili distribusi data pelatihan. Dataset ini digunakan untuk memperkirakan rentang dinamis (*dynamic range*) nilai aktivasi sehingga *scale*, *zero-point*, dan *clipping range* dapat ditentukan sebelum model dikonversi ke representasi integer 8-bit (Gholami et al., 2021).

Kehadiran *representative dataset* dalam proses ini menjadi krusial sebagai sampel kalibrasi untuk menentukan rentang dinamis nilai aktivasi pada citra apusan mikroskopis, sehingga dapat meminimalkan risiko penurunan akurasi akibat efek pembulatan (*quantization noise*) (Gholami et al., 2021). Meskipun komputasi internal model dilakukan menggunakan representasi integer berpresisi rendah, format data pada antarmuka masukan (*input*) dan keluaran (*output*) dapat berbeda bergantung pada skema kuantisasi yang digunakan. Secara umum, pendekatan tersebut bertujuan meningkatkan efisiensi komputasi tanpa mengubah mekanisme inferensi model secara signifikan (Gholami et al., 2021).

2.7 Metrik Evaluasi

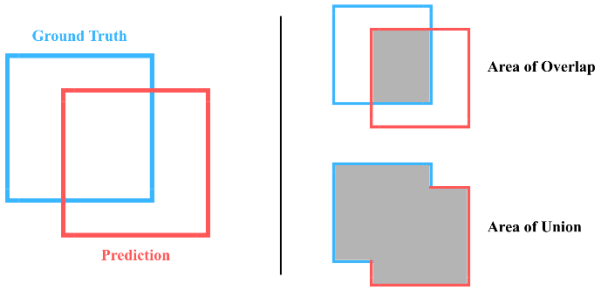
Evaluasi performa sistem deteksi objek berbasis *deep learning* memerlukan kerangka metrik yang tepat agar hasil yang diperoleh dapat diinterpretasikan secara valid serta dibandingkan secara adil antararsitektur. Pemilihan metrik tidak hanya bersifat teknis, tetapi juga mencerminkan aspek performa yang diprioritaskan sesuai dengan konteks aplikasi yang dikaji. Dalam penelitian deteksi bakteri TB, metrik evaluasi berfokus pada kemampuan model dalam mendeteksi objek berukuran kecil secara akurat sekaligus meminimalkan kesalahan deteksi yang berdampak klinis.

2.7.1 *Confusion Matrix* dan Intersection over Union

Seluruh metrik evaluasi dalam deteksi objek berakar pada dua konsep utama, yaitu *confusion matrix* dan *Intersection over Union* (IoU), yang saling berkaitan dalam menentukan kualitas prediksi model. IoU digunakan sebagai ukuran kesesuaian antara *bounding box* hasil prediksi dan *ground truth*, yang dinyatakan sebagai rasio antara luas irisan terhadap luas gabungan dari kedua area tersebut (Terven & Cordova-Esparza, 2024). Secara matematis, perhitungan IoU dirumuskan pada Persamaan (2.1)

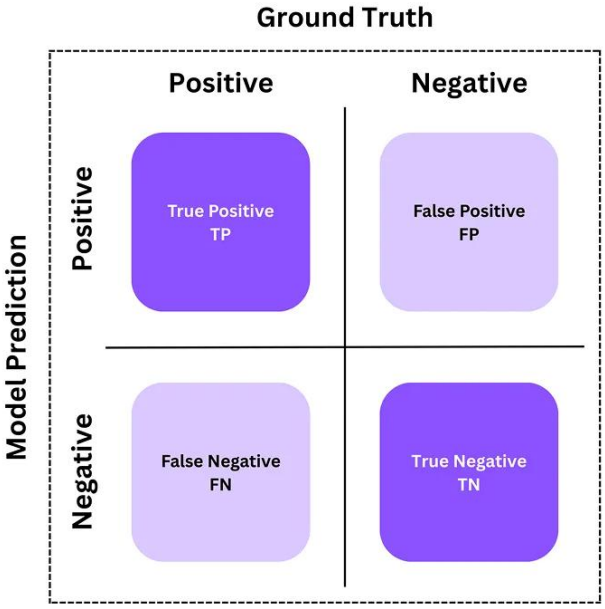
$$\text{IoU} = \frac{Area\ of\ Overlap}{Area\ of\ Union} \qquad (2.1)$$

Untuk memperjelas konsep tersebut, ilustrasi perhitungan IoU antara *bounding box* prediksi dan *ground truth* ditunjukkan pada Gambar 2.1.



Gambar 2.1 Ilustrasi perhitungan Intersection over Union (IoU)

Nilai IoU ini kemudian dibandingkan dengan suatu ambang batas (*threshold*) untuk menentukan kategori hasil prediksi dalam kerangka *confusion matrix.* Prediksi diklasifikasikan sebagai *True Positive* (TP) apabila nilai IoU melebihi *threshold* yang ditentukan. Sebaliknya, prediksi dengan nilai IoU di bawah *threshold* dikategorikan sebagai *False Positive* (FP), sedangkan *False Negative* (FN) merujuk pada objek *ground truth* yang tidak berhasil dideteksi oleh model atau dapat berupa prediksi yang dihasilkan tidak memenuhi *threshold* IoU yang ditetapkan. Adapun *True Negative* (TN) tidak didefinisikan secara bermakna dalam deteksi objek karena jumlah area tanpa objek dalam citra bersifat tidak terbatas. Untuk memberikan gambaran yang lebih jelas mengenai hubungan antar komponen evaluasi tersebut, *confusion matrix* dalam konteks deteksi objek disajikan pada Gambar 2.2.



Gambar 2.2 *Confusion Matrix*

Sumber: Vina (2025b)

2.7.2 *Precision*, *Recall*, dan *F1-Score*

Berdasarkan komponen *confusion matrix*, dua metrik utama yang digunakan untuk mengevaluasi performa deteksi adalah *Precision* dan *Recall*. *Precision* mengukur proporsi prediksi yang benar terhadap seluruh prediksi yang dihasilkan model, sedangkan *Recall* mengukur proporsi objek *ground truth* yang berhasil dideteksi (Terven & Cordova-Esparza, 2024). Secara matematis, *Precision* dan *Recall* dapat dirumuskan pada Persamaan (2.2) dan (2.3).

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$

a. TP = *True Positive*, yaitu total data *true* yang berhasil diprediksi *true*.
b. FP = *False Positive*, yaitu total data *false*, namun diprediksi *true.*
c. FN = *False Negative*, yaitu total data *true*, namun diprediksi *false*

Kedua metrik ini memiliki hubungan *trade-off* yang dikendalikan oleh nilai *confidence threshold*. Penurunan *threshold* akan meningkatkan *Recall* namun menurunkan *Precision*, sedangkan peningkatan *threshold* akan menghasilkan kondisi sebaliknya (Vina, 2025a). Oleh

karena itu, diperlukan metrik tambahan yang mampu merepresentasikan keseimbangan antara keduanya.

F1-score digunakan sebagai ukuran harmonik antara *Precision* dan *Recall*, sehingga memberikan representasi tunggal terhadap performa model yang mempertimbangkan kedua aspek tersebut secara simultan. Secara matematis, F1-score dirumuskan pada Persamaan (2.4).

$$\mathrm{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.4)$$

Nilai F1-score yang tinggi menunjukkan bahwa model mampu menjaga keseimbangan yang baik antara *Precision* dan *Recall*. Dalam konteks deteksi bakteri TB, metrik ini menjadi penting karena model diharapkan tidak hanya mampu mendeteksi sebanyak mungkin bakteri (*Recall* tinggi), tetapi juga meminimalkan kesalahan deteksi terhadap artefak (*Precision* tinggi).

Dalam konteks diagnosis tuberkulosis, terdapat asimetri konsekuensi antara *false positive* dan *false negative*. *False negative* berpotensi menyebabkan *missed diagnosis*, sedangkan *false positive* hanya memerlukan verifikasi lanjutan. Oleh karena itu, *Recall* dapat menjadi prioritas utama pada lingkungan medis, sementara F1-score digunakan untuk mengevaluasi keseimbangan performa model secara keseluruhan (Vina, 2025a).

2.7.3 Average *Precision* dan Mean Average *Precision*

Untuk memberikan evaluasi yang lebih komprehensif, digunakan metrik *Average Precision* (AP) dan *mean Average Precision* (mAP). AP dihitung sebagai luas area di bawah kurva *Precision–Recall*, yang menggambarkan performa model pada berbagai nilai *threshold*.

Sementara itu, mAP merupakan rata-rata nilai AP pada seluruh kelas atau pada berbagai *threshold* IoU, seperti mAP@50 dan mAP@50–95. Dalam penelitian deteksi bakteri TB yang menggunakan satu kelas objek, nilai mAP secara langsung merepresentasikan performa keseluruhan model dalam mendeteksi bakteri.

Nilai *threshold* IoU berperan sebagai hiperparameter evaluasi yang menentukan tingkat ketatnya kriteria kesesuaian lokasi. Threshold sebesar 0,5 (dikenal sebagai mAP@50) merupakan standar yang umum digunakan, di mana prediksi dianggap benar jika memiliki minimal 50% tumpang tindih dengan *ground truth*. Sedangkan mAP@50-95 merupakan rata-

rata dari mAP yang dihitung pada berbagai ambang batas IoU, mulai dari 0,50 hingga 0,95. Ini memberikan gambaran komprehensif tentang kinerja model di berbagai tingkat kesulitan deteksi. Penggunaan *threshold* yang lebih tinggi, seperti 0,75 atau 0,95, menuntut presisi lokalisasi yang lebih tinggi dan umumnya menghasilkan nilai *Average Precision* (AP) yang lebih rendah karena lebih banyak prediksi yang tidak memenuhi kriteria tersebut. Perhitungan mAP dapat ditulis pada Persamaan (2.5).

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} APi \qquad (2.5)$$

a. N = jumlah kelas
b. AP = *Average Precision*

2.7.4 Metrik Efisiensi Komputasi dan Relevansinya

Selain metrik performa, evaluasi sistem deteksi objek dalam konteks klinis perlu mempertimbangkan aspek efisiensi komputasi yang menentukan kelayakan implementasi pada lingkungan nyata. Hal ini menjadi krusial terutama pada skenario *edge deployment*, di mana keterbatasan sumber daya perangkat dapat menjadi faktor pembatas utama. Oleh karena itu, penelitian ini mengadopsi tiga metrik utama efisiensi, yaitu *inference time*, jumlah parameter, dan ukuran model.

Inference time atau latensi per citra (dalam milidetik) merepresentasikan waktu yang dibutuhkan model untuk memproses satu citra. Metrik ini secara langsung menentukan *throughput* sistem, yaitu jumlah citra yang dapat diproses dalam satuan waktu tertentu. Dalam konteks sistem pendukung diagnosis *real-time* di laboratorium, latensi yang rendah menjadi kebutuhan utama.

Jumlah parameter model, yang dinyatakan dalam jutaan parameter (M), dapat digunakan sebagai salah satu indikator kompleksitas arsitektur. Secara umum, model dengan jumlah parameter yang lebih besar cenderung membutuhkan memori RAM/VRAM yang lebih besar selama inferensi, serta semakin besar potensi kapasitas representasi model. Namun demikian, peningkatan jumlah parameter tidak selalu berbanding lurus dengan peningkatan performa, khususnya pada tugas deteksi objek berukuran kecil seperti bakteri TB. Oleh karena itu, jumlah parameter perlu dianalisis bersama metrik akurasi dan efisiensi komputasi.

Ukuran file model setelah dikonversi ke format *deployment*, seperti TensorFlow Lite (TFLite), memberikan gambaran konkret mengenai kebutuhan penyimpanan pada perangkat target. Metrik ini menjadi sangat penting pada perangkat dengan keterbatasan sumber daya dan kapasitas memori penyimpanan. Model berukuran lebih kecil umumnya lebih mudah

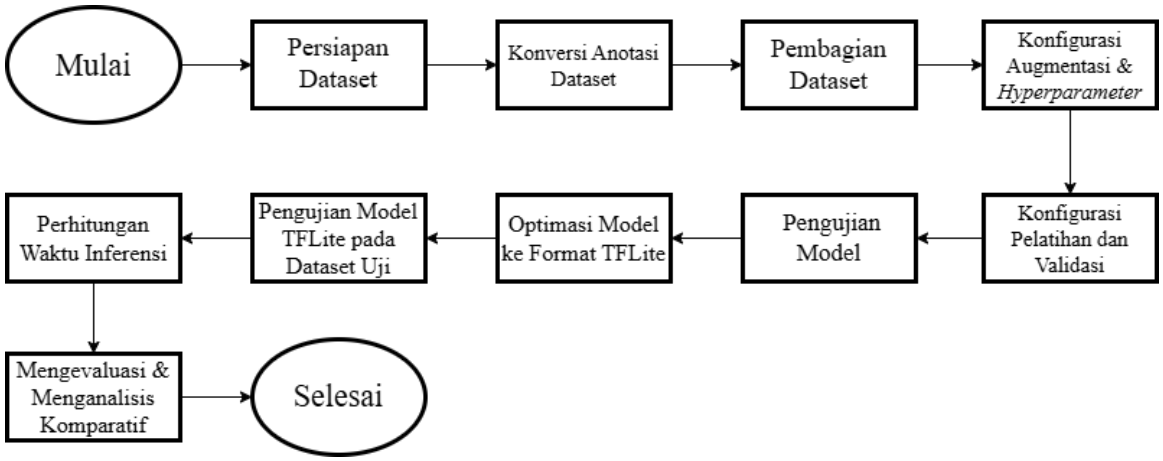
didistribusikan dan diimplementasikan, terutama pada skenario penggunaan berskala luas atau fasilitas dengan infrastruktur terbatas.

BAB III
METODOLOGI PENELITIAN

Pada bagian ini dijelaskan perancangan penelitian yang mencakup gambaran alur program yang digunakan. Alur tersebut disajikan dalam bentuk *flowchart* yang ditampilkan pada Gambar 3.1, yang menggambarkan secara sistematis tahapan-tahapan dalam proses penelitian ini, mulai dari persiapan data hingga evaluasi model.

Penelitian ini menggunakan pendekatan eksperimental komparatif dengan paradigma kuantitatif. Pendekatan ini dipilih karena tujuan utama penelitian adalah membandingkan performa delapan varian arsitektur YOLO (YOLOv8n, YOLOv8s, YOLOv10n, YOLOv10s, YOLO11n, YOLO11s, YOLO26n, dan YOLO26s) dari empat generasi yang berbeda secara sistematis dan terukur dalam mendeteksi *Mycobacterium tuberculosis* (MTB) pada citra apusan mikroskopis sputum. Dengan mempertahankan *dataset*, pembagian data, konfigurasi pelatihan, prosedur augmentasi, hiperparameter, jumlah *epoch*, dan lingkungan evaluasi secara konsisten, perbedaan performa yang diamati dapat lebih valid dikaitkan dengan perbedaan arsitektur model.

Seluruh eksperimen dilakukan pengulangan sebanyak 3 kali menggunakan *random seed* yang berbeda-beda, interpretasi hasil difokuskan pada analisis deskriptif-komparatif yang menekankan pada pola dan tren yang konsisten antar-metrik evaluasi. Oleh karena itu, hasil yang diperoleh lebih tepat diinterpretasikan sebagai indikasi atau kecenderungan performa pada konfigurasi percobaan.



Gambar 3.1 Flowchart Rancangan Penelitian

3.1 Persiapan *Dataset*

Dataset yang digunakan dalam penelitian ini adalah Tuberculosis Image Dataset yang tersedia secara publik melalui platform Kaggle (https://www.kaggle.com/datasets/saife245/tuberculosis-image-datasets/data). *Dataset* terdiri dari citra mikroskopis apusan sputum dengan resolusi asli 1.632 × 1.224 piksel. Berdasarkan hasil verifikasi terhadap dataset yang diunduh dari Kaggle, *dataset* yang digunakan dalam penelitian ini terdiri atas 1.265 citra mikroskopis sputum beserta 1.265 file anotasi berformat XML. Seluruh anotasi merepresentasikan satu kelas objek, yaitu *TBbacillus*. Meskipun deskripsi dataset pada halaman Kaggle menyebutkan bahwa dataset terdiri atas 928 citra dengan 3.734 anotasi *bounding box*, hasil verifikasi terhadap berkas dataset menunjukkan jumlah sebanyak 1.265 citra. Oleh karena itu, penelitian ini menggunakan keseluruhan data yang tersedia pada saat proses pengunduhan.

Citra diperoleh melalui proses akuisisi menggunakan perangkat laboratorium dan kamera *handphone* yang dirancang untuk menangkap sampel sputum secara mikroskopis. Meskipun karakteristik pada *dataset* ini relatif konsisten, citra tetap mengandung variasi *debris* dan artefak latar belakang yang cukup tinggi, sehingga menimbulkan tantangan dalam membedakan bakteri TB dari non-objek. Selain itu, *Dataset* yang digunakan pada penelitian ini tidak menyertakan metadata seperti identitas pasien, nomor preparat (*slide*), namun terdapat informasi asal institusi untuk setiap citra yaitu berasal dari Mulago National Referral Hospital.

3.2 Konversi Anotasi *Dataset*

Sebelum digunakan pada proses pelatihan model, seluruh data anotasi pada *dataset* yang telah diunduh lalu dikonversi terlebih dahulu ke dalam berkas teks berekstensi (*.txt*) yang struktur isinya mengikuti aturan penulisan anotasi standar YOLO. Pada format ini, setiap objek di dalam citra direpresentasikan menggunakan satu baris yang terdiri atas lima nilai numerik. Kelima nilai tersebut meliputi indeks kelas (*class id*), koordinat titik pusat *bounding box* pada sumbu-x (*x_center*), koordinat titik pusat pada sumbu-y (*y_center*), lebar *bounding box* (*width*), dan tinggi *bounding box* (*height*). Seluruh nilai koordinat dan dimensi tersebut telah dinormalisasi terhadap ukuran citra asli sehingga nilainya berada pada rentang 0 hingga 1. Setiap citra akan memiliki satu berkas teks (*.txt*) anotasi yang terpisah dan saling berpasangan dengan gambar *dataset*. Contoh isi berkas anotasi hasil konversi dari format XML ke format YOLO ditunjukkan pada Gambar 3.2.

```
0 0.234687 0.516667 0.018125 0.033333
0 0.320937 0.690417 0.026875 0.029167
0 0.117188 0.738750 0.053125 0.030833
0 0.669375 0.915000 0.017500 0.026667
0 0.655312 0.736667 0.025625 0.036667
0 0.505000 0.395417 0.015000 0.027500
0 0.603750 0.020000 0.012500 0.023333
0 0.772188 0.269167 0.023125 0.033333
```

Gambar 3.2 Contoh Isi File Anotasi Format YOLO

Setelah proses konversi anotasi selesai, dilakukan verifikasi integritas data untuk memastikan kesesuaian antara citra dan label. Tahap ini mencakup pengecekan jumlah berkas citra dan anotasi, validasi nilai koordinat *bounding box* agar tetap berada pada rentang 0 hingga 1, serta identifikasi berkas yang rusak, kosong, atau tidak lengkap. Apabila ditemukan berkas citra atau anotasi yang mengalami kerusakan (*corrupted*) maupun tidak lengkap akibat ketiadaan pasangan berkas (*mismatch*), berkas tersebut akan dieliminasi dari *dataset* guna menghindari kegagalan sistem (*error*) saat proses pelatihan model. Sementara itu, untuk berkas citra yang diidentifikasi kosong atau tidak mengandung objek bakteri, citra tersebut tetap dipertahankan. Citra tersebut digunakan selama pelatihan untuk membantu model membedakan latar belakang dari objek target dan berpotensi mengurangi deteksi positif palsu (*false positive*). Verifikasi ini dilakukan secara menyeluruh untuk meminimalkan potensi kesalahan dan memastikan stabilitas selama proses pelatihan model.

3.3 Pembagian *Dataset*

Tahap pembagian *dataset* dilakukan setelah seluruh proses konversi anotasi dan verifikasi data selesai dilakukan. Pembagian *dataset* bertujuan untuk memisahkan data yang digunakan pada proses pelatihan, validasi, dan pengujian sehingga evaluasi performa model dapat dilakukan secara objektif dan terukur. Pada penelitian ini, *dataset* dibagi ke dalam tiga subset yang saling terpisah, yaitu *training set*, *validation set*, dan *testing set*, dengan rasio pembagian sebesar 70:20:10 (Jocher & Raimbekovm, 2024).

Berdasarkan rasio tersebut, diperoleh sebanyak 885 citra untuk data pelatihan, 253 citra untuk data validasi, dan 127 citra untuk data pengujian. Data pelatihan digunakan untuk proses pembelajaran model dalam mengenali pola visual bakteri *Mycobacterium tuberculosis* pada citra mikroskopis. Sementara itu, data validasi digunakan untuk memantau performa model selama proses pelatihan, khususnya dalam mendeteksi indikasi *overfitting* dan mengevaluasi kemampuan generalisasi model terhadap data yang tidak digunakan secara langsung dalam

proses pembelajaran. Adapun data pengujian digunakan pada tahap akhir eksperimen untuk mengevaluasi performa akhir model terhadap data yang benar-benar belum pernah dilihat sebelumnya (*unseen data*).

Proses pembagian *dataset* dilakukan secara acak pada tingkat citra (*image-level split*) menggunakan *random seed* deterministik (seed = 42, 45, dan 50) untuk memastikan reproduktibilitas eksperimen. Dengan penggunaan *seed* yang konsisten, komposisi data pada setiap subset akan tetap sama meskipun proses pembagian dijalankan ulang. Hal tersebut penting untuk menjaga konsistensi hasil eksperimen dan memungkinkan penelitian direplikasi pada kondisi yang identik.

Setelah proses pembagian *dataset* selesai, tahap selanjutnya adalah konfigurasi *dataset* didefinisikan melalui berkas YAML yang memuat informasi jalur direktori *dataset*, jumlah kelas objek, serta nama kelas yang digunakan dalam proses pelatihan. Berkas konfigurasi ini digunakan oleh *framework* YOLO untuk mengakses *dataset* secara terstruktur selama proses pelatihan dan evaluasi model.

3.4 Konfigurasi Augmentasi dan Hiperparameter

3.4.1 Konfigurasi Augmentasi

Pada penelitian ini, augmentasi data diterapkan menggunakan konfigurasi bawaan (*default augmentation pipeline*) yang disediakan oleh *framework* Ultralytics YOLO (Jocher, 2025). Augmentasi dijalankan secara otomatis selama proses pelatihan melalui fungsi model .train() tanpa implementasi kode augmentasi kustom secara eksplisit. Konfigurasi augmentasi yang digunakan mengikuti parameter standar *framework* Ultralytics, yang secara otomatis aktif ketika parameter augmentasi tertentu tidak di-*override* pada konfigurasi pelatihan.

Penerapan augmentasi bertujuan untuk meningkatkan keragaman data pelatihan secara artifisial sehingga model memiliki kemampuan generalisasi yang lebih baik terhadap variasi citra yang mungkin ditemui pada kondisi nyata. Augmentasi hanya diterapkan pada *training set*, sedangkan data validasi dan pengujian tetap menggunakan citra asli tanpa transformasi agar evaluasi performa model tetap objektif.

Berdasarkan konfigurasi pelatihan yang digunakan oleh *framework* Ultralytics, augmentasi yang diterapkan meliputi *horizontal flipping* (fliplr = 0,5), translasi (translate = 0,1), penyesuaian ruang warna HSV (hsv_h = 0,015, hsv_s = 0,7, hsv_v = 0,4), *mosaic augmentation*, *RandAugment* (auto_augment = randaugment), dan *random erasing* (erasing = 0,4). Sementara itu, augmentasi seperti *vertical flipping*, rotasi besar, *mixup*, dan *cutmix* tidak

digunakan karena konfigurasi default yang diterapkan tidak mengaktifkan transformasi tersebut.

3.4.2 Konfigurasi Hiperparameter

Untuk memastikan keadilan perbandingan antararsitektur, seluruh model dilatih menggunakan konfigurasi hiperparameter yang identik. Pendekatan ini dilakukan agar perbedaan performa yang diperoleh dapat dikaitkan dengan perbedaan arsitektur model, bukan dengan variasi konfigurasi pelatihan.

Sebelum menentukan konfigurasi akhir, dilakukan uji coba awal pada model YOLOv8n menggunakan jumlah *epoch* sebanyak 500 dengan nilai *patience* 100 *epoch*, mengikuti nilai default yang lebih besar untuk mengeksplorasi potensi performa maksimum model. Hasil uji coba tersebut menunjukkan bahwa nilai *train loss* terus mengalami penurunan, sedangkan *val loss* justru menunjukkan tren peningkatan setelah melewati sejumlah *epoch* tertentu, yang mengindikasikan terjadinya *overfitting*. Berdasarkan temuan ini, serta mempertimbangkan efisiensi penggunaan sumber daya komputasi untuk melatih delapan arsitektur model sekaligus, konfigurasi *epoch* dan *patience* untuk seluruh eksperimen selanjutnya disesuaikan menjadi lebih rendah.

Setelah mempertimbangkan dari pelatihan dengan menggunakan 500 *epoch*, selanjutnya pelatihan dilakukan dengan jumlah *epoch* sebanyak 100. Untuk *batch size* sebanyak 16 dengan ukuran citra input 640 × 640 piksel, sesuai dengan konfigurasi *default* yang ditetapkan oleh *framework* Ultralytics (Jocher & Munawar, 2023). Mekanisme *early stopping* diterapkan dengan nilai *patience* sebesar 30 *epoch*, yang lebih rendah dibandingkan nilai default sebesar 100 *epoch*, untuk mengefisienkan penggunaan sumber daya komputasi selama pelatihan delapan arsitektur model sekaligus mencegah *overfitting*. Seluruh eksperimen menggunakan optimizer Stochastic Gradient Descent (SGD) dengan *initial learning rate* sebesar 0,01 yang merupakan nilai *default* yang ditetapkan oleh Ultralytics (Jocher & Munawar, 2023).

Untuk meningkatkan reliabilitas hasil pelatihan, setiap arsitektur dilatih sebanyak tiga kali menggunakan *random* seed yang berbeda, yaitu 42, 45, dan 50. Penggunaan beberapa nilai *seed* bertujuan untuk mengurangi pengaruh variasi akibat inisialisasi bobot dan proses pengacakan data selama pelatihan, sehingga kestabilan performa model dapat dievaluasi. Nilai performa yang dilaporkan pada penelitian ini disajikan dalam bentuk rata-rata (mean) dan standar deviasi dari ketiga hasil pelatihan tersebut. Selanjutnya, model dengan nilai mAP@50

tertinggi dari ketiga pengulangan dipilih sebagai *best checkpoint* untuk digunakan pada tahap evaluasi inferensi dan proses konversi ke format TensorFlow Lite INT8.

Selain hiperparameter utama yang ditentukan secara eksplisit, *framework* Ultralytics juga menerapkan beberapa konfigurasi bawaan selama pelatihan, seperti *warmup* selama 3 *epoch*, bobot *loss* untuk *bounding box* (box = 7,5), klasifikasi (cls = 0,5), dan *Distribution Focal Loss* (dfl = 1,5), serta *threshold* IoU untuk *Non-Maximum Suppression* sebesar 0,7 (Jocher & Munawar, 2023). Seluruh model diinisialisasi menggunakan *pretrained weights* dari *dataset* COCO melalui pendekatan *transfer learning*.

3.5 Konfigurasi Pelatihan dan Validasi Model

Setelah proses konversi label, pembagian *dataset*, dan penentuan hiperparameter selesai dilakukan, tahapan berikutnya adalah pelatihan dan validasi model deteksi objek, yang berjalan secara terintegrasi dalam satu proses pelatihan menggunakan kerangka kerja Ultralytics YOLO. Proses ini bertujuan untuk memungkinkan model mempelajari pola visual dan karakteristik bakteri TB pada citra mikroskopis sputum, sehingga model mampu melakukan deteksi lokasi objek secara otomatis, sekaligus memantau performa model pada data validasi guna mendeteksi potensi *overfitting*.

Sebelum proses pelatihan penuh dilakukan, terlebih dahulu dilakukan tahap uji coba awal selama 3 *epoch*. Tahap ini bertujuan untuk memvalidasi fungsionalitas seluruh alur arsitektur model, memastikan kelancaran alokasi memori pada GPU, serta mendeteksi potensi kegagalan sistem (*runtime error*) sebelum model mengeksekusi keseluruhan iterasi pelatihan.

Penelitian ini melatih delapan varian model dari empat generasi arsitektur YOLO, yaitu YOLOv8, YOLOv10, YOLO11, dan YOLO26, yang masing-masing terdiri atas varian *nano* dan *small*. Pemilihan arsitektur tersebut didasarkan pada perkembangan desain YOLO modern yang berfokus pada peningkatan efisiensi, stabilitas pelatihan, dan kemampuan deteksi objek kecil, yang relevan dengan karakteristik basil TB pada citra mikroskopis. Selain itu, varian ringan seperti *nano* dan *small* dipilih karena lebih sesuai untuk skenario implementasi *edge computing* dengan keterbatasan sumber daya komputasi (Sapkota & Karkee, 2026).

Pemilihan arsitektur YOLOv8, YOLOv10, YOLO11, dan YOLO26 dalam penelitian ini didasarkan pada karakteristik teknis masing-masing generasi yang telah dijelaskan pada Subbab 2.4. Keempat arsitektur ini secara bersama-sama membentuk matriks perbandingan lintas generasi yang relevan untuk mengevaluasi perkembangan kemampuan deteksi objek berukuran kecil pada citra mikroskopis TB.

Seluruh model dilatih menggunakan konfigurasi pelatihan yang identik untuk menjaga konsistensi eksperimen dan memastikan bahwa perbedaan performa yang diperoleh berasal dari perbedaan arsitektur model. Data validasi digunakan secara otomatis pada setiap akhir *epoch* selama proses pelatihan untuk memantau performa model dan mendeteksi potensi *overfitting*, mengacu pada rasio pembagian data dan jumlah detail citra yang telah dijabarkan pada Subbab 3.3.

Model terbaik (*best* model) dipilih berdasarkan performa pada data validasi, bukan berdasarkan *epoch* terakhir pelatihan. Pemilihan ini dilakukan karena performa validasi lebih merepresentasikan kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya. Adapun metrik yang menjadi acuan utama sistem dalam menentukan model terbaik tersebut adalah *fitness score*, yaitu kombinasi tertimbang dari *Precision*, *Recall*, mAP@50, dan mAP@50–95, dengan bobot default sebesar 1,0 pada mAP@50–95 (Jocher & Raimbekovm, 2026). Dengan demikian, *checkpoint* best.pt merepresentasikan *epoch* dengan nilai mAP@50-95 tertinggi yang dicapai sepanjang tahapan validasi.

3.6 Pengujian Model

Pengujian model dilakukan menggunakan subset data pengujian yang telah dipisahkan pada tahap pembagian *dataset*. Rasio pembagian data serta jumlah detail citra untuk masing-masing subset merujuk pada ketentuan yang telah dijabarkan pada Subbab 3.3. Berbeda dengan data validasi yang digunakan selama proses pelatihan, data pengujian digunakan sebagai data yang tidak pernah dilihat model (unseen data) untuk mengevaluasi kemampuan generalisasi model secara objektif pada tahap akhir penelitian.

Pada tahap ini, model terbaik (*best model*) dalam format PyTorch (*.pt*) atau file best.pt dimuat kembali menggunakan *framework* Ultralytics YOLO. File best.pt merupakan *checkpoint* dengan nilai *fitness score* tertinggi sebagaimana dijelaskan pada Subbab 3.5. Selanjutnya, setiap model dievaluasi menggunakan fungsi model.val() pada *test set* untuk memperoleh berbagai metrik performa deteksi objek. Metrik yang dihitung meliputi *Precision*, *Recall*, *F1-Score*, *mAP@50*, dan *mAP@50–95*. Seluruh hasil evaluasi kemudian dikompilasi ke dalam bentuk tabel dan dicatat untuk memudahkan analisis komparatif antararsitektur.

3.7 Optimasi Model ke Format TensorFlow Lite

Pertimbangan atas skenario *edge computing* tersebut mendasari keputusan untuk mengarahkan proses optimasi model pada penelitian ini menuju format yang ringan dan efisien,

yaitu TensorFlow Lite (TFLite). Pengujian efisiensi model pada penelitian ini dilakukan melalui simulasi menggunakan TFLite interpreter pada lingkungan Google Colab, tanpa melibatkan *deployment* langsung pada perangkat *edge* fisik seperti Raspberry Pi atau perangkat lainnya.

Proses konversi dieksekusi menggunakan fungsi ekspor pada *framework* Ultralytics dengan menetapkan parameter format='tflite' dan int8=True. Selain itu, parameter data=DATA_YAML_PATH dilibatkan untuk menyediakan *representative dataset* yang bersumber dari file konfigurasi data penelitian. Berdasarkan karakteristik teknis tersebut, optimasi yang dihasilkan dikategorikan sebagai *Full Integer* Post-Training Quantization (PTQ). Dalam skema ini, kompresi tidak hanya dilakukan pada bobot (*weights*) internal model, melainkan juga pada nilai aktivasi di setiap layer arsitektur YOLO yang ditransformasikan dari presisi *floating-point* 32-bit (FP32) menjadi integer 8-bit (INT8).

Setelah seluruh tahapan konversi selesai, berkas model TensorFlow Lite hasil optimasi akan tersimpan dalam format .tflite. Ukuran berkas akhir ini kemudian dicatat dan dibandingkan dengan ukuran model PyTorch asli (.pt) untuk mengukur efisiensi ruang penyimpanan. Melalui penerapan kuantisasi INT8 ini, model diharapkan mampu mereduksi ukuran penyimpanan secara signifikan sekaligus mempercepat waktu inferensi, dengan tetap mempertahankan performa deteksi bakteri *Mycobacterium tuberculosis* yang optimal untuk skenario implementasi pada perangkat *edge*.

3.8 Pengujian Model TFLite pada *Dataset* Uji

Setelah masing-masing model berhasil dikonversi ke dalam format TensorFlow Lite (TFLite), tahap selanjutnya adalah melakukan pengujian terhadap model hasil konversi menggunakan *dataset* uji. Pengujian ini dilakukan untuk membandingkan performa model sebelum dan sesudah proses konversi ke format TFLite INT8, sehingga dapat diketahui pengaruh proses kuantisasi terhadap kemampuan deteksi objek. Selain itu, evaluasi terhadap model TFLite dilakukan sebagai langkah awal dalam menilai efisiensi model untuk skenario implementasi pada perangkat *edge computing*, meskipun pengujian pada penelitian ini masih dilakukan pada lingkungan CPU Google Colab dan belum menggunakan perangkat *edge* fisik seperti Raspberry Pi. Metrik evaluasi yang digunakan meliputi Precision, Recall, mAP@50, mAP@50-95, dan F1-Score.

Pengujian model TFLite dilakukan menggunakan *library* Ultralytics yang mendukung pemuatan dan evaluasi berkas model berformat .tflite secara langsung tanpa memerlukan

konversi tambahan dan menyertakan *path* berkas masing-masing model. Sebelum proses evaluasi dimulai, direktori penyimpanan hasil pengujian disiapkan terlebih dahulu secara otomatis guna memastikan seluruh keluaran hasil pengujian dapat tersimpan secara terstruktur dan rapi.

Proses evaluasi pada masing-masing model dilakukan menggunakan fungsi val() yang disediakan oleh library Ultralytics. Sama seperti tahap pengujian sebelum dikonversi, fungsi ini dikonfigurasi dengan parameter data yang merujuk pada berkas konfigurasi *dataset* dalam format YAML, parameter split='test' yang mengarahkan proses evaluasi untuk menggunakan subset data uji secara eksklusif, serta parameter imgsz=640 yang menetapkan ukuran citra masukan sebesar 640×640 piksel. Hasil evaluasi model sebelum dan sesudah konversi kemudian dibandingkan untuk mengidentifikasi perubahan performa akibat proses kuantisasi TFLite INT8, sekaligus mengevaluasi apakah efisiensi yang diperoleh masih mempertahankan tingkat akurasi deteksi yang memadai. Selanjutnya, seluruh nilai metrik dirangkum ke dalam tabel untuk dianalisis secara komparatif.

3.9 Metode Perhitungan Waktu Inferensi

Pengujian inferensi model dilakukan pada platform Google Colab dengan runtime CPU sebagai simulasi lingkungan komputasi terbatas, bukan pada perangkat *edge computing* fisik seperti Raspberry Pi atau perangkat *embedded* lainnya. Oleh karena itu, evaluasi efisiensi *deployment* dilakukan menggunakan metrik komparatif berupa ukuran model TFLite dan waktu inferensi yang diukur pada lingkungan CPU Google Colab. Metrik tersebut digunakan sebagai indikator awal untuk mengidentifikasi arsitektur yang paling berpotensi diterapkan pada perangkat dengan keterbatasan sumber daya komputasi. Dengan demikian, hasil yang diperoleh dalam penelitian ini belum dapat sepenuhnya merepresentasikan performa aktual pada perangkat target. Pengujian empiris secara langsung pada perangkat *edge* fisik direkomendasikan sebagai pengembangan penelitian lanjutan untuk memvalidasi performa inferensi, konsumsi memori, serta efisiensi energi pada kondisi implementasi nyata.

Tahap pengukuran inferensi bertujuan untuk mengevaluasi performa model yang telah dioptimasi dalam format TensorFlow Lite (TFLite) dengan kuantisasi INT8. Pengukuran dilakukan menggunakan skrip pengujian (*benchmark*) yang dirancang khusus untuk mensimulasikan kondisi inferensi yang nyata dan konsisten. Proses diawali dengan memuat model TFLite menggunakan *interpreter* dari TensorFlow Lite, kemudian mengalokasikan

memori yang dibutuhkan serta membaca konfigurasi input dan output model secara otomatis, termasuk ukuran citra masukan yaitu 640×640 piksel dengan tiga kanal warna.

Sebelum pengukuran utama dimulai, model terlebih dahulu menjalani fase *warm-up* sebanyak 10 iterasi menggunakan data acak. Fase ini dilakukan untuk memastikan sistem komputasi, seperti CPU, *cache*, dan alokasi memori, telah berada dalam kondisi stabil sehingga hasil pengukuran tidak terdistorsi oleh *overhead* yang terjadi pada saat inisialisasi awal model.

Dataset pengujian yang digunakan dalam penelitian ini terdiri dari 127 citra. Namun, khusus untuk mengukur waktu inferensi, digunakan satu citra representatif dari *dataset* pengujian yang diproses secara berulang sebanyak 100 kali agar diperoleh estimasi waktu yang stabil dan bebas dari variasi akibat perbedaan konten antarcitra. Waktu inferensi pada setiap pengulangan diukur secara presisi menggunakan fungsi time.perf_counter(), dihitung sejak data citra dimasukkan ke dalam model hingga proses inferensi selesai dijalankan, kemudian dikonversi ke satuan milidetik. Dari keseluruhan 100 pengukuran tersebut dihitung rata-rata waktu inferensi yang mencerminkan kecepatan umum model. Metrik waktu inferensi ini kemudian dibandingkan antarmodel untuk menentukan model mana yang lebih optimal dalam hal kecepatan inferensi, terutama jika model ditargetkan untuk digunakan pada perangkat dengan sumber daya komputasi terbatas.

3.10 Metode Evaluasi dan Analisis Komparatif

Evaluasi dilakukan terhadap seluruh varian model YOLO menggunakan dua kelompok metrik, yaitu metrik performa (*Precision*, Recall, F1-Score, mAP@50, dan mAP@50–95) dan metrik efisiensi komputasi (waktu inferensi, jumlah parameter, serta ukuran model TensorFlow Lite/TFLite). Kombinasi kedua kelompok metrik ini digunakan untuk membandingkan performa antarmodel secara komprehensif, baik dari sisi performa deteksi maupun kelayakannya untuk diimplementasikan pada perangkat dengan sumber daya komputasi terbatas.

Ukuran model diukur dalam satuan megabyte (MB) untuk dua format, yaitu format PyTorch (.pt) sebelum konversi dan format TFLite INT8 (.tflite) setelah konversi. Pengukuran ini digunakan untuk mengestimasi kelayakan *deployment* pada perangkat *edge* yang memiliki keterbatasan memori. Perlu dicatat bahwa pengujian langsung pada perangkat *edge* tidak dilakukan dalam penelitian ini, sehingga ukuran model dan *inference time* berfungsi sebagai indikator komparatif antar-model, bukan sebagai ukuran performa absolut pada target

hardware. Penjelasan masing-masing metrik beserta relevansinya dalam konteks penelitian ini disajikan pada Tabel 3.1.

Tabel 3.1 Metrik Evaluasi Performa dan Efisiensi Model

Metrik	Definisi Singkat	Relevansi dalam Penelitian
Precision	Proporsi prediksi positif yang benar terhadap seluruh prediksi positif yang dihasilkan model, dihitung sebagai TP / (TP + FP)	Mengukur kemampuan model dalam menghindari deteksi palsu (*false positive*) yang dapat muncul akibat keberadaan debris pada citra
Recall	Proporsi objek positif yang berhasil dideteksi terhadap seluruh objek positif yang sebenarnya ada, dihitung sebagai TP / (TP + FN)	Mengukur kemampuan model dalam menemukan seluruh bakteri MTB yang ada pada citra, di mana nilai *Recall* yang rendah berpotensi menurunkan sensitivitas model
F1-Score	Rata-rata harmonik antara precision dan *Recall*, dihitung sebagai 2 × (*Precision* × *Recall*) / (*Precision* + *Recall*)	Memberikan gambaran performa model secara keseluruhan *Precision* dan *Recall*
mAP@50	Rata-rata Average *Precision* seluruh kelas pada ambang IoU sebesar 0,50	Digunakan sebagai metrik utama komparasi antar-arsitektur karena merupakan standar yang paling umum digunakan dalam penelitian deteksi objek berbasis YOLO
mAP@50-95	Rata-rata Average *Precision* pada rentang ambang IoU 0,50 hingga 0,95 dengan interval 0,05	Memberikan evaluasi yang lebih ketat terhadap kualitas lokalisasi *bounding box* dibandingkan mAP@50
Jumlah Parameter	Total bobot yang dapat dilatih dalam arsitektur model, dinyatakan dalam satuan juta parameter (M)	Digunakan sebagai indikator kompleksitas arsitektur dan efisiensi model, serta sebagai pembanding antar-varian *nano* dan *small* dalam konteks keterbatasan komputasi perangkat *edge*
Ukuran Model	Besar file model dalam satuan megabyte (MB), diukur pada format PyTorch (.pt) dan TFLite INT8 (.tflite)	Digunakan untuk mengestimasi kelayakan *deployment* pada perangkat *edge* yang memiliki keterbatasan kapasitas memori
Inference Time	Waktu yang dibutuhkan model untuk memproses satu citra, diukur dalam milidetik (ms), pada format PyTorch dan TFLite INT8	Digunakan sebagai estimasi kecepatan inferensi model pada kondisi sebelum dan sesudah konversi, sebagai salah satu parameter kelayakan *deployment* pada perangkat *edge*

BAB IV
HASIL DAN PEMBAHASAN

Tahapan yang terdapat pada diagram alur Gambar 3.1 diimplementasikan menggunakan bahasa pemrograman Python versi 3.12.13 dengan dukungan beberapa pustaka utama, yaitu PyTorch versi 2.11.0 sebagai backend *deep learning* untuk pelatihan dan inferensi model YOLO melalui kerangka kerja Ultralytics, TensorFlow versi 2.20.0 untuk pengembangan dan *deployment* model *deep learning*, serta OpenCV versi 4.13.0 untuk pengolahan citra. Seluruh implementasi dilakukan secara sistematis sesuai dengan urutan tahapan yang ditunjukkan pada diagram alir penelitian. Proses pengembangan dan eksekusi program dilakukan pada lingkungan Google Colab karena platform ini menyediakan lingkungan komputasi berbasis cloud yang mendukung eksekusi Python secara interaktif dan memiliki akses GPU tanpa memerlukan konfigurasi perangkat keras lokal yang kompleks.

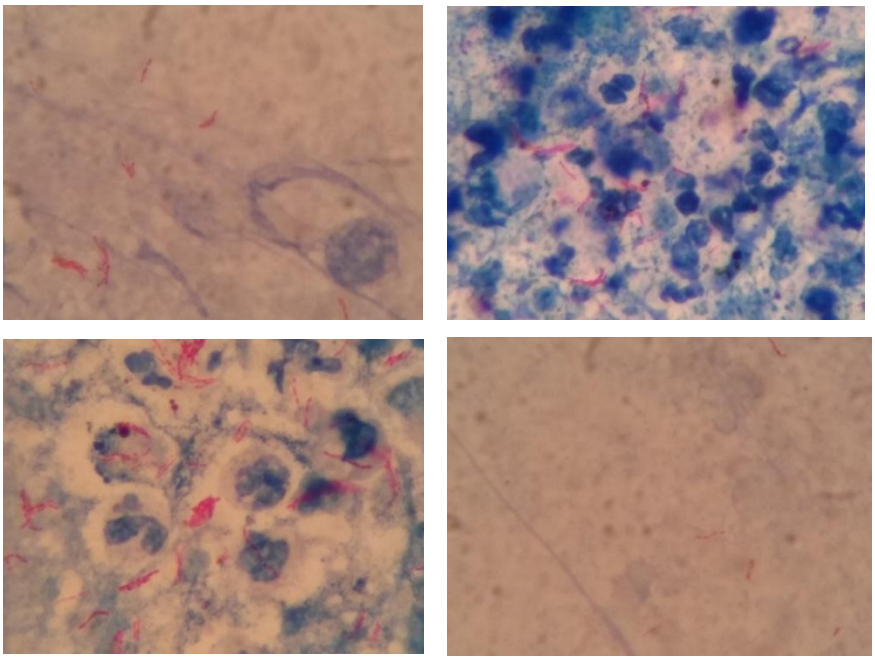
Pada tahap pelatihan model (*training*), penelitian ini memanfaatkan GPU NVIDIA Tesla T4 yang tersedia pada Google Colab dengan kapasitas VRAM sebesar 15 GB, RAM sistem sebesar 12,7 GB, dan ruang penyimpanan sementara sebesar 112,6 GB. Penggunaan GPU diperlukan untuk mempercepat proses komputasi selama pelatihan model *deep learning* yang melibatkan operasi matriks dan konvolusi dalam jumlah besar.

Sementara itu, pengukuran waktu inferensi pada data pengujian dilakukan menggunakan CPU berbasis arsitektur x86_64 dengan prosesor Intel Xeon CPU @ 2.00 GHz yang memiliki dua inti pemrosesan (*2 cores*). Penggunaan CPU dalam pengujian inferensi dilakukan untuk memperoleh gambaran efisiensi komputasi model pada lingkungan yang lebih mendekati skenario *deployment* umum dibandingkan dengan pengukuran berbasis GPU.

Beberapa pustaka Python tambahan juga digunakan untuk mendukung pengelolaan data dan otomatisasi proses penelitian, diantaranya seperti Pustaka os yang digunakan untuk pengelolaan direktori dan file, glob digunakan untuk mengambil kumpulan file berdasarkan pola tertentu, sedangkan shutil digunakan untuk operasi manipulasi file dan direktori seperti penyalinan, pemindahan, dan penghapusan data. Seluruh tahapan implementasi program, termasuk proses pelatihan, evaluasi, konversi model, dan pengujian inferensi, dijelaskan lebih lanjut pada bagian Hasil dan Pembahasan beserta potongan kode program yang relevan.

4.1 Hasil Persiapan *Dataset*

Tahap persiapan *dataset* menghasilkan kumpulan data citra mikroskopis sputum yang bersumber dari platform Kaggle, sebagaimana telah diuraikan pada Subbab 3.1. *Dataset* yang digunakan dalam penelitian ini terdiri atas 1.265 citra dengan resolusi asli 1.632 x 1.224 piksel dan memiliki 9.969 anotasi *bounding box* yang seluruhnya termasuk dalam satu kelas, yaitu 'TBbacillus'. Selain citra yang mengandung objek bakteri, *dataset* juga mencakup 47 citra yang tidak memiliki anotasi *bounding box* atau tidak mengandung bakteri *Mycobacterium tuberculosis*. Keberadaan citra tanpa objek tersebut tetap dipertahankan dalam *dataset* karena dapat membantu model mempelajari karakteristik latar belakang (*background*), sehingga diharapkan mampu mengurangi kesalahan deteksi (*false positive*) selama proses pelatihan. Contoh citra dari *dataset* ini ditampilkan pada Gambar 4.1.



Gambar 4.1 Contoh Citra Mikroskopis Sputum

Sumber: Dataset Kaggle, 2020

Berdasarkan inspeksi visual terhadap sampel *dataset*, citra menunjukkan karakteristik citra sputum yang konsisten, di mana bakteri MTB tampak sebagai batang merah terang (pink) pada latar berwarna biru. *Dataset* juga mengandung variasi debris yang cukup tinggi berupa partikel atau struktur latar belakang yang secara visual menyerupai bakteri MTB, sehingga berpotensi menjadi sumber *false positive* pada proses deteksi. Seluruh citra berasal dari satu sumber *dataset* publik, yang menjadi salah satu keterbatasan penelitian ini sebagaimana dibahas pada Subbab 3.1.

Proses pengunduhan *dataset* dilakukan secara manual melalui akun Kaggle, kemudian seluruh berkas *dataset* disimpan pada Google Drive untuk memudahkan pengelolaan data dan integrasi dengan lingkungan komputasi Google Colab. Setelah *dataset* berhasil disimpan pada Google Drive, langkah berikutnya adalah melakukan proses *mounting* Google Drive pada Google Colab. Proses ini bertujuan untuk menghubungkan ruang penyimpanan Google Drive dengan lingkungan kerja Google Colab sehingga file *dataset* dapat diakses secara langsung melalui *notebook*. Dengan metode ini, seluruh proses pengolahan data, pelatihan model, hingga penyimpanan hasil eksperimen dapat dilakukan secara terpusat tanpa perlu mengunggah ulang *dataset* setiap sesi eksekusi.

4.2 Hasil Konversi Anotasi *Dataset*

Dataset dari Kaggle menyediakan anotasi objek dalam format XML yang umumnya digunakan pada skema anotasi *Pascal VOC* seperti yang ditunjukkan pada Gambar 4.2. Format tersebut menyimpan informasi koordinat *bounding box* dalam posisi piksel terhadap dimensi citra. Namun, model YOLO membutuhkan format anotasi yang berbeda, yaitu dalam bentuk file teks (*.txt*) dengan representasi koordinat yang telah dinormalisasi seperti pada Gambar 4.3. Oleh karena itu, dilakukan proses konversi anotasi dari format XML ke format YOLO agar *dataset* dapat digunakan dalam pelatihan model deteksi objek berbasis YOLO. Struktur Anotasi dalam format XML (Pascal VOC), setiap objek memiliki atribut koordinat sebagai berikut:

- xmin: koordinat x kiri atas
- ymin: koordinat y kiri atas
- xmax: koordinat x kanan bawah
- ymax: koordinat y kanan bawah

```
<object>
    <label>TBbacillus</label>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <bndbox>
        <xmin>655</xmin>
        <ymin>1033</ymin>
        <xmax>711</xmax>
        <ymax>1085</ymax>
    </bndbox>
</object>
```

Gambar 4.2 Struktur Anotasi Format XML

Pada format YOLO, setiap objek direpresentasikan dalam satu baris dengan format class_id, x_center, y_center, width, height. Dengan keterangan sebagai berikut:

- class_id: indeks kelas objek
- x_center, y_center: titik tengah *bounding box* (relatif)
- width, height: lebar dan tinggi *bounding box* (relatif)

Seluruh nilai koordinat dinormalisasi terhadap ukuran citra (bernilai antara 0 dan 1).

```
0 0.611520 0.472222 0.037990 0.062092
0 0.405637 0.602941 0.029412 0.053922
0 0.359681 0.793301 0.035539 0.044118
0 0.567096 0.765523 0.050858 0.065359
0 0.378676 0.129085 0.028186 0.084967
0 0.376838 0.025327 0.031863 0.039216
```

Gambar 4.3 Struktur Anotasi Format TXT

Proses konversi label (berkas anotasi) data dilakukan untuk mengubah format anotasi Pascal VOC (XML) menjadi format YOLO yang dibutuhkan dalam pelatihan model deteksi objek. Fungsi convert_xml_to_yolo() yang terdapat pada Gambar 4.4 menerima tiga parameter masukan, yaitu berkas XML hasil anotasi, lebar gambar (*image width*), dan tinggi gambar (*image height*). Pertama, fungsi melakukan *parsing* berkas XML menggunakan modul xml.etree.ElementTree. Apabila terjadi kesalahan dalam proses *parsing*, misalnya berkas rusak atau format tidak valid, fungsi akan mencetak pesan peringatan dan mengembalikan nilai *None* sehingga proses dapat dilanjutkan tanpa menghentikan eksekusi program.

```
import xml.etree.ElementTree as ET

def convert_xml_to_yolo(xml_file, img_width, img_height):
    """Mengonversi Pascal VOC XML ke format YOLO yang ternormalisasi"""
```

```
    tree = ET.parse(xml_file)
    root = tree.getroot()
    yolo_annotations = []

    for obj in root.findall('object'):
        bbox = obj.find('bndbox')

        # Ekstraksi koordinat absolut dari XML
        xmin = float(bbox.find('xmin').text)
        ymin = float(bbox.find('ymin').text)
        xmax = float(bbox.find('xmax').text)
        ymax = float(bbox.find('ymax').text)

        # Transformasi ke format YOLO (x_center, y_center, width, height)
        x_center = ((xmin + xmax) / 2) / img_width
        y_center = ((ymin + ymax) / 2) / img_height
        width = (xmax - xmin) / img_width
        height = (ymax - ymin) / img_height

        # Pembatasan nilai (clipping) dalam rentang [0, 1]
        x_center = max(0, min(1, x_center))
        y_center = max(0, min(1, y_center))
        width = max(0, min(1, width))
        height = max(0, min(1, height))

        # Format label: [class_id] [x_center] [y_center] [width] [height]
        yolo_annotations.append(f"0{x_center:.6f}{y_center:.6f}{width:.6f}
{height:.6f}")

    return yolo_annotations
```

Gambar 4.4 Kode Program untuk Konversi Format Label

Selanjutnya, fungsi mengiterasi setiap elemen objek yang terdapat di dalam berkas XML untuk mengekstrak koordinat *bounding box* dalam format Pascal VOC, yaitu nilai xmin, ymin, xmax, dan ymax yang merepresentasikan sudut kiri atas dan sudut kanan bawah kotak pembatas. Koordinat tersebut kemudian dikonversi ke format YOLO yang merepresentasikan posisi objek sebagai titik pusat (x_center, y_center) beserta lebar dan tinggi kotak pembatas (width, height), seluruhnya dinormalisasi terhadap dimensi gambar sehingga nilainya berada dalam rentang 0 hingga 1. Normalisasi ini penting agar anotasi bersifat independen terhadap resolusi gambar. Sebagai langkah pengamanan tambahan, setiap nilai hasil konversi di-*clip* ke rentang 0 hingga 1 menggunakan fungsi max() dan min() untuk mencegah nilai yang melampaui batas akibat ketidakakuratan anotasi. Hasil akhir dari fungsi ini adalah daftar (*list*) *string* anotasi berformat YOLO, di mana setiap baris merepresentasikan satu objek dengan label kelas 0 diikuti oleh keempat nilai koordinat yang telah dinormalisasi dalam format *txt* seperti pada Gambar 4.3 yang siap digunakan dalam fase pelatihan model.

4.3 Hasil Pembagian *Dataset*

Setelah seluruh anotasi berhasil dikonversi ke dalam format YOLO, tahap berikutnya adalah melakukan pembagian *dataset* (*dataset splitting*) dan pengorganisasian file ke dalam

struktur direktori yang sesuai dengan kebutuhan *framework* YOLO. Tahap ini bertujuan untuk memastikan bahwa data yang digunakan pada proses pelatihan, validasi, dan pengujian berada pada subset yang terpisah sehingga evaluasi performa model dapat dilakukan secara objektif.

Proses pembagian *dataset* dilakukan secara otomatis menggunakan fungsi train_test_split dari pustaka Scikit-Learn. *Dataset* dibagi menjadi tiga subset, yaitu 70% data latih (*training set*), 20% data validasi (*validation set*), dan 10% data uji (*testing set*).

Untuk menjaga konsistensi hasil eksperimen, proses pembagian *dataset* dilakukan menggunakan *random seed* deterministik dengan nilai seed = 42. Penggunaan seed ini memastikan bahwa komposisi data pada setiap subset tetap sama meskipun proses dijalankan ulang, sehingga mendukung reproduktibilitas penelitian. Implementasi teknis dari prosedur pembagian dan pengorganisasian data ini direpresentasikan melalui potongan algoritma pada Gambar 4.5.

```
# Inisialisasi rasio pembagian data
train_ratio, val_ratio, test_ratio = 0.7, 0.2, 0.1
seed = 42

train_data, temp = train_test_split(valid_pairs, test_size=(val_ratio + test_ratio),
random_state=seed)
val_data,  test_data  =  train_test_split(temp,  test_size=test_ratio/(val_ratio  +
test_ratio), random_state=seed)

# Pemetaan grup data untuk proses iterasi folder
splits = {'train': train_data, 'val': val_data, 'test': test_data}

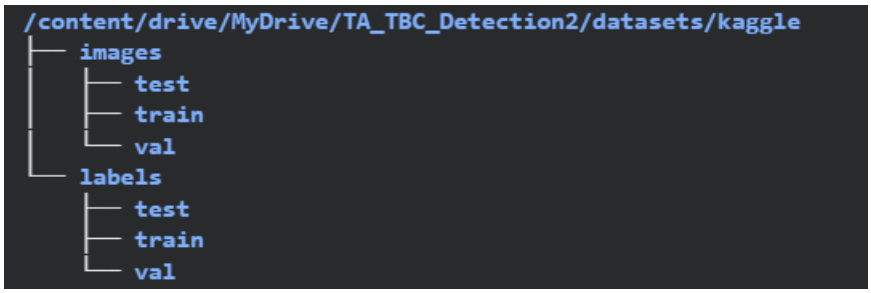
for split_name, data in splits.items():
    # Pembuatan direktori
    img_dir = Path(f'{dest_path}/images/{split_name}')
    lbl_dir = Path(f'{dest_path}/labels/{split_name}')
    img_dir.mkdir(parents=True, exist_ok=True)
    lbl_dir.mkdir(parents=True, exist_ok=True)

        # Penulisan koordinat YOLO ke file teks (.txt)
        with open(lbl_dir / f"{item['original_name']}.txt", 'w') as f:
            f.write('\n'.join(item['yolo_annotations']))
```

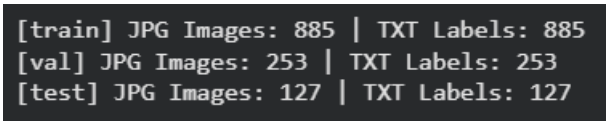
Gambar 4.5 Kode Program Pembagian *Dataset*

Agar sesuai dengan struktur *dataset* yang dipersyaratkan oleh *framework* YOLO, sistem secara otomatis membangun struktur direktori yang memisahkan file citra dan file anotasi seperti yang ditunjukkan pada Gambar 4.6. Folder *images* digunakan untuk menyimpan citra, sedangkan folder *labels* digunakan untuk menyimpan file anotasi dalam format .txt. Masing-masing folder kemudian dibagi kembali ke dalam subfolder *train*, *val*, dan *test* sesuai subset data yang telah ditentukan. Penamaan file anotasi dibuat identik dengan nama file citra untuk menjaga sinkronisasi antara gambar dan label. Melalui prosedur ini, *dataset* berhasil disusun secara terstruktur dan konsisten sehingga siap digunakan pada tahap pelatihan dan evaluasi

model deteksi objek. Informasi mengenai jumlah *dataset* yang tersimpan pada folder kerja setelah proses pembagian *dataset* ditunjukkan pada Gambar 4.7.



Gambar 4.6 Struktur Folder Dataset



Gambar 4.7 Jumlah File pada Folder Dataset

4.4 Hasil Augmentasi dan Pengaturan Hiperparameter

Pada tahap pelatihan model, proses augmentasi data dilakukan secara otomatis menggunakan konfigurasi bawaan (*default augmentation*) yang disediakan oleh *framework* Ultralytics YOLO, sebagaimana telah dijelaskan pada Subbab 3.4. Pendekatan ini dipilih karena konfigurasi bawaan Ultralytics telah dirancang untuk meningkatkan kemampuan generalisasi model melalui berbagai transformasi citra yang sesuai untuk tugas deteksi objek (Jocher, 2025).

Selain augmentasi, dilakukan pula pengaturan hiperparameter pelatihan yang digunakan secara konsisten pada seluruh model agar perbandingan performa antararsitektur dapat dilakukan secara adil. Implementasi program ditampilkan pada Gambar 4.8. Kode pada gambar mendefinisikan konfigurasi pelatihan model YOLO yang disimpan dalam variabel TRAIN_CONFIG. Parameter device diatur secara otomatis, yaitu menggunakan GPU apabila tersedia, atau CPU sebagai alternatif jika GPU tidak terdeteksi. Proses pelatihan dijalankan selama maksimal 100 *epoch* dengan ukuran *batch* sebesar 16 dan resolusi citra masukan 640 piksel, serta dilengkapi mekanisme *early stopping* melalui parameter *patience* sebesar 30 *epoch* untuk mencegah *overfitting*. Optimizer yang digunakan adalah Stochastic Gradient Descent (SGD) dengan learning rate awal (lr0) sebesar 0,01.

```
DEVICE = 0 if torch.cuda.is_available() else 'cpu'

TRAIN_CONFIG = {
    'epochs'   : 100,
```

```
    'batch'    : 16,
    'imgsz'    : 640,
    'patience' : 30,
    'optimizer': 'SGD’,
    'lr0'      : 0.01,
    'device': DEVICE,
}
```

Gambar 4.8 Kode Program Konfigurasi Hiperparameter

4.5 Hasil Pelatihan dan Validasi Model

Setelah seluruh hiperparameter ditentukan, tahapan berikutnya adalah proses pelatihan model (*training*). Pelatihan dilakukan menggunakan *framework* Ultralytics YOLO yang dijalankan pada lingkungan Google Colab dengan dukungan akselerasi GPU NVIDIA Tesla T4.

Implementasi proses pelatihan dilakukan menggunakan pustaka Ultralytics yang menyediakan antarmuka terintegrasi untuk memuat model dan melakukan pelatihan. Contoh kode program pelatihan model ditunjukkan pada Gambar 4.9.

```
model_n = YOLO('yolov8n.pt')

results_n = model_n.train(
    data    = DATA_YAML_PATH,
    name    = 'yolov8n_tbc',
    project = OUTPUT_PATH,
    save    = True,
    plots   = True,
    **TRAIN_CONFIG,
    Seed    = 42 #45, 50
)

BEST_PT_N = f'{OUTPUT_PATH}/yolov8n_tbc/weights/best.pt'
print("Best weights:", BEST_PT_N)
```

Gambar 4.9 Kode Program Pelatihan Model

Pada kode program pada Gambar 4.9, proses pelatihan dimulai dengan memuat arsitektur YOLOv8n menggunakan perintah YOLO ('yolov8n.pt'). Berkas yolov8n.pt merupakan *pretrained weights* dari model YOLOv8 varian *nano* yang sebelumnya telah dilatih pada *dataset* COCO dan diunduh secara otomatis oleh kerangka kerja Ultralytics dari repositori GitHub resmi. Mekanisme yang sama berlaku untuk seluruh arsitektur yang digunakan dalam penelitian ini, dengan tautan unduhan masing-masing berkas *pretrained weights* ditunjukkan pada Tabel 4.1. Penggunaan *pretrained weights* ini bertujuan untuk menerapkan pendekatan *transfer learning*, sehingga model tidak memulai pelatihan dari bobot yang diinisialisasi secara acak (*random initialization*), melainkan dari *pretrained weights* yang telah mempelajari representasi fitur dari dataset COCO (Raimbekovm, 2026).

Tabel 4.1 Sumber Berkas *Pretrained Weights* Model YOLO

Model	Berkas	Tautan Unduhan
YOLOv8n	yolov8n.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolov8n.pt
YOLOv8s	yolov8s.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolov8s.pt
YOLOv10n	yolov10n.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolov10n.pt
YOLOv10s	yolov10s.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolov10s.pt
YOLO11n	yolo11n.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo11n.pt
YOLO11s	yolo11s.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo11s.pt
YOLO26n	yolo26n.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n.pt
YOLO26s	yolo26s.pt	https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26s.pt

Selanjutnya, proses pelatihan dilakukan menggunakan fungsi train(). Parameter data mengacu pada file konfigurasi *dataset* (data.yaml) yang berisi informasi jalur *dataset*, jumlah kelas, dan nama kelas objek. Parameter name digunakan untuk menentukan nama folder hasil pelatihan, sedangkan parameter *project* menentukan direktori utama penyimpanan hasil eksperimen. Opsi save=True digunakan agar bobot model hasil pelatihan disimpan secara otomatis, sementara plots=True digunakan untuk menghasilkan visualisasi proses pelatihan seperti grafik *loss*, *Precision*, *Recall*, dan mAP.

Konfigurasi hiperparameter pelatihan diterapkan melalui TRAIN_CONFIG, yang memuat pengaturan seperti jumlah *epoch*, ukuran *batch*, ukuran citra input, *optimizer*, dan *learning rate* pada tahap konfigurasi hiperparameter pada Subbab 4.4. Penggunaan **TRAIN_CONFIG memungkinkan seluruh konfigurasi hiperparameter dipanggil secara otomatis ke dalam fungsi pelatihan tanpa perlu menuliskannya satu per satu.

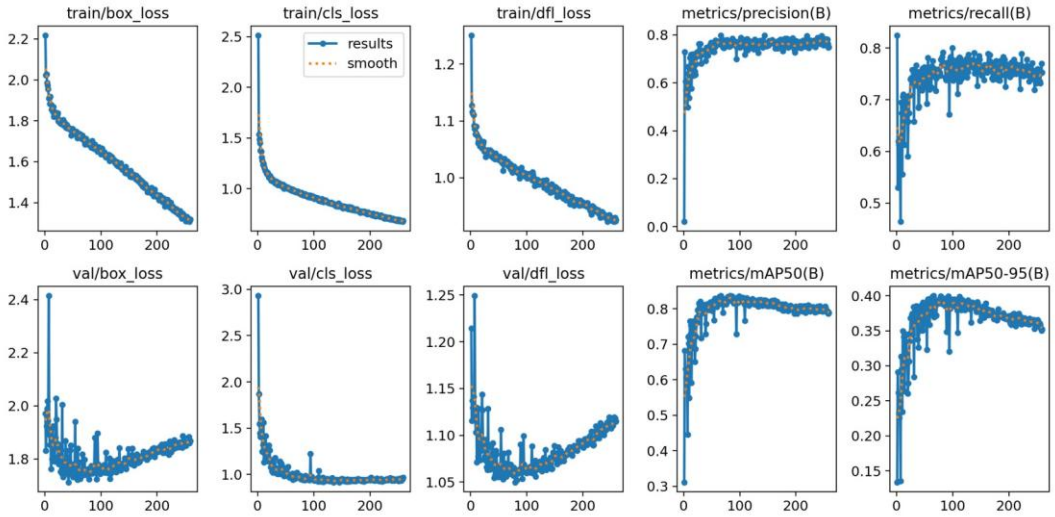
Setelah proses pelatihan selesai, program secara otomatis menyimpan model dengan performa terbaik pada file best.pt, yang ditentukan berdasarkan hasil evaluasi pada data validasi selama proses pelatihan berlangsung. Lokasi penyimpanan bobot terbaik tersebut kemudian disimpan pada variabel BEST_PT_N untuk digunakan pada tahap evaluasi dan pengujian selanjutnya.

Karena penelitian ini membandingkan beberapa generasi YOLO, proses pergantian arsitektur dilakukan secara sederhana dengan mengganti nama bobot model yang dipanggil, misalnya yolov8n.pt, yolov10n.pt, yolo11s.pt, atau yolo26s.pt dan seterusnya, tanpa perlu mengubah struktur utama kode program. Pendekatan ini memungkinkan seluruh model diuji pada *pipeline* pelatihan yang identik sehingga hasil perbandingan performa dapat dilakukan secara lebih objektif dan konsisten.

Penelitian ini juga melakukan analisis pada *box loss*, karena komponen ini paling relevan dengan tujuan utama penelitian, yaitu merepresentasikan tingkat kesalahan model dalam

memprediksi posisi dan ukuran *bounding box* menggunakan fungsi Complete Intersection over Union (CIoU). Mengingat *dataset* penelitian ini hanya terdiri atas satu kelas objek (TBbacillus), kontribusi cls *loss* terhadap perbedaan performa antararsitektur menjadi kurang signifikan dibandingkan pada kasus deteksi multi-kelas, sehingga *box loss* dipilih sebagai indikator utama untuk menilai kualitas konvergensi dan kemampuan lokalisasi setiap model selama proses pelatihan dan validasi.

Sebelum dilakukan pelatihan pada seluruh delapan arsitektur model dengan konfigurasi akhir, dilakukan uji coba awal pada model YOLOv8n menggunakan jumlah *epoch* sebanyak 500 dan nilai *patience* 100 *epoch*. Uji coba ini bertujuan untuk mengeksplorasi potensi performa maksimum model apabila diberikan ruang pelatihan yang lebih panjang. Hasil uji coba yang ditampilkan pada Gambar 4.10 menunjukkan bahwa nilai *train loss* (box, classification, dan DFL) terus mengalami penurunan secara konsisten hingga akhir pelatihan, sedangkan *val loss* menunjukkan pola yang berbeda, yaitu menurun pada tahap awal, mencapai titik stabil pada rentang *epoch* menengah, kemudian menunjukkan tren peningkatan yang konsisten menjelang akhir pelatihan seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Hasil Grafik Loss Pelatihan Awal

Pola serupa juga teramati pada metrik mAP@50 dan mAP@50-95, yang mencapai nilai puncak pada rentang *epoch* tersebut sebelum kembali melandai. Kesenjangan yang semakin melebar antara *train loss* yang terus menurun dan *val loss* yang justru meningkat ini mengindikasikan bahwa model mulai mengalami *overfitting*, yaitu kondisi di mana model semakin menyesuaikan diri terhadap data pelatihan namun mengalami penurunan kemampuan generalisasi terhadap data validasi. Pelatihan uji coba ini berhenti pada *epoch* ke-259 akibat

mekanisme *early stopping*, yang mengindikasikan bahwa performa terbaik model telah tercapai pada epoch ke-159. Hal ini menjadi pertimbangan dalam penentuan konfigurasi *epoch* dan *patience* yang digunakan pada pelatihan seluruh model selanjutnya, sebagaimana telah dijelaskan pada Subbab 3.4, yaitu *epoch*=100 dan *patience*=30. Selain pertimbangan generalisasi model, keputusan ini juga didasari oleh efisiensi penggunaan sumber daya komputasi mengingat penelitian ini melibatkan pelatihan pada delapan arsitektur model sekaligus.

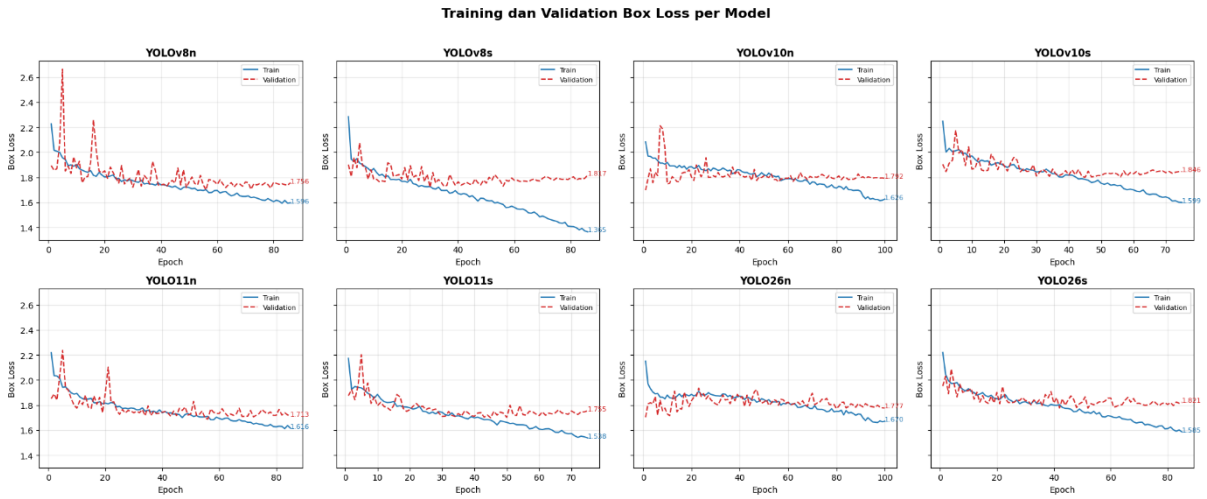
Untuk meningkatkan reliabilitas hasil pelatihan, setiap arsitektur dilatih sebanyak tiga kali menggunakan *random seed* yang berbeda, yaitu 42, 45, dan 50. Penggunaan beberapa nilai *seed* bertujuan untuk mengurangi pengaruh variasi akibat proses pengacakan data dan inisialisasi selama pelatihan. Kestabilan proses pelatihan kemudian dievaluasi berdasarkan nilai mAP@50–95 pada *validation set*, karena metrik tersebut digunakan oleh framework Ultralytics sebagai dasar pemilihan model terbaik (*best.pt*). Oleh karena itu, nilai mAP@50–95 pada setiap arsitektur disajikan dalam bentuk rata-rata (*mean*) dan standar deviasi dari ketiga kali pelatihan. Selanjutnya, model *best.pt* dengan nilai mAP@50–95 validasi tertinggi digunakan pada tahap pengujian menggunakan dataset uji dan proses konversi ke format TensorFlow Lite.

Hasil pelatihan sebanyak tiga kali untuk setiap arsitektur disajikan pada Tabel 4.2. Berdasarkan hasil tersebut, seluruh model menunjukkan nilai standar deviasi yang relatif kecil, yaitu berkisar antara 0,001 hingga 0,005, sehingga mengindikasikan bahwa proses pelatihan menghasilkan performa yang cukup konsisten terhadap variasi *random seed*. Berdasarkan nilai rata-rata (*mean*), YOLO11n dan YOLO11n memperoleh nilai mAP@50-95 tertinggi sebesar 0,413 dengan masing-masing nilai standar deviasi 0,003 dan 0,001. Sementara itu, YOLOv8n dan YOLOv8s memperoleh nilai rata-rata masing-masing sebesar 0,407 ± 0,003 dan 0,400 ± 0,002, sedangkan YOLOv10n dan YOLOv10s memperoleh nilai rata-rata sebesar 0,402 ± 0,001 dan 0,403 ± 0,001. Selain itu, YOLO26n dan YOLO26s memperoleh 0,406 ± 0,005 dan 0,408 ± 0,001.

Tabel 4.2 Hasil Pelatihan dan Evaluasi Kinerja Model Berdasarkan Variasi *Random Seed*

Model	Seed 42	Seed 45	Seed 50	Mean ± SD
YOLOv8n	0,404	0,408	**0,410**	0,407 ± 0,003
YOLOv8s	**0,403**	0,3977	0,399	0,400 ± 0,002
YOLOv10n	**0,403**	0,401	0,401	0,402 ± 0,001
YOLOv10s	**0,404**	0,402	0,403	0,403 ± 0,001
YOLO11n	**0,413**	0,407	0,410	0,410 ± 0,003
YOLO11s	**0,413**	0,412	0,411	0,412 ± 0,001
YOLO26n	0,407	0,401	**0,411**	0,406 ± 0,005
YOLO26s	0,409	**0,410**	0,407	0,408 ± 0,001

Gambar 4.11 menunjukkan grafik Training Box Loss dan Validation Box Loss dari seluruh delapan model selama proses pelatihan menggunakan 100 *epoch* dan *patience* 30 dari nilai mAP@50-95 tertinggi dari beberapa *seed*. Pada grafik, sumbu x menunjukkan jumlah *epoch* pelatihan, sedangkan sumbu y menunjukkan nilai *loss* yang diperoleh selama proses pelatihan. Training Box Loss merepresentasikan tingkat kesalahan model dalam memprediksi posisi dan ukuran *bounding box* terhadap anotasi *ground truth* ditandai dengan garis berwarna biru. Semakin rendah nilai *loss* maka semakin baik kemampuan model dalam melakukan lokalisasi objek, Di sisi lain, Validation *Box loss* secara khusus merepresentasikan kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya.



Gambar 4.11 Grafik Training dan Validation Box Loss

Pada *epoch* awal, seluruh model menunjukkan nilai Train Loss yang relatif tinggi, sebagai konsekuensi dari bobot model yang masih berada pada kondisi inisialisasi awal, sebelum kemudian menurun secara signifikan pada beberapa *epoch* berikutnya. Nilai awal dan akhir dari Training Box Loss dan Validation Box Loss untuk masing-masing model YOLO

tercatat pada Tabel 4.3 . Angka yang relatif tinggi tersebut umum terjadi pada proses *transfer learning*, di mana bobot awal hasil *pretraining* pada *dataset* COCO masih beradaptasi terhadap karakteristik citra sputum yang memiliki distribusi fitur visual berbeda dibandingkan *dataset* umum.

Tabel 4.3 Perbandingan Hasil *Box loss* dan *Epoch* Best Model

Model	Train Box Awal	Train Box Akhir	Val Box Awal	Val Box Akhir	Epoch best.pt	Total Epoch
YOLOv8n	2,227	1,596	1,893	1,756	56	86
YOLOv8s	2,284	1,365	1,901	1,817	56	86
YOLOv10n	2,083	1,626	1,697	1,792	100	100
YOLOv10s	2,248	1,599	1,903	1,846	45	75
YOLO11n	2,219	1,616	1,851	1,713	55	85
YOLO11s	2,173	1,538	1,875	1,755	45	75
YOLO26n	2,150	1,670	1,703	1,777	80	100
YOLO26s	2,220	1,585	1,952	1,821	55	85

Memasuki pertengahan hingga akhir pelatihan, seluruh arsitektur memperlihatkan tren penurunan Training Box Loss yang konsisten, meskipun dengan tingkat konvergensi yang berbeda. YOLOv8s menunjukkan penurunan *Train loss* yang paling kecil dengan nilai akhir sebesar 1,365 dan YOLO11s sebesar 1,538, mengindikasikan kapasitas representasi fitur yang baik dalam mempelajari pola lokalisasi bakteri TB. Posisi ketiga terbaik ditempati oleh YOLO26s dengan nilai akhir 1,585, sedangkan YOLOv8n berada di posisi keempat dengan capaian akhir sebesar 1,596.

Sementara itu, model YOLOv10s dan YOLO11n memperlihatkan penurunan yang relatif stabil, masing-masing mencapai nilai akhir sebesar 1,599 dan 1,616 pada akhir pelatihan. Di sisi lain, YOLOv10n dan YOLO26n menghasilkan nilai *loss* yang sedikit lebih tinggi dibandingkan model lainnya, masing-masing sebesar 1,626 dan 1,670, meskipun keduanya tetap menunjukkan kecenderungan konvergen tanpa indikasi divergensi. Beberapa fluktuasi kecil antar-*epoch* dapat dipengaruhi oleh dinamika *optimizer*, augmentasi data, maupun karakteristik distribusi sampel pada *mini-batch* pelatihan. Secara umum, pola ini menunjukkan bahwa seluruh varian *small* mencapai nilai Training Box Loss akhir yang lebih rendah dibandingkan varian *nano* dari arsitektur yang sama.

Selain kurva Training Box Loss, pada gambar juga terdapat kurva Validation Box Loss yang berfungsi untuk mengukur tingkat kesalahan model dalam memprediksi posisi dan ukuran *bounding box* terhadap anotasi *ground truth* pada data yang tidak digunakan selama proses training, pada gambar ditandai dengan garis putus-putus berwarna merah. Semakin rendah nilai

Validation Box Loss, semakin baik kemampuan model dalam melakukan generalisasi terhadap data baru yang belum pernah dilihat sebelumnya.

Pada fase awal, seluruh model memperlihatkan fluktuasi Validation Box Loss yang relatif tinggi, dengan beberapa model bahkan mengalami lonjakan nilai *loss* pada beberapa *epoch* pertama, seperti YOLOv8n yang melampaui nilai 2,6 pada Gambar 4.11. Fenomena ini umum terjadi pada tahap awal *transfer learning* ketika model masih beradaptasi terhadap karakteristik citra mikroskopis sputum yang berbeda dari *dataset* COCO. Seiring bertambahnya *epoch*, fluktuasi tersebut secara bertahap berkurang. Memasuki pertengahan pelatihan, yaitu sekitar *epoch* ke-30 hingga *epoch* ke-50, kurva Validation Box Loss mulai mengalami penurunan yang lebih landai dan cenderung stabil, menunjukkan bahwa proses pembelajaran telah mencapai fase konvergensi awal. Kesenjangan (*gap*) antara *train loss* dan *validation loss* yang tetap bertahan hingga akhir pelatihan ini mengindikasikan adanya *overfitting* pada tingkat tertentu.

Hasil menunjukkan bahwa tidak seluruh model menyelesaikan pelatihan hingga *epoch* ke-100. Seluruh varian *small* berhenti lebih awal (*early stopping*) akibat tidak adanya peningkatan performa validasi selama 30 *epoch* berturut-turut dan secara otomatis diberhentikan untuk menghindari pemborosan komputasi. Sebaliknya, seluruh varian *nano* menyelesaikan pelatihan penuh hingga *epoch* ke-100 sebagaimana tercantum pada Tabel 4.3. Menariknya, meskipun seluruh model menggunakan *learning rate* awal yang identik, varian *small* justru mencapai kondisi konvergen lebih cepat dibandingkan varian *nano*, ditandai dengan terpicunya *early stopping* pada *epoch* yang jauh lebih awal. Hal ini sejalan dengan kapasitas parameter varian *small* yang lebih besar dibandingkan varian *nano*, sehingga model mampu mempelajari representasi fitur yang relevan dengan lebih efisien dalam jumlah *epoch* yang lebih sedikit, sekaligus lebih cepat mencapai titik jenuh (saturasi) performa pada data validasi. Sebaliknya, varian *nano* dengan kapasitas parameter yang lebih kecil cenderung membutuhkan *epoch* yang lebih banyak untuk mencapai tingkat representasi fitur yang sebanding, sehingga tetap menunjukkan peningkatan performa secara konsisten hingga batas maksimum *epoch* tercapai.

Tabel 4.3 menunjukkan bahwa *epoch* pengambilan *checkpoint* terbaik (*best.pt*) bervariasi antarmodel, yaitu berkisar antara *epoch* ke-45 (YOLOv8s) hingga *epoch* ke-100 (YOLO11n). Variasi ini mengindikasikan bahwa titik konvergensi dari setiap arsitektur berbeda-beda, sehingga pemilihan *checkpoint* dilakukan berdasarkan performa validasi terbaik, bukan dari

epoch terakhir. Hal ini bertujuan untuk memastikan setiap model dievaluasi pada kondisi optimalnya masing-masing.

Pada Tabel 4.3, YOLO11n menghasilkan nilai Validation Box Loss akhir terendah sebesar 1,713, diikuti oleh YOLO11s sebesar 1,755. Nilai *loss* yang lebih rendah menunjukkan bahwa kesalahan lokalisasi *bounding box* pada data validasi relatif lebih kecil dibandingkan model lainnya. Sementara itu, YOLOv10s dan YOLO26s memperoleh nilai Validation Box Loss yang paling tinggi, masing-masing sebesar 1,846 dan 1,821. Adapun YOLOv8n, YOLOv8s, YOLOv10n, dan YOLO26n berada pada rentang nilai menengah antara 1,756 hingga 1,817. Secara umum, seluruh model menunjukkan nilai Validation Box Loss yang relatif berdekatan, sehingga perbedaan kemampuan deteksi tidak dapat disimpulkan hanya dari metrik ini saja.

Hal penting yang didapat dari Gambar 4.11 dan Tabel 4.3 di atas adalah bahwa nilai Training Box Loss yang lebih rendah tidak selalu berbanding lurus dengan performa terbaik model, sebagaimana terlihat pada YOLOv8s yang memiliki Training Box Loss akhir terendah (1,365) namun bukan Validation Box Loss terendah. Hal ini wajar mengingat *box loss* pada tahap pelatihan berfokus pada proses membangun dan mengoptimasi model terhadap data yang sudah diketahui, sedangkan kemampuan model dalam mengenali objek pada data yang sama sekali belum pernah dilihat hanya dapat diukur secara objektif melalui tahap pengujian. Oleh karena itu, performa akhir model dievaluasi lebih lanjut menggunakan metrik *Precision*, Recall, F1-Score, mAP@50, dan mAP@50–95 pada data pengujian, sebagaimana akan dibahas pada Subbab 4.6.

4.6 Hasil Pengujian Model

Sebagaimana telah disinggung pada subbab sebelumnya, nilai *training loss* yang rendah tidak serta-merta mengindikasikan performa model yang optimal pada data baru. Oleh karena itu, evaluasi pada penelitian ini dilakukan menggunakan metrik yang lebih merepresentasikan kemampuan generalisasi model, yaitu *Precision*, *Recall*, F1-Score, mAP@50, dan mAP@50–95, yang diukur pada *test set* (data yang tidak dilihat model selama proses pelatihan). Dengan demikian, interpretasi performa model dalam penelitian ini tidak hanya didasarkan pada nilai *training loss*, melainkan mempertimbangkan hasil evaluasi secara menyeluruh pada data pengujian.

Subbab ini menyajikan hasil pengujian seluruh model yang telah dilatih pada data uji (*test set*) yang tidak digunakan selama proses pelatihan maupun validasi. Pengujian dilakukan

untuk mengevaluasi kemampuan generalisasi model dalam mendeteksi bakteri TB pada citra mikroskopis sputum. Seluruh proses pengujian dilakukan menggunakan 127 citra data uji yang diperoleh dari hasil pembagian *dataset*. Hasil pengujian untuk masing-masing model kemudian dibandingkan dan dianalisis untuk menentukan arsitektur yang memberikan keseimbangan terbaik antara performa deteksi dan efisiensi komputasi pada tugas deteksi *Mycobacterium tuberculosis*.

Berdasarkan Tabel 4.4, YOLO11n menunjukkan performa terbaik secara keseluruhan dibandingkan model lainnya. Model ini memperoleh nilai mAP@50 sebesar 0,840, mAP@50–95 sebesar 0,415, *Precision* sebesar 0,792, *Recall* sebesar 0,774, dan *F1-Score* sebesar 0,783. Di antara seluruh model yang diuji, YOLO11n mencatatkan nilai tertinggi pada metrik mAP@50, mAP@50–95, *Precision*, dan *F1-Score*, meskipun bukan model dengan nilai *Recall* tertinggi. Model ini juga menawarkan efisiensi komputasi yang tinggi dengan jumlah parameter sekitar 2,5 juta, sehingga menghasilkan rasio performa terhadap kompleksitas model yang lebih baik dibandingkan model-model lainnya.

Tabel 4.4 Perbandingan Hasil Pengujian

Model	**Parameter**	**mAP@50**	**mAP@50-95**	***Precision***	***Recall***	**F1**
YOLOv8n	3 M	0,836	0,414	0,774	0,777	0,775
YOLOv8s	11,1 M	0,819	0,396	0,769	0,768	0,769
YOLOv10n	2,3 M	0,810	0,403	0,784	0,726	0,754
YOLOv10s	7,2 M	0,835	0,412	0,762	**0,784**	0,773
YOLO11n	2,5 M	**0,840**	**0,415**	**0,792**	0,774	**0,783**
YOLO11s	9,4 M	0,834	0,407	0,780	0,766	0,773
YOLO26n	2,3 M	0,820	0,401	0,760	0,760	0,760
YOLO26s	9,5 M	0,829	0,401	0,769	**0,784**	0,777

Selain itu, nilai mAP@50 dan mAP@50–95 yang tertinggi pada YOLO11n menunjukkan bahwa model tidak hanya mampu mendeteksi keberadaan bakteri MTB dengan tingkat keberhasilan yang tinggi, tetapi juga menghasilkan lokalisasi *bounding box* yang lebih konsisten pada berbagai ambang Intersection over Union (IoU). Dengan kata lain, representasi fitur yang dipelajari oleh YOLO11n mampu menangkap karakteristik visual bakteri MTB secara lebih baik dibandingkan model-model lain yang diuji.

Keunggulan YOLO11n mengindikasikan bahwa penyempurnaan arsitektur pada generasi YOLO11 mampu menghasilkan performa deteksi yang lebih baik dibandingkan generasi lainnya pada dataset yang digunakan dalam penelitian ini. Hasil tersebut menunjukkan bahwa peningkatan performa tidak semata-mata ditentukan oleh kebaruan arsitektur, melainkan juga oleh kesesuaian desain model dengan karakteristik data yang dihadapi.

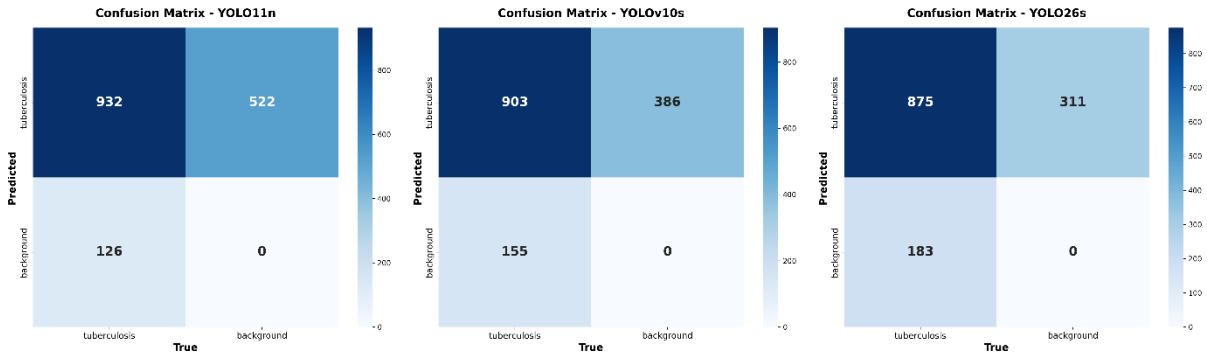
Menariknya, meskipun YOLO26 dirancang dengan mekanisme *one-to-one label assignment* dan pendekatan NMS-free yang dirancang untuk mengurangi prediksi duplikat, serta memperkenalkan STAL untuk meningkatkan pembelajaran pada objek berukuran kecil. Mekanisme tersebut tidak secara otomatis menjamin *Precision* yang lebih tinggi pada semua dataset. Pada penelitian ini, YOLO26s justru menghasilkan *Recall* tertinggi, yaitu 0,784. Model ini memperoleh mAP@50–95 sebesar 0,401 dan F1-score sebesar 0,777. Penyebab hasil performa tersebut belum dapat dipastikan karena penelitian ini tidak melakukan *ablation study* maupun optimasi hiperparameter secara khusus untuk setiap arsitektur. Selain itu, karakteristik dataset yang mengandung debris, artefak pewarnaan, dan objek dengan kemiripan visual terhadap bakteri TB juga berpotensi memengaruhi munculnya *false positive*.

Perbandingan antar-generasi juga menunjukkan tren yang konsisten. Generasi YOLO11 secara konsisten mengungguli generasi YOLOv8 pada seluruh metrik evaluasi. YOLO11n mencatat mAP@50 sebesar 0,840, lebih tinggi dibandingkan YOLOv8n sebesar 0,836 maupun YOLOv8s sebesar 0,819. Bahkan, YOLOv8n mampu melampaui YOLOv8s meskipun memiliki jumlah parameter yang jauh lebih sedikit (3 M berbanding 11,1 M).

Pada generasi YOLOv10, varian *nano* memperoleh mAP@50 sebesar 0,810 dan *F1-Score* sebesar 0,754, sedangkan varian *small* memperoleh mAP@50 sebesar 0,835 dan *F1-Score* sebesar 0,773. Jika dibandingkan dengan generasi sebelumnya, YOLOv10n (0,810) dan YOLOv10s (0,835) menghasilkan mAP@50 yang lebih rendah dibandingkan YOLOv8n (0,836). Selisih performa yang kecil ini menunjukkan bahwa kebaruan arsitektur yang diperkenalkan pada generasi YOLOv10 belum memberikan keunggulan yang signifikan dibandingkan generasi sebelumnya pada *dataset* ini.

Secara keseluruhan, hasil pengujian menunjukkan bahwa YOLO11n merupakan kandidat terbaik dalam penelitian ini karena mampu memberikan kombinasi paling seimbang antara performa model. Walaupun beberapa model unggul pada metrik tertentu, seperti *Recall* pada YOLOv10s dan YOLO26s atau mAP@50 dan mAP@50-95 yang kompetitif yaitu model YOLOv8n, namun tidak ada model lain yang mampu secara bersamaan mencapai nilai tertinggi pada sebagian besar metrik evaluasi utama sebagaimana yang ditunjukkan oleh YOLO11n.

Untuk memperdalam analisis terhadap pola kesalahan deteksi, **Error! Reference source not found.** menyajikan *confusion matrix* dari YOLO11n, YOLO10s, dan YOLO26s. Ketiga model dipilih karena merepresentasikan karakteristik performa yang berbeda, yaitu YOLO11n sebagai model dengan performa keseluruhan terbaik berdasarkan hasil pengujian, lalu YOLO10s dan YOLO26n sebagai model yang menunjukkan sensitivitas deteksi (Recall) tertinggi.



Gambar 4.12 *Confusion Matrix* Tiga Model YOLO

Berdasarkan evaluasi pada data pengujian, hasil *confusion matrix* YOLO11n mencatatkan 932 deteksi *true positive* (TP), 522 *false positive* (FP), dan 126 *false negative* (FN). Sementara itu, YOLO10s menghasilkan 903 TP, 386 FP, dan 155 FN. Sedangkan YOLO26s mencatatkan 863 TP, 338 FP, dan 195 FN. *Confusion matrix* ini kemungkinan divisualisasikan pada *confidence threshold* conf=0,25, yaitu nilai default yang umum digunakan Ultralytics untuk visualisasi, sedangkan nilai *Precision*, *Recall*, dan F1-Score pada Tabel 4.4 dihitung berdasarkan integrasi penuh kurva Precision-Recall sesuai standar evaluasi deteksi objek. Oleh karena itu, nilai *Precision* dan *Recall* hasil perhitungan langsung dari TP, FP, dan FN pada *confusion matrix* tidak sama dengan nilai pada Tabel 4.4, meskipun keduanya berasal dari proses evaluasi data pengujian yang sama.

Jumlah FN yang lebih tinggi pada YOLO26s mengindikasikan bahwa YOLO26s melewatkan lebih banyak objek bakteri TB pada citra pengujian. Di sisi lain, jumlah FP pada YOLO26s yang paling rendah diantara model YOLOv10s dan YOLO11n menunjukkan bahwa model ini menghasilkan lebih sedikit kesalahan deteksi berupa artefak latar belakang yang keliru diklasifikasikan sebagai bakteri TB.

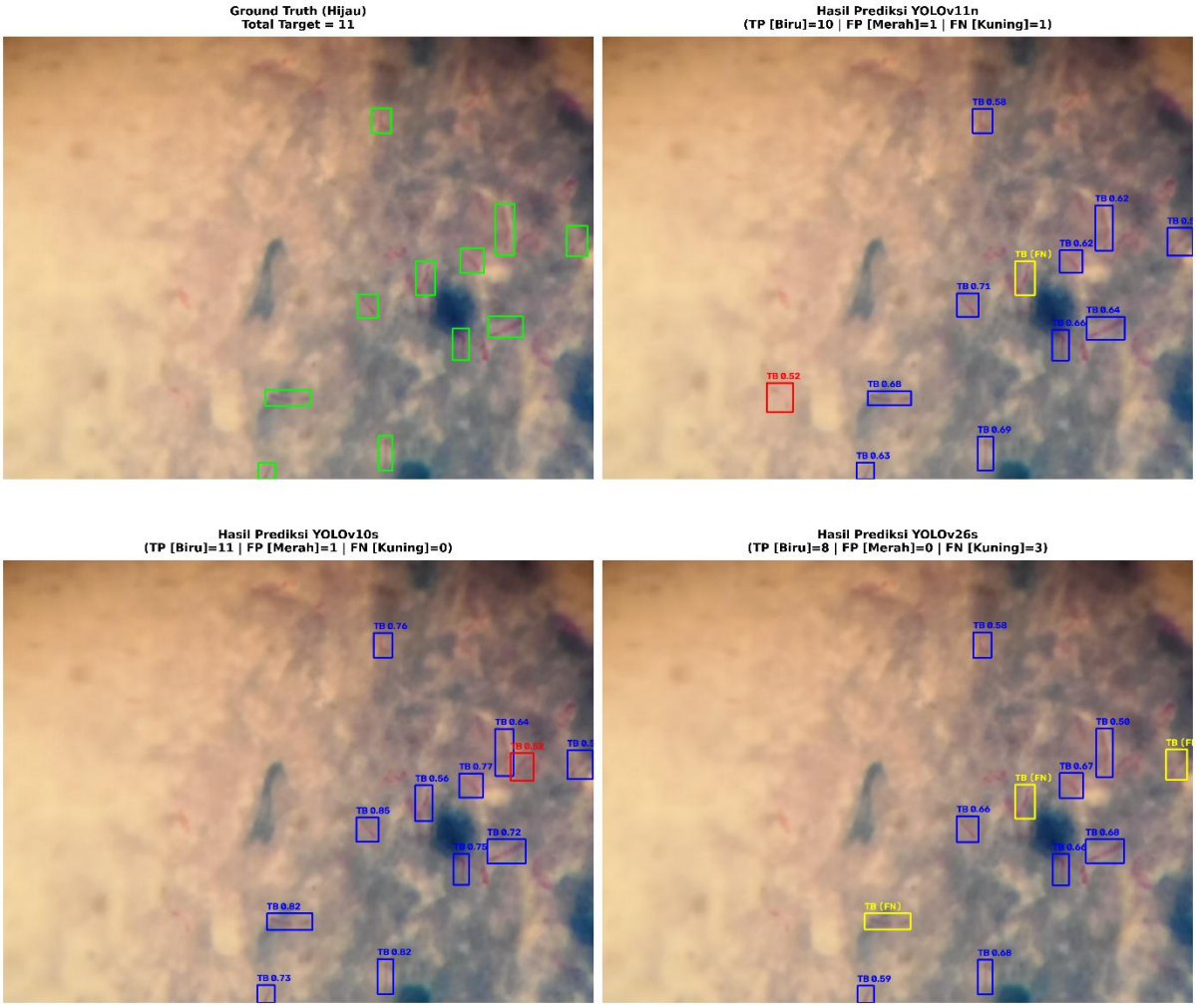
Perbedaan nilai antara Tabel 4.4 dan *confusion matrix* disebabkan oleh perbedaan metodologi perhitungan. *Precision* dan *Recall* pada Tabel 4.4 dihitung melalui fungsi model.val(), yang menentukan *confidence threshold* optimal berdasarkan kurva Precision-

Recall untuk memaksimalkan F1-score, kemudian dirata-ratakan. Sementara itu, *confusion matrix* umumnya dihitung menggunakan *confidence threshold* tetap (default 0,25) dan IoU *threshold* tetap (default 0,45) untuk mencocokkan prediksi terhadap *ground truth*. Pada YOLO11n, Precision menurun dari 0,792 menjadi 0,641, sementara Recall meningkat dari 0,774 menjadi 0,881. Hasil tersebut menunjukkan *threshold* tetap yang digunakan *confusion matrix* tampak menghasilkan lebih banyak deteksi dibandingkan *threshold* optimal dari perhitungan model.val(), sehingga *Recall* meningkat namun *Precision* menurun akibat bertambahnya *False Positive*. Pola yang sama terjadi pada YOLOv10s (0,762 ke 0,701 dan 0,784 ke 0,853). Hasil tersebut mengindikasikan bahwa hubungan antara kedua metode perhitungan tersebut bergantung pada distribusi *confidence score* masing-masing model. Sama seperti YOLO26s menghasilkan penurunan *Precision* (0,769 ke 0,737) serta kenaikan *Recall* (0,784 ke 0,827).

YOLO11n menghasilkan *Recall* yang lebih tinggi pada perhitungan confusion matrix dan dari model.val(), YOLO11n secara keseluruhan tetap unggul pada mAP@50, mAP@50–95, Precision, dan F1-Score, yang membuktikan keseimbangan dan kemampuan generalisasi yang lebih baik untuk mendeteksi bakteri TB pada citra mikroskopis sputum.

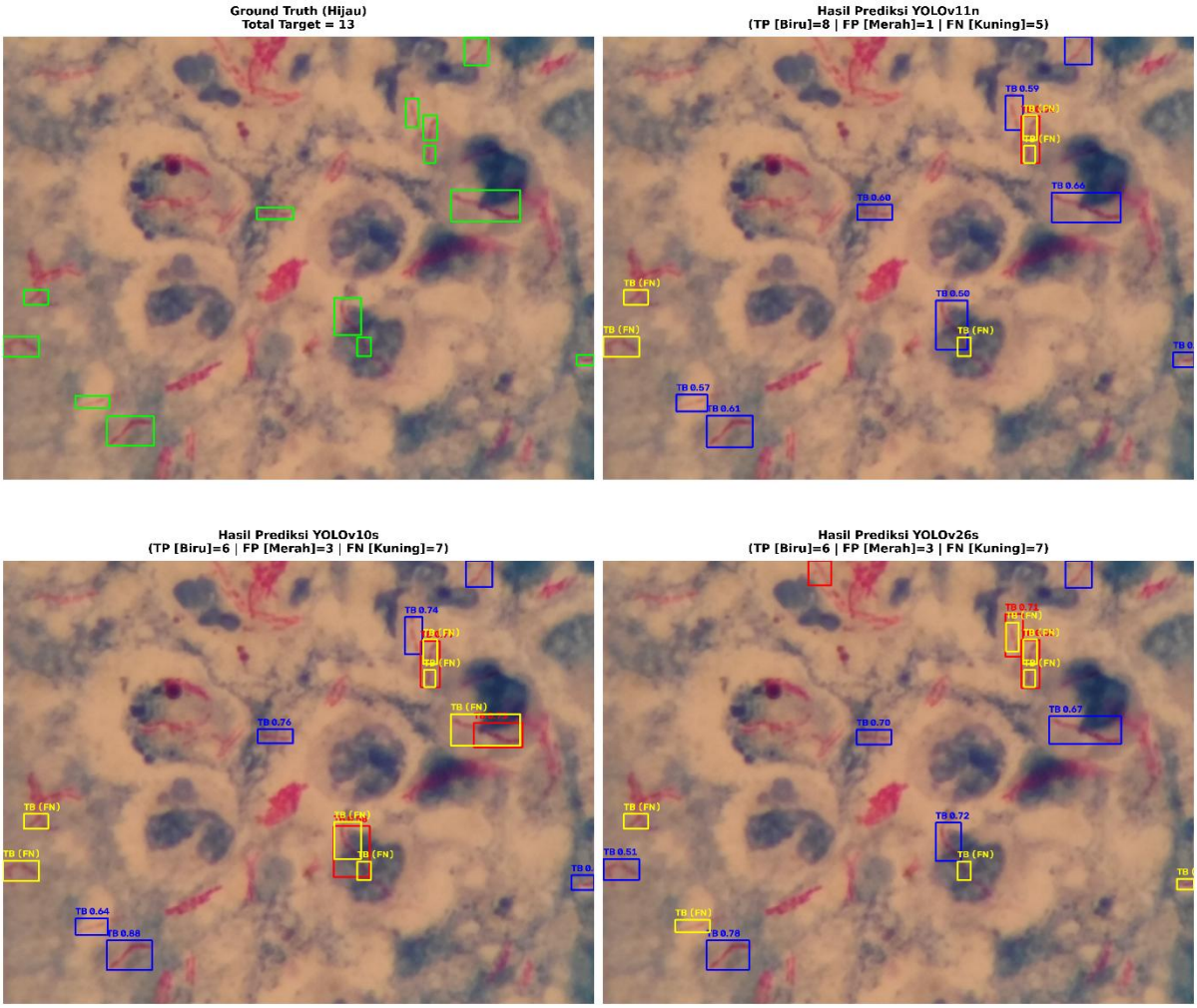
Gambar 4.13 dan Gambar 4.14 menyajikan perbandingan hasil prediksi YOLO11n, YOLOv10s, dan YOLO26s pada dua sampel citra pengujian yang berbeda. Visualisasi ini bertujuan untuk memberikan gambaran mengenai kemampuan kedua model dalam mendeteksi bakteri *Mycobacterium tuberculosis* pada kondisi citra yang bervariasi.

Pada sampel pertama Gambar 4.13, terdapat 11 objek bakteri MTB berdasarkan anotasi *ground truth* yang ditandai dengan warna hijau. YOLO11n berhasil mendeteksi 10 objek sebagai TP ditandai dengan warna biru, 1 FP ditandai dengan warna merah dan 1 FN ditandai dengan warna kuning, sedangkan YOLOv10s mendeteksi 11 objek sebagai TP, dengan 1 FP dan tidak ada FN. Selain itu, YOLO26s mendeteksi 8 TP, dan terdapat 3 FN. Hasil ini menunjukkan bahwa pada sampel pertama, YOLOv10s mampu melakukan lokalisasi objek secara lebih lengkap dibandingkan YOLO11n dan YOLO26s, karena berhasil mendeteksi bakteri secara menyeluruh dan tidak ada FN .



Gambar 4.13 Hasil Prediksi pada Data Testing Pertama

Pada sampel kedua Gambar 4.14, citra memiliki tingkat kompleksitas yang lebih tinggi akibat keberadaan debris dan struktur seluler lain yang menyerupai karakteristik visual bakteri. Dari total 13 objek bakteri MTB yang tersedia pada *ground truth*, YOLO11n menghasilkan 8 TP, 1 FP, dan 5 FN. Sementara itu, YOLOv10s menghasilkan 6 TP, 3 FP, dan 7 FN dan YOLO26s menghasilkan jumlah yang sama. Pada contoh citra ini, YOLO11n mampu mendeteksi lebih banyak objek bakteri serta menghasilkan kesalahan deteksi yang lebih sedikit dibandingkan YOLOv10s dan YOLO26s.



Gambar 4.14 Hasil Prediksi pada Data Testing Kedua

Perbandingan kedua sampel tersebut menunjukkan bahwa performa model dapat dipengaruhi oleh karakteristik visual citra yang dianalisis. Pada sampel dengan latar belakang yang relatif lebih bersih, YOLOv10s menunjukkan hasil deteksi yang lebih lengkap, sedangkan pada sampel dengan latar belakang yang lebih kompleks akibat keberadaan debris dan struktur seluler yang menyerupai bakteri, YOLO11n menunjukkan jumlah *false positive* yang lebih rendah. Meskipun demikian, evaluasi kualitatif pada sejumlah contoh citra tidak dapat digunakan sebagai dasar utama untuk menentukan model terbaik. Untuk memastikan kecenderungan ini secara sahih, diperlukan pengelompokan seluruh data uji berdasarkan tingkat kompleksitas visual secara sistematis dan analisis kuantitatif per kelompok, yang berada di luar cakupan penelitian ini dan dapat menjadi arah bagi penelitian selanjutnya. Oleh karena itu, penentuan performa akhir tetap mengacu pada hasil evaluasi kuantitatif pada seluruh data pengujian yang disajikan pada Tabel 4.4. Berdasarkan evaluasi tersebut, YOLO11n tetap

menjadi model dengan performa keseluruhan terbaik karena memperoleh nilai mAP@50, mAP@50-95, *Precision*, dan *F1-Score* tertinggi di antara seluruh model yang diuji.

4.7 Hasil Optimasi Model ke Format TensorFlow Lite

Setelah tahap pengujian selesai dilakukan, model dengan bobot terbaik (*best.pt*) dari setiap model selanjutnya dikonversi ke format TensorFlow Lite (TFLite). Proses ini bertujuan untuk menghasilkan model yang lebih ringan dan efisien sehingga lebih sesuai untuk implementasi pada perangkat *edge computing* dengan sumber daya terbatas, seperti Raspberry Pi atau perangkat lainnya (Ultralytics, 2026).

Konversi dilakukan menggunakan fungsi export() yang disediakan oleh *framework* Ultralytics YOLO. Pada penelitian ini, model hasil pelatihan dalam format PyTorch (.pt) diekspor ke format TensorFlow Lite (.tflite) dengan mengaktifkan parameter int8=True, sehingga proses Post Training Quantization (PTQ) berbasis INT8 diterapkan secara otomatis. Teknik kuantisasi INT8 mengubah representasi bobot dan aktivasi model dari format *floating-point* 32-bit (FP32) menjadi *integer* 8-bit (INT8), yang umumnya dapat mengurangi ukuran model secara signifikan dan meningkatkan efisiensi inferensi pada perangkat yang mendukung operasi integer.

Gambar 4.15 menampilkan contoh implementasi proses konversi untuk model YOLOv8n dan YOLOv8s. Pemilihan kedua model tersebut pada kode bertujuan untuk menyederhanakan penyajian implementasi tanpa mengurangi substansi penelitian. Prosedur konversi yang sama juga diterapkan pada seluruh model yang dibandingkan, yaitu YOLOv10n, YOLOv10s, YOLO11n, YOLO11s, YOLO26n, dan YOLO26s, dengan hanya mengganti file bobot (*best.pt*) sesuai model yang akan diekspor. Dengan demikian, seluruh model menghasilkan versi TensorFlow Lite INT8 yang selanjutnya digunakan pada tahap analisis efisiensi ukuran model dan waktu inferensi.

```
BEST_PT_N = f'{OUTPUT_PATH}/yolov8n_tbc/weights/best.pt'
BEST_PT_S = f'{OUTPUT_PATH}/yolov8s_tbc/weights/best.pt'

def export_tflite(best_pt, model_name):
    print(f"\nExport {model_name} ke TFLite INT8...")
    m = YOLO(best_pt)
    exported = m.export(
        format = 'tflite',
        imgsz  = TRAIN_CONFIG['imgsz'],
        int8   = True,
        data   = DATA_YAML_PATH,
    )
    dst = f'{TFLITE_PATH}/{model_name}_int8.tflite'
    shutil.copy(str(exported), dst)
    print(f"Tersimpan: {dst}")
    return dst
```

```
tflite_n = export_tflite(BEST_PT_N, 'yolov8n_tbc')
tflite_s = export_tflite(BEST_PT_S, 'yolov8s_tbc')
```

Gambar 4.15 Kode Program untuk Optimasi ke Format TFLite

Kode program pada Gambar 4.15 menunjukkan implementasi proses konversi model yang digunakan dalam penelitian ini. Pertama, file bobot terbaik hasil pelatihan (best.pt) dimuat menggunakan kelas YOLO. Selanjutnya, fungsi export() dijalankan dengan parameter format='tflite', imgsz=640, int8=True, dan data=DATA_YAML_PATH. Parameter imgsz digunakan untuk mempertahankan ukuran input model sebesar 640 × 640 piksel, sedangkan parameter data digunakan oleh Ultralytics selama proses kuantisasi INT8 untuk menyediakan data kalibrasi yang diperlukan dalam menentukan rentang aktivasi model.

Hasil konversi kemudian disimpan ke direktori khusus menggunakan format penamaan yang terpisah untuk setiap model, sehingga memudahkan proses pengujian dan analisis lebih lanjut. Model TFLite yang dihasilkan selanjutnya digunakan untuk mengevaluasi efisiensi ukuran berkas dan waktu inferensi sebagai indikator potensi implementasi pada lingkungan *edge computing*.

Tabel 4.5 menyajikan perbandingan ukuran model sebelum dan sesudah dikonversi ke format TensorFlow Lite (TFLite) dengan kuantisasi INT8. Hasil menunjukkan bahwa seluruh model mengalami penurunan ukuran berkas yang signifikan setelah proses konversi. Pengurangan ukuran ini terjadi karena representasi bobot model yang semula menggunakan format *floating-point* 32-bit (FP32) diubah menjadi format *integer* 8-bit (INT8), sehingga kebutuhan ruang penyimpanan menjadi lebih kecil.

Tabel 4.5 Hasil Ukuran Model

Model	Ukuran Awal (MB)	Ukuran Setelah Export ke TFlite (MB)
YOLOv8n	5,95	3,19
YOLOv8s	21,47	11,01
YOLOv10n	5,48	**2,58**
YOLOv10s	15,75	7,43
YOLO11n	5,21	2,85
YOLO11s	18,28	9,46
YOLO26n	5,13	2,70
YOLO26s	19,36	9,58

Pada kelompok varian *nano*, YOLOv10n menghasilkan ukuran model TFLite terkecil yaitu 2,58 MB dari ukuran awal 5,48 MB, diikuti oleh YOLO26n sebesar 2,70 MB dari ukuran awal 5,13 MB. Sementara itu, YOLO11n memiliki ukuran 2,85 MB dari ukuran awal 5,21 MB, dan YOLOv8n sebesar 3,19 MB dari ukuran awal 5,95 MB. Hasil ini menunjukkan bahwa seluruh varian *nano* berhasil diperkecil hingga berada pada rentang 2,58–3,19 MB sehingga

berpotensi untuk diimplementasikan pada perangkat *edge computing* dengan keterbatasan kapasitas penyimpanan dan memori.

Pada kelompok varian *small*, YOLOv10s menghasilkan ukuran TFLite terkecil sebesar 7,43 MB dari ukuran awal 15,75 MB. Selanjutnya, YOLO11s dan YOLO26s memiliki ukuran akhir masing-masing sebesar 9,46 MB dan 9,58 MB, sedangkan YOLOv8s menjadi model dengan ukuran terbesar yaitu 11,01 MB dari ukuran awal 21,47 MB. Meskipun memiliki ukuran lebih besar dibandingkan varian *nano*, seluruh model *small* tetap mengalami reduksi ukuran yang cukup signifikan setelah proses kuantisasi.

Jika ditinjau dari persentase reduksi ukuran, seluruh model mengalami pengurangan sekitar 45–55% dibandingkan ukuran awalnya. Hasil ini menunjukkan bahwa proses konversi TFLite INT8 efektif dalam menekan kebutuhan penyimpanan tanpa mengubah arsitektur model. Pengurangan ukuran model ini menjadi salah satu faktor penting dalam mendukung implementasi pada perangkat *edge computing*, karena memungkinkan proses *deployment* dilakukan dengan kebutuhan memori yang lebih rendah serta waktu pemuatan model yang lebih cepat.

Meskipun proses konversi menghasilkan ukuran model yang lebih kecil, pengurangan ukuran tersebut berpotensi memengaruhi performa deteksi yang dihasilkan. Oleh karena itu, analisis selanjutnya dilakukan dengan membandingkan performa model sebelum dan sesudah dikonversi ke format TensorFlow Lite untuk mengevaluasi dampak konversi terhadap akurasi sekaligus efisiensi komputasi.

4.8 Hasil Pengujian Model TensorFlow Lite pada *Dataset* Uji

Setelah seluruh model berhasil dikonversi ke format TensorFlow Lite (TFLite) dengan kuantisasi INT8, tahap berikutnya adalah melakukan pengujian untuk memastikan model hasil konversi dapat digunakan dengan baik pada proses inferensi. Pengujian ini bertujuan untuk memverifikasi bahwa model TFLite berhasil dimuat oleh TensorFlow Lite Interpreter serta mampu menghasilkan prediksi pada citra masukan tanpa mengalami kesalahan fungsional.
Pengujian pada tahap ini tidak bertujuan untuk mengevaluasi kembali performa model, karena metrik deteksi seperti mAP@50, mAP@50-95, *Precision*, *Recall*, dan *F1-Score* telah diperoleh pada tahap pengujian model PyTorch sebelumnya. Fokus utama pada tahap ini adalah menganalisis dampak konversi TFLite terhadap efisiensi komputasi serta mengidentifikasi model yang paling berpotensi untuk diterapkan pada perangkat *edge computing*.

Hasil evaluasi pasca-konversi yang ditunjukkan pada Tabel 4.6 bahwa YOLO11n tetap mempertahankan posisinya sebagai model dengan performa terbaik pada metrik mAP@50-95 yaitu sebesar 0,413 dan F1-Score sebesar 0,781. Setelah konversi TFLite, terjadi pergeseran pemegang nilai mAP@50 dan *Precision* tertinggi. Sebelum konversi, nilai mAP@50 dan *Precision* tertinggi dipegang oleh YOLO11n masing-masing sebesar 0,840 dan 0,792. Namun setelah proses kuantisasi, kedua posisi tersebut beralih ke YOLOv8n, yang nilai mAP@50-nya menurun tipis dari 0,836 menjadi 0,835, dan nilai *Precision*-nya meningkat dari 0,774 menjadi 0,792, menjadikan YOLOv8n sebagai model dengan mAP@50 dan *Precision* tertinggi setelah konversi. Fenomena ini menunjukkan bahwa proses *quantization* INT8 tidak selalu menghasilkan degradasi pada semua metrik secara serentak, melainkan dapat menggeser titik keseimbangan Precision-Recall pada model tertentu, sehingga posisi model terbaik pada suatu metrik dapat berpindah meskipun perubahan nilainya relatif kecil.

Selain itu, posisi Recall tertinggi tetap dipegang oleh YOLOv10s baik sebelum maupun sesudah konversi, yaitu sebesar 0,787 karena mengalami peningkatan sedikit dari nilai sebelum konversi. Hal ini mengindikasikan bahwa karakteristik deteksi model tersebut terhadap objek target relatif stabil terhadap proses kuantisasi INT8.

Tabel 4.6 Perbandingan Hasil Pengujian Setelah Konversi

Model	**mAP@50**	**mAP@50-95**	***Precision***	***Recall***	**F1**
YOLOv8n	**0,835**	0,412	**0,792**	0,768	0,780
YOLOv8s	0,814	0,394	0,777	0,760	0,768
YOLOv10n	0,812	0,402	0,734	0,781	0,757
YOLOv10s	0,826	0,408	0,756	**0,787**	0,771
YOLO11n	0,833	**0,413**	0,787	0,775	**0,781**
YOLO11s	0,830	0,406	0,779	0,778	0,778
YOLO26n	0,813	0,393	0,761	0,770	0,766
YOLO26s	0,812	0,395	0,747	0,777	0,761

Untuk mengevaluasi dampak proses kuantisasi INT8 terhadap performa deteksi, dilakukan perbandingan antara hasil evaluasi model sebelum dan sesudah konversi ke format TFLite. Tabel 4.7 menyajikan selisih nilai mAP@50, mAP@50-95, *Precision*, *Recall*, dan *F1-Score* yang diperoleh masing-masing model, di mana nilai positif menunjukkan peningkatan

dan nilai negatif menunjukkan penurunan performa setelah proses kuantisasi. Secara umum, hasil pada tabel menunjukkan bahwa proses konversi ke TFLite INT8 tidak menyebabkan penurunan performa yang signifikan pada sebagian besar model.

Tabel 4.7 Selisih Performa Sebelum dan Sesudah Konversi

Model	**mAP@50**	**mAP@50-95**	***Precision***	***Recall***	**F1**
YOLOv8n	-0,001	-0,002	+0,018	-0,009	+0,005
YOLOv8s	-0,005	-0,002	+0,008	-0,008	-0,001
YOLOv10n	+0,002	-0,001	-0,050	+0,055	+0,003
YOLOv10s	-0,009	-0,004	-0,006	+0,003	-0,002
YOLO11n	-0,007	-0,002	-0,005	+0,001	-0,002
YOLO11s	-0,004	-0,001	-0,001	+0,012	+0,005
YOLO26n	-0,007	-0,008	+0,001	+0,010	+0,006
YOLO26s	-0,017	-0,006	-0,022	-0,007	-0,016

Ditinjau dari stabilitas metrik mAP@50, selisih mAP@50 berkisar antara -0,017 hingga +0,002, sedangkan selisih mAP@50-95 berkisar antara -0,008 hingga -0,001. Penurunan terbesar pada mAP@50 terjadi pada model YOLO26s sebesar -0,017, sedangkan satu-satunya model yang justru mengalami peningkatan mAP@50 setelah konversi adalah YOLOv10n sebesar +0,002.

Pola yang menarik terlihat pada model YOLOv10n, yang mengalami penurunan *Precision* cukup besar sebesar -0,050 namun diimbangi dengan peningkatan *Recall* sebesar +0,055. Kondisi ini mengindikasikan adanya pergeseran karakteristik prediksi model setelah konversi, di mana model menjadi lebih sensitif dalam mendeteksi objek namun disertai peningkatan jumlah *false positive*. Meskipun *Precision* menurun cukup signifikan, kenaikan *Recall* yang lebih besar menyebabkan nilai F1 tetap meningkat tipis sebesar +0,003, mengindikasikan bahwa penurunan *Precision* belum sampai mendominasi keseimbangan antara *Precision* dan *Recall* pada perhitungan F1-Score.

Model YOLO26s menunjukkan penurunan performa paling konsisten di antara seluruh model yang diuji, dengan selisih negatif pada hampir seluruh metrik, yaitu mAP@50 (-0,017), mAP@50-95 (-0,006), *Precision* (-0,022), *Recall* (-0,007), dan F1 (-0,016). Hal ini

mengindikasikan bahwa proses kuantisasi INT8 memberikan dampak yang relatif lebih besar terhadap model varian s pada arsitektur YOLO26 dibandingkan model lainnya.

Di sisi lain, beberapa model seperti YOLOv8n, YOLO11s, dan YOLO26n justru menunjukkan peningkatan nilai F1 setelah konversi, masing-masing sebesar +0,005, +0,005, dan +0,006. Peningkatan ini menunjukkan bahwa proses konversi tidak selalu berdampak negatif terhadap performa deteksi, dan pada beberapa kasus proses kuantisasi dapat menghasilkan keseimbangan precision-recall yang lebih baik.

Secara keseluruhan, hasil pada Tabel 4.7 menunjukkan bahwa kuantisasi INT8 memberikan dampak yang tidak terlalu signifikan terhadap performa deteksi seluruh model. Dengan penurunan terbesar hanya pada metrik *Precision* sebesar -0,050 poin dan sebagian model bahkan mengalami peningkatan pada metrik tertentu, proses konversi TensorFlow Lite dapat dikatakan berhasil meningkatkan efisiensi model tanpa menyebabkan degradasi performa yang berarti. Temuan ini mendukung penggunaan format TFLite INT8 sebagai solusi yang sesuai untuk implementasi deteksi bakteri *Mycobacterium tuberculosis* pada perangkat *edge computing* dengan sumber daya terbatas.

Hasil tersebut menunjukkan bahwa konversi TensorFlow Lite INT8 mampu menghasilkan model yang lebih ringan tanpa mengorbankan performa deteksi secara signifikan. Namun, efisiensi model untuk implementasi pada perangkat *edge computing* tidak hanya ditentukan oleh ukuran model dan performa, melainkan juga oleh kecepatan inferensinya. Oleh karena itu, dilakukan pengukuran waktu inferensi untuk membandingkan efisiensi komputasi dari setiap model yang telah dikonversi.

4.9 Hasil Perhitungan Waktu Inferensi

Setelah dilakukan optimasi dan konversi model ke format TensorFlow Lite INT8, tahap selanjutnya adalah mengukur waktu inferensi untuk mengevaluasi efisiensi komputasi masing-masing model. Pengukuran dilakukan menggunakan TFLite Interpreter secara langsung, tanpa melalui pipeline YOLO.predict(), sehingga waktu yang terukur merupakan waktu inferensi murni tanpa overhead pre-processing maupun post-processing dari *framework* YOLO. Pengujian dijalankan pada CPU Google Colab dengan spesifikasi Intel Xeon @ 2.00 GHz, namun jumlah *thread* yang digunakan dibatasi sebanyak 4 *thread* untuk mensimulasikan kondisi Raspberry Pi 4 yang memiliki 4 inti prosesor ARM Cortex-A72. Nilai pembatasan tersebut hanya menyamakan tingkat paralelisme dan tidak merepresentasikan perbedaan arsitektur. Oleh karena itu, pengujian ini tidak dapat dianggap sebagai simulasi penuh terhadap

kondisi Raspberry Pi 4, dan nilai waktu inferensi yang dihasilkan tidak merepresentasikan performa aktual pada perangkat *edge* fisik. Pengukuran ini berfungsi sebagai indikator komparatif relatif antar-model pada kondisi *hardware* yang identik, bukan sebagai prediksi performa pada perangkat target *deployment*.

Proses pengukuran diimplementasikan sebagaimana ditunjukkan pada Gambar 4.16. Setiap model dimuat menggunakan tf.lite.Interpreter dengan alokasi memori dilakukan secara eksplisit melalui allocate_tensors(). Citra masukan berukuran 640×640 piksel dikonversi ke format uint8 tanpa normalisasi, sesuai dengan persyaratan input model TFLite INT8. Sebelum pengukuran utama, setiap model menjalani fase *warm-up* sebanyak 10 iterasi untuk memastikan CPU dan alokasi memori berada dalam kondisi stabil. Selanjutnya, satu citra sampel representatif dari *dataset* pengujian diproses secara berulang sebanyak 100 iterasi, dengan waktu setiap iterasi diukur menggunakan time.perf_counter() dalam satuan milidetik. Pendekatan pengulangan satu citra dipilih untuk mengisolasi variabilitas konten citra sehingga hasil pengukuran murni mencerminkan karakteristik kecepatan inferensi masing-masing arsitektur.

```
# Konfigurasi pengukuran
NUM_RUNS     = 100   # iterasi pengukuran
WARM_UP_RUNS = 10    # iterasi warm-up
IMG_SIZE     = 640
NUM_THREADS  = 4     # simulasi 4-core Raspberry Pi 4

def preprocess_image(img_path, input_details):
    img = Image.open(img_path).convert('RGB').resize((IMG_SIZE, IMG_SIZE))
    img_array = np.array(img, dtype=np.uint8)
    return np.expand_dims(img_array, axis=0)

for model_name in all_models:
    interpreter = tf.lite.Interpreter(
        model_path=TFL_FILES[model_name],
        num_threads=NUM_THREADS
    )
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    input_data    = preprocess_image(SAMPLE_IMG_PATH, input_details)

    for _ in range(WARM_UP_RUNS):
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()

    times = []
    for _ in range(NUM_RUNS):
        start = time.perf_counter()
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()
        times.append((time.perf_counter() - start) * 1000)

    avg_ms = round(np.mean(times), 2)
```

Gambar 4.16 Kode Program untuk Pengukuran Waktu Inferensi

Tabel 4.8 menyajikan hasil pengukuran waktu inferensi seluruh model TFLite INT8. Semakin kecil nilai rata-rata waktu inferensi, semakin cepat model memproses citra masukan sehingga semakin sesuai untuk implementasi pada perangkat dengan sumber daya komputasi terbatas.

Tabel 4.8 Hasil Pengukuran Waktu Inferensi

Model	**Sebelum Konversi (ms)**	**Setelah Konversi (ms)**
YOLOv8n	122,44	126,58
YOLOv8s	335,19	397,71
YOLOv10n	111,72	133,60
YOLOv10s	300,86	339,10
YOLO11n	115,76	119,05
YOLO11s	320,40	336,91
YOLO26n	**104,91**	**114,83**
YOLO26s	296,73	350,95

Berdasarkan Tabel 4.8, model dengan waktu inferensi tercepat adalah YOLO26n dengan rata-rata sebelum dan sesudah konversi sebesar 104,91 ms dan 114,83 ms, sedangkan model dengan waktu inferensi paling lambat adalah YOLOv8s dengan rata-rata waktu sebelum dan sesudah konversi sebesar 335,19 ms dan 397,71 ms. Hasil ini menunjukkan bahwa terdapat perbedaan efisiensi komputasi yang signifikan antararsitektur meskipun seluruh model telah melalui proses kuantisasi INT8 yang sama.

Jika ditinjau berdasarkan ukuran model, varian *nano* secara konsisten menunjukkan waktu inferensi yang lebih cepat dibandingkan varian *small* pada seluruh arsitektur yang diuji. Rasio perbedaan waktu inferensi antara varian *nano* dan *small* pada masing-masing arsitektur adalah sebagai berikut, YOLOv8s 3 kali lebih lambat dari YOLOv8n (397,71 ms dan 126,58 ms), YOLOv10s 2,5 kali lebih lambat dari YOLOv10n (339,10 ms dan 133,60 ms), YOLO11s 2,8 kali lebih lambat dari YOLO11n (336,91 ms dan 119,05 ms), dan YOLO26s 3 kali lebih lambat dari YOLO26n (350,95 ms dan 114,83 ms). Perbedaan ini dapat dijelaskan oleh jumlah parameter yang lebih sedikit pada varian *nano* sehingga beban komputasi selama proses inferensi menjadi lebih rendah. Sebaliknya, varian *small* memiliki kapasitas representasi fitur yang lebih besar, namun memerlukan waktu pemrosesan yang relatif lebih lama.

Pada lingkungan pengujian berbasis CPU x86 (Intel Xeon), seluruh model TFLite menunjukkan waktu inferensi yang lebih tinggi dibandingkan model PyTorch-nya pada kolom sebelum konversi. Perbedaan waktu inferensi antara format PyTorch dan TFLite INT8 pada lingkungan pengujian dipengaruhi oleh perbedaan presisi komputasi (FP32 dan INT8) serta tingkat dukungan optimasi *framework* terhadap arsitektur CPU yang digunakan. Pada penelitian Jain et al., 2020, menemukan bahwa TFLite dioptimalkan untuk eksekusi pada perangkat *edge* berbasis ARM, namun tidak memiliki dukungan eksekusi yang efisien pada CPU berbasis Intel maupun GPU Nvidia. Studi yang sama juga menunjukkan bahwa pada CPU Intel Xeon Cascade Lake, model terkuantisasi hanya dapat mencapai performa optimal apabila dijalankan menggunakan compiler atau jalur eksekusi khusus, dengan peningkatan kecepatan hingga 2,35 kali dibandingkan eksekusi tanpa optimasi tersebut. Temuan ini konsisten dengan hasil pengujian pada penelitian ini, di mana model TFLite INT8 dijalankan pada CPU Intel Xeon tanpa dukungan delegasi akselerasi khusus untuk operasi kuantisasi.

Pada perangkat target implementasi berbasis ARM, seperti Raspberry Pi 4, TFLite secara arsitektural memang dirancang dan diprioritaskan untuk perangkat tersebut sebagaimana dikemukakan oleh Jain et al., 2020, sehingga secara teoretis berpotensi memperoleh dukungan eksekusi yang lebih sesuai dibandingkan CPU x86 desktop atau server. Namun demikian, tanpa pengujian langsung pada perangkat tersebut, penelitian ini tidak dapat memastikan besarnya percepatan yang benar-benar akan diperoleh, sehingga hasil pengukuran waktu inferensi pada penelitian ini lebih tepat diinterpretasikan sebagai perbandingan relatif antarmodel dalam kondisi pengujian yang identik, bukan sebagai representasi performa aktual pada perangkat target.

Model YOLO11n, yang sebelumnya memperoleh performa deteksi terbaik, mencatat waktu inferensi sebesar 119,05 ms pada format TFLite INT8. Adapun sebelum konversi, YOLO11n mencatat waktu inferensi sebesar 115,76 ms. Hasil ini menunjukkan bahwa YOLO11n tidak hanya mampu mempertahankan performa deteksi yang tinggi setelah kuantisasi, tetapi juga tetap memiliki efisiensi komputasi yang kompetitif, berada di posisi kedua tercepat di antara seluruh model TFLite yang diuji setelah YOLO26n.

Model YOLO26n menawarkan waktu inferensi paling rendah sebesar 114,83 ms pada format TFLite dan 104,91 ms sebelum konversi, menjadikannya model tercepat pada kedua kondisi pengujian. Namun demikian, keputusan pemilihan model akhir perlu mempertimbangkan hasil performa deteksi yang telah dibahas pada subbab sebelumnya. Dengan demikian, evaluasi kelayakan model untuk implementasi *edge computing* tidak hanya

didasarkan pada kecepatan inferensi, tetapi juga mempertimbangkan keseimbangan antara performa deteksi, ukuran model, dan efisiensi komputasi.

Secara keseluruhan, meskipun angka waktu inferensi TFLite INT8 pada lingkungan pengujian ini tidak mencerminkan performa optimal akibat keterbatasan dukungan akselerasi sebagaimana telah dibahas, seluruh varian *nano* tetap tercatat tidak lebih dari 140 ms dan seluruh varian *small* tidak lebih dari 400 ms bahkan tanpa akselerasi tersebut. Hasil ini tetap menunjukkan bahwa format TFLite INT8 merupakan pendekatan yang layak untuk mendukung implementasi sistem deteksi bakteri TB pada lingkungan *edge computing*, dengan potensi efisiensi waktu komputasi yang lebih baik lagi apabila dijalankan pada perangkat dengan dukungan akselerasi yang sesuai.

4.10 Hasil Evaluasi dan Analisis Komparatif

Subbab ini menyajikan hasil eksperimen menyeluruh terhadap seluruh hasil pengujian yang telah dibahas pada subbab-subbab sebelumnya, meliputi metrik performa dan metrik efisiensi dari seluruh model dalam format TFLite INT8. Analisis komparatif ini bertujuan untuk menentukan model arsitektur YOLO yang paling optimal untuk implementasi sistem deteksi bakteri TB pada perangkat *edge computing*, dengan mempertimbangkan keseimbangan antara performa deteksi dan efisiensi komputasi.

Tabel 4.9 merangkum seluruh metrik performa dan efisiensi dari delapan model TFLite INT8 yang diuji. Berdasarkan tabel tersebut, analisis komparatif dilakukan dari dua dimensi utama, yaitu dimensi performa yang mencakup mAP@50, mAP@50-95, *Precision*, *Recall*, dan *F1-Score*, serta dimensi efisiensi yang mencakup parameter, ukuran model, dan waktu inferensi.

Tabel 4.9 Perbandingan Hasil Metrik Model TFLite

Model	**Metrik Performa**					**Metrik Efisiensi**		
	mAP @50	**mAP @50-95**	***Precision***	***Recall***	**F1-Score**	**Parameter**	**Ukuran (Mb)**	**Waktu Inferensi (ms)**
YOLOv8n	**0,835**	0,412	**0,792**	0,768	0,780	3 M	3,19	126,58
YOLOv8s	0,814	0,394	0,777	0,760	0,768	11,1 M	11,01	397,71
YOLOv10n	0,812	0,402	0,734	0,781	0,757	2,3 M	**2,58**	133,60
YOLOv10s	0,826	0,408	0,756	**0,787**	0,771	7,2 M	7,43	339,10

YOLO11n	0,833	**0,413**	0,787	0,775	**0,781**	2,5 M	2,85	119,05
YOLO11s	0,830	0,406	0,779	0,778	0,778	9,4 M	9,46	336,91
YOLO26n	0,813	0,393	0,761	0,770	0,766	2,3 M	2,70	**114,83**
YOLO26s	0,812	0,395	0,747	0,777	0,761	9,5 M	9,58	350,95

Berdasarkan hasil perbandingan pada Tabel 4.9, kedelapan model menunjukkan karakteristik performa dan efisiensi yang beragam, sehingga tidak terdapat satu model yang unggul secara mutlak pada seluruh metrik pengujian. Dari dimensi performa, YOLOv8n mencatat mAP@50 tertinggi sebesar 0,835 sekaligus *Precision* tertinggi sebesar 0,792, menunjukkan proporsi prediksi positif yang benar paling tinggi di antara seluruh model.

Sementara itu, YOLO11n mencatat mAP@50-95 tertinggi sebesar 0,413 dan F1-Score tertinggi sebesar 0,781, menjadikannya model dengan keseimbangan performa terbaik secara keseluruhan, mengingat F1-Score merepresentasikan harmonisasi antara *Precision* dan *Recall*. Adapun nilai *Recall* tertinggi dipegang oleh YOLOv10s sebesar 0,787, menunjukkan kemampuan terbaik dalam mendeteksi seluruh bakteri yang ada pada citra. Dalam konteks medis, *Recall* yang tinggi memiliki peran penting karena kegagalan mendeteksi bakteri TBC yang sebenarnya ada (*false negative*) berpotensi menyebabkan pasien tidak mendapat diagnosis dan penanganan yang tepat.

Dari dimensi efisiensi, YOLO26n menempati posisi teratas dengan waktu inferensi tercepat sebesar 114,83 ms, ukuran model sebesar 2,70 Mb, serta parameter sebanyak 2,3 juta. Posisi kedua ditempati oleh YOLO11n dengan waktu inferensi 119,05 ms, hanya terpaut 4,22 ms atau sekitar 3,68 persen lebih lambat dari YOLO26n, sehingga tetap kompetitif meskipun bukan yang tercepat. Adapun model dengan ukuran paling ringan adalah YOLOv10n dengan ukuran 2,58 Mb dan parameter yang sama dengan YOLO26n, yaitu 2,3 juta, meskipun waktu inferensinya berada di posisi keempat, yaitu 133,60 ms. Fenomena ini menarik karena meskipun YOLOv10n telah mengadopsi pendekatan NMS-free yang serupa dengan YOLO26 untuk memangkas latensi *post-processing* (A. Wang et al., 2024), model ini tidak menjadi kandidat yang bersaing ketat dari sisi kecepatan, sehingga menunjukkan bahwa ukuran file dan kuantitas parameter yang kecil tidak selalu berkorelasi langsung dengan waktu inferensi yang cepat.

Mengingat keunggulan setiap model tersebar pada metrik yang berbeda-beda, pemilihan model akhir dalam penelitian ini tidak diarahkan pada satu model tunggal yang dianggap

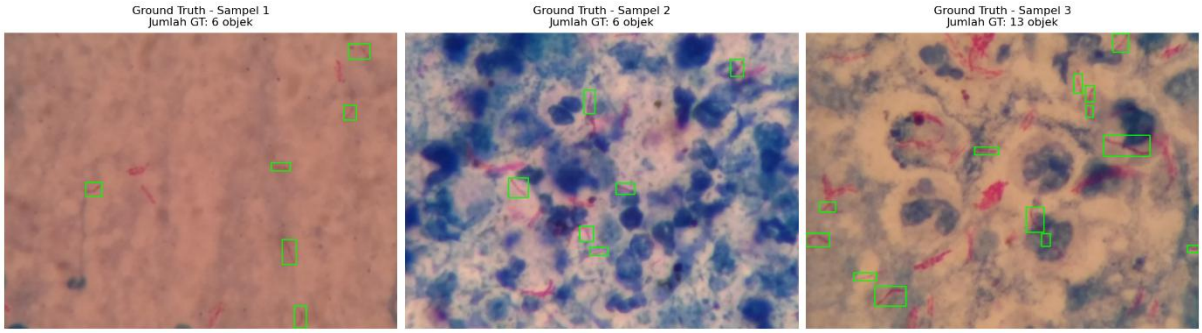
terbaik secara umum, melainkan disesuaikan dengan prioritas kebutuhan implementasi. Apabila prioritas utama adalah sensitivitas deteksi yang tinggi untuk meminimalkan risiko bakteri TBC yang luput terdeteksi, YOLOv10s menjadi pilihan yang paling sesuai berkat nilai *Recall* tertinggi sebesar 0,787. Apabila prioritas utama adalah akurasi prediksi dan minimalisasi kesalahan deteksi positif palsu, YOLOv8n menjadi pilihan yang lebih sesuai berkat *Precision* dan mAP@50 tertinggi. Apabila prioritas utama adalah kecepatan pemrosesan untuk kebutuhan *real-time* pada perangkat *edge* dengan keterbatasan daya komputasi, YOLO26n menjadi pilihan yang paling sesuai berkat waktu inferensi tercepat dengan ukuran model yang tetap ringan. Apabila prioritas utama adalah efisiensi penyimpanan pada perangkat dengan kapasitas memori sangat terbatas, YOLOv10n menjadi kandidat yang paling sesuai berkat ukuran model terkecil. Sementara itu, apabila kebutuhan implementasi menuntut keseimbangan antara performa deteksi dan efisiensi komputasi secara umum tanpa dominasi kebutuhan spesifik pada salah satu aspek, YOLO11n dapat menjadi pertimbangan yang relevan, mengingat F1-Score tertinggi yang dimilikinya turut disertai efisiensi yang tetap kompetitif, dengan ukuran 2,85 Mb dan waktu inferensi 119,05 ms yang hanya terpaut tipis dặri YOLO26n.

Dengan demikian, pemilihan model dalam implementasi sistem deteksi bakteri *Mycobacterium tuberculosis* pada perangkat *edge computing* sebaiknya disesuaikan dengan kebutuhan spesiflk pengguna, baik itu prioritas terhadap sensitivitas deteksi, akurasi prediksi, kecepatan pemrosesan, efisiensi penyimpanan, maupun keseimbangan menyeluruh antara performa dan efisiensi, alih-alih berpatokan pada satu model tunggal yang dianggap paling optimal secara universal.

Selain analisis kuantitatif terhadap metrik performa dan efisiensi, dilakukan pula analisis terhadap visualisasi kesalahan deteksi yang dihasilkan oleh model. Analisis ini difokuskan pada YOLO11n dan YOLO26n sebagai dua model dengan performa dan efisiensi terbaik berdasarkan hasil pada Tabel 4.9, dengan tujuan untuk memahami karakteristik kesalahan yang masih terjadi meskipun kedua model tersebut menunjukkan hasil kuantitatif yang baik.

Gambar 4.17 merupakan visualisasi yang menunjukkan *ground truth* beserta jumlah anotasi objek bakteri TBC pada tiga sampel citra uji. Visualisasi ini merepresentasikan letak dan sebaran riil bakteri (*actual bounding box*) yang digunakan sebagai acuan kebenaran (*gold standard*) untuk memvalidasi performa deteksi model YOLO. Selain itu, pada Gambar 4.18, *bounding box* ditandai dengan warna biru yang dihasilkan model menunjukkan tingkat kesesuaian yang baik dengan posisi dan ukuran objek MTB terhadap *ground truth* (hijau).

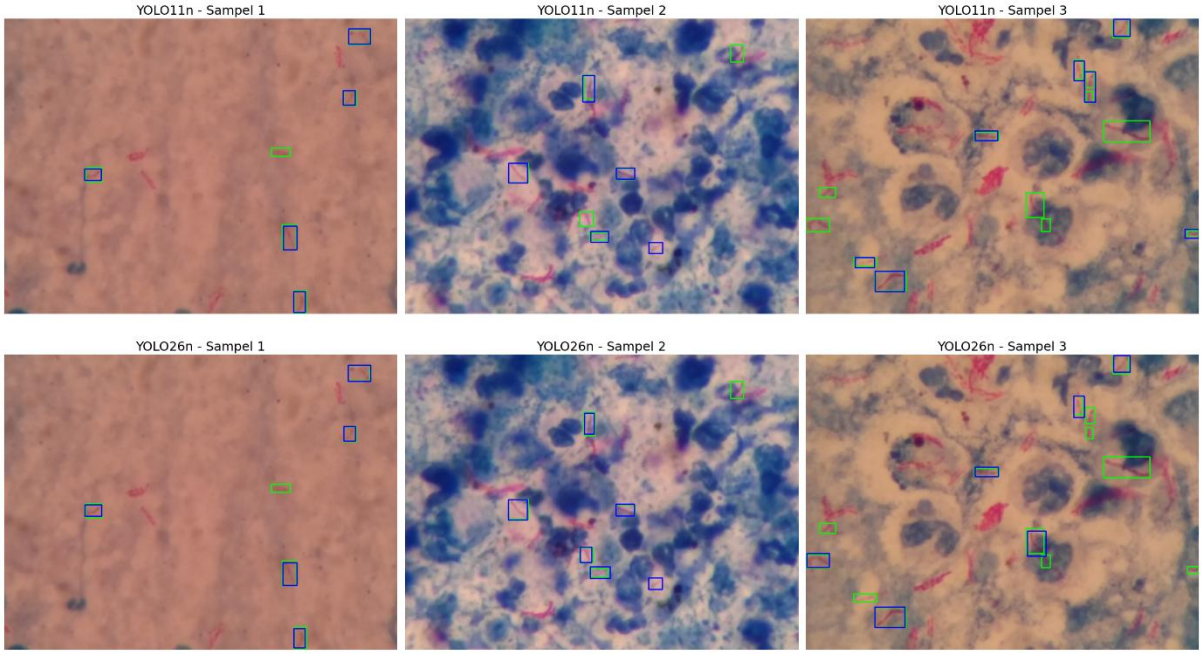
Secara umum, lokalisasi objek telah dilakukan secara akurat sehingga *bounding box* mampu merepresentasikan area objek dengan baik. Akurasi lokalisasi ini mendukung proses deteksi dan penghitungan jumlah basil secara lebih andal pada implementasi sistem deteksi tuberkulosis berbasis citra mikroskopis.



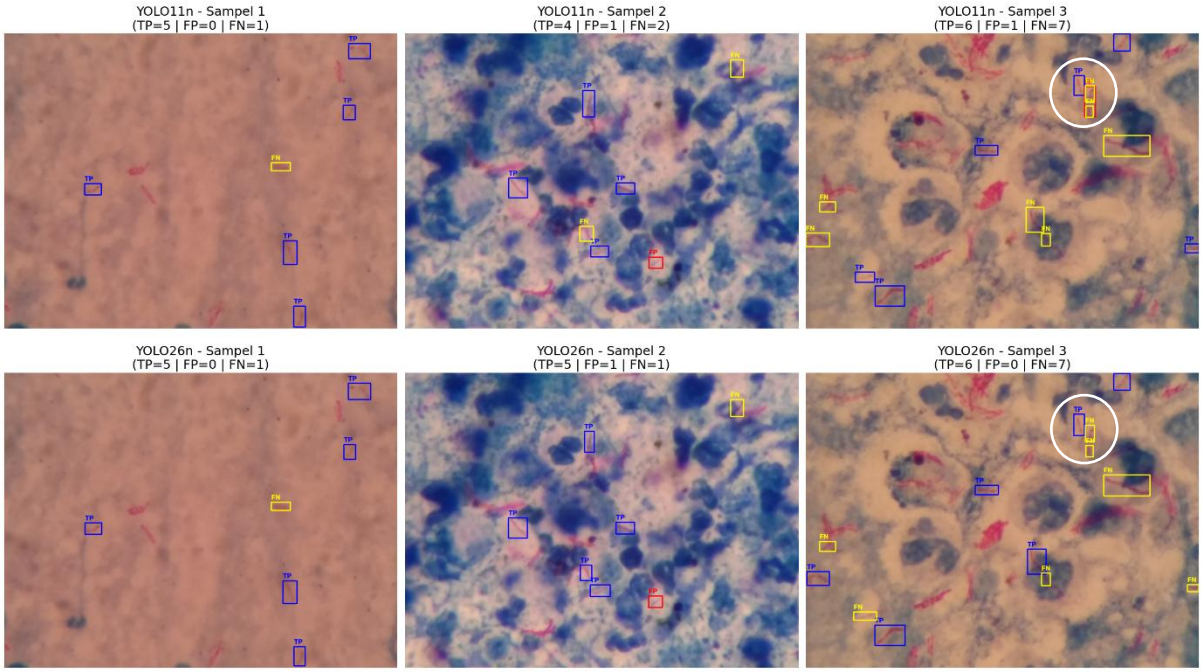
Gambar 4.17 Visualisasi *Ground Truth* dan Jumlah Anotasi Objek pada Tiga Sampel Citra

Setelah menganalisis *bounding box* terhadap *ground truth*, setiap hasil deteksi diklasifikasikan ke dalam kategori *true positive* (TP) ditandai dengan warna biru, *false positive* (FP) ditandai dengan warna merah, dan *false negative* (FN) ditandai dengan warna kuning menggunakan nilai *Intersection over Union* (IoU) *threshold* sebesar 0,5. Dengan nilai tersebut, suatu prediksi dikategorikan sebagai TP apabila memiliki nilai IoU $\geq$ 0,5 terhadap *ground truth*, sedangkan prediksi yang tidak memenuhi kriteria tersebut dikategorikan sebagai FP atau FN sesuai dengan kondisi deteksinya. Selanjutnya, dilakukan analisis kualitatif melalui pengamatan visual terhadap hasil prediksi YOLO11n dan YOLO26n pada tiga sampel citra uji dengan tingkat kompleksitas yang berbeda, sebagaimana ditunjukkan pada Gambar 4.19.

Gambar 4.19 menunjukkan pada sampel dengan kepadatan basil yang rendah (Sampel 1), kedua model menunjukkan hasil deteksi yang sangat baik. Tidak ditemukan FP pada kedua model. YOLO11n dan YOLO26n menghasilkan satu FN yaitu masih terdapat satu bakteri MTB yang tidak berhasil dideteksi. Pada sampel dengan kepadatan objek dan kompleksitas latar belakang yang lebih tinggi dan cukup banyak *noise* (Sampel 2), mulai muncul kesalahan berupa FP pada artefak yang memiliki karakteristik visual menyerupai bakteri *Mycobacterium tuberculosis*, seperti objek warna merah muda (*pink*) dan bentuk memanjang. Selain itu, kedua model juga menghasilkan FN pada beberapa bakteri MTB yang memiliki intensitas warna lebih pudar atau berukuran sangat kecil sehingga lebih sulit dibedakan dari latar belakang.



Gambar 4.18 Perbandingan Kesesuaian Bounding Box Prediksi terhadap Ground Truth

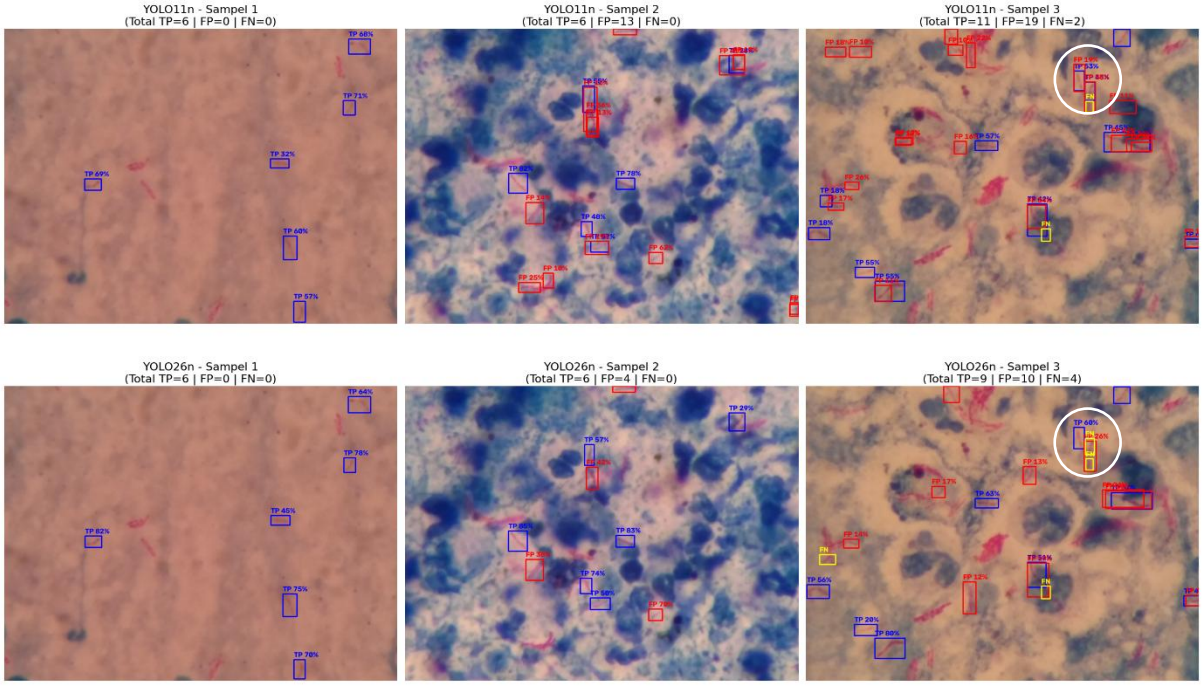


Gambar 4.19 Perbandingan Hasil Deteksi YOLO11n & YOLO26n pada Tiga Sampel Citra

Pada sampel dengan basil yang bergerombol dan saling berdekatan (Sampel 3), jumlah FP yang diperoleh adalah 1 pada YOLO11n. Pada citra tersebut terdapat dua anotasi yang saling berdampingan yang ditandai dengan area lingkaran berwarna putih. Model YOLO11n mendeteksi dua objek yang berdampingan tersebut sebagai satu objek, sehingga hasil deteksi dikategorikan sebagai FP.

Berbeda dengan YOLO11n, hasil deteksi YOLO26n terhadap dua objek basil MTB yang saling berdampingan tidak berhasil terdeteksi, jadi kedua objek tersebut dikategorikan sebagai FN. Temuan ini menunjukkan bahwa objek basil yang saling berhimpitan masih menjadi tantangan bagi kedua arsitektur model, terutama dalam memisahkan objek yang berdekatan pada citra dengan kepadatan basil tinggi.

Gambar 4.20 menyajikan hasil prediksi YOLO11n dan YOLO26n menggunakan *confidence threshold* sebesar 0,10. Nilai *threshold* yang lebih rendah ini digunakan agar prediksi dengan skor *confidence* rendah, yang pada *threshold* sebelumnya akan tersaring dan tidak tervisualisasikan, tetap dapat ditampilkan. Dengan demikian, karakteristik deteksi kedua model dapat diamati secara lebih menyeluruh, termasuk kasus prediksi dengan lokalisasi yang sebenarnya yang memiliki IoU minimal 0,5 namun disertai skor *confidence* yang rendah.



Gambar 4.20 Distribusi *Confidence Score* Hasil Deteksi

Pada YOLO11n nilai *confidence* TP pada Sampel 1 cenderung berada pada rentang menengah hingga tinggi, yaitu 32% hingga 71%. Pada Sampel 2 ditemukan TP dengan *confidence* score rendah (di bawah 50%), yaitu sebesar 48%. Pada Sampel 3 ditemukan sejumlah TP dengan *confidence* score di bawah 20%. Namun yang menarik dari hasil deteksi YOLO11n pada Sampel 3 adalah terdapat beberapa struktur yang secara visual menyerupai basil MTB pada area yang bergerombol, tetapi tidak memiliki anotasi bounding box pada ground truth. Oleh karena itu, interpretasi seluruh FP pada sampel ini perlu dilakukan secara

hati-hati karena tidak dapat dipastikan apakah struktur tersebut merupakan basil MTB yang luput dianotasi akibat keterbatasan proses anotasi manual atau memang bukan termasuk objek target dalam dataset sehingga benar-benar merupakan kesalahan deteksi model. Diperlukan validasi terhadap struktur tersebut oleh tenaga medis atau laboran untuk memastikan status objek secara pasti, tetapi validasi klinis semacam ini berada di luar cakupan penelitian dan menjadi salah satu batasan yang diakui.

Pada Sampel 1 nilai *confidence* TP mencapai 45%-78% untuk YOLO26n. Pada Sampel 2 nilai terkecil dari *confidence* TP YOLO26n sebesar 29% dan pada Sampel 3 20%. Dari beberapa hasil analisis sebelumnya, menunjukkan bahwa tidak seluruh hasil TP memiliki *confidence* tinggi, dan kompleksitas visual citra turut memengaruhi tingkat keyakinan model terhadap prediksi yang secara lokasi sudah benar.

Hasil deteksi FP pada kedua model umumnya memiliki *confidence score* pada rentang menengah hingga rendah dan cenderung mendekati ambang deteksi yang digunakan. Hal ini mengindikasikan bahwa sebagian besar kesalahan klasifikasi terjadi pada objek yang dikenali dengan tingkat keyakinan model yang relatif rendah. Pola tersebut terlihat konsisten pada kedua arsitektur, dengan jumlah FP yang meningkat seiring bertambahnya kompleksitas citra dan kepadatan basil, sebagaimana ditunjukkan pada Sampel 3, yaitu sebanyak 19 FP pada YOLO11n dan 10 FP pada YOLO26n.

Pada YOLO11n tidak ditemukan hasil deteksi FP pada Sampel 1. sedangkan pada Sampel 2 *confidence score* berada pada rentang 10%–62% dan pada Sampel 3 berada pada rentang 10%–58%. Sama seperti YOLO11n, YOLO26n tidak menghasilkan deteksi FP pada Sampel 1. Pada Sampel 2, *confidence score* FP berada pada rentang 16%–70%, sedangkan pada Sampel 3 berada pada rentang 10%–26%.

Berbeda dengan TP dan FP, FN tidak memiliki nilai *confidence score* apabila model sama sekali tidak menghasilkan prediksi yang sesuai pada *threshold* yang ditentukan terhadap *ground truth*. Namun, pada beberapa kasus ditemukan bahwa objek yang semula dikategorikan sebagai FN pada *confidence threshold* 0,5 berubah menjadi TP ketika diturunkan menjadi 0,1. Hal ini menunjukkan bahwa sebagian FN bukan disebabkan oleh kegagalan model mendeteksi objek, melainkan karena nilai *confidence score* prediksi berada di bawah ambang batas yang ditetapkan.

Oleh karena itu, penyesuaian *confidence threshold* dapat mengurangi jumlah FN pada objek yang telah diprediksi oleh model tetapi memiliki *confidence score* rendah. Selain itu, untuk FN yang terjadi karena model tidak menghasilkan prediksi sama sekali, perubahan

confidence threshold tidak akan memengaruhi hasil deteksi. Dengan demikian, pemilihan *confidence threshold* merupakan *trade-off* antara meningkatkan jumlah TP sekaligus berpotensi meningkatkan jumlah FP.

Adapun FN yang tetap muncul meskipun *confidence threshold* telah diturunkan seperti objek yang berdekatan ditandai dengan lingkaran putih pada Sampel 3 kemungkinan disebabkan oleh keterbatasan model dalam mendeteksi objek dengan karakteristik visual tertentu, seperti ukuran basil MTB yang kecil, kontras pewarnaan yang rendah terhadap latar belakang, objek yang saling berhimpitan, atau keberadaannya di antara struktur seluler pada citra mikroskopis sputum.

Secara keseluruhan, analisis terhadap sampel citra ini melengkapi hasil evaluasi kuantitatif pada Tabel 4.9 dengan memberikan pemahaman yang lebih mendalam mengenai karakteristik kesalahan deteksi yang masih dialami oleh YOLO11n dan YOLO26n. Analisis kualitatif menunjukkan bahwa kedua model masih menghadapi tantangan dalam mendeteksi basil MTB pada kondisi tertentu, seperti objek yang berukuran kecil, saling berhimpitan, memiliki kontras yang rendah terhadap latar belakang, serta adanya struktur lain yang menyerupai basil MTB. Meskipun demikian, kedua model tetap mampu menunjukkan performa deteksi yang baik secara keseluruhan berdasarkan hasil evaluasi kuantitatif dan kualitatif yang telah dilakukan.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan yang telah dilakukan, diperoleh kesimpulan sebagai berikut.

1. Berdasarkan hasil pengujian terhadap delapan varian model YOLO (YOLOv8n, YOLOv8s, YOLOv10n, YOLOv10s, YOLO11n, YOLO11s, YOLO26n, dan YOLO26s), diperoleh hasil bahwa seluruh model mampu mendeteksi bakteri Mycobacterium tuberculosis pada citra mikroskopis sputum. Tingkat performa yang diperoleh bervariasi antar arsitektur maupun antar varian ukuran model (nano dan small), yang menunjukkan bahwa performa deteksi tidak hanya dipengaruhi oleh generasi arsitektur YOLO, tetapi juga oleh kompleksitas dan jumlah parameter tiap varian. Secara umum, hasil ini menunjukkan bahwa arsitektur YOLO layak digunakan untuk tugas deteksi objek berukuran kecil seperti bakteri TB.
2. Berdasarkan perbandingan performa, hasil pengujian menunjukkan bahwa seluruh model sebelum dikonversi memperoleh nilai metrik mAP@50 pada rentang 0,811 – 0,840, mAP@50-95 pada rentang 0,392 – 0,412, *Precision* pada rentang 0,711 – 0,792, *Recall* pada rentang 0,731 – 0,802, serta F1-Score pada rentang 0,753 – 0,781. Hasil tersebut menunjukkan bahwa seluruh model mampu mempertahankan keseimbangan yang baik dalam mendeteksi bakteri TB. Selain itu, hasil tersebut juga menunjukkan bahwa setiap arsitektur memiliki karakteristik performa yang berbeda sesuai dengan metrik evaluasi yang digunakan.
3. Ditinjau dari efisiensi komputasi, kedelapan model menunjukkan variasi pada ukuran file, waktu inferensi, dan dampak konversi ke format TensorFlow Lite INT8. Hasil pengujian menunjukkan bahwa ukuran file model berbanding lurus dengan waktu inferensi, sebagaimana terlihat pola pada model yang memiliki ukuran parameter lebih kecil selalu menghasilkan waktu inferensi yang lebih cepat. Selain itu, proses konversi ke format TensorFlow Lite INT8 berhasil menghasilkan model dengan ukuran yang lebih ringan tanpa menyebabkan penurunan performa deteksi yang signifikan, sehingga sesuai untuk implementasi pada perangkat *edge computing*.
4. Hasil komparasi performa deteksi dan efisiensi komputasi pada Bab IV menunjukkan bahwa tidak terdapat satu model yang unggul secara mutlak pada seluruh aspek pengujian. Oleh karena itu, pemilihan model paling optimal untuk implementasi pada

perangkat *edge computing* perlu disesuaikan dengan prioritas kebutuhan pengguna. Model dengan *Recall* tertinggi lebih sesuai digunakan pada konteks yang mengutamakan sensitivitas deteksi, model dengan *Precision* tertinggi lebih sesuai pada konteks yang mengutamakan minimalisasi kesalahan deteksi positif palsu, sementara model dengan waktu inferensi tercepat dan ukuran model terkecil lebih sesuai untuk kebutuhan pemrosesan *real-time* pada perangkat dengan keterbatasan sumber daya. Sebagai rekomendasi umum yang menawarkan keseimbangan antara performa dan efisiensi, YOLO11n dapat menjadi pilihan yang relevan untuk implementasi sistem deteksi bakteri TB, tanpa mengesampingkan model lain sebagai alternatif sesuai kebutuhan spesifik penggunaannya.

5.2 Saran

Berdasarkan hasil penelitian, terdapat beberapa keterbatasan yang perlu ditindaklanjuti pada penelitian selanjutnya. Pertama, hasil evaluasi yang diperoleh pada penelitian ini belum dapat dipastikan bersifat konsisten atau justru dipengaruhi oleh keterbatasan eksperimental yang digunakan. Oleh karena itu, penelitian selanjutnya perlu melakukan hiperparameter *tuning* yang lebih mendalam serta menggunakan *dataset* dengan jumlah citra yang lebih besar dan lebih beragam, sehingga kemampuan generalisasi model terhadap variasi kondisi preparat dan proses pewarnaan dapat diuji secara lebih menyeluruh.

Kedua, pengujian model pada penelitian ini masih dilakukan pada lingkungan simulasi dan belum diujikan secara langsung pada perangkat *edge computing* fisik. Penelitian selanjutnya disarankan untuk melakukan pengujian langsung pada perangkat seperti Raspberry Pi atau perangkat *embedded* lainnya, sehingga performa model dapat dievaluasi pada kondisi implementasi yang sesungguhnya.

Ketiga, evaluasi kualitatif yang dilakukan pada sejumlah contoh citra belum dapat dijadikan dasar utama untuk menentukan model terbaik secara sahih, karena tingkat kompleksitas visual pada seluruh data uji belum dianalisis secara sistematis. Oleh sebab itu, penelitian selanjutnya perlu melakukan pengelompokan seluruh data uji berdasarkan tingkat kompleksitas visual, sehingga kecenderungan performa model terhadap tingkat kompleksitas citra dapat dipastikan secara lebih objektif.

Keempat, sebagaimana telah diuraikan pada Bab IV, terdapat sejumlah hasil deteksi berupa *false positive* yang secara visual menyerupai basil MTB namun tidak memiliki anotasi pada *ground truth*, sehingga status objek tersebut tidak dapat dipastikan tanpa validasi lebih lanjut. Penelitian selanjutnya perlu melibatkan tenaga medis atau laboran yang kompeten untuk

melakukan validasi terhadap struktur-struktur tersebut, sehingga dapat dipastikan apakah struktur tersebut merupakan basil MTB yang luput dianotasi akibat keterbatasan proses anotasi manual atau memang bukan termasuk objek target, sekaligus meningkatkan validitas kualitas anotasi dataset yang digunakan.

DAFTAR PUSTAKA

Altaie, U. K., Abdelkareem, A. E., & Alhasanat, A. (2025). Lightweight Optimization of YOLO Models for Resource-Constrained Devices: A Comprehensive Review. In *Diyala Journal of Engineering Sciences* (Vol. 18, Number 4, pp. 1–26). University of Diyala. https://doi.org/10.24237/djes.2025.18401

Aulia, S., Suksmono, A. B., Mengko, T. R., & Alisjahbana, B. (2024). A Novel Digitized Microscopic Images of ZN-Stained Sputum Smear and Its Classification Based on IUATLD Grades. *IEEE Access*, *12*, 51364–51380. https://doi.org/10.1109/ACCESS.2024.3386208

Boukili, M., Jebbor, N., Errajraji, K., & Elbathaoui, M. (2026). CNN model acceleration and optimization: CPU and GPU performance analysis and evaluation for multiclass classification. *Array*, *30*. https://doi.org/10.1016/j.array.2026.100746

Chen, H., Gu, W., Zhang, H., Yang, Y., & Qian, L. (2024). Research on improved YOLOv8s model for detecting mycobacterium tuberculosis. *Heliyon*, *10*(18). https://doi.org/10.1016/j.heliyon.2024.e38088

Davidson, G., Davidson, D. U., Okoye, O. K., Mensah, L. S., Ukaegbu, E. C., Agbor, D. B. A., Adegbola, M. O., Eneyo, U. S., Osuluku, B. A., Dinyain, T. O., Okeke, A. A., Okah, M. J., Owulu, A. I., Peterson, J. C., Onyekweli, S. O., Omoruyi, G. O., Adebiyi, O. S., Atoyebi, F. A., Omeje, A. C., … Uche, C. J. (2024). Overview of Tuberculosis: Causes, Symptoms and Risk Factors. *Asian Journal of Research in Infectious Diseases*, *15*(9), 8–19. https://doi.org/10.9734/ajrid/2024/v15i9370

de Carvalho Brito, V., dos Santos, P. R. S., de Carvalho Filho, A. O., & Diniz, J. O. B. (2026). Multi-task deep learning for bacilli detection and counting in sputum smear microscopy. *Biomedical Signal Processing and Control*, *112*. https://doi.org/10.1016/j.bspc.2025.108531

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2021). *A Survey of Quantization Methods for Efficient Neural Network Inference*. http://arxiv.org/abs/2103.13630

Hussain, M. (2024). *YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision*. http://arxiv.org/abs/2407.02988

Jain, A., Bhattacharya, S., Masuda, M., Sharma, V., & Wang, Y. (2020). *Efficient Execution of Quantized Deep Learning Models: A Compiler Approach*. http://arxiv.org/abs/2006.10226

Jocher, G. (2024). *Export YOLO to TFLite for Edge Devices | Ultralytics*. https://docs.ultralytics.com/integrations/tflite#why-should-you-export-to-tflite

Jocher, G. (2025). *Data Augmentation using Ultralytics YOLO | Ultralytics*. https://docs.ultralytics.com/guides/yolo-data-augmentation#example-configurations

Jocher, G., & Munawar, R. (2023). *YOLO Configuration | Ultralytics*. https://docs.ultralytics.com/usage/cfg#augmentation-settings

Jocher, G., & Raimbekovm. (2024). *Computer Vision Project Steps | Ultralytics*. https://docs.ultralytics.com/guides/steps-of-a-cv-project#step-3-data-augmentation-and-splitting-your-dataset

Jocher, G., & Raimbekovm. (2025). *Ultralytics YOLO26 - Ultralytics YOLO Docs*. Website. https://docs.ultralytics.com/models/yolo26/

Jocher, G., & Raimbekovm. (2026). *Customizing Trainer | Ultralytics*. https://docs.ultralytics.com/guides/custom-trainer#saving-the-best-model-by-custom-metric

Kamaluddin, N. A., Kassim, M., Salmah Bakar, N., & Saaidin, S. (2026). *Deep Learning-Driven Identification of Tubercle Bacilli in Ziehl-Neelsen-Stained Tissue Biopsies within Malaysian Healthcare Contexts*. https://ssrn.com/abstract=6692053

Raimbekovm. (2026). *How to Fine-Tune YOLO on a Custom Dataset | Ultralytics*. https://docs.ultralytics.com/guides/finetuning-guide#fine-tuning-vs-training-from-scratch

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. http://arxiv.org/abs/1506.02640

Rodrick, N., Sserwadda, A., & Favor, E. (2025). YOLOv8 for Real-Time Tuberculosis Detection from Low-Resolution Images Using Smartphone Cameras. *EAS Journal of Anaesthesiology and Critical Care*, *7*(05), 109–115. https://doi.org/10.36349/easjacc.2025.v07i05.007

Sapkota, R., & Karkee, M. (2026). *Ultralytics YOLO Evolution: An Overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 Object Detectors for Computer Vision and Pattern Recognition*. http://arxiv.org/abs/2510.09653

Saurina, N., Chamidah, N., Rulaningtyas, R., & Aryati, A. (2025). Multimodal deep learning from sputum image segmentation to classify Mycobacterium tuberculosis using IUATLD assessment. *Bulletin of Electrical Engineering and Informatics*, *14*(2), 1579–1590. https://doi.org/10.11591/eei.v14i2.9250

Sigit, R., Yuniarti, H., Karlita, T., Kusumawati, R., & Anwar. (2025). Automatic Detection of Tuberculosis Bacteria Using Microscope Imaging System Based on Motorized Servo and Deep Learning Techniques. *International Journal of Innovative Computing, Information and Control*, *21*(2), 273–286. https://doi.org/10.24507/ijicic.21.02.273

Sigit, R., Yuniarti, H., Karlita, T., Kusumawati, R., & Hanif Maulana, F. (2025). *Real-Time Tuberculosis Bacteria Detection Using YOLOv8*. *International Journal on Informatics Visualization journal homepage : www.joiv.org/index.php/joiv*

Tan, L., Huangfu, T., Wu, L., & Chen, W. (2021). Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification. *BMC Medical Informatics and Decision Making*, *21*(1). https://doi.org/10.1186/s12911-021-01691-8

Terven, J., & Cordova-Esparza, D. (2024). *A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS*. https://doi.org/10.3390/make5040083

Ultralytics. (2026). *Model Quantization for AI: Faster Inference | Ultralytics*. https://www.ultralytics.com/glossary/model-quantization#implementing-quantization-with-ultralytics

Vina, A. (2025a). *Accuracy, Precision & Recall: When Each Matters*. https://www.ultralytics.com/blog/accuracy-precision-recall#what-is-recall

Vina, A. (2025b). *Mean Average Precision (mAP): Object Detection*. https://www.ultralytics.com/blog/mean-average-precision-map-in-object-detection

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). *YOLOv10: Real-Time End-to-End Object Detection*. http://arxiv.org/abs/2405.14458

Wang, Y., Pan, L., & Shu, X. (2024). YOLO-FMS: A Lightweight and Efficient Model for Medical Microscopic Smear Detection. *IEEE Access*, *12*, 125253–125265. https://doi.org/10.1109/ACCESS.2024.3455555

WHO. (2025). *Global tuberculosis report 2025*.

Wulandari, S. A., Muqtadiroh, F. A., Yuniarno, E. M., Ketut Eddy Purnama, I., & Purnomo, M. H. (2026). A Digital Microscopy System and New Sputum Dataset for

Deep Learning-based Tuberculosis Detection. *International Journal of Intelligent Engineering and Systems*, *19*(3), 443–454. https://doi.org/10.22266/ijies2026.0331.27

Yeerjiang, A., Wang, Z., & Huang, X. (2024). YOLOv1 to YOLOv10: A Comprehensive Review of YOLO Variants and Their Application in Medical Image Detection. *Journal of Artificial Intelligence Practice*, *7*(3). https://doi.org/10.23977/jaip.2024.070314

Yuniarti, H., Fatichah, C., & Sigit, R. (2024). Detection and Counting Bacilli Tuberculosis Images using Classical Image Processing and Deep Learning. *ACM International Conference Proceeding Series*, 47–52. https://doi.org/10.1145/3676581.3676589

LAMPIRAN