

**TUGAS AKHIR - EF234801**

# **Deteksi Objek Multi-Kamera Berbasis Sumber Daya Menggunakan Model YOLO**

**Muhammad Fath Mushaffa Azhar**

NRP 5025201051

Dosen Pembimbing

**Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D.**

NIP 198106202005011003

**Dini Adni Navastara, S.Kom, M.Sc.**

NIP 198510172015042001

**Program Studi S1 Teknik Informatika**

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2024

*(Halaman ini sengaja dikosongkan)*



**TUGAS AKHIR - EF234801**

# **Deteksi Objek Multi-Kamera Berbasis Sumber Daya Menggunakan Model YOLO**

**Muhammad Fath Mushaffa Azhar**

NRP 5025201051

Dosen Pembimbing

**Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D.**

NIP 198106202005011003

**Dini Adni Navastara, S.Kom, M.Sc.**

NIP 198510172015042001

**Program Studi S1 Teknik Informatika**

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2024

*(Halaman ini sengaja dikosongkan)*



**FINAL PROJECT - EF234801**

## **Resource-Aware Based Multi-Camera Object Detection Using YOLO Model**

**Muhammad Fath Mushaffa Azhar**

NRP 5025201051

**Advisor**

**Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D.**

NIP 198106202005011003

**Dini Adni Navastara, S.Kom, M.Sc.**

NIP 198510172015042001

**Study Program Bachelor of Informatics**

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2024

*(Halaman ini sengaja dikosongkan)*

## **LEMBAR PENGESAHAN**

### **DETEKSI OBJEK MULTI-KAMERA BERBASIS SUMBER DAYA MENGGUNAKAN MODEL YOLO**

#### **TUGAS AKHIR**

Diajukan untuk memenuhi salah satu syarat  
Memperoleh gelar Sarjana Komputer pada  
Program Studi S-1 Teknik Informatika  
Departemen Teknik Informatika  
Fakultas Teknologi Elektro dan Informatika Cerdas  
Institut Teknologi Sepuluh Nopember

Oleh : Muhammad Fath Mushaffa Azhar

NRP. 5025201051

Disetujui oleh Tim Penguji Tugas Akhir:

1. Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D. Pembimbing
2. Dini Adni Navastara, S.Kom, M.Sc. Ko-pembimbing
3. Dr. Wahyu Suadi, S.Kom., MM., M.Kom. Penguji
4. Baskoro Adi Pratomo, S.Kom., M.Kom., Ph.D. Penguji

**SURABAYA**  
**Juli, 2024**

*(Halaman ini sengaja dikosongkan)*

## **APPROVAL SHEET**

### **RESOURCE-AWARE BASED MULTI-CAMERA OBJECT DETECTION USING YOLO MODEL**

#### **FINAL PROJECT**

Submitted to fulfill one of the requirements  
for obtaining a degree Bachelor of Computer at  
Undergraduate Study Program of Informatics  
Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology  
Institut Teknologi Sepuluh Nopember

By: Muhammad Fath Mushaffa Azhar

NRP. 5025201051

Approved by Final Project Examiner Team:

1. Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D. Advisor
2. Dini Adni Navastara, S.Kom, M.Sc. Co-Advisor
3. Dr. Wahyu Suadi, S.Kom., MM., M.Kom. Examiner
4. Baskoro Adi Pratomo, S.Kom., M.Kom., Ph.D. Examiner

**SURABAYA**  
**July, 2024**

*(Halaman ini sengaja dikosongkan)*

## **PERNYATAAN ORISINALITAS**

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : Muhammad Fath Mushaffa Azhar / 5025201051  
Departemen : Teknik Informatika  
Dosen Pembimbing / NIP : Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D /  
198106202005011003

Dengan ini menyatakan bahwa Tugas Akhir dengan judul “Deteksi Objek Multi-Kamera Berbasis Sumber Daya Menggunakan Model YOLO” adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari dietemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima saksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 10 Juli 2024

Mengetahui

Dosen Pembimbing

Mahasiswa

(Ary Mazharuddin Shiddiqi, S.Kom,  
M.Comp.Sc, Ph.D)  
NIP. 198106202005011003

(Muhammad Fath Mushaffa Azhar)  
NRP. 5025201051

*(Halaman ini sengaja dikosongkan)*

## **STATEMENT OF ORIGINALITY**

The undersigned below:

Name of student / NRP : Muhammad Fath Mushaffa Azhar / 5025201051  
Department : Informatics  
Advisor / NIP : Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D /  
198106202005011003

Hereby declare that Final Project with the title of “Resource-Aware Based Multi-Camera Object Detection Using YOLO Model” is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, 10 July 2024

Acknowledge

Advisor

Student

(Ary Mazharuddin Shiddiqi, S.Kom,  
M.Comp.Sc, Ph.D)  
NIP. 198106202005011003

(Muhammad Fath Mushaffa Azhar)  
NRP. 5025201051

*(Halaman ini sengaja dikosongkan)*

## **DETEKSI OBJEK MULTI-KAMERA BERBASIS SUMBER DAYA MENGGUNAKAN MODEL YOLO**

**Nama Mahasiswa / NRP** : Muhammad Fath Mushaffa Azhar / 5025201051  
**Departemen** : Teknik Informatika FTEIC – ITS  
**Dosen Pembimbing** : Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D

### **Abstrak**

Dalam beberapa dekade terakhir, jaringan kamera telah menjadi tren utama yang semakin signifikan. Pencatatan jangka panjang dari lingkungan yang berubah terus-menerus, menghasilkan jumlah informasi besar. Perubahan terus-menerus dalam lingkungan tersebut menyebabkan distribusi data yang fundamental juga berubah seiring waktu, yang dikenal sebagai *concept drift*. Penggunaan multi kamera untuk deteksi objek membawa tantangan besar dalam analisis data *real-time* karena volume data yang signifikan. Penelitian sebelumnya yang mengimplementasikan *Algorithm Granularity Settings* (AGS) pada deteksi orang dengan satu kamera menunjukkan akurasi tinggi, namun tetap ada batasannya dalam hal jumlah objek yang dideteksi. Oleh karena itu, penelitian ini mengusulkan sebuah sistem pemantauan dengan dua kamera dan beberapa model YOLO dengan *Resource-aware Framework* untuk menawarkan tingkat akurasi yang tinggi sambil mempertahankan penggunaan sumber daya, yaitu CPU, RAM dan *disk storage*.

Dalam penelitian ini, model YOLO yang digunakan meliputi YOLOv4-Tiny, YOLOv4, YOLOv5s, YOLOv5m, YOLOv8s, dan YOLOv8m. Untuk mendukung penggunaan dua kamera, penelitian ini akan mengimplementasikan dua kamera dengan dua metode: metode *schedule* (dua kamera bergantian) dan metode paralel (dua kamera bersamaan). Penelitian ini mencakup empat skenario berbeda, di mana setiap skenario akan menjalankan semua model YOLO. Skenario 1 melibatkan penggunaan satu kamera tanpa Resource-aware Framework, skenario 2 menggunakan satu kamera dengan Resource-aware Framework. Skenario 3 melibatkan penggunaan dua kamera tanpa Resource-aware Framework, sedangkan skenario 4 melibatkan penggunaan dua kamera dengan Resource-aware Framework. Demikian, penelitian ini akan melakukan 36 pengujian, dengan setiap program berjalan selama 3 hingga 6 jam.

Hasil penelitian menunjukkan bahwa penggunaan sumber daya pada dua kamera dengan metode *schedule* memiliki hasil dengan peningkatan yang sedikit dibandingkan dengan metode paralel. Pada metode *schedule* peningkatan CPU dan RAM hanya meningkat sekitar 0 – 6%, sedangkan metode paralel penggunaan CPU meningkat hingga 84% dan penggunaan RAM meningkat hingga 60%. Hal ini menunjukan bahwa metode paralel membutuhkan sumber daya yang lebih besar daripada metode *schedule*. Penggunaan *Resource-aware Framework* dapat mengurangi penggunaan sumber daya hingga 68% untuk penggunaan CPU, 13% untuk penggunaan RAM, dan memperlambat peningkatan penggunaan disk hingga enam kali lipat. Hal ini menunjukkan bahwa *Resource-aware Framework* dapat mengoptimalkan penggunaan sumber daya baik untuk satu kamera maupun dua kamera. Di antara beberapa model YOLO yang diuji, YOLOv8m menunjukkan tingkat akurasi tertinggi sebesar 98,77% dan tingkat akurasi kesesuaian jumlah orang sebesar 68,03%. Sementara itu, YOLOv4-Tiny memiliki *inference time* tercepat, yaitu 0,995s.

**Kata kunci:** *Multi Kamera, YOLO, Resource-Aware Framework, Deteksi Objek*

*(Halaman ini sengaja dikosongkan)*

# **RESOURCE-AWARE BASED MULTI-CAMERA OBJECT DETECTION USING YOLO MODEL**

**Student Name / NRP : Muhammad Fath Mushaffa Azhar / 5025201051**

**Department : Informatics ELECTICS - ITS**

**Advisor : Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D**

## **Abstract**

In recent decades, camera networks have become an increasingly significant trend. Long-term recording of continuously changing environments generates vast amounts of information. These continuous changes cause the fundamental data distribution to shift over time, known as concept drift. The use of multi-camera systems for object detection presents substantial challenges in real-time data analysis due to the significant data volume. Previous research implementing Algorithm Granularity Settings (AGS) for person detection with a single camera demonstrated high accuracy, but there were limitations in the number of detected objects. Therefore, this study proposes a monitoring system with two cameras and several YOLO models with a Resource-aware Framework to offer high accuracy while maintaining resource usage, including CPU, RAM, and disk storage.

In this study, the YOLO models used include YOLOv4-Tiny, YOLOv4, YOLOv5s, YOLOv5m, YOLOv8s, and YOLOv8m. To support the use of two cameras, the study will implement two cameras using two methods: the schedule method (alternating between two cameras) and the parallel method (simultaneously using two cameras). The study encompasses four different scenarios, where each scenario will run all YOLO models. Scenario 1 involves using one camera without the Resource-aware Framework, scenario 2 uses one camera with the Resource-aware Framework. Scenario 3 involves using two cameras without the Resource-aware Framework, while scenario 4 involves using two cameras with the Resource-aware Framework. Thus, the study will conduct 36 tests, with each program running for 3 to 6 hours.

The results show that resource usage with two cameras and the scheduling method has minimal increases compared to the parallel method. With the scheduling method, CPU and RAM usage increases by only 0 – 6%, while the parallel method shows an 84% increase in CPU usage and a 60% increase in RAM usage. This indicates that the parallel method requires significantly more resources than the scheduling method. The Resource-aware Framework can reduce resource usage by up to 68% for CPU, 13% for RAM, and slow down disk usage increase by up to six times. This suggests that the Resource-aware Framework can optimize resource usage for both single and dual-camera setups. Among the tested YOLO models, YOLOv8m achieved the highest accuracy of 98.77% and the highest person count accuracy of 68.03%, while YOLOv4-Tiny had the fastest inference time of 0.995 seconds.

**Keywords:** *Multi-Camera, YOLO, Resource-Aware Framework, Object Detection*

*(Halaman ini sengaja dikosongkan)*

## **KATA PENGANTAR**

Puji syukur ke hadirat Allah SWT atas segala Rahmat, petunjuk, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul "Deteksi Objek Multi-Kamera Berbasis Sumber Daya Menggunakan Model YOLO". Tugas akhir ini tidak dapat diselesaikan tanpa bantuan serta dukungan dari banyak pihak yang terlibat, baik secara langsung maupun tidak langsung. Oleh karena itu, penulis ingin mengucapkan rasa terima kasih kepada:

1. Bapak Ary Mazharuddin Shiddiqi, S.Kom, M.Comp.Sc, Ph.D selaku dosen pembimbing utama yang senantiasa memberikan ide penelitian, bimbingan, dukungan, saran, dan masukan selama pengerjaan Tugas Akhir ini,
2. Ibu Dini Adni Navastara, S.Kom, M.Sc. selaku dosen pembimbing yang senantiasa memberikan bimbingan, dukungan, saran dan masukan selama pengerjaan Tugas Akhir ini.
3. Keluarga, terutama Ayah dan Ibu yang senantiasa memberikan perhatian, kasih sayang, dukungan, motivasi, dan doa sebagai penyemangat selama masa perkuliahan dan pengerjaan Tugas Akhir ini.
4. Teman-teman dari Pondok Pesantren Mahasiswa Khoirul Huda 2 yang memberikan semangat dan dukungan dalam membantu pengumpulan data Tugas Akhir ini.
5. Pihak lain yang membantu memberikan saran, dukungan, serta doa kepada penulis yang tidak dapat penulis sebutkan satu per satu.

Dalam penyusunan Tugas Akhir ini, penulis menyadari masih terdapat banyak kekurangan sehingga kritik dan saran yang membangun sangat diharapkan untuk perbaikan. Semoga Tugas Akhir ini dapat memberikan manfaat.

Surabaya, 10 Juli 2024

Penulis

*(Halaman ini sengaja dikosongkan)*

## DAFTAR ISI

HALAMAN JUDUL.....	i
LEMBAR PENGESAHAN .....	v
PERNYATAAN ORISINALITAS .....	ix
ABSTRAK.....	xiii
KATA PENGANTAR .....	xvii
DAFTAR ISI.....	xix
DAFTAR GAMBAR/GRAFIK/DIAGRAM .....	xxi
DAFTAR TABEL.....	xxiii
DAFTAR LAMPIRAN .....	xxv
BAB I PENDAHULUAN .....	1
1.1    Latar Belakang.....	1
1.2    Rumusan Permasalahan.....	2
1.3    Batasan Masalah .....	2
1.4    Tujuan.....	2
1.5    Manfaat.....	2
BAB II TINJAUAN PUSTAKA.....	3
2.1    Penelitian Terkait.....	3
2.2    Dasar Teori .....	4
2.2.1    Data Stream Mining.....	4
2.2.2    Deep Learning untuk Deteksi Objek .....	5
2.2.3    Algoritma YOLO.....	8
2.2.4    Resource-Aware Framework.....	10
2.2.5    Raspberry Pi .....	13
2.2.6    IP Camera ColorVu .....	14
BAB III METODOLOGI.....	17
3.1    Metode yang Dirancang.....	17
3.1.1    Studi Literatur.....	17
3.1.2    Implementasi Sistem.....	17
3.1.3    Pengujian dan Evaluasi Hasil .....	22
3.2    Peralatan pendukung.....	22
3.3    Implementasi .....	23
BAB IV HASIL DAN PEMBAHASAN .....	29
4.1    Hasil Pengujian.....	29

4.1.1	Skenario 1: Satu Kamera tanpa Resource-aware Framework .....	29
4.1.2	Skenario 2: Satu Kamera Menggunakan Resource-aware Framework.....	32
4.1.3	Skenario 3: Dua Kamera tanpa Resource-aware Framework .....	35
4.1.4	Skenario 4: Dua Kamera Menggunakan Resource-aware Framework .....	41
4.2	Pembahasan.....	46
4.2.1	Model YOLO .....	46
4.2.2	CPU .....	49
4.2.3	RAM.....	52
4.2.4	Disk .....	54
	BAB V KESIMPULAN DAN SARAN .....	58
5.1	Kesimpulan.....	59
5.2	Saran.....	60
	DAFTAR PUSTAKA .....	61
	LAMPIRAN-LAMPIRAN ATAU APPENDIKS .....	63
	BIODATA PENULIS .....	93

## DAFTAR GAMBAR/GRAFIK/DIAGRAM

Gambar 2.1	<i>Deep Learning</i> merupakan Bagian dari <i>Machine Learning</i> , yang merupakan Bagian dari <i>Artificial Intelligence</i> (Jakhar & Kaur, 2020) .....	6
Gambar 2.2	Domain Aplikasi <i>Deep Learning</i> (Shinde & Shah, 2018) .....	6
Gambar 2.3	Deteksi Objek (Jiang, Zhao, Li, & Jia, 2020) .....	7
Gambar 2.4	<i>Object Detection Milestone</i> (Zou, Chen, Shi, Guo, & Ye, 2019) .....	7
Gambar 2.5	Arsitektur <i>Backbone</i> , <i>Neck</i> , dan <i>Head</i> (Terven & Cordova-Esparza, 2023) .....	8
Gambar 2.6	<i>A timeline of YOLO Version</i> (Terven & Cordova-Esparza, 2023).....	9
Gambar 2.7	<i>Algorithm Granularity Settings General Process</i> (Gaber & Yu, 2006) .....	11
Gambar 2.8	Penggunaan RAM pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023).....	12
Gambar 2.9	Penggunaan CPU pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023).....	12
Gambar 2.10	Penggunaan Disk Storage pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023).....	13
Gambar 2.11	Raspberry Pi 4 Model B.....	13
Gambar 2.12	Perbedaan Kamera Konvensional dengan Kamera ColorVu (James, 2020) ....	14
Gambar 2.13	Aperture Kamera Konvensional dan Kamera Colorvu (James, 2020) .....	15
Gambar 3.1	Arsitektur Sistem .....	17
Gambar 3.2	<i>Flowchart</i> Keseluruhan Sistem.....	19
Gambar 3.3	<i>Flowchart YOLO Thread</i> .....	21
Gambar 4.1	Persentase Penggunaan CPU YOLOv4-Tiny pada Skenario 1 .....	30
Gambar 4.2	Persentase Penggunaan RAM YOLOv4-Tiny pada Skenario 1 .....	31
Gambar 4.3	Persentase Penggunaan Disk YOLOv4-Tiny pada Skenario 1.....	31
Gambar 4.4	Persentase Penggunaan CPU YOLOv5s pada Skenario 2.....	34
Gambar 4.5	Persentase Penggunaan RAM YOLOv5s pada Skenario 2 .....	34
Gambar 4.6	Persentase Penggunaan Disk YOLOv5s pada Skenario 2 .....	34
Gambar 4.7	Persentase Penggunaan CPU YOLOv5m dengan Metode <i>Schedule</i> pada Skenario 3 .....	37
Gambar 4.8	Persentase Penggunaan RAM YOLOv5m dengan Metode <i>Schedule</i> pada Skenario 3 .....	38
Gambar 4.9	Persentase Penggunaan Disk YOLOv5m dengan Metode <i>Schedule</i> pada Skenario 3 .....	38
Gambar 4.10	Persentase Penggunaan CPU YOLOv8m dengan Metode Paralel pada Skenario 3.....	39
Gambar 4.11	Persentase Penggunaan RAM YOLOv8m dengan Metode Paralel pada Skenario 3 .....	39
Gambar 4.12	Persentase Penggunaan Disk YOLOv8m dengan Metode Paralel pada Skenario 3.....	39
Gambar 4.13	Persentase Penggunaan CPU YOLOv4 dengan Metode <i>Schedule</i> pada Skenario 4.....	43
Gambar 4.14	Persentase Penggunaan RAM YOLOv4 dengan Metode <i>Schedule</i> pada Skenario 4 .....	43
Gambar 4.15	Persentase Penggunaan Disk YOLOv4 dengan Metode <i>Schedule</i> pada Skenario 4.....	43

Gambar 4.16	Persentase Penggunaan CPU YOLOv8m dengan Metode Paralel pada Skenario 4 .....	44
Gambar 4.17	Persentase Penggunaan RAM YOLOv8m dengan Metode Paralel pada Skenario 4 .....	44
Gambar 4.18	Persentase Penggunaan Disk YOLOv8m dengan Metode Paralel pada Skenario 4 .....	45
Gambar 4.19	Perbandingan Format pada Model YOLOv8s (Jocher, Chaurasia, & Qiu, 2023) .....	48
Gambar 4.20	Perbandingan Penggunaan CPU YOLOv8s Format NCNN dan PyTorch .....	49
Gambar 4.21	Persentase Penggunaan CPU YOLOv4-tiny pada Satu Kamera di Pengujian Skenario 1 .....	51
Gambar 4.22	Persentase Penggunaan CPU YOLOv4-tiny pada Dua Kamera dengan Metode Paralel di Pengujian Skenario 3.....	51
Gambar 4.23	Persentase Penggunaan CPU dalam Rentang Waktu 10.000 Detik .....	52
Gambar 4.24	Perbandingan Penggunaan Persentase Penggunaan RAM pada Beberapa Model YOLO dalam Interval 100 Detik.....	54

## DAFTAR TABEL

Tabel 2.1	Perbandingan antara <i>Traditional Data Mining</i> dan <i>Data Stream Mining</i> Utama (Alothali, Alashwal, & Harous, 2019) .....	5
Tabel 2.2	Perbandingan <i>Average Precision</i> Beberapa Versi Model YOLO (Terven & Cordova-Esparza, 2023).....	9
Tabel 2.3	Perubahan Kondisi yang Dipengaruhi Algoritma (Gaber & Yu, 2006) .....	11
Tabel 3.1	Hasil Pelatihan Beberapa Model YOLO .....	18
Tabel 4.1	Perbandingan Hasil Pengujian Skenario 1 pada Model YOLO .....	29
Tabel 4.2	Selisih Waktu Proses Deteksi YOLOv4-Tiny pada Skenario 1 dalam Interval 1000 Detik.....	31
Tabel 4.3	Perbandingan Hasil Pengujian Skenario 2 pada Beberapa Model YOLO .....	32
Tabel 4.4	Selisih Waktu Proses Deteksi YOLOv5s pada Skenario 2 dalam Interval 1000 Detik.....	35
Tabel 4.5	Perbandingan Hasil Pengujian Skenario 3 pada Beberapa Model YOLO .....	36
Tabel 4.6	Selisih Waktu Proses Deteksi YOLOv5m dengan Metode <i>Schedule</i> pada Skenario 3 dalam Interval 1000 Detik.....	38
Tabel 4.7	Selisih Waktu Kamera 1 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 3 dalam Interval 1000 Detik .....	40
Tabel 4.8	Selisih Waktu Kamera 2 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 3 dalam Interval 1000 Detik .....	40
Tabel 4.9	Perbandingan Hasil Pengujian Skenario 4 pada Beberapa Model YOLO .....	41
Tabel 4.10	Selisih Waktu Proses Deteksi YOLOv4 dengan Metode <i>Schedule</i> pada Skenario 4 dalam Interval 1000 Detik.....	44
Tabel 4.11	Selisih Waktu Kamera 1 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 4 dalam Interval 1000 Detik .....	45
Tabel 4.12	Selisih Waktu Kamera 2 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 4 dalam Interval 1000 Detik .....	45
Tabel 4.13	Perbandingan Tingkat Akurasi pada Beberapa Model YOLO .....	46
Tabel 4.14	Perbandingan Rata-rata Jumlah Deteksi Antar Model YOLO .....	47
Tabel 4.15	Perbandingan <i>Inference Time</i> pada Hasil Pengujian Beberapa Model YOLO .....	48
Tabel 4.16	Perbandingan Rata-rata Penggunaan CPU pada Semua Skenario .....	49
Tabel 4.17	Persentase Penurunan Penggunaan CPU dengan menggunakan <i>Resource-aware Framework</i> .....	50
Tabel 4.18	Persentase Peningkatan Penggunaan CPU dengan Perbandingan Penggunaan Satu Kamera dan Dua Kamera .....	50
Tabel 4.19	Perbandingan Rata-rata Penggunaan RAM pada Semua Skenario .....	52
Tabel 4.20	Persentase Penurunan Penggunaan RAM dengan menggunakan <i>Resource-aware Framework</i> .....	53
Tabel 4.21	Persentase Peningkatan Penggunaan RAM dengan Perbandingan Penggunaan Satu Kamera dan Dua Kamera .....	54
Tabel 4.22	Perbandingan Waktu Disk Naik 1% pada Skenario 1 dan 2 .....	55
Tabel 4.23	Perbandingan Waktu Disk Naik 1% pada Skenario 3 dan 4 .....	55
Tabel 4.24	Persentase Peningkatan Waktu Disk Naik 1% dengan menggunakan <i>Resource-aware Framework</i> .....	56
Tabel 4.25	Persentase Penurunan Waktu Disk Naik 1% pada Penggunaan Dua Kamera dengan Metode <i>Schedule</i> .....	56
Tabel 4.26	Persentase Penurunan Waktu Disk Naik 1% pada Penggunaan Dua Kamera dengan Metode Paralel.....	57

*(Halaman ini sengaja dikosongkan)*

## **DAFTAR LAMPIRAN**

Lampiran 1. Gambar Penggunaan Sumber Daya pada Skenario 1.....	63
Lampiran 2. Gambar Penggunaan Sumber Daya pada Skenario 2.....	67
Lampiran 3. Gambar Penggunaan Sumber Daya pada Skenario 3.....	71
Lampiran 4. Gambar Penggunaan Sumber Daya pada Skenario 4.....	80
Lampiran 5. Data Hasil <i>Running</i> .....	89

*(Halaman ini sengaja dikosongkan)*

## BAB I PENDAHULUAN

### 1.1 Latar Belakang

Tren utama dalam beberapa dekade terakhir adalah meningkatnya signifikansi jaringan kamera. Pencatatan dalam jangka waktu yang panjang, yakni berjam-jam, berhari-hari, dan mungkin bahkan bertahun-tahun, menghasilkan sumber informasi yang sangat besar dari lingkungan yang terus berubah. Lingkungan ini mengalami evolusi secara konstan di mana metode pembelajaran konvensional tidak mampu mengikuti perubahan yang terjadi. Di tengah lingkungan semacam itu, distribusi data pokoknya berubah seiring waktu (yang dikenal sebagai *concept drift*) baik disebabkan oleh perubahan intrinsik seperti perubahan pose atau gerakan, maupun perubahan ekstrinsik seperti kondisi pencahayaan, perubahan latar belakang yang dinamis, kompleksitas latar belakang objek, perubahan sudut kamera, dan lain sebagainya (Khoshrou, Cardoso, & Teixeira, 2014).

Sementara itu, dalam konteks penggunaan multi kamera untuk deteksi objek, volume data yang signifikan yang dihasilkan oleh sensor yang terhubung ke dalam sistem menggambarkan tantangan besar dalam analisis data secara *real-time*. Analisis data *real-time* yang dilakukan menggunakan data dari multi kamera ini menjadi krusial dalam mendeteksi perubahan atau penyimpangan dalam sistem dengan segera. Ketersediaan memori yang cukup dan sumber daya yang memadai menjadi kunci dalam memproses data yang terus menerus. Penggunaan *data stream* berupa gambar dari multi kamera menambah kompleksitas dalam sumber daya yang efisien. Oleh karena itu, perlu pengaturan yang tepat untuk memastikan penggunaan sumber daya yang minimal tanpa mengorbankan performa.

Pendekatan menggunakan teknik *concept drift* atau perubahan konsep data seiring waktu dapat menjadi solusi dalam menangani masalah ini (Patil & Wadhai, 2012). Pendekatan ini melibatkan penggunaan *classifier*, teknik statistik, *windows based*, dan *ensemble*, masing-masing memiliki kelebihan dan kelemahan tersendiri (Wares, Isaacs, & Elyan, 2019). Referensi lainnya, seperti penelitian yang dilakukan oleh Gaber dan Yu (Gaber & Yu, 2006) (Cuzzocrea, Gaber, & Shiddiqi, 2015) menyoroti pentingnya pengaturan ketersediaan sumber daya selama proses *data mining*. Konsep pengaturan granularity pada input, proses, dan output, disebut sebagai *Algorithm Granularity Settings* (AGS), menjadi kunci dalam menjaga ketersediaan sumber daya.

Pada penelitian lain yang dilakukan oleh Shiddiqi, Yogatama, dan Navastara (Shiddiqi, Yogatama, & Navastara, 2023), telah berhasil mengimplementasikan AGS pada sistem deteksi orang dengan satu kamera dengan algoritma YOLOv4-Tiny. Sistem yang diusulkan memiliki akurasi deteksi sebesar 99,21%, dihitung dengan membagi jumlah frame video yang terdeteksi dengan benar dengan kebenaran dasar pengamatan manusia. Dari jumlah tersebut, 43,65% merupakan deteksi yang sepenuhnya benar, dan 56,34% merupakan deteksi yang sebagian benar, yang menunjukkan bahwa sistem dapat mendeteksi beberapa tetapi tidak semua orang di dalam ruangan atau mendeteksi mereka sebagai objek lain. Meskipun demikian, sistem ini memenuhi tantangannya karena secara akurat mendeteksi ada atau tidaknya orang di dalam ruangan.

Sebagai solusi untuk mengatasi tantangan dalam deteksi objek dengan mempertimbangkan ketersedian sumber daya, Tugas Akhir ini mengusulkan sistem pemantauan dengan menggunakan dua kamera dan menerapkan beberapa model YOLO dengan *Resource-aware Framework*. Hal ini diharapkan dapat meningkatkan akurasi deteksi objek sambil tetap mempertahankan penggunaan sumber daya.

## 1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana implementasi sistem pemantauan menggunakan multi kamera dengan memperhatikan pengaturan dan optimalisasi penggunaan sumber daya?
2. Bagaimana hasil kinerja penggunaan sumber daya menggunakan *Resource-aware Framework* dengan multi-kamera?
3. Bagaimana hasil evaluasi kinerja beberapa varian Algoritma YOLO terhadap pemantauan multi kamera, dengan memperhatikan aspek penggunaan sumber daya?

## 1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, diantaranya sebagai berikut:

1. Gambar yang ditangkap bersumber dari dua *IP camera*.
2. Lingkup uji coba di ruang laboratorium Komputasi Berbasis Jaringan di Gedung Teknik Informatika.
3. Perangkat yang digunakan yaitu Raspberry Pi 4 model B.
4. Sumber daya yang dipantau adalah CPU, RAM, dan *disk storage*.

## 1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Merancang sistem pemantauan multi kamera yang efisien menggunakan *Resource-aware Framework*, memastikan optimalitas penggunaan sumber daya dalam lingkungan pemantauan yang dinamis.
2. Mengevaluasi efektivitas dan efisiensi penggunaan sumber daya pada sistem multi-kamera dengan menerapkan *Resource-aware Framework*.
3. Membandingkan kinerja berbagai varian Algoritma YOLO dalam konteks pemantauan multi kamera untuk memastikan akurasi deteksi objek yang tinggi sekaligus mempertahankan penggunaan sumber daya.

## 1.5 Manfaat

Tugas Akhir ini diharapkan dapat menghasilkan sistem pemantauan multi-kamera yang efisien secara *Resource-aware Framework*. Melalui perbandingan kinerja berbagai varian Algoritma YOLO dalam konteks pemantauan multi-kamera, untuk menentukan solusi yang dapat mengoptimalkan penggunaan sumber daya tanpa mengorbankan akurasi deteksi objek. Sehingga dapat memberikan kontribusi berharga dalam pengembangan teknologi pemantauan yang murah, efisien dan handal.

## BAB II TINJAUAN PUSTAKA

### 2.1 Penelitian Terkait

Telah banyak penelitian yang dilakukan untuk mengoptimalkan penggunaan sumber daya supaya dapat memproses *data stream* secara *real time* dengan lebih efektif. *Data stream* dapat dihasilkan dari berbagai macam perangkat ataupun sensor, salah satunya sensor *Internet of Things* (IoT) yang penting untuk pemantauan secara real time. Arsitektur umum dari perangkat IoT mencakup beberapa komponen dasar sebagai berikut (Alieksieiev & Gaiduchok, 2017)

- *Device*: sebuah *remote computer* seperti Raspberry Pi atau perangkat khusus yang dapat menyertakan berbagai macam sensor.
- *Internet*: segala jenis koneksi nirkabel internet melalui Wi-Fi, GPRS, 3G/4G atau apapun yang disertakan di jaringan seluler.
- *Cloud*: media IoT yang popular untuk menyimpan dan memproses *big data*.

Penambangan *data stream* merupakan data yang sulit diprediksi karena cenderung berubah seiring waktu dengan pola yang tidak terduga. Oleh karena itu, *concept drift* menjadi relevan. Fenomena *concept drift* diakui sebagai faktor utama yang menyebabkan penurunan efektivitas pada berbagai sistem informasi yang berbasis data (Lu, Liu, Dong, Gu, Gama, & Zhang, 2019). Terdapat banyak sekali pendekatan yang telah dilakukan untuk mengatasi *concept drift*, salah satunya dari pendekatan kerangka *class-based ensemble* pada sistem *parallel stream* (Khoshrou, Cardoso, & Teixeira, 2014).

Pada pengolahan *data stream mining*, terdapat beragam pendekatan yang digunakan, salah satunya adalah melalui pendekatan *Algorithm Granularity Settings* (AGS). Merujuk pada jurnal yang berjudul *A Holistic Approach for Resource-aware Adaptive Data Stream Mining* (Gaber & Yu, 2006) mengidentifikasi tiga pengaturan utama yang menentukan ketersediaan sumber daya saat melakukan proses *data stream mining*, yaitu input, proses, dan output. Pengaturan input disebut sebagai *Algorithm Input Granularity* (AIG), pengaturan proses disebut sebagai *Algorithm Processing Granularity* (APG), dan pengaturan output disebut sebagai *Algorithm Output Granularity* (AOG). Ketiga pengaturan tersebut, secara kolektif, membentuk keseluruhan dari *Algorithm Granularity Settings* (AGS). Hasil penelitian menunjukkan bahwa dengan menggunakan pendekatan AGS dapat menghemat sumber daya saat memproses *data stream*. Pada jurnal lain yang berjudul *Distributed Classification of Data Streams: An Adaptive Technique* (Cuzzocrea, Gaber, & Shiddiqi, 2015) menggunakan pendekatan *Algorithm Granularity Settings* (AGS) pada teknik klasifikasi *data stream* terdistribusi yang menggunakan sensor jaringan Sun SPOT. Penelitian ini secara khusus mengimplementasikan seluruh aspek dari pendekatan AGS, termasuk pengaturan input, proses, dan output, dalam sistem yang mampu beroperasi pada lingkungan yang terdistribusi.

Penelitian ini mengacu pada jurnal yang berjudul Resource-aware video streaming (RAViS) framework for object detection system using deep learning algorithm (Shiddiqi, Yogatama, & Navastara, 2023) yang membahas penggunaan *Resource-aware Framework* untuk deteksi orang pada sebuah CCTV. Pada penelitian tersebut dapat menekan penggunaan sumber daya sambil mempertahankan tingkat akurasi yang tinggi. Sistem yang diusulkan memiliki tingkat akurasi deteksi sebesar 99,21%. Dari jumlah tersebut, 43,65% merupakan deteksi yang sepenuhnya benar, dan 56,34% merupakan deteksi yang sebagian benar, yang

menunjukkan bahwa sistem dapat mendeteksi beberapa tetapi tidak semua orang di dalam ruangan atau mendeteksi mereka sebagai objek lain. Terkait penggunaan satu kamera yang memiliki keterbatasan dalam mendeteksi manusia, saran penggunaan beberapa kamera didasarkan pada jurnal yang berjudul Deep Multi-Camera People Detection (Chavdarova & Fleuret, 2017). Jurnal ini menyarankan penggunaan multi-kamera yang dapat menyediakan sudut pandang yang beragam, tidak dapat diperoleh oleh satu kamera. Penggunaan multi-kamera terbukti dapat meningkatkan tingkat akurasi dan confidence dalam klasifikasi jika dibandingkan dengan penggunaan satu kamera.

Dalam jurnal YOLO Multi-Camera Object Detection and Distance Estimation (Strbac, Gostovic, Lukac, & Samardzija, 2020), pendekatan penggunaan multi-kamera ditawarkan dengan tujuan menentukan estimasi jarak antara objek menggunakan dua sudut pandang yang berbeda. Metode ini memanfaatkan algoritma YOLOv3 untuk mendeteksi objek dan algoritma Stereoscopy untuk menentukan estimasi jarak antar objek. Penelitian ini menghasilkan tingkat kesalahan rata-rata sebesar 4%, menunjukkan tingkat akurasi yang tinggi dalam estimasi jarak antar objek. Tugas Akhir ini akan mengimplementasikan Algorithm Granularity Settings (AGS) pada data stream berupa gambar yang diambil dari multi-kamera. mengusulkan sistem pemantauan dengan menggunakan dua kamera dan menerapkan beberapa varian model YOLO dengan *Resource-aware Framework*. Hal ini diharapkan dapat meningkatkan akurasi deteksi objek sambil tetap mempertahankan penggunaan sumber daya.

## 2.2 Dasar Teori

Pada subbab ini menjelaskan dasar teori yang digunakan dalam Tugas Akhir ini yang meliputi domain masalah dan teknologi.

### 2.2.1 Data Stream Mining

*Data stream mining* menjadi lingkup pengetahuan yang semakin banyak diminati seiring pentingnya penggunaan dan penyebaran generator *data stream* (Gaber & Yu, 2006). *Data stream* merujuk pada serangkaian data yang terus menerus dihasilkan, diproses, dan dikonsumsi dari waktu ke waktu (Shiddiqi, Yogatama, & Navastara, 2023). *Data stream* adalah sekumpulan data berurutan secara *real-time*, berkelanjutan, dan teratur secara implisit berdasarkan waktu kedatangan atau secara eksplisit berdasarkan stempel waktu (Mohammed & Soliman, 2010). Secara sederhana *data stream* adalah data informasi yang terus menerus mengalir.

*Data stream mining* memiliki banyak karakteristik unik yang menjadi perbedaan jika dibandingkan dengan penambangan data tradisional seperti yang ditunjukkan dalam Tabel 2.1. Banyak penelitian yang telah mengulas secara rinci karakteristik-karakteristik data stream, Secara umum, karakteristik *data stream* ini dibagi dalam tiga poin utama (Alothali, Alashwal, & Harous, 2019):

- Ukuran: Ukuran besar *data stream* membuatnya tidak mungkin disimpan sepenuhnya untuk *data mining* selanjutnya, maka hanya ringkasan *data stream mining* berisi informasi penting yang akan diambil.
- Kecepatan Tinggi: Produksi data yang cepat memerlukan efisiensi yang tinggi dalam *data mining*. Kedatangan dinamis dari *data stream* dapat berubah seiring waktu yang menyebabkan rentan terhadap kesalahan.

- Multidimensional: *Data stream* berasal dari jenis data yang beragam, maka dibutuhkan algoritma yang efisien supaya dapat menganalisis data stream yang berukuran besar dalam satu kali proses dan komputasi.

Tabel 2.1 Perbandingan antara *Traditional Data Mining* dan *Data Stream Mining* Utama (Alothali, Alashwal, & Harous, 2019)

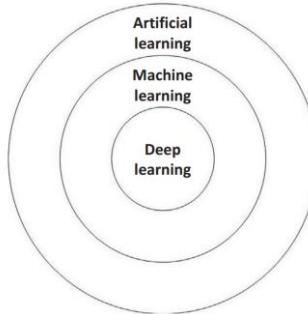
Fitur	Traditional Data Mining	Data Stream Mining
Proses	<i>Offline</i> setiap rekaman	<i>Real-time</i>
Penyimpanan	Bisa dilakukan	Tidak bisa dilakukan
Volume	Terbatas	Tidak terbatas
Pembuatan data	Statis	Dinamis
Waktu	Lebih lama	Satu kali pemindaian
Tipe data	Homogen	Heterogen
Hasil	Tepat	Perkiraan

Pada penelitian lain yang berjudul *Data Stream Mining* (Mohammed & Soliman, 2010), terdapat dua jenis *data stream* yaitu *transactional data stream* dan *measurement data stream*. *Transactional data stream* merupakan *data stream* yang mencatat interaksi antara atribut data, seperti pembelian suatu barang atau data panggilan telepon. *Measurement data stream* merupakan *data stream* yang memantau perubahan status entitas yang berasak dari data sensor, misalnya memprediksi tren cuaca dan suhu berdasarkan data historis cuaca dan suhu. Berbagai metode matematis dan algoritmik telah diusulkan untuk memproses data stream. *Sampling* dan proyeksi digunakan untuk menangani laju data yang tinggi pada aliran data. *Sampling* mengacu pada proses pemilihan item data berdasarkan ukuran statistik, digunakan untuk memilih *subset* dari aliran data. Di sisi lain, proyeksi digunakan untuk mengurangi dimensi data dengan menggunakan teknik *sketching*. Selain itu, teknik-teknik seperti pengujian *group testing*, *tree method*, *robust approximation*, dan *exponential histograms* telah diterapkan sebagai solusi algoritmik guna mengurangi kompleksitas ruang dan waktu. (Mohammed & Soliman, 2010).

### 2.2.2 Deep Learning untuk Deteksi Objek

*Machine Learning* merupakan bagian dari *Artificial Intelligence* (AI) yang melibatkan proses di mana mesin belajar dari data tanpa perlu diprogram secara eksplisit. Tujuannya adalah melatih mesin menggunakan data dan algoritma yang diberikan, memungkinkan mesin untuk belajar mengambil keputusan dari data yang diproses. Keunikan *Machine Learning* terletak pada dinamisnya yang memungkinkannya untuk menyesuaikan diri dengan data baru, dengan tujuan meminimalkan kesalahan dan meningkatkan akurasi prediksi (Jakhar & Kaur, 2020).

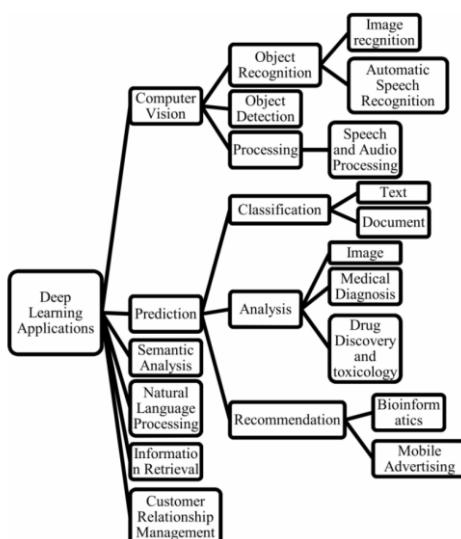
*Deep Learning* merupakan bagian dari *Machine Learning* yang menggunakan model komputasi dan algoritma yang meniru arsitektur jaringan saraf biologis di otak (*Artificial Neural Networks /ANNs*). Konsep utama *Deep Learning* terinspirasi dari cara otak manusia memproses informasi, membandingkannya dengan informasi yang sudah diketahui, dan mengkategorikannya (Pesapane, Codari, & Sardanelli, 2018).



Gambar 2.1 *Deep Learning* merupakan Bagian dari *Machine Learning*, yang merupakan Bagian dari *Artificial Intelligence* (Jakhari & Kaur, 2020)

Istilah 'Deep' mengacu pada jumlah lapisan dalam *Artificial Neural Networks* (ANN). Ada tiga jenis lapisan, yaitu lapisan masukan (menerima data masukan), lapisan keluaran (menghasilkan hasil pengolahan data) dan lapisan tersembunyi (mengekstraksi pola-pola di dalam data. (Pesapane, Codari, & Sardanelli, 2018). Semakin banyaknya penelitian tentang *deep learning* dan *distributed learning* dikarenakan perkembangan data yang tidak bisa diprediksi, perkembangan teknologi *hardware* yang semakin maju dan murah seperti *High Performance Computing* (HPC), dan peningkatan kemampuan *processing* seperti GPU (Alzubaidi et al., 2021) (Mishra & Gupta, 2017).

Penggunaan *deep learning* menyebar luas dalam berbagai domain aplikasi seperti *computer vision*, *prediction*, *semantic analysis*, *natural language processing*, *information retrieval*, dan *customer relationship management*. Pada bidang *computer vision*, terdapat sub-domain seperti pengenalan objek, deteksi objek, dan pemrosesan yang melibatkan tugas-tugas seperti pengenalan ucapan otomatis, pengenalan gambar, pemrosesan ucapan dan audio, serta pemrosesan seni visual. Pada bidang *prediction* terdapat klasifikasi, analisis, dan rekomendasi. Selain itu, teknik deep learning kini juga diterapkan dalam *semantic analysis*, *natural language processing*, *information retrieval* yang sebelumnya telah dieksplorasi menggunakan *machine learning*. *Deep learning* juga memberikan manfaat dalam *customer relationship management* dengan menganalisis data pelanggan untuk meningkatkan hubungan bisnis (Shinde & Shah, 2018).



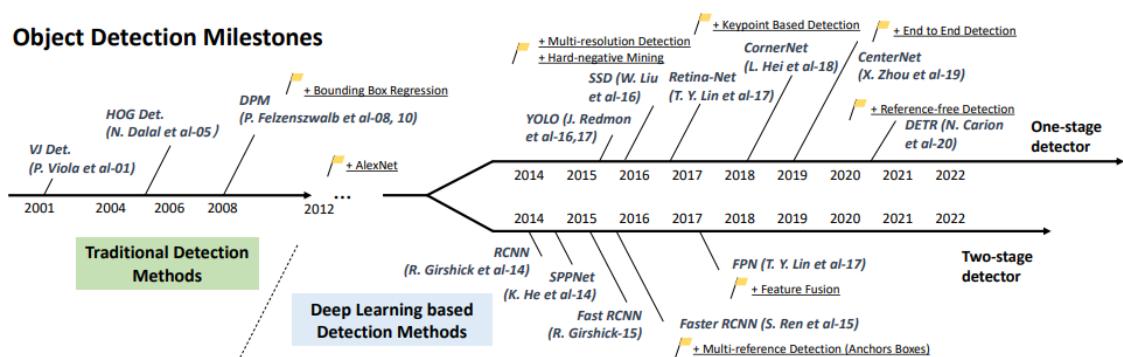
Gambar 2.2 Domain Aplikasi *Deep Learning* (Shinde & Shah, 2018)

Dalam subdomain deteksi objek menggunakan *deep learning* terdapat dua tipe, yaitu *region proposal-based two-staged method* dan *regression-based method (one-staged method)*. Meskipun metode *two-staged method* menawarkan tingkat akurasi yang lebih tinggi dibandingkan dengan *one-staged method*, namun *one-staged method* memiliki kecepatan deteksi yang lebih cepat. *One-staged method* lebih cocok untuk penerapan dalam situasi di mana deteksi objek secara *real-time* (Jiang, Zhao, Li, & Jia, 2020).



Gambar 2.3 Deteksi Objek (Jiang, Zhao, Li, & Jia, 2020)

Deteksi objek menyangkut klasifikasi dan juga lokalisasi pada gambar dengan beberapa objek terdapat pada satu gambar. Kebanyakan deteksi objek terdiri dari 2 langkah, langkah pertama menentukan lokasi objek pada gambar dan langkah selanjutnya mengklasifikasikannya. Sebagian besar algoritma tradisional untuk deteksi objek seperti Viola-Jones, *Histogram of Oriented Gradients* (HOG), dan *Deformable Parts Model* (DPM) bergantung pada ekstraksi fitur secara manual seperti *edges*, *corner*, gradien dari gambar, serta algoritma klasik *machine learning*. Perubahan besar terjadi setelah arsitektur *Convolutional Neural Network* (CNN) berhasil meningkatkan hasil dalam klasifikasi gambar, khususnya setelah prestasi yang dicapai dalam ImageNet (Sharma, 2022).



Gambar 2.4 *Object Detection Milestone* (Zou, Chen, Shi, Guo, & Ye, 2019)

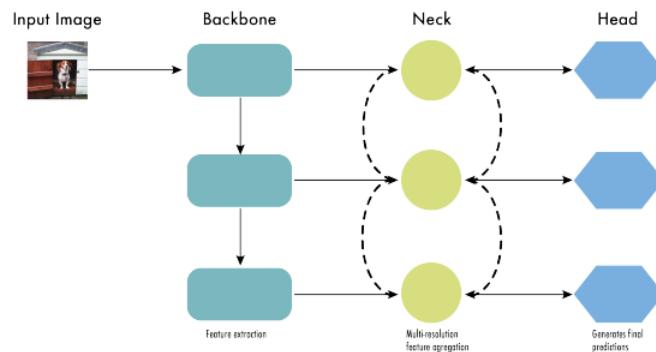
*Convolutional Neural Network* (CNN) adalah salah satu jaringan *deep learning* yang paling populer digunakan (Alzubaidi, 2021). Keunggulan utama dalam CNN dibandingkan dengan pendahulunya adalah kemampuannya untuk secara otomatis mengidentifikasi fitur – fitur yang

tidak relevan tanpa definisi dari manusia. Berbeda dengan *fully connected networks* konvensional, CNN membuat shared weights dan koneksi local untuk menggunakan data input dengan struktur dua dimensi seperti sinyal gambar. Banyak deteksi objek berbasis CNN telah diusulkan, dan struktur metode berbasis CNN memberikan kemajuan yang pesat terhadap akurasi deteksi objek. Metode yang diusulkan cenderung mengadopsi struktur yang kompleks untuk meningkatkan kemampuan representasi dari jaringan seperti contoh R-CNN, Fast-RCNN, dan Faster-RCNN (Li, Zhang, Xiang, Li, & Zou, 2021).

### 2.2.3 Algoritma YOLO

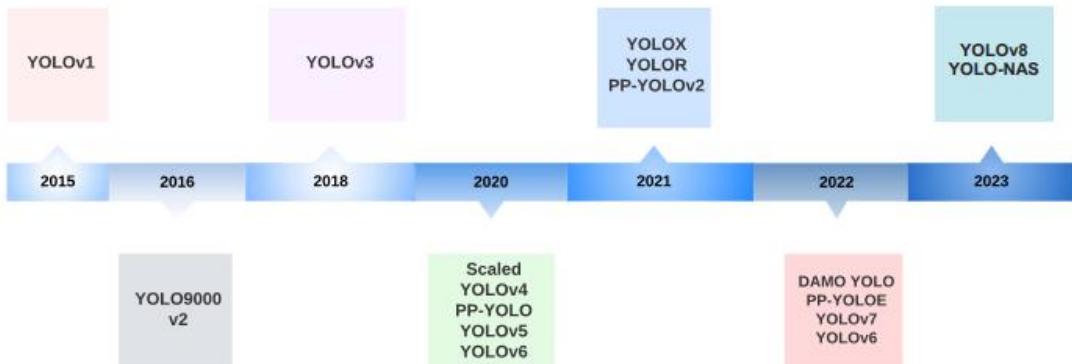
You Only Look Once (YOLO), sebuah metode baru dalam deep learning yang digagas oleh Redmon, Divvala, Girshick, & Farhadi pada tahun 2015, yang bertujuan untuk mendeteksi objek pada citra. YOLO menggunakan metode *one-staged detector* yang didasarkan pada CNN. YOLO merupakan pendekatan baru untuk mendeteksi objek. Sebelumnya deteksi objek menggunakan kembali pengklasifikasi untuk dapat mendeteksi, namun pada YOLO menggunakan satu jaringan saraf untuk secara langsung memprediksi kotak pembatas dan probabilitas kelas dari gambar masukan dalam satu evaluasi (Redmon, Divvala, Girshick, & Farhadi, 2016) (Fang, Wang, & Ren, 2020). Cara kerja YOLO adalah dengan menggunakan satu *convolutional networks* yang secara bersamaan memprediksi beberapa kotak pembatas (*bounding box*) dan probabilitas kelas yang terkandung di dalam kotak pembatas tersebut. Ukuran dari gambar input dipotong ke beberapa potong bagian, dan dipisahkan kedalam bentuk *unit grid* berukuran SxS. Setiap sel *grid* memprediksi B *bounding box*, dan posisi informasi, *confidence* dan kategori dari *bounding box* didapatkan dengan regresi di lapisan output (Liu, Fang Xi, Yin, & Yang, 2022).

Dalam konteks jaringan neural untuk deteksi objek, konsep *backbone*, *neck*, dan *head* merujuk pada bagian-bagian penting dari arsitektur detector. *backbone*, biasanya berupa *convolutional neural network* (CNN) yang berperan dalam mengekstraksi fitur-fitur penting dari gambar pada berbagai skala yang berbeda. Kemudian, *neck* mengolah kembali fitur-fitur ini dengan tujuan untuk menyempurnakan dan meningkatkan *spatial and semantic information* yang terkandung di dalamnya. Terakhir, *head* menggunakan fitur-fitur yang telah disempurnakan ini untuk membuat prediksi dalam tugas deteksi objek, memanfaatkannya sebagai dasar untuk membuat penilaian yang akurat terkait keberadaan dan karakteristik objek dalam gambar yang diberikan. Keseluruhan arsitektur ini bekerja bersama untuk menciptakan sistem deteksi objek yang efisien dan akurat dalam mengidentifikasi objek dalam gambar (Terven & Cordova-Esparza, 2023).



Gambar 2.5 Arsitektur *Backbone*, *Neck*, dan *Head* (Terven & Cordova-Esparza, 2023)

YOLO terkenal sebagai salah satu metode pendekripsi objek tercepat dengan kinerja *real-time* yang unggul dan tingkat akurasi yang tinggi, yang terus ditingkatkan sejak pertama kali diusulkannya. Sampai penelitian ini dilaksanakan telah terdapat 16 versi YOLO yang sudah dikembangkan (Terven & Cordova-Esparza, 2023).



Gambar 2.6 A timeline of YOLO Version (Terven & Cordova-Esparza, 2023)

Dalam pengujian pemodelan, beberapa versi dari YOLO akan diimplementasikan dan dibandingkan untuk mengevaluasi kinerjanya, yaitu YOLOv4, YOLOv5, dan YOLOv8. Keputusan untuk memilih model YOLO tersebut didasarkan pada banyaknya penelitian sebelumnya yang telah menggunakan versi tersebut. Hal ini bertujuan untuk memudahkan penggunaan model tersebut berdasarkan pengalaman yang sudah terdokumentasi dalam berbagai penelitian sebelumnya. Tabel berikut merupakan perbandingan beberapa versi YOLO (Jocher, 2020) (Terven & Cordova-Esparza, 2023).

Tabel 2.2 Perbandingan Average Precision Beberapa Versi Model YOLO (Terven & Cordova-Esparza, 2023)

Versi	Tanggal	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	63.4
YOLOv2	2016	Yes	Darknet	Darknet24	63.4
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

## 2.2.4 Resource-Aware Framework

Sistem perangkat lunak kini telah membutuhkan lebih banyak sumber daya, sering melebihi kemampuan *hardware* dan menyebabkan masalah *resource scarcity*. Untuk menghindari masalah ini, sistem perangkat lunak harus menjadi *resource-aware* dan sesuaikan kebutuhan komputasinya dengan kondisi sumber daya perangkat keras (Shiddiqi, Yogatama, & Navastara, 2023). Oleh karena itu, diperlukan suatu *framework* yang dapat memantau dan mengelola sumber daya yang diperlukan. Dalam penelitian yang dilakukan oleh Moreau & Queinnec, mereka mengembangkan sebuah kerangka kerja yang dikenal sebagai *Resource-aware Programming*. *Resource-aware Programming* merupakan suatu *framework* yang memungkinkan pengguna untuk memantau penggunaan sumber daya yang digunakan oleh sebuah program, serta membuat aturan yang mengatur manajemen sumber daya tersebut. Pemberitahuan asinkronus digunakan untuk memberi peringatan terkait beban sumber daya yang berlebihan, dan untuk menghentikan komputasi jika diperlukan, hal ini dapat dikendalikan oleh kode yang dibuat oleh pengguna. Metode ini juga memungkinkan pengguna untuk menjalankan kontrol atas struktur hirarki program yang sedang berjalan (Moreau & Queinnec, 2005).

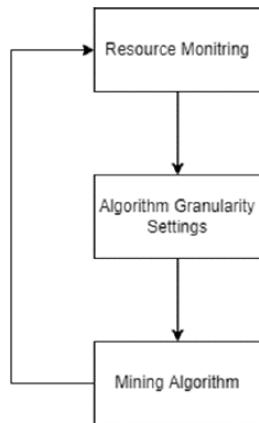
Kerangka kerja *Resource-aware* memanfaatkan algoritma berbasis *granularity settings* yang dapat mengubah konfigurasi dari input, output, dan proses akhir dari suatu pemrosesan. Dalam kerangka kerja *Algorithm Granularity Settings* yang diusulkan oleh Gaber dan Yu, sumber daya komputasi yang dipengaruhi oleh *data stream mining* mencakup *memory*, *processing cycle*, *bandwidth*, dan daya baterai. Berdasarkan kerangka kerja *Algorithm Granularity Settings* tersebut, pengukuran ketersediaan sumber daya dapat dilakukan secara berkala dan pengaturan algoritma dapat disesuaikan. Komponen utama dalam proses pengukuran ketersediaan sumber daya ini adalah pola konsumsi sumber daya dan pengaturan algoritma. Pola konsumsi sumber daya menggambarkan perubahan pola penggunaan sumber daya pada setiap periode dalam suatu rentang waktu tertentu. Sementara itu, pengaturan algoritma mengacu pada konfigurasi input, output, dan proses yang dapat diubah sewaktu-waktu untuk mengatasi masalah ketersediaan sumber daya dan kecepatan *data stream*. Terdapat tiga jenis *granularity* yang diatur pada algoritma, antara(Gaber & Yu, 2006):

- *Algorithm Input Granularity* (AIG) merupakan teknik yang mengubah laju aliran data yang memasuki algoritma *stream mining*. Teknik ini mencakup penggunaan metode *sampling*, *load shedding*, dan *synopsis of data*.
- *Algorithm Output Granularity* (AOG) adalah proses mengubah ukuran output dari algoritma untuk mempertahankan keterbatasan memori. Besarnya output dapat diubah dengan menggunakan tingkat *granular output*, di mana semakin rinci outputnya, semakin rendah nilai *granularity*-nya, dan sebaliknya.
- *Algorithm Processing Granularity* (APG) adalah proses mengubah parameter dari algoritma untuk menjaga konsumsi daya atau baterai.

Dalam algoritma tersebut, terdapat interaksi yang memengaruhi sumber daya yang disajikan dalam Tabel 2.3. Perubahan tersebut juga akan memengaruhi perubahan kondisi dari sumber daya lainnya. Gambaran umum dari arsitektur yang diusulkan dapat dilihat pada Gambar 2.7.

Tabel 2.3 Perubahan Kondisi yang Dipengaruhi Algoritma (Gaber & Yu, 2006)

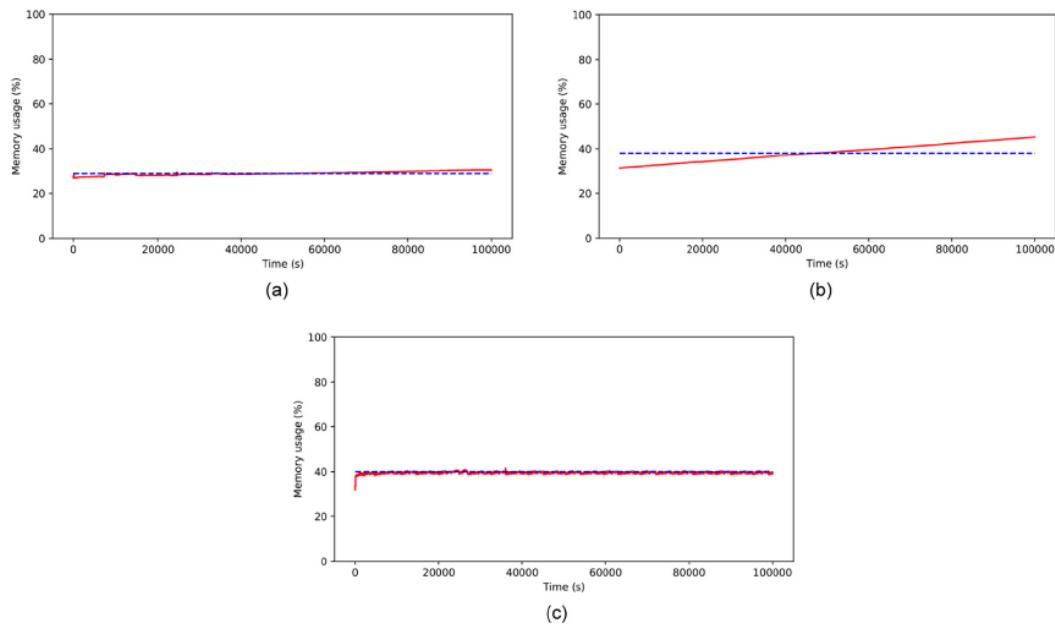
Teknik	Sumber Daya yang Dipengaruhi
<i>Algorithm Input Granularity</i> (AIG)	Kecepatan aliran data dan baterai
<i>Algorithm Output Granularity</i> (AOG)	Memory
<i>Algorithm Processing Granularity</i> (APG)	CPU



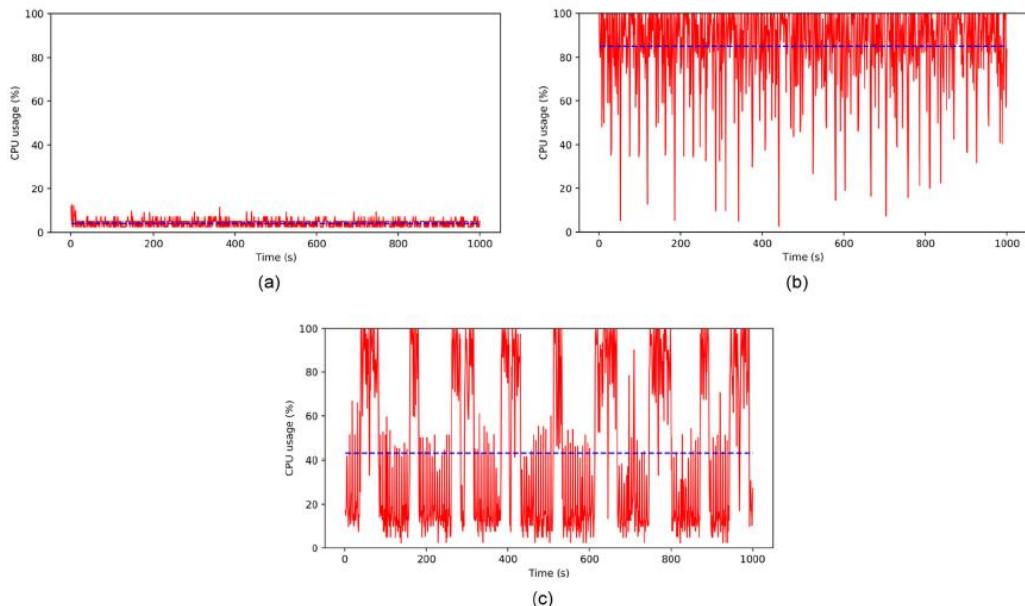
Gambar 2.7 *Algorithm Granularity Settings General Process* (Gaber & Yu, 2006)

*Resource Monitoring* adalah komponen yang mengamati semua kebutuhan secara statistik terhadap konsumsi sumber daya dalam periode waktu. *Algorithm Granularity Settings* adalah perubahan parameter antara AIG, AOG atau APG sesuai dengan output dari *resource monitoring*. Setelah itu *mining algorithm* merupakan *mining* yang beradaptasi dari perubahan *algorithm granularity settings* (Gaber & Yu, 2006). Dalam penelitian yang dilakukan oleh Shiddiqi, Yogatama, dan Navastara, menggunakan adaptasi konsep AGS untuk *Resource-aware Framework* melalui AOG dan APG dilakukan dengan mengasumsikan bahwa perangkat menggunakan daya dari listrik gedung, bukan dari baterai. AOG bertanggung jawab dalam menyesuaikan penggunaan memori atau RAM perangkat untuk mengatur kecepatan pengambilan data. Selain memori, *disk storage* juga menjadi sumber daya yang memengaruhi kecepatan pengambilan data. Di sisi lain, APG mengatur adaptasi CPU pada sistem untuk mempengaruhi proses pengolahan data yang telah diperoleh sehingga dapat menghasilkan keluaran yang diinginkan.

Perbandingan antara penerapan *Resource-aware Framework* dan pendekatan yang tidak menerapkan hal tersebut dapat ditemukan dalam gambar 2.8 untuk CPU, 2.9 untuk memori, dan 2.10 untuk penyimpanan disk. (Shiddiqi, Yogatama, & Navastara, 2023). Pada gambar 2.8c penggunaan RAM menggunakan *Resource-aware Framework*, Ketika penggunaan RAM melebihi nilai *threshold* yang ditentukan, sistem akan mengadaptasi sehingga kecepatan data yang disimpan akan semakin menurun hingga penggunaan RAM turun di bawah nilai *threshold* yang ditentukan. Setelah penggunaan di bawah nilai *threshold*, maka kecepatan data akan berjalan seperti pada awalnya.

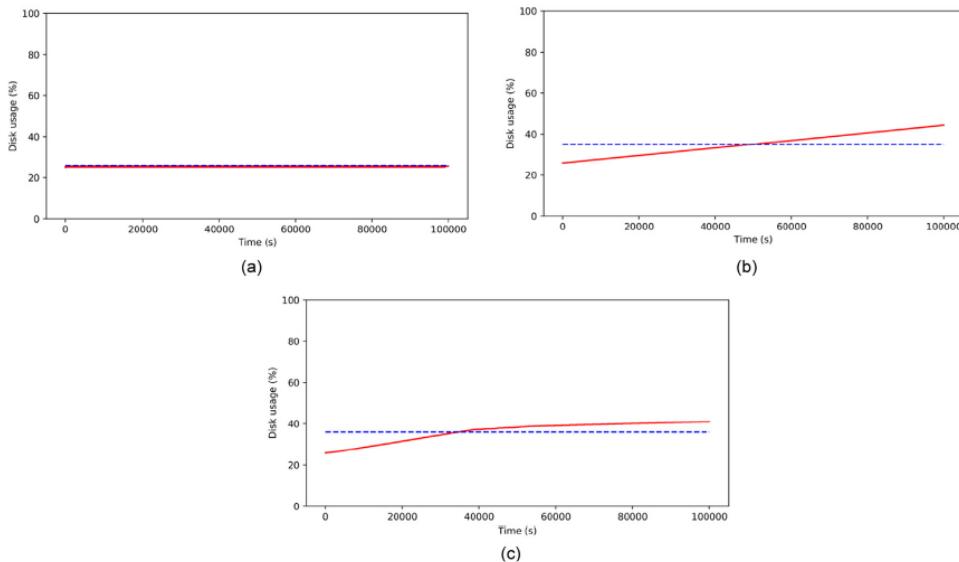


Gambar 2.8 Penggunaan RAM pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023)



Gambar 2.9 Penggunaan CPU pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023)

Pada gambar 2.9c penggunaan CPU menggunakan *Resource-aware Framework*, ketika penggunaan CPU melebihi nilai *threshold* yang ditentukan, sistem akan menghitung batasan ketersediaan CPU untuk menyesuaikan diri. Dampak dari penyesuaian ini adalah penurunan penggunaan CPU. Selama sistem beroperasi dan terjadi peningkatan jumlah data yang masuk, penggunaan CPU akan tetap berada dalam batasan yang telah ditetapkan.

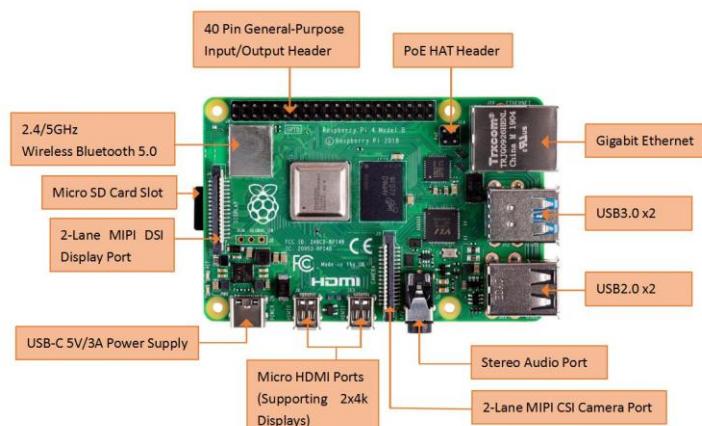


Gambar 2.10 Penggunaan Disk Storage pada (a) Aliran Data Numerik, (b) Aliran Video tanpa Kerangka Kerja Ravis, dan (c) Aliran Video dengan Kerangka Kerja Ravis (Shiddiqi, Yogatama, & Navastara, 2023)

Pada gambar 2.10c penggunaan *Disk Storage* menggunakan *Resource-aware Framework* ketika penggunaan penyimpanan telah mencapai batas *threshold* yang telah ditetapkan, kecepatan pengolahan data akan menurun, menyebabkan *disk storage* menjadi lebih lambat karena mendekati kapasitas penuh. Setelah mencapai kapasitas penuh, proses pembersihan akan dilakukan terhadap data lama yang telah diproses oleh sistem.

## 2.2.5 Raspberry Pi

Raspberry Pi adalah serangkaian komputer *single-board* dengan biaya murah, ukuran kecil, dan portabel. Diluncurkan oleh Raspberry Pi Foundation pada tahun 2012, perangkat ini memiliki ukuran kecil namun mampu menjalankan berbagai aplikasi dari yang sederhana hingga kompleks seperti IoT dan *media center*. Para programmer bisa mengembangkan *script* atau program menggunakan bahasa pemrograman Python yang merupakan bahasa utama dalam sistem operasi Raspbian (Wai, Jegatheesan, & Loon, 2015).



Gambar 2.11 Raspberry Pi 4 Model B

Dipersenjatai dengan prosesor Broadcom BCM2711 quad-core Cortex-A72 64-bit SoC berkecepatan clock hingga 1.5GHz, Raspberry Pi 4 hadir dalam varian RAM 2GB, 4GB, dan 8GB LPDDR4-3200 SDRAM. *Board* ini memiliki kemampuan grafis yang kuat dengan dukungan dari prosesor grafis VideoCore VI yang mampu menampilkan resolusi hingga 4K pada 60fps melalui dua port micro-HDMI. Konektivitasnya terdiri dari dua port USB 3.0, dua port USB 2.0, port Gigabit Ethernet, dan slot kartu microSD. Untuk koneksi nirkabel, Raspberry Pi 4 dilengkapi dengan WiFi 802.11ac dual-band (2.4GHz dan 5GHz) dan Bluetooth 5.0. Selain itu, terdapat 40 pin header GPIO yang memungkinkan pengguna untuk menghubungkan berbagai sensor dan perangkat eksternal. Dengan kemampuan menjalankan berbagai sistem operasi seperti Raspberry Pi OS (sebelumnya dikenal sebagai Raspbian), Ubuntu, dan distribusi Linux lainnya, Raspberry Pi 4 menawarkan peningkatan signifikan dalam kinerja, cocok untuk berbagai proyek komputasi, multimedia, dan pengembangan aplikasi (Raspberry Pi Foundation, n.d.).

## 2.2.6 IP Camera ColorVu

Teknologi *IP Camera* ColorVu dari Hikvision merupakan solusi yang memungkinkan pengguna untuk mendapatkan gambar berkualitas tinggi dan warna yang jelas, bahkan dalam kondisi gelap yang biasanya sulit dijangkau oleh kamera konvensional. Berbeda dengan kebanyakan *IP camera* lainnya, ColorVu menggunakan pendekatan yang berbeda dengan algoritma *noise* yang lebih agresif, infra merah, dan penyesuaian rentang dinamis. Teknologi ini mengatasi masalah umum terkait kualitas gambar yang buruk pada kondisi cahaya rendah. Perbedaan antara *IP camera* konvensional dengan teknologi ColorVu milik Hikvision tergambar jelas dalam Gambar 2.12, yang menunjukkan kemampuan unggul dalam menangkap gambar dengan detail dan warna yang lebih baik dalam kondisi pencahayaan yang minim.



Gambar 2.12 Perbedaan Kamera Konvensional dengan Kamera ColorVu (James, 2020)

Teknologi ColorVu memiliki dua komponen utama. Pertama, lensa ColorVu dirancang untuk menangkap lebih banyak cahaya dengan tingkat akurasi yang tinggi. Lensa ini memiliki aperture yang lebih lebar, memungkinkan kamera untuk menangkap lebih banyak cahaya bahkan dalam kondisi pencahayaan minim. Hal ini menyebabkan sensor ColorVu dapat menerima empat kali lipat cahaya dibandingkan dengan kondisi gelap pada kamera konvensional. Perbedaan yang signifikan terletak pada nilai aperture, di mana *kebanyakan IP camera* biasanya menggunakan aperture F2.0, sementara ColorVu menggunakan F1.0. Gambar 2.12 mengilustrasikan perbedaan nilai aperture yang menjadi faktor kunci dalam kemampuan ColorVu dalam menangkap cahaya lebih banyak dan meningkatkan kualitas gambar dalam kondisi pencahayaan yang rendah.



Gambar 2.13 Aperture Kamera Konvensional dan Kamera Colorvu (James, 2020)

Sensor CMOS yang digunakan juga merupakan versi yang lebih lanjut, mampu mengonversi cahaya menjadi sinyal digital dengan lebih efisien dan akurat jika dibandingkan dengan sensor yang digunakan pada kebanyakan *IP camera* konvensional. Ketika kondisi pencahayaan sangat minim atau hampir tidak ada cahaya alami yang masuk ke kamera, teknologi ColorVu dapat menghasilkan cahaya lembut yang dipancarkan di sekitar sudut pandang kamera. Hal ini membantu dalam meningkatkan kemampuan pengambilan gambar dan memungkinkan kamera untuk tetap menghasilkan gambar yang jelas dan berwarna dalam kondisi pencahayaan yang sangat rendah (James, 2020).

*(Halaman ini sengaja dikosongkan)*

## BAB III METODOLOGI

### 3.1 Metode yang Dirancang

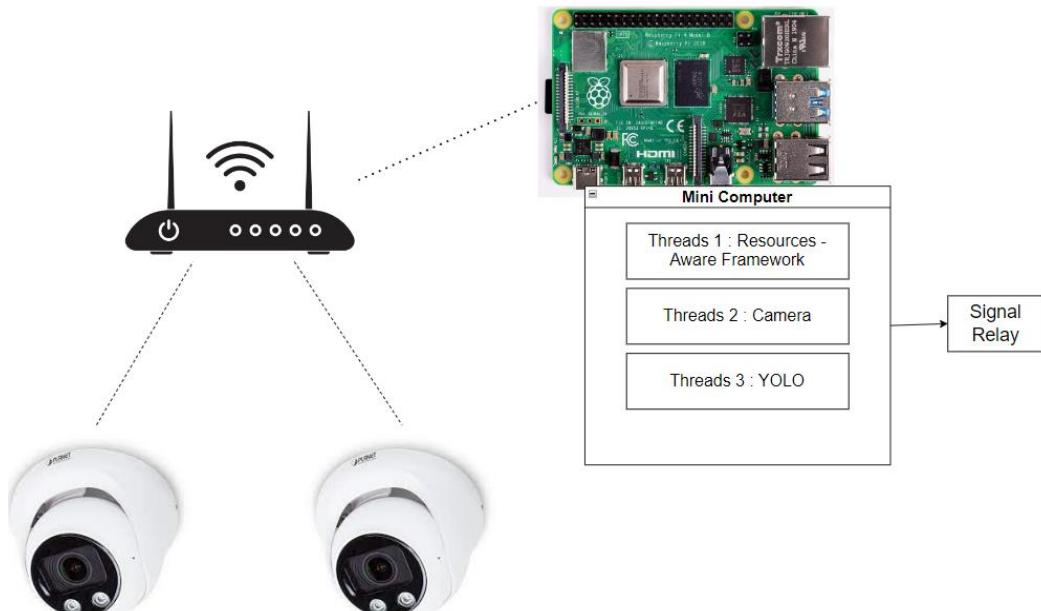
Bab ini akan menguraikan metode yang digunakan pada tugas akhir yang terdiri dari tiga tahapan, yaitu studi literatur, implementasi sistem, serta pengujian dan evaluasi hasil.

#### 3.1.1 Studi Literatur

Dalam melakukan studi literatur, beberapa referensi akan dipelajari sebagai panduan dalam pengerjaan tugas akhir. Hal tersebut meliputi multi-kamera, *Resource-aware Framework (Granularity Settings Algorithm)*, algoritma YOLO, dan deteksi objek.

#### 3.1.2 Implementasi Sistem

Tahap implementasi sistem ini melibatkan studi kasus di dalam sebuah ruangan untuk mendeteksi keberadaan orang. Sistem ini akan bekerja secara terus menerus untuk memantau kondisi ruangan tersebut. Arsitektur sistem digambarkan pada gambar 3.1



Gambar 3.1 Arsitektur Sistem

Dalam arsitektur sistem tersebut terdiri dari beberapa perangkat, seperti *mini computer*, dua kamera IP, dan *wifi access point*. *Mini computer* yang digunakan berupa Raspberry Pi 4 Model B yang memiliki spesifikasi, yaitu prosesor Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz, RAM 4 GB LPDDR4-3200, Disk Storage 32 GB serta dukungan konektivitas nirkabel IEEE 802.11ac pada frekuensi 2.4 GHz dan 5.0 GHz. Raspberry Pi dipilih karena memiliki sumber daya yang terbatas, sehingga perlu dilakukan penyesuaian jika sumber daya sudah terpakai penuh. Dua buah kamera IP Hikvision ColorVu 4MP 2.8mm yang terhubung dengan *wifi access point* sekaligus terhubung juga dengan *mini computer* dalam satu jaringan. *Mini computer* dan kamera IP terhubung ke *wifi access point* melalui jaringan LAN, sehingga memungkinkan *mini computer* untuk mengakses *IP Camera* melalui protokol *Real Time Streaming Protocol* (RTSP) yang mendukung.

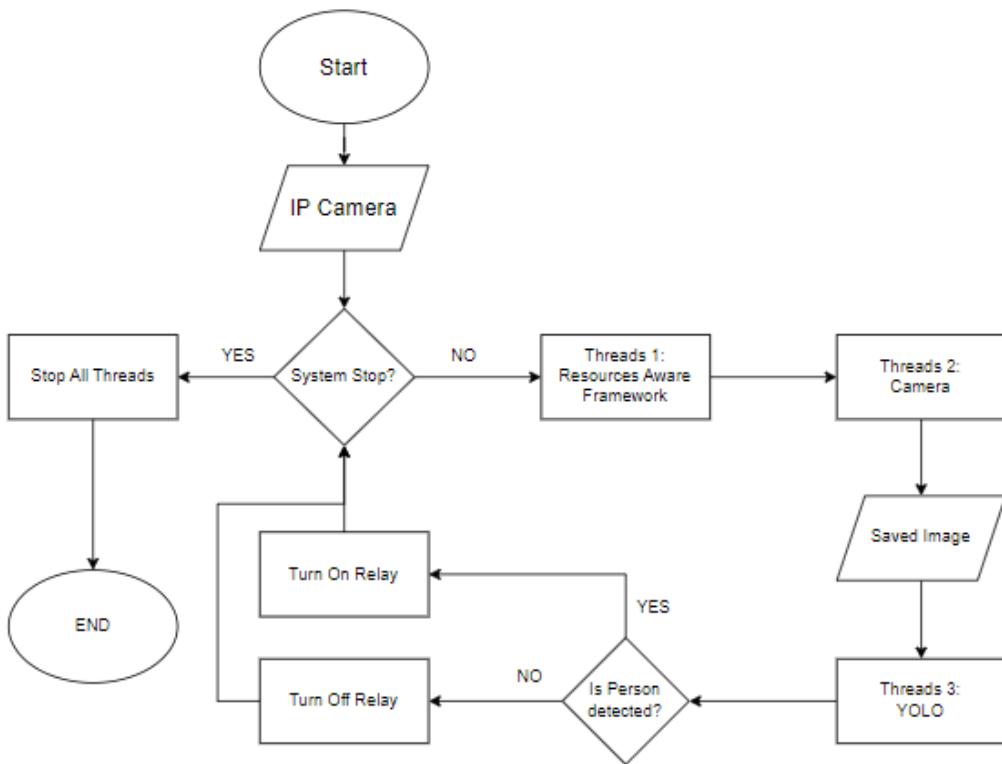
Sistem ini dikembangkan dengan menggunakan Python3 sebagai bahasa pemrograman utamanya, hal ini dikarenakan Python3 memiliki kemudahan dalam mengakses sumber daya yang tersedia pada perangkat dan kemudahan dalam penggunaan *thread* pada *mini computer*. Python juga menjadi pilihan yang tepat jika berinteraksi dengan perangkat *Internet of Things* (IoT). Selain itu, Python memudahkan proses *deep learning* yang digunakan untuk mendeteksi keberadaan orang dalam suatu ruangan. Model *deep learning* yang digunakan adalah model YOLO (You Only Look Once) dengan beberapa versi, yaitu YOLOv4, YOLOv4-tiny, YOLOv5s (*small*), YOLOv5m (*medium*), YOLOv8s (*small*), dan YOLOv8m (*medium*).

Proses *training* model YOLO dilakukan di *personal computer* dengan *operating system* berbasis linux yaitu Ubuntu 22.04 LTS. Spesifikasi dari personal computer tersebut antara lain, Processor Intel Core i5-12500H 12C (4P + 8E) / 16T, P-core 2.5 / 4.5GHz, E-core 1.8 / 3.3GHz, RAM 16 GB DDR4-3200, GPU NVIDIA GeForce RTX 3050 4GB GDDR6. Dengan menggunakan dataset COCO yang memiliki label “*person*” sebanyak 64.115 data, dengan pembagian sebanyak 51.292 data untuk *training*, 4.489 data untuk *validation*, dan 8.334 data untuk *test*. Hasil training beberapa model YOLO sebagai berikut

Tabel 3.1 Hasil Pelatihan Beberapa Model YOLO

No	Model	Lama Proses	mAP(0.5)	mAP(0.5:0.95)	Ukuran Model
1	YOLOv4-Tiny	2.25 jam	62,9%	30,7%	23.5 MB
2	YOLOv4	6.97 jam	75,7%	42,9%	256 MB
3	YOLOv5s	2.87 jam	79,1%	52,5%	14.8 MB
4	YOLOv5m	6.92 jam	81,7%	57,1%	42.8 MB
5	YOLOv8s	3.16 jam	75,7%	53,2%	22.5 MB
6	YOLOv8m	7.12 jam	80,8%	58,9%	52 MB

Berdasarkan hasil *training*, model dengan akurasi mAP(0.5) tertinggi adalah YOLOv5m dengan nilai 81,7%. Namun, untuk akurasi mAP(0.5:0.95), model YOLOv8m memiliki nilai tertinggi sebesar 58,9%. Di sisi lain, YOLOv4 memiliki ukuran model yang cukup besar, yaitu 256MB, yang empat kali lebih besar dibandingkan YOLOv8m, model dengan ukuran terbesar kedua. Pada *Mini Computer* akan menjalankan tiga *thread* secara bersamaan, dimana *thread* pertama bertugas menjalankan *Resources-Aware Framework*, *thread* kedua bertanggung jawab untuk menangkap gambar secara *real-time* dari kamera, dan *thread* ketiga untuk mengeksekusi proses algoritma YOLO. *Flowchart* keseluruhan sistem dapat dilihat pada gambar 3.2.



Gambar 3.2 Flowchart Keseluruhan Sistem

### 1. Thread 1: Resource-aware Framework

Dalam *Resource-aware Framework* menggunakan pendekatan *Algorithm Granularity Settings* yang memiliki tiga proses yang berbeda yaitu memantau ketersediaan CPU, RAM dan *disk storage* yang dijalankan pada *thread* yang terpisah. Setiap proses ini bertanggung jawab untuk memantau dan memberikan masukan kepada proses lainnya. Pada proses pemantauan RAM, jika penggunaan RAM mencapai 100%, maka proses akan dihentikan sementara untuk menjalankan *commad* untuk mengalokasikan RAM yaitu menghapus *cache* program. Jika penggunaan RAM kurang dari nilai *threshold* (nilai *threshold* ditetapkan tergantung dari model YOLO), nilai penghitung perubahan akan diatur kembali ke satu. Jika penggunaan RAM berada di antara 45% dan 100%, maka dilakukan perhitungan untuk memberikan waktu dimana sistem harus mengambil gambar, dengan persamaan perhitungan sebagai berikut:

$$imageCaptureTime = 1 + (1 - thresholdRAM) \times counterUpthRAM \quad (1)$$

*imageCaptureTime* adalah interval waktu yang digunakan untuk memperlambat pengambilan data gambar yang kemudian dikomunikasikan ke camera *thread*. Nilai *thresholdRAM* merupakan konstanta batas yang menunjukkan nilai penggunaan RAM yang memerlukan adaptasi. Sementara itu, *counterUpthRAM* adalah penghitung yang mencatat jumlah penyesuaian yang dilakukan sebelum penggunaan RAM berada di bawah *thresholdRAM*. Pada proses memantau *disk storage* kurang lebih memiliki perhitungan yang sama dengan dengan pemantauan RAM. Ketika *disk storage* mencapai kapasitas 100%, sistem akan menghapus data lama. Namun, jika kapasitas *disk storage* berada di bawah nilai *threshold* yang telah ditentukan, yakni sekitar 50%, maka data akan tetap disimpan, dan penghitung perubahan akan dikembalikan ke satu. Sementara itu, jika kapasitas *disk storage* yang

digunakan dalam kisaran nilai *threshold* dan kurang dari 100%, maka akan dilakukan perhitungan serupa seperti dalam persamaan satu. Jika salah satu pemantauan sumber daya, seperti RAM atau *disk storage*, mencapai nilai *threshold* dan melakukan adaptasi, maka pemantauan sumber daya lainnya akan terdampak. Sebagai contoh, ketika penggunaan *disk storage* telah melebihi nilai *threshold*, waktu pengambilan gambar akan diatur sesuai dengan persamaan satu, yang juga akan mempengaruhi penggunaan RAM.

Pemantauan penggunaan CPU berkaitan erat dengan waktu yang dibutuhkan untuk mendeteksi objek menggunakan model YOLO. Semakin besar model YOLO yang digunakan, semakin besar pula beban penggunaan CPU. Apabila penggunaan CPU mencapai 100%, maka proses deteksi gambar akan dihentikan sementara selama 10 detik. Jika penggunaan CPU berada di bawah nilai *threshold*, yaitu 80%, program akan tetap berjalan normal dan variabel penghitung perubahan CPU akan diatur kembali menjadi satu. Namun, jika penggunaan CPU berada dalam kisaran antara *threshold* (80%) dan 100%, maka perhitungan akan dilakukan untuk menentukan waktu penghentian sementara menggunakan persamaan berikut:

$$\text{processDelayTime} = \text{random}(1,5) \times \text{counterUpthCPU} \quad (2)$$

*processDelayTime* adalah waktu yang digunakan oleh YOLO *thread* untuk menghentikan sementara proses deteksi objek. *counterUpthCPU* merupakan penghitung jumlah penyesuaian yang dilakukan agar CPU dapat beradaptasi dan penggunaan berada di bawah nilai *threshold*. Selanjutnya, nilai *counterUpthCPU* dikalikan dengan angka acak dari 1 hingga 5. Ini memastikan bahwa sistem dapat beradaptasi dengan beban CPU yang tinggi, memungkinkan proses deteksi objek berjalan lebih efisien tanpa menyebabkan penurunan kinerja yang signifikan.

## 2. Thread 2: Camera

*Camera thread* berperan sebagai proses yang menangkap *frame* gambar selama *stream* berlangsung. Proses ini akan menangkap *frame* saat itu dan menyimpannya sebagai data gambar ke *directory* yang telah ditentukan sebelumnya. Waktu penangkapan dan penyimpanan gambar sebagai data akan disesuaikan berdasarkan umpan balik yang diterima *Resource-aware Framework* melalui *main stream*. Oleh karena itu, jarak antara setiap penangkapan gambar tidak selalu satu detik, tetapi akan disesuaikan dengan ketersediaan sumber daya pada saat itu. Meskipun demikian, untuk inisiasi awal, setiap 1 detik, *camera thread* akan menangkap dan menyimpan gambar. Terdapat dua skenario berbeda pada *camera thread*, yaitu penggunaan satu kamera dan dua kamera. Penggunaan satu kamera hanya melibatkan satu *thread*. Sedangkan, penggunaan dua kamera akan diuji dengan dua metode:

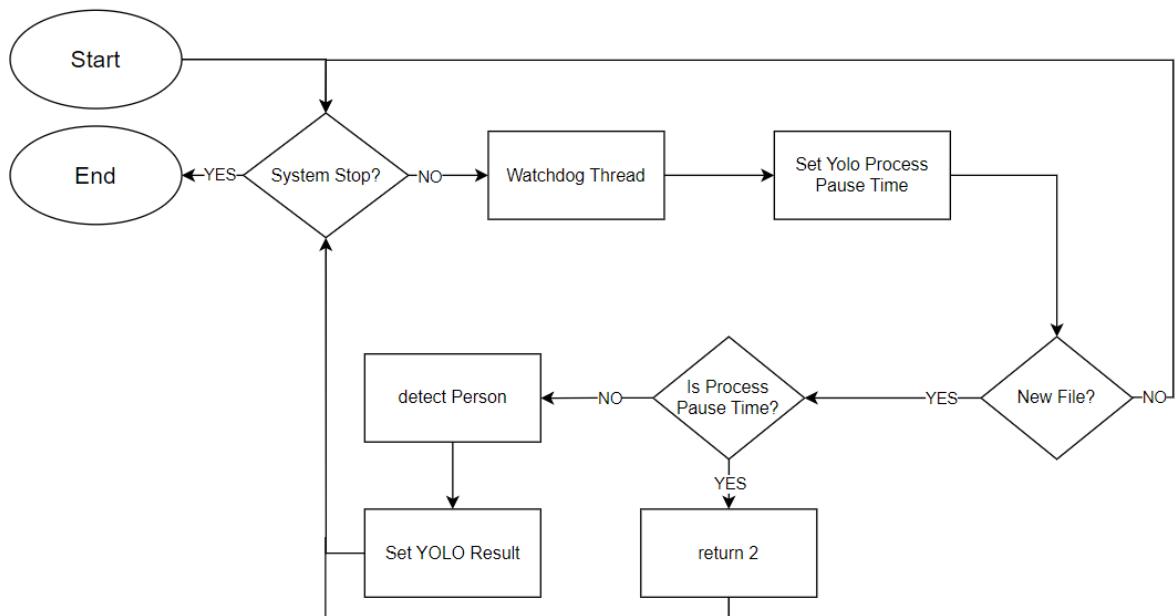
- *Schedule* (Jadwal): Metode jadwal bergantian menggunakan dua kamera yang beroperasi secara bergiliran setiap beberapa menit, dengan keduanya tetap berjalan pada satu *thread* yang sama.
- Paralel: Metode ini menggunakan konsep paralel, di mana dua kamera menangkap gambar secara bersamaan dalam dua *thread* yang terpisah.

Maka pengujian dua kamera ini akan menangkap seluruh gambar dari kedua modul kamera dan menyimpannya di direktori yang berbeda, dengan mempertimbangkan waktu penangkapan dan penyimpanan gambar berdasarkan umpan balik yang diterima *Resource-aware Framework*.

### 3. Thread 3: YOLO

*Thread* berikutnya merupakan bagian dari proses deteksi yang menggunakan beberapa versi dari algoritma YOLO. Pada *thread* ini, menggunakan *package watchdog* yang berjalan pada *thread* terpisah. *Watchdog* adalah suatu *open-source* Python API yang digunakan untuk memantau *event* pada *file system*. Oleh karena itu, *watchdog* digunakan untuk memonitor kemunculan gambar baru yang disimpan oleh *camera thread*. Ketika *watchdog* mendeteksi gambar baru dalam direktori yang telah ditentukan, gambar tersebut akan dimasukkan ke dalam sistem *queue* untuk mendeteksi keberadaan manusia dalam ruangan menggunakan model algoritma YOLO yang telah ditentukan. Hasil dari deteksi ini akan menghasilkan skor, dan skor-skornya akan diambil rata-rata. *Thread* ini akan memproses setiap gambar yang telah disimpan, kecuali jika menerima umpan balik dari *Resource-aware Framework*, dalam hal ini proses dapat dihentikan selama beberapa detik sesuai dengan umpan balik yang diberikan.

Ketika proses dihentikan akibat umpan balik dari *Resource-aware Framework*, maka output yang dihasilkan akan tetap bernilai 2. Namun, jika proses gagal atau tidak dapat mendeteksi keberadaan manusia pada gambar, dan nilai deteksi yang diperoleh berada di bawah *threshold* yang telah ditentukan, maka output yang dihasilkan akan menjadi 0. Jika terdapat manusia dalam gambar, output akan berada dalam kisaran antara *threshold* dan 1.



Gambar 3.3 Flowchart YOLO Thread

Setelah menerima respons dari deteksi yang dilakukan oleh algoritma YOLO hasilnya akan disalurkan ke proses Relay yang dijalankan oleh *thread* tersendiri. Pada proses Relay, terdapat *queue* pendek digunakan untuk menyimpan respons dari proses deteksi. *Queue* ini memiliki kapasitas maksimum sebanyak lima elemen. Ketika *queue* telah mencapai kapasitas maksimum, respons lama akan dikeluarkan (*pop*) dan digantikan dengan hasil respons yang baru diterima (*push*). Jika jumlah elemen bernilai "True" dari *queue* itu mencapai tiga atau lebih, maka proses Relay akan memberikan sinyal untuk menyala. Sebaliknya, jika jumlah elemen bernilai "False" mencapai tiga atau lebih, maka proses Relay akan memberikan sinyal untuk mati. Terdapat dua skenario yang berbeda dalam penggunaan YOLO *thread*, tergantung

pada jumlah kamera yang digunakan dan metodenya. Jika hanya satu kamera yang aktif, maka hanya satu YOLO *thread* yang digunakan. Namun, jika terdapat dua kamera, penggunaan *thread* akan disesuaikan dengan metode yang diterapkan. Dalam metode jadwal bergantian, cukup satu *thread* yang diperlukan. Sedangkan dalam metode paralel, digunakan dua *thread*, sesuai dengan jumlah kamera yang tersedia.

### 3.1.3 Pengujian dan Evaluasi Hasil

Pengujian ini dijalankan dengan berbagai skenario pengujian dengan tujuan untuk memahami berbagai aspek. Berikut skenario pengujian sistem ini:

1. Sistem akan dijalankan menggunakan satu kamera tanpa menerapkan *Resource-aware Framework*, dengan maksud untuk mengetahui konsumsi sumber daya yang terlibat.
2. Sistem akan dijalankan menggunakan satu kamera dengan menerapkan *Resource-aware Framework*, dengan maksud untuk mengetahui seberapa efisien komsumsi sumber daya pada satu kamera.
3. Sistem akan dijalankan menggunakan dua kamera tanpa menerapkan *Resource-aware Framework*, dengan maksud untuk mengetahui konsumsi sumber daya yang terlibat.
4. Sistem akan dijalankan menggunakan dua kamera dengan menerapkan *Resource-aware Framework*, dengan maksud untuk mengetahui seberapa efisien komsumsi sumber daya pada dua kamera.

Setiap skenario dalam penelitian ini dijalankan menggunakan beberapa versi algoritma YOLO, yaitu YOLOv4, YOLOv4-tiny, YOLOv5-small, YOLOv5-medium, YOLOv8-small, dan YOLOv8-medium. Pada pengujian dengan dua kamera, digunakan dua metode: *schedule* dan paralel. Hal ini menghasilkan total 36 pengujian berbeda. Setiap pengujian menghasilkan file CSV yang berisi foto yang diambil serta data mengenai penggunaan CPU, RAM, dan *disk storage*. Dalam evaluasi hasil pengujian, akan dilakukan perbandingan antara model YOLO untuk mengetahui seberapa besar pengaruh penggunaan sumber daya saat menjalankan sistem. Selain itu, akan dilakukan perbandingan antara skenario untuk mencapai tujuan berikut:

1. Skenario 1 dan 2, serta Skenario 3 dan 4 akan dianalisis untuk mengevaluasi efektivitas *Resource-aware Framework* dalam penggunaan sumber daya.
2. Skenario 1 dan 3, serta Skenario 2 dan 4, akan dianalisis untuk mengevaluasi penggunaan sumber daya ketika sistem dijalankan menggunakan dua kamera dibandingkan dengan satu kamera.

## 3.2 Peralatan pendukung

Dalam penggeraan ini, penulis menggunakan beberapa alat pendukung, yaitu Geany sebagai editor teks untuk menulis kode program, VLC Media Player yang digunakan sebagai sarana untuk memutar video streaming secara *real-time*, RealVNC Viewer yang digunakan sebagai alat untuk mengakses perangkat *Mini Computer* secara remote dari perangkat lain, dan *hardisk* eksternal sebagai media penyimpanan untuk file hasil program yang telah dijalankan.

### 3.3 Implementasi

Dalam implementasi, sistem dibuat menjadi beberapa modul sesuai dengan *thread* yang sudah dijelaskan untuk memudahkan penggunaan sumber kode. Modul-modul ini dibuat berdasarkan kaidah *object-oriented programming*, antara lain *class camera*, *class Resource-aware Framework* (*class AGS*), *class YOLO*, dan *class Relay*. Seluruh objek yang dibuat akan berjalan pada *class main*. Sumber kode seluruh sistem dibuat seperti pada Pseudocode 3.1, dalam satu direktori yang berisikan sumber kode Python, bash script, direktori gambar, dan direktori model yang berisi file weight dan XML untuk YOLOv4 serta file PyTorch untuk YOLOv5 dan YOLOv8.

```
1. image/
2.   - cam0/
3.   - cam1/
4. model/
5.   - YOLOv4-tiny.cfg
6.   - YOLOv4-tiny.weights
7.   - YOLOv4.cfg
8.   - YOLOv4.weights
9.   - YOLOv5s.pt
10.  - YOLOv5m.pt
11.  - YOLOv8s.pt
12.  - YOLOv8m.pt
13. ags.py
14. camera.py
15. constant.py
16. main.py
17. relay.py
18. yolo.py
19. freeRAM.sh
20. freeSpace.sh
```

Pseudocode 3.1 Project Directory

Modul *camera.py* dirancang untuk mengintegrasikan IP *camera* dengan sistem. Modul ini bertanggung jawab atas penangkapan gambar dari IP *camera* menggunakan *Real Time Streaming Protocol* (RTSP) dengan bantuan *package OpenCV*. Setiap kamera diidentifikasi menggunakan indeks yang ditentukan dalam RTSP, seperti alamat standar untuk IP camera Hikvision seperti “rtsp://username:password@ipaddress:554/h264Preview\_01\_main”. Hasil penangkapan gambar disimpan dalam direktori ‘image’. Untuk mendukung penggunaan multi kamera dengan metode paralel, modul ini mengatur penyimpanan gambar sesuai dengan masing-masing kamera. Ini dilakukan dengan membuat direktori tambahan seperti ‘cam0/’, ‘cam1/’, dan seterusnya, sesuai dengan jumlah kamera yang digunakan.

Setelah berhasil mengakses IP *camera*, gambar akan diambil dan disimpan setiap t detik, yang diatur berdasarkan umpan balik dari *Resource-aware Framework*. Inisiasi awal untuk penangkapan gambar dilakukan setiap 1 detik. Gambar yang disimpan adalah gambar mentah tanpa adanya pengolahan tambahan seperti *resize* atau penambahan cahaya. Semua gambar disimpan dalam direktori ‘image/’ dengan format PNG. Nama file gambar dibuat berdasarkan waktu penyimpanan gambar tersebut untuk membedakan setiap proses penyimpanan. Sistem ini dibangun dengan *loop main thread* untuk membaca IP *camera*, sedangkan proses penangkapan dan penyimpanan gambar berjalan pada loop tersendiri pada *camera thread*. Garis besar proses tersebut dilihat pada Pseudocode 3.2.

```

1. CLASS Camera:
2.   PUBLIC PROCEDURE stream:
3.     ret, frame = read_stream_video()
4.     IF terminate:
5.       stop()
6.   END PROCEDURE
7.
8.   PUBLIC PROCEDURE setTimeToCapture(AGS_FEEDBACK):
9.     timeToCapture = AGS_FEEDBACK
10.  END PROCEDURE
11.
12.  PUBLIC PROCEDURE capture:
13.    WHILE
14.      IF frame:
15.        saveImage()
16.      END IF
17.      time.sleep(timeToCapture)
18.    END WHILE
19.  END PROCEDURE
20. END CLASS

```

### Pseudocode 3.2 Camera Class

Pada pseudocode 3.2, baris 2 – 6 merupakan proses inisiasi membaca video stream yang dilakukan pada *main thread*. Pada baris 8 – 10 merupakan proses penyesuaian waktu pengambilan gambar yang didapatkan dari umpan balik *Resource-aware Framework*. Pada baris 12 – 19 merupakan proses pengambilan dan penyimpanan gambar pada direktori yang sudah ditentukan, dengan jeda waktu sesuai umpan balik dari *Resource-aware Framework*.

Setelah gambar disimpan pada direktori 'image/', maka proses akan lanjut pada modul yolo.py untuk mendeteksi objek pada gambar. Gambar baru yang tersimpan pada direktori 'image/' akan ditangkap oleh *event listener* menggunakan *watchdog package*. Pada proses ini terdapat tiga *class* yang diperlukan, yaitu *class YOLO*, *class EventHandler*, dan *class EventObserver*. Pada *class EventObserver* menjalankan proses pemantauan direktori yang telah ditentukan. Jika terdapat perubahan, seperti penambahan file baru dalam direktori tersebut, maka akan ditangani oleh *class EventHandler*. Proses pemantauan direktori berjalan dalam sebuah *thread*, sehingga selama sistem berjalan, pemantauan tersebut tetap aktif. Selanjutnya *class EventHandler* bertanggung jawab untuk menangani perubahan yang telah dipantau oleh *class EventObserver*. Ketika terjadi perubahan berupa penambahan file baru, method 'on\_created' dipanggil untuk menjalankan deteksi objek menggunakan *class YOLO*. Hasil dari deteksi ini kemudian akan disimpan dan dapat diakses menggunakan method 'getYoloResult'.

Pada *class YOLO* menjalankan proses deteksi objek sesuai dengan model YOLO yang telah *di-training* sebelumnya. Proses *load model* dilakukan dengan menggunakan package yang sesuai dengan versi model. Untuk load model YOLOv4 digunakan package OpenCV dengan method cv2.dnn.readNet, yang memerlukan file weight dan config sebagai parameter. Model YOLOv5 menggunakan package PyTorch dengan method torch.hub.load, dengan parameter berupa file PyTorch. Sedangkan untuk YOLOv8 menggunakan package ultralytics dengan method YOLO dengan parameter berupa file PyTorch. Setelah load model, langkah selanjutnya adalah me-resize gambar menjadi 640x640 sebelum dimasukkan sebagai input ke dalam model yang telah dipilih. Hasil dari deteksi adalah rata-rata nilai *confidence* dari objek yang dideteksi dan jumlah objek yang terdeteksi. Namun jika terjadi kesalahan atau tidak ada objek yang

terdeteksi maka akan mengembalikan nilai 0. Pada *class* ini tidak semua gambar yang tersimpan akan diproses. Waktu yang diperoleh dari umpan balik *Resource-aware Framework* digunakan sebagai penanda untuk menunda proses deteksi gambar selama t detik untuk mengurangi beban CPU. Proses deteksi akan mengembalikan nilai 2 jika sedang dalam waktu tunggu. Garis besar dari proses ini dapat dilihat melalui Pseudocode 3.3

```
1. CLASS YOLO
2. PUBLIC PROCEDURE detect(image_path):
3.   IF processPaused == TRUE:
4.     RETURN 2
5.   ELSE IF processPaused == FALSE:
6.     PRIVATE confidences[] = inference_with_yolo(image_path)
7.     RETURN avarage(confidences)
8.   END IF
9. END PROCEDURE
10. END CLASS
11.
12. CLASS EventHandler:
13.   PUBLIC PROCEDURE on_created(event):
14.     time.sleep()
15.     PRIVATE result = yoloDetect(image_path)
16.     yoloResult = result
17.   END PROCEDURE
18. END CLASS
19.
20. CLASS EventObserver()
21.   PUBLIC PROCEDURE run:
22.     Observer.schedule(event_handler, image_dir)
23.     Observer.start()
24.   END PROCEDURE
25. END CLASS
```

Pseudocode 3.3 *Class YOLO*

Dalam *class* YOLO terdapat procedure yang berada pada baris 2 – 9. Procedure ini digunakan untuk mendeteksi objek dan mengembalikan nilai rata-rata confidence dari semua objek yang terdeteksi. Pada *class* EventHandler, baris 13 – 17 berisi procedure yang mengatasi perubahan file, yang akan menjalankan proses deteksi objek setiap kali ada perubahan. Pada *class* EventObserver, baris 21 – 25 berisi procedure untuk memantau direktori yang telah ditentukan dan menjalankan objek EventHandler menggunakan *thread*. Output dari *class* YOLO dikomunikasikan dengan *class* Relay di *class* main. *Class* Relay memiliki list pendek dengan panjang hanya 5 elemen. List ini nanti akan bertipe Boolean. Apabila nilai return dari *class* YOLO lebih dari 0 dan bukan 2, maka nilai True akan ditambahkan ke dalam list. Jika tidak, nilai Flase yang akan ditambahkan. Pengecekan ini terdapat pada *main class*, sebagai media komunikasi antar *class*. Setelah list penuh, akan dilakukan pop pada nilai indeks 0, menjadikan list ini berfungsi seperti queue. List ini menentukan kapan relay harus dihidupkan atau dimatikan. Pseudocode berikut menunjukkan garis besar dari *class* Relay.

```

1. CLASS Relay
2. PUBLIC PROCEDURE run:
3.   WHILE TRUE
4.     IF length_of_TRUE(yoloRes)>=3 AND length_of_TRUE(yoloRes)<=4:
5.       turnOn_relay()
6.     ELSE IF length_of_FALSE(yoloRes)>=3 AND length_of_FALSE(yoloRes)<=4:
7.       turnOff_relay()
8.     END IF
9.   END WHILE
10.  END PROCEDURE
11.
12. PUBLIC PROCEDURE appendYoloRes(Boolean)
13.   yoloRes.append(Boolean)
14. END PROCEDURE
15. END CLASS

```

Pseudocode 3.4 Class Relay

Procedure pada *class relay* ini cukup sederhana. Baris 2 – 10 merupakan procedure untuk mengirim sinyal ke relay. Sementara itu, procedure di baris 12 – 14 digunakan untuk menambahkan nilai boolean pada list. Untuk pengecekan, dilakukan pada *main class* seperti yang telah dijelaskan. *Class Resource-aware Framework (class AGS)* adalah yang paling penting dalam proses ini karena return dari *class Resource-aware Framework* dapat mempengaruhi penggunaan sumber daya pada Raspberry Pi. *Class Resource-aware Framework* memiliki tiga *thread* yang berbeda untuk memantau sumber daya CPU, RAM, dan *disk storage*. *Class* ini menggunakan package psutil untuk memonitor kondisi sumber daya pada Raspberry Pi. Setiap proses pemantauan berjalan pada *thread* yang berbeda di dalam metode yang berbeda di *class Resource-aware Framework*. Setiap loop di dalam *thread* tersebut akan terus memperbarui kondisi sumber daya Raspberry Pi sehingga perhitungan dapat dilakukan dan hasilnya dikomunikasikan melalui *main class* menggunakan setter dan getter yang telah dibuat. Pemantauan sumber daya secara umum dijelaskan pada Pseudocode 3.5.

```

1. CLASS AGS
2. PUBLIC PROCEDURE WatchCPU:
3.   WHILE
4.     PUBLIC cpu = monitorCPUPrecentage()
5.     IF cpu > ThresholdCPU and cpu < 100:
6.       cpuWarning = FALSE
7.       timeToProcess = random(1,5) x counterUpthCPU
8.       counterUpthCPU = counterUpthCPU + 1
9.     ELSE IF cpu >= 100:
10.      cpuWarning = TRUE
11.    ELSE ram < ThresholdCPU:
12.      cpuWarning = FALSE
13.      counterUpthCPU = 1
14.    END IF
15.  END WHILE
16. END PROCEDURE
17.
18. PUBLIC PROCEDURE WatchRAM:
19.   WHILE
20.     PUBLIC ram = monitorRAMPrecentage()
21.     IF ram > ThresholdRAM and ram < 100:
22.       ramWarning = FALSE
23.       timeToCapture = 1 + (1 - (ThresholdRAM/100)) x counterUpthRAM
24.       counterUpthRAM = counterUpthRAM + 1
25.     ELSE IF ram >= 100:

```

```

26.         ramWarning = TRUE
27.     ELSE ram < ThresholdRAM:
28.         counterUpthRAM = 1
29.         ramWarning = FALSE
30.     END IF
31. END WHILE
32. END PROCEDURE
33.
34. PUBLIC PROCEDURE WatchDisk:
35.     WHILE
36.         PUBLIC disk = monitorDiskPrecentage()
37.         IF disk > ThresholdDisk AND disk < 100:
38.             diskWarning = FALSE
39.             timeToCapture = 1 + (1 - (ThresholdDisk/100)) x counterUpthDisk
40.             counterUpthDisk = counterUpthDisk + 1
41.         ELSE IF disk >= 100:
42.             diskWarning = TRUE
43.         ELSE disk < ThresholdDisk:
44.             counterUpthDisk = 1
45.             diskWarning = FALSE
46.         END IF
47.     END WHILE
48. END PROCEDURE
49. END CLASS

```

### Pseudocode 3.5 Class Resource-aware Framework

Pada Pseudocode 3.5, *thread* untuk memantau CPU terdapat pada baris 2 – 16, yang melakukan pengecekan terhadap persentase penggunaan CPU pada sistem dan melakukan perhitungan sesuai dengan yang telah dijelaskan. Pengamatan penggunaan RAM dan *disk storage* serupa, hanya saja sumber daya yang diamati berbeda. Prosedur pengamatan RAM terdapat pada baris 18 – 32, sedangkan pengamatan *disk storage* terdapat pada baris 35 – 48. Seluruh proses dalam *class* akan dibuat objeknya di *main class*. Komunikasi antar *thread* dari tiap objek terjadi di *main class*, yang merupakan *main thread*. Setiap *thread* akan dijalankan ketika *main thread* sudah membaca IP camera dari objek Camera yang dibuat. Komunikasi dari objek *Resource-aware Framework* juga dilakukan untuk dikirim ke tiap *thread*. Penanganan terkait *disk storage* yang penuh dilakukan dengan mengeksekusi bash script yang menghentikan sementara sistem untuk mencegah error. Ketika RAM mencapai batas maksimum, sebuah bash script akan dieksekusi untuk mengatur ulang kondisi RAM, dan program akan menghapus cache untuk memulai program kembali. Gambaran besar pseudocode dari *main class* yang merupakan *main thread* pada sistem dijelaskan dalam Pseudocode 3.6.

```

1. CLASS Main:
2.     PUBLIC PROCEDURE start:
3.         Start_all_thread()
4.     WHILE true:
5.         Camera.stream()
6.         Camera.setTimeCapture(ags.getTimeCapture())
7.
8.         Yolo.setTimeout(ags.getTimeout())
9.
10.        IF cpu_warning:
11.            Yolo.setTimeout(10)
12.        END IF
13.
14.        IF ram_warning:
15.            Restart()
16.        END IF

```

```

17.
18.      IF disk_warning:
19.          Free space
20.          Sleep(5)
21.      END IF
22.
23.      IF yolo.result is not None:
24.          IF yolo.result>0 and yolo.result<2:
25.              Relay.push(true)
26.          ELSE IF yolo.result <=0:
27.              Relay.push(false)
28.          END IF
29.      END IF
30.  END WHILE
31. END PROCEDURE
32. END CLASS

```

### Pseudocode 3.6 Main Class

Pada Pseudocode 3.6, baris 2 menunjukkan proses di mana semua *thread* dijalankan. Selanjutnya, baris 5 membaca stream video dari IP camera. Baris 6 – 7 mencakup komunikasi antara AGS dengan *thread* camera dan *thread* YOLO. Baris 10 – 20 adalah penanganan ketika sumber daya terkait penuh. Sumber daya CPU akan memberhentikan proses deteksi selama 10 detik, RAM akan menghapus cache dari program, dan *disk storage* akan menghapus data lama pada Raspberry Pi serta menghentikan sementara sistem agar direktori tidak mengalami kegagalan dalam memproses. Pengiriman hasil deteksi objek ke *thread* relay dilakukan pada baris 23 – 29. Setelah program dihentikan dengan interupsi pada keyboard, seluruh *thread* yang berjalan akan diberhentikan. Hasil dari sistem akan disimpan dalam file CSV yang berisi pencatatan deteksi objek, mencakup timestamp proses deteksi, nama file gambar, nilai rata-rata confidence, dan jumlah deteksi. Selain itu, sistem juga mencatat penggunaan sumber daya CPU, RAM, dan disk dalam bentuk file CSV. File ini berisi timestamp pencatatan sumber daya, durasi penggunaan setelah sistem dijalankan dalam satuan detik, serta persentase sumber daya yang digunakan.

## BAB IV HASIL DAN PEMBAHASAN

### 4.1 Hasil Pengujian

Bagian ini mencakup hasil pengujian yang dilakukan dalam jangka waktu terbatas, di mana setiap program dijalankan selama tiga hingga enam jam. Upaya dilakukan untuk memastikan durasi waktu yang sama bagi setiap program guna memperoleh hasil yang konsisten dan dapat dibandingkan secara akurat. Namun, keterbatasan waktu ini mungkin menyebabkan data penggunaan sumber daya yang tercatat tidak optimal untuk analisis menyeluruh, mengingat waktu yang terbatas dan banyaknya pengujian. Data yang dikumpulkan selama menjalankan sistem mencakup *timestamp* pencatatan sumber daya, durasi penggunaan setelah sistem dijalankan dalam satuan detik, serta persentase sumber daya yang digunakan.

#### 4.1.1 Skenario 1: Satu Kamera tanpa *Resource-aware Framework*

Dalam skenario satu yaitu pengujian satu kamera tanpa *Resource-aware Framework*, rata-rata setiap pengujian berlangsung selama tiga jam. Skenario ini dirancang dengan menghilangkan interaksi antara *main thread* dan *thread Resource-aware Framework*. Akibatnya, tidak ada adaptasi sumber daya ketika *threshold* tercapai. Skenario ini melibatkan enam model YOLO yang berbeda. Berikut adalah tabel perbandingan hasil pengujian skenario satu pada beberapa model YOLO.

Tabel 4.1 Perbandingan Hasil Pengujian Skenario 1 pada Model YOLO

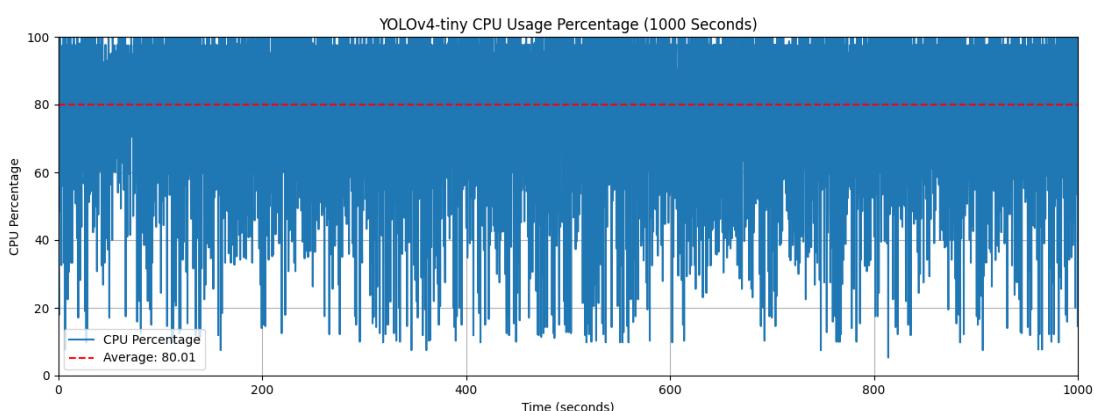
No	Model	Rata-rata CPU (%)	Rata-rata RAM (%)	Waktu Disk Naik 1% (s)	Peningkatan Selisih Waktu/1000s (s)	Rata-rata Jumlah Deteksi/1000 detik
1	YOLOv4-Tiny	80	21,13	181,48	30	826,1
2	YOLOv4	86,97	68,29	185,52	850	103,4
3	YOLOv5s	92,52	23,72	180,18	300	565,1
4	YOLOv5m	93,56	26,81	182,14	600	253,5
5	YOLOv8s	92,78	23,29	179,85	500	362,6
6	YOLOv8m	94,25	25,89	180,83	700	191,5

Dalam hal penggunaan CPU, model YOLOv8m menunjukkan rata-rata penggunaan CPU tertinggi sebesar 94,25%, sedangkan penggunaan CPU terendah ada pada model YOLOv4-Tiny sebesar 80%. Hal ini menunjukkan bahwa YOLOv8m membutuhkan sumber daya pemrosesan yang lebih besar dibandingkan model lainnya. Penggunaan RAM tertinggi tercatat pada model YOLOv4 dengan angka 68,29%, dikarenakan ukuran model yang cukup besar sehingga membutuhkan kapasitas RAM yang lebih besar saat memuat model tersebut. Sebaliknya, YOLOv4-Tiny menunjukkan penggunaan RAM terendah, yaitu sebesar 21,13%. Dalam hal penggunaan disk, semua model menggunakan ukuran totak *disk storage* yang sama sebesar 32 GB. Waktu yang diperlukan untuk peningkatan penggunaan disk sebesar 1% dari total

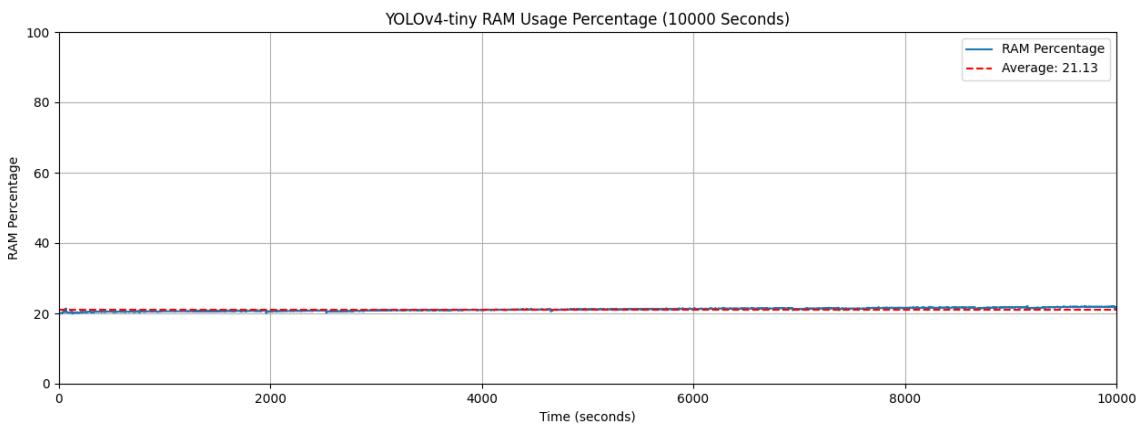
penyimpanan disk berkisar antara 179 – 185 detik. Ini menunjukkan bahwa perbedaan dalam model YOLO tidak berdampak signifikan pada penggunaan disk selama pengujian.

Peningkatan selisih waktu antara waktu pengambilan gambar dan waktu pemrosesan deteksi objek setiap interval 1000 detik menunjukkan variasi yang signifikan di antara berbagai model YOLO. Hal ini menunjukkan semakin lama sistem dijalankan, maka semakin besar pula selisih waktu antara waktu pengambilan gambar dan waktu pemrosesan deteksi objek. Selisih waktu terbesar terjadi pada YOLOv4, di mana selisih waktu meningkat sebesar 850 detik setiap interval 1000 detik. Model lainnya seperti YOLOv5s, YOLOv5m, YOLOv8s, dan YOLOv8m juga menunjukkan performa yang kurang optimal dalam kondisi tanpa *Resource-aware Framework*, dengan peningkatan selisih waktu masing-masing sebesar 300 detik, 600 detik, 500 detik, dan 700 detik. Sebaliknya, YOLOv4-Tiny menunjukkan selisih waktu terkecil, yaitu sekitar 30 detik setiap interval 1000 detik. Meskipun ini lebih baik dibandingkan model-model YOLO lainnya, YOLOv4-Tiny tetap tidak ideal untuk pemrosesan deteksi objek tanpa *Resource-aware Framework*, karena selisih waktu cenderung meningkat seiring berjalananya waktu. Namun, performa YOLOv4-Tiny masih lebih baik daripada beberapa model YOLO lainnya.

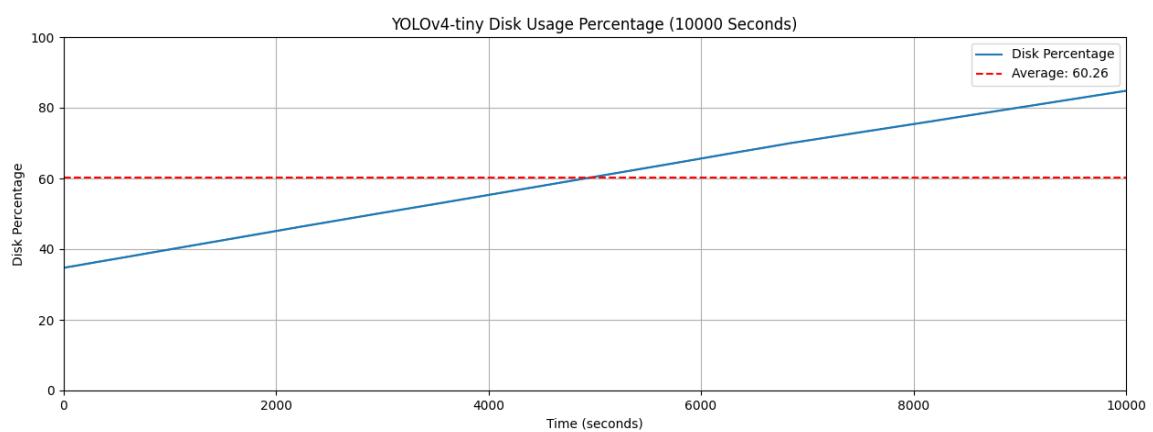
Jumlah deteksi yang dihasilkan oleh berbagai model YOLO menunjukkan perbedaan yang cukup signifikan. Model YOLOv4-Tiny menghasilkan jumlah deteksi terbanyak, dengan rata-rata 826,1 deteksi per interval 1000 detik. Rata-rata jumlah deteksi YOLOv4-Tiny mencapai sekitar delapan kali lipat lebih banyak dibandingkan dengan rata-rata deteksi YOLOv4, yang hanya berjumlah 103,4 deteksi per interval 1000 detik. YOLOv4 merupakan model dengan rata-rata jumlah deteksi objek paling sedikit dibandingkan dengan model YOLO lainnya. Di sisi lain, YOLOv5s dan YOLOv8s juga menunjukkan rata-rata jumlah deteksi objek yang lebih banyak dibandingkan dengan model YOLO versi medium yang setara. Berikut merupakan grafik dan tabel dari hasil pengujian pada model YOLOv4-Tiny. Grafik penggunaan CPU dapat dilihat pada gambar 4.1, penggunaan RAM pada gambar 4.2, penggunaan *disk storage* pada gambar 4.3, dan tabel selisih waktu proses deteksi objek pada tabel 4.2. Grafik untuk model lainnya dapat dilihat pada Lampiran 1.



Gambar 4.1 Persentase Penggunaan CPU YOLOv4-Tiny pada Skenario 1



Gambar 4.2 Persentase Penggunaan RAM YOLOv4-Tiny pada Skenario 1



Gambar 4.3 Persentase Penggunaan Disk YOLOv4-Tiny pada Skenario 1

Tabel 4.2 Selisih Waktu Proses Deteksi YOLOv4-Tiny pada Skenario 1 dalam Interval 1000 Detik

<b>Waktu Berlalu (s)</b>	<b>Waktu Proses Deteksi</b>	<b>Waktu Pengambilan Gambar</b>	<b>Selisih Waktu (s)</b>	<b>Jumlah Deteksi</b>
0	19:55:11	19:55:09	2	1
1000	20:11:51	20:11:46	5	855
2000	20:28:31	20:28:29	2	863
2999	20:45:10	20:45:09	1	860
3999	21:01:50	21:01:49	1	851
5000	21:18:31	21:18:30	1	860
6000	21:35:11	21:35:10	1	862
7000	21:51:51	21:51:44	7	845
8000	22:08:31	22:07:46	45	753
9000	22:25:11	22:23:57	74	756
10000	22:41:51	22:40:03	108	756

#### 4.1.2 Skenario 2: Satu Kamera Menggunakan *Resource-aware Framework*

Dalam skenario dua yaitu pengujian satu kamera menggunakan *Resource-aware Framework*, rata-rata setiap pengujian berlangsung selama enam jam. Pengujian skenario ini dilakukan beberapa kali untuk menentukan nilai *threshold* yang optimal pada setiap sumber daya. Penentuan nilai *threshold* sangat penting karena *threshold* yang terlalu tinggi akan mengakibatkan sulitnya memperoleh efek adaptasi dari *Resource-aware Framework*. Sebaliknya, jika nilai *threshold* yang digunakan terlalu rendah, kemampuan sistem untuk mendeteksi objek akan menjadi tidak efektif, terutama saat terdapat objek yang sudah ada pada gambar. Setelah mengumpulkan data dari pengujian sebelumnya dan skenario satu, nilai *threshold* untuk skenario dua ditentukan sebagai berikut: *threshold* CPU sebesar 80%, *threshold disk storage* sebesar 50%, dan *threshold RAM* yang bervariasi tergantung pada model YOLO yang digunakan. Nilai *threshold RAM* pada model YOLOv4-Tiny sebesar 21%, YOLOv4 sebesar 68%, YOLOv5s sebesar 24%, YOLOv5m sebesar 26%, YOLOv8s sebesar 23%, dan YOLOv8m sebesar 27%. Berikut adalah tabel perbandingan hasil pengujian skenario dua, yaitu deteksi objek menggunakan *Resource-aware Framework* pada beberapa model YOLO.

Tabel 4.3 Perbandingan Hasil Pengujian Skenario 2 pada Beberapa Model YOLO

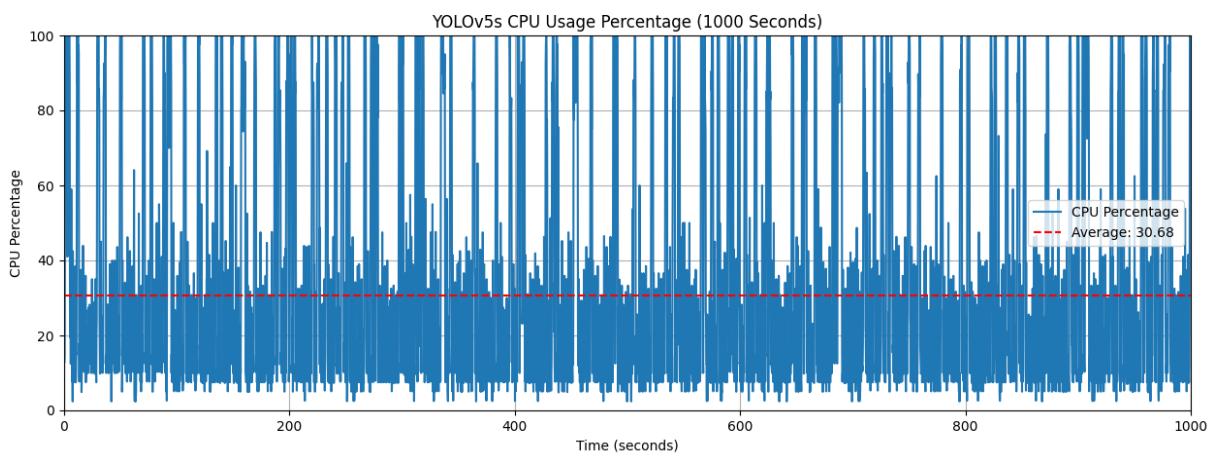
No	Model	Rata-rata CPU (%)	Rata-rata RAM (%)	Waktu Disk Naik 1% sebelum Threshold (s)	Waktu Disk Naik 1% sesudah Threshold (s)	Selisih Waktu (s)	Rata-rata Jumlah Deteksi/1000 detik
1	YOLOv4-Tiny	25,1	21,13	186,77	1045,94	1 – 3	97,7
2	YOLOv4	70,91	68,35	252,14	1310	9 – 28	72,5
3	YOLOv5s	30,68	23,41	204,15	1110,38	1 – 2	99,7
4	YOLOv5m	41,27	25,22	189,11	971,25	3 – 7	70,7
5	YOLOv8s	31,69	21,71	221,19	1190	2 – 5	102,2
6	YOLOv8m	44,63	26,15	189,32	968,25	4 – 10	76,3

Dalam rata-rata penggunaan CPU, YOLOv4 menunjukkan bahwa penggunaan tertinggi mencapai 70,91%, sementara YOLOv4-Tiny memiliki rata-rata penggunaan terendah sekitar 25%. Model-model lainnya mengalami rata-rata penggunaan CPU di bawah 50%, dengan rentang antara 30% hingga 45%. Ini disebabkan oleh adanya waktu tunda antara proses deteksi objek yang membantu meringankan beban kerja CPU, sehingga tidak mencapai puncak 90% hingga 100% seperti yang terjadi pada skenario pertama. Selain pengaruh waktu tunda ini, faktor lain yang mempengaruhi adalah lamanya proses deteksi objek. YOLOv4, meskipun memiliki waktu tunda, masih menunjukkan penggunaan CPU yang tinggi karena proses deteksi objeknya yang memakan waktu lama. Hal ini menandakan bahwa YOLOv4 membutuhkan sumber daya pemrosesan yang lebih besar dibandingkan model lainnya, sehingga kurang cocok digunakan pada perangkat dengan sumber daya terbatas seperti Raspberry Pi.

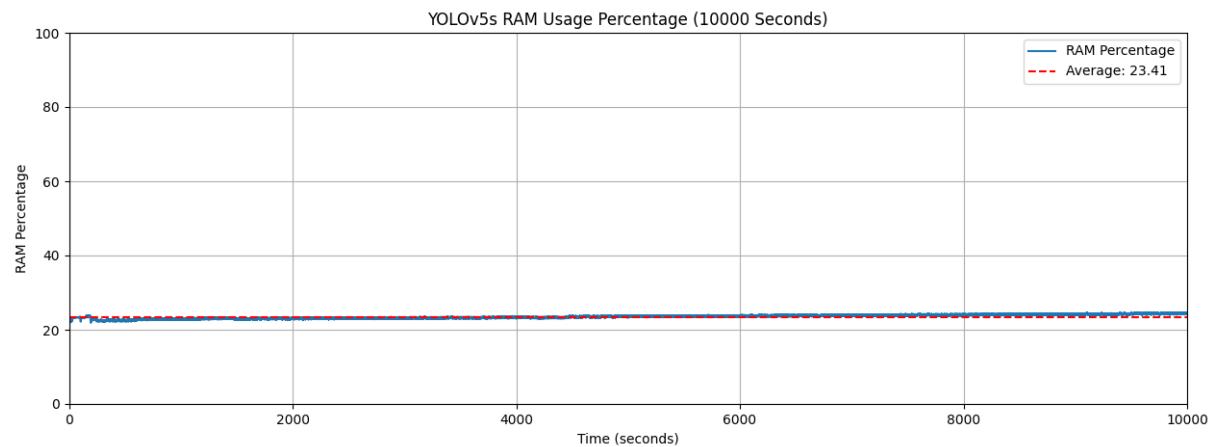
Rata-rata penggunaan RAM tertinggi tercatat pada model YOLOv4 dengan angka 68,35%, dikarenakan ukuran model yang cukup besar sehingga membutuhkan kapasitas RAM yang lebih besar saat memuat model tersebut. Sebaliknya, YOLOv4-Tiny menunjukkan rata-rata penggunaan RAM terendah, yaitu sebesar 21,13%. Model YOLO yang lain memiliki rata-rata penggunaan RAM berkisar antara 21% – 26%. Rata-rata penggunaan RAM skenario ini mengalami penurunan yang sedikit dibandingkan dengan skenario pertama. Dalam hal penggunaan disk, semua model menunjukkan pola yang hampir serupa dengan peningkatan 1% disk sebelum mencapai *threshold* dalam rentang waktu 186 hingga 253 detik. Namun, setelah mencapai *threshold*, peningkatan 1% disk membutuhkan waktu lebih lama, yaitu antara 968 hingga 1310 detik. Perbandingan antara peningkatan disk setelah mencapai *threshold* dan sebelum mencapai *threshold* menunjukkan bahwa setelah mencapai *threshold*, peningkatan disk melambat sebanyak lima kali lipat dibandingkan dengan sebelum mencapai *threshold*. Selain itu, penggunaan disk sebelum mencapai *threshold* pada skenario ini memiliki selisih yang kecil dibandingkan dengan penggunaan disk pada skenario pertama, karena adaptasi sumber daya belum diterapkan sebelum *threshold* tercapai.

Selisih waktu antara pengambilan data gambar dan proses deteksi menunjukkan variasi yang signifikan di antara berbagai model YOLO. YOLOv4 memiliki selisih waktu terbesar, yaitu antara 9 hingga 30 detik. Hal ini menunjukkan bahwa meskipun menggunakan *Resource-aware Framework*, model YOLOv4 tetap memerlukan sumber daya yang besar. Model lainnya seperti YOLOv5m, YOLOv8s, dan YOLOv8m juga menunjukkan performa yang kurang optimal dalam kondisi menggunakan *Resource-aware Framework*, dengan selisih waktu masing-masing sebesar 3 hingga 7 detik, 2 hingga 5 detik, dan 4 hingga 10 detik. Sebaliknya, YOLOv4-Tiny dan YOLOv5s menunjukkan selisih waktu yang paling kecil, hanya sekitar 1 hingga 3 detik dan 1 hingga 2 detik masing-masing. Rata-rata jumlah deteksi setiap interval 1000 detik antar model YOLO berkisar 70 hingga 102. Pada YOLOv4-Tiny, YOLOv5s, dan YOLOv8s memiliki rata-rata jumlah deteksi yang hampir sama, yaitu berkisar antara 97 – 102. Sebaliknya YOLOv4, YOLOv5m, dan YOLOv8m memiliki rata-rata jumlah YOLO berkisar 70 – 76. Hal ini menunjukkan bahwa model YOLO varian *small* lebih cepat dalam memproses deteksi objek dibandingkan dengan varian *medium*. Namun, jika dibandingkan dengan skenario pertama, rata-rata jumlah deteksi mengalami penurunan yang signifikan. Misalnya, YOLOv4-Tiny pada skenario pertama memiliki rata-rata jumlah deteksi sebesar 826, sedangkan pada skenario ini hanya mencapai 97, yang merupakan penurunan lebih dari delapan kali lipat. Hal ini disebabkan oleh adanya waktu tunda dalam proses deteksi yang terjadi ketika CPU mencapai *threshold*, sehingga sistem beradaptasi dengan waktu tunda tersebut.

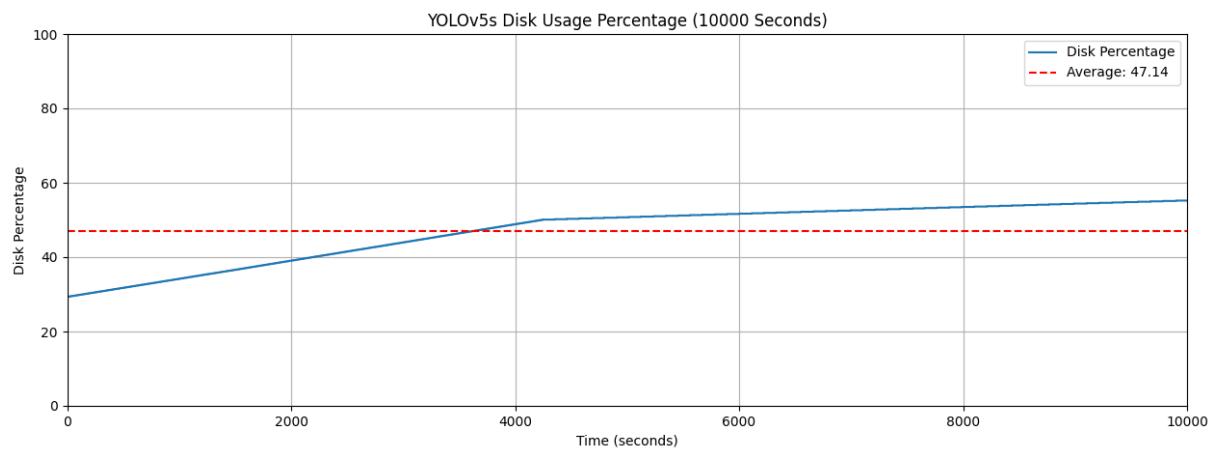
Berikut merupakan grafik dan tabel dari hasil pengujian pada model YOLOv5s. Grafik penggunaan CPU dapat dilihat pada gambar 4.4, penggunaan RAM pada gambar 4.5, penggunaan *disk storage* pada gambar 4.6, dan tabel selisih waktu proses deteksi objek pada tabel 4.4. Grafik untuk model lainnya dapat dilihat pada Lampiran 2.



Gambar 4.4 Persentase Penggunaan CPU YOLOv5s pada Skenario 2



Gambar 4.5 Persentase Penggunaan RAM YOLOv5s pada Skenario 2



Gambar 4.6 Persentase Penggunaan Disk YOLOv5s pada Skenario 2

Tabel 4.4 Selisih Waktu Proses Deteksi YOLOv5s pada Skenario 2 dalam Interval 1000 Detik

Waktu Berlalu (s)	Waktu Proses Deteksi	Waktu Pengambilan Gambar	Selisih Waktu (s)	Jumlah Deteksi
0	22:25:21	22:25:19	2	1
997	22:41:58	22:41:57	1	110
1996	22:58:37	22:58:36	1	117
3005	23:15:26	23:15:25	1	99
4005	23:32:06	23:32:04	2	112
5001	23:48:42	23:48:40	2	90
6000	00:05:21	00:05:19	2	93
7005	00:22:06	00:22:04	2	90
8004	00:38:45	00:38:44	1	88
8997	00:55:18	00:55:17	1	107
9997	01:11:58	01:11:57	1	102

#### 4.1.3 Skenario 3: Dua Kamera tanpa *Resource-aware Framework*

Pada skenario tiga, yaitu pengujian dua kamera tanpa menggunakan *Resource-aware Framework*, rata-rata setiap pengujian berlangsung selama tiga jam. Skenario ini sama dengan skenario pertama, yang menghapus komunikasi antara *main thread* dengan *thread Resource-aware Framework*, sehingga tidak ada adaptasi penggunaan sumber daya. Dalam skenario ini, dua metode akan diterapkan, yaitu metode jadwal (*schedule*) dan metode paralel. Dalam metode jadwal (*Schedule*), dua kamera akan menangkap gambar secara bergantian setiap dua menit. Metode ini hanya menggunakan satu proses YOLO yang akan berpindah-pindah mengikuti kamera yang aktif. Sedangkan metode paralel, kedua kamera akan mengambil gambar secara bersamaan. Setiap kamera akan memiliki satu proses YOLO terpisah, sehingga total jumlah proses YOLO yang berjalan sama dengan jumlah kamera yang digunakan. Setiap proses YOLO berjalan secara paralel, sehingga tidak mengganggu proses YOLO lainnya. Pendekatan ini dilakukan untuk memungkinkan pemrosesan deteksi objek yang memerlukan dua kamera atau lebih dapat menyesuaikan penggunaan sumber dayanya secara tepat dan efisien. Berikut adalah tabel perbandingan hasil pengujian skenario tiga, yaitu deteksi objek beberapa model YOLO tanpa menggunakan *Resource-aware Framework* pada dua kamera dengan dua metode.

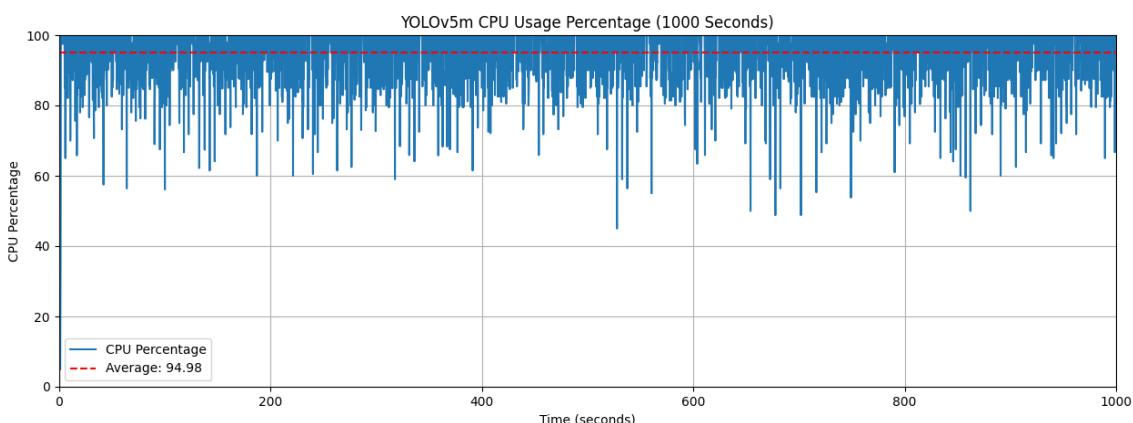
Tabel 4.5 Perbandingan Hasil Pengujian Skenario 3 pada Beberapa Model YOLO

No	Model - Metode	Rata-rata CPU (%)	Rata-rata RAM (%)	Waktu Disk Naik 1% (s)	Peningkatan Selisih Waktu / 1000 (s)	Rata-rata Jumlah deteksi (1 kamera)
1	YOLOv4-Tiny – <i>Schedule</i>	79,48	21,35	165,28	250	686,1
2	YOLOv4 – <i>Schedule</i>	85,61	68,94	171,23	900	90,8
3	YOLOv5s – <i>Schedule</i>	93,42	25,22	176,05	500	456,4
4	YOLOv5m – <i>Schedule</i>	94,98	27,07	174,21	700	212,2
5	YOLOv8s – <i>Schedule</i>	93,95	23,8	163,93	600	318,5
6	YOLOv8m - <i>Schedule</i>	94,27	27,09	163,39	800	164,5
7	YOLOv4-Tiny – Paralel	74,74	33,77	111,68	500	379,6
8	YOLOv4 – Paralel	NaN	NaN	NaN	NaN	NaN
9	YOLOv5s – Paralel	99,06	27,48	121,63	650	260,1
10	YOLOv5m – Paralel	99,47	31,48	128,14	800	122,1
11	YOLOv8s – Paralel	99,27	26,57	122,4	750	169,1
12	YOLOv8m – Paralel	99,52	34,95	129,74	900	84,8

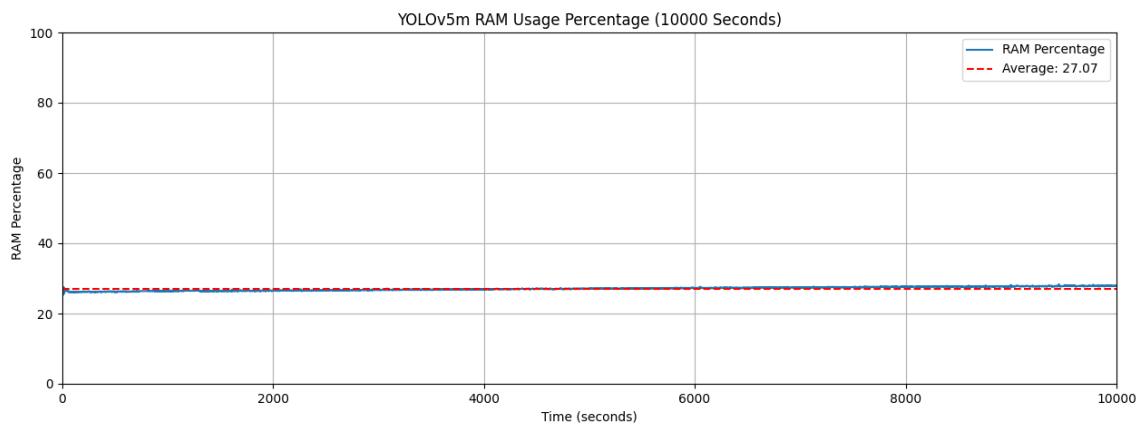
Dalam pengujian skenario 3 menggunakan beberapa model YOLO dengan dua metode, *schedule* dan paralel, ditemukan bahwa rata-rata penggunaan CPU bervariasi secara signifikan. Pada metode *schedule*, model YOLOv4-Tiny menunjukkan rata-rata penggunaan CPU terendah sebesar 79,48%, sementara model YOLOv8m memiliki rata-rata penggunaan CPU tertinggi sebesar 94,27%. Hal ini menunjukkan bahwa YOLOv4-Tiny lebih efisien dalam konsumsi CPU dibandingkan dengan model lain dalam metode ini. Metode paralel memerlukan CPU yang besar karena harus memproses deteksi objek pada dua gambar sekaligus. Model-model seperti YOLOv5s, YOLOv5m, YOLOv8s, dan YOLOv8m menunjukkan penggunaan CPU yang sangat tinggi, dengan rata-rata sekitar 99%. Penggunaan CPU yang tinggi ini tidak boleh dilakukan terus-menerus karena dapat menyebabkan Raspberry Pi mengalami *overheat* dan kerusakan cepat. Di sisi lain, meskipun harus memproses dua gambar sekaligus, penggunaan CPU pada model YOLOv4-Tiny terbilang lebih rendah, yaitu 74,74%. Ini menunjukkan bahwa saat memproses deteksi objek, penggunaan CPU pada model YOLOv4-Tiny lebih ringan dibandingkan model lainnya, membuatnya lebih cocok untuk aplikasi yang memerlukan pemrosesan gambar secara paralel.

Penggunaan RAM pada metode paralel tentu saja lebih tinggi dibandingkan dengan metode *schedule*. Bahkan, pada YOLOv4-Tiny, kenaikan penggunaan RAM mencapai hingga 50%. Model lainnya mengalami kenaikan yang bervariasi antara 8% hingga 30%. Meski demikian, penggunaan RAM tertinggi tercatat pada model YOLOv4 dengan metode *schedule*, yang berada di kisaran 68,29%. Hal ini disebabkan oleh ukuran model yang cukup besar, sehingga membutuhkan kapasitas RAM yang lebih besar saat memuat model tersebut. Menariknya, ketika dilakukan pengujian dengan metode paralel, terjadi *freeze* pada Raspberry Pi karena penggunaan RAM melebihi batas (*Out of Memory*). Dalam hal penggunaan disk, metode paralel menunjukkan peningkatan penggunaan disk 1% dalam waktu yang lebih singkat, yaitu antara 110 hingga 130 detik. Sebaliknya, metode *schedule* memerlukan waktu lebih lama, yaitu antara 160 hingga 180 detik. Perbedaan ini terjadi karena metode paralel menangkap dua gambar secara bersamaan, yang menyebabkan peningkatan penggunaan disk yang lebih cepat. Dengan menangkap dua gambar sekaligus, metode paralel mempercepat laju penulisan data ke disk, yang mengakibatkan waktu yang diperlukan untuk mencapai kenaikan penggunaan disk sebesar 1% menjadi lebih singkat dibandingkan metode *schedule*.

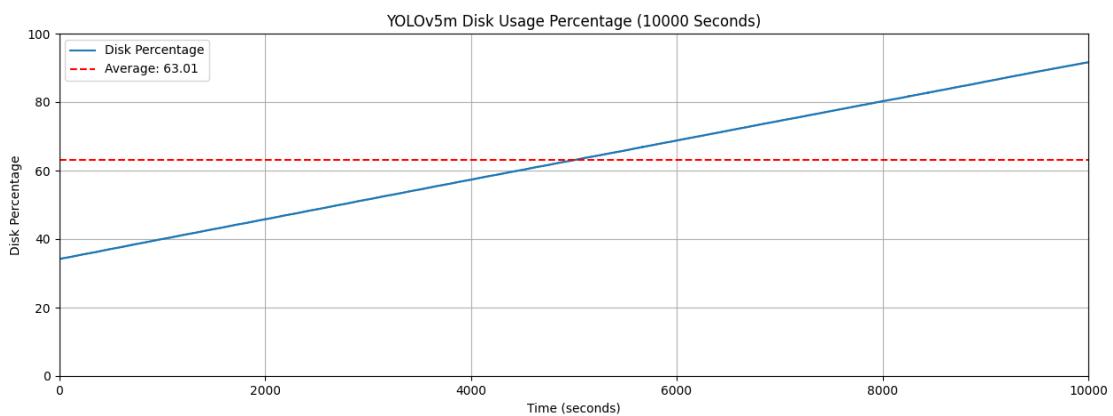
Peningkatan selisih waktu setiap interval 1000 detik pada metode paralel mengalami lonjakan dibandingkan dengan metode *schedule*. Hal ini disebabkan oleh dua proses yang berjalan secara bersamaan, yang menyebabkan pembagian sumber daya Raspberry Pi menjadi dua bagian. Antara metode paralel dan *schedule*, terjadi peningkatan yang bervariasi antara 10% hingga 50%. Meskipun begitu, kedua metode tersebut tidak ideal untuk pemrosesan deteksi objek secara terus-menerus karena selisih waktu cenderung meningkat seiring berjalannya waktu. Pada jumlah deteksi satu kamera pada metode paralel mengalami penurunan hingga 50% dibandingkan dengan metode *schedule*. Akan tetapi jika hasil deteksi dari dua kamera digabungkan, metode paralel memberikan jumlah deteksi lebih banyak. Namun demikian, hal ini menunjukkan adanya keterbatasan sumber daya dalam memproses deteksi objek. Berikut merupakan grafik dan tabel dari hasil pengujian pada model YOLOv5m dengan metode *schedule* dan model YOLOv8m dengan metode paralel. Grafik untuk model lainnya dapat dilihat pada Lampiran 3.



Gambar 4.7 Persentase Penggunaan CPU YOLOv5m dengan Metode *Schedule* pada Skenario 3



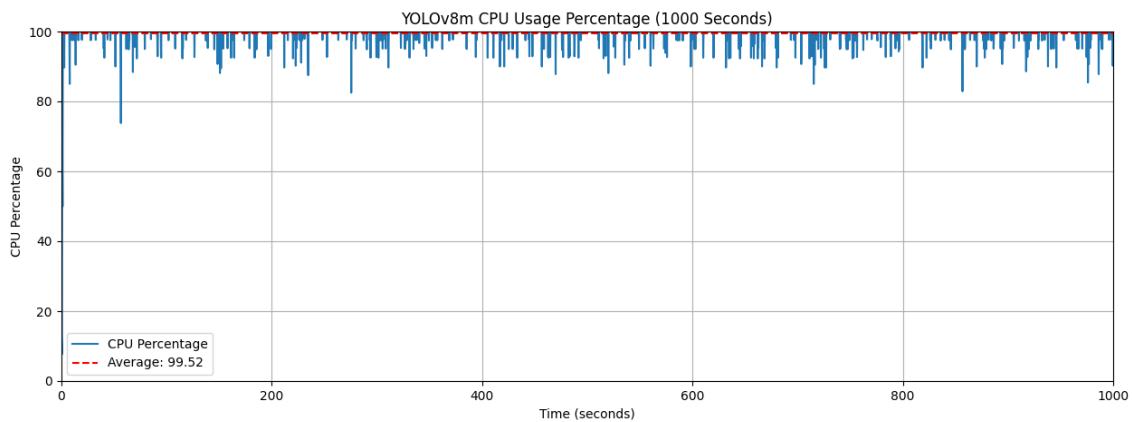
Gambar 4.8 Persentase Penggunaan RAM YOLOv5m dengan Metode *Schedule* pada Skenario 3



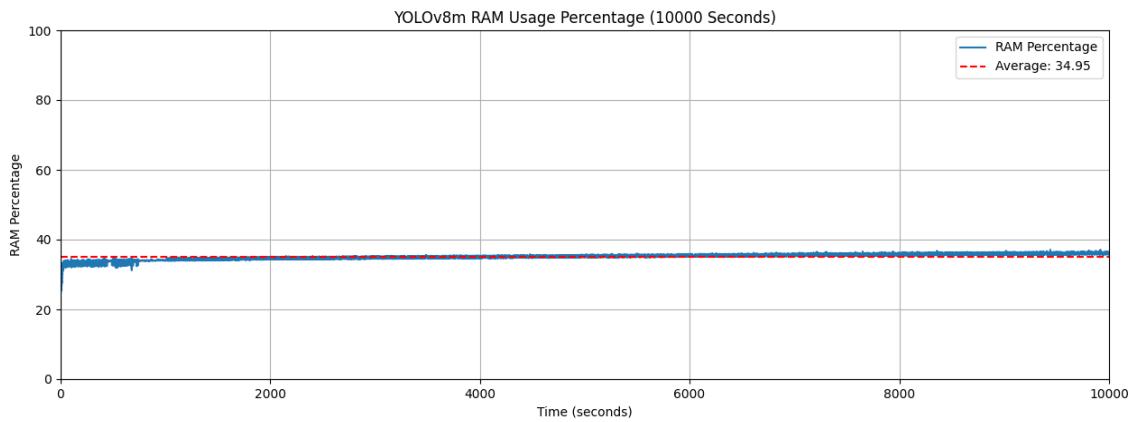
Gambar 4.9 Persentase Penggunaan Disk YOLOv5m dengan Metode *Schedule* pada Skenario 3

Tabel 4.6 Selisih Waktu Proses Deteksi YOLOv5m dengan Metode *Schedule* pada Skenario 3 dalam Interval 1000 Detik

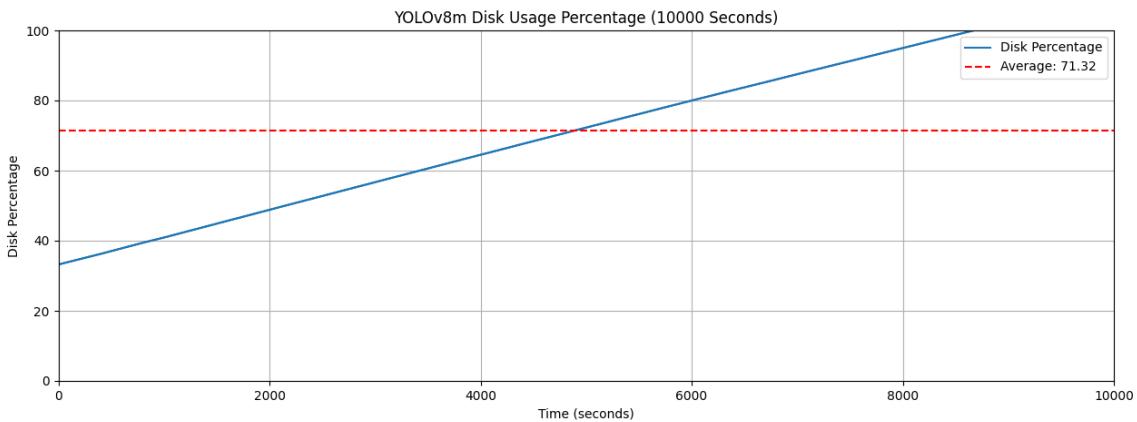
Waktu Berlalu (s)	Waktu Proses Deteksi	Waktu Pengambilan Gambar	Selisih Waktu (s)	Jumlah Deteksi
0	08:38:42	08:38:37	5	1
1001	08:55:23	08:42:12	791	214
2002	09:12:04	08:45:45	1579	214
3001	09:28:43	08:49:14	2369	211
4001	09:45:23	08:52:51	3152	218
4997	10:01:59	08:56:34	3925	223
6002	10:18:44	09:00:03	4721	210
7000	10:35:22	09:03:32	5510	210
8001	10:52:03	09:06:59	6304	208
9000	11:08:42	09:10:28	7094	207
9997	11:25:19	09:13:56	7883	207



Gambar 4.10 Persentase Penggunaan CPU YOLOv8m dengan Metode Paralel pada Skenario 3



Gambar 4.11 Persentase Penggunaan RAM YOLOv8m dengan Metode Paralel pada Skenario 3



Gambar 4.12 Persentase Penggunaan Disk YOLOv8m dengan Metode Paralel pada Skenario 3

Tabel 4.7 Selisih Waktu Kamera 1 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 3 dalam Interval 1000 Detik

<b>Waktu Berlalu (s)</b>	<b>Waktu Proses Deteksi</b>	<b>Waktu Pengambilan Gambar</b>	<b>Selisih Waktu (s)</b>	<b>Jumlah Deteksi</b>
0	11:16:27	11:15:49	38	1
1004	11:33:11	11:21:12	719	77
1999	11:49:46	11:23:09	1597	84
3004	12:06:31	11:25:08	2483	85
3995	12:23:02	11:27:05	3357	84
4998	12:39:45	11:29:07	4238	86
6004	12:56:31	11:31:07	5124	86
7005	13:13:12	11:33:08	6004	85
7996	13:29:43	11:35:07	6876	85
9004	13:46:31	11:37:11	7760	88
9998	14:03:05	11:39:23	8622	94

Tabel 4.8 Selisih Waktu Kamera 2 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 3 dalam Interval 1000 Detik

<b>Waktu Berlalu (s)</b>	<b>Waktu Proses Deteksi</b>	<b>Waktu Pengambilan Gambar</b>	<b>Selisih Waktu (s)</b>	<b>Jumlah Deteksi</b>
0	11:16:27	11:15:49	38	1
1002	11:33:09	11:21:10	719	76
2004	11:49:51	11:23:05	1606	84
3001	12:06:28	11:24:58	2490	83
4003	12:23:10	11:26:52	3378	84
5002	12:39:49	11:28:50	4259	84
5994	12:56:21	11:30:47	5134	84
6994	13:13:01	11:32:44	6017	84
7997	13:29:44	11:34:39	6905	84
8997	13:46:24	11:36:37	7787	86
9998	14:03:05	11:38:49	8656	93

#### 4.1.4 Skenario 4: Dua Kamera Menggunakan *Resource-aware Framework*

Pada skenario empat, yang melibatkan pengujian dua kamera dengan menggunakan *Resource-aware Framework*, rata-rata setiap sesi pengujian berlangsung selama enam jam. Dalam skenario ini, diterapkan dua metode seperti pada skenario ketiga, yaitu metode jadwal (*schedule*) dan metode paralel. Penentuan nilai *threshold* diambil dari analisis hasil skenario sebelumnya, yaitu skenario kedua dan ketiga. Hal ini sangat krusial karena nilai *threshold* yang terlalu tinggi dapat menyulitkan adaptasi yang efektif dari *Resource-aware Framework*. Setelah mengumpulkan data dari skenario kedua dan ketiga, maka nilai *threshold* untuk skenario empat ini ditentukan sebagai berikut: *threshold CPU* sebesar 80%, *threshold disk storage* sebesar 50%, dan *threshold RAM* yang bervariasi tergantung pada model YOLO dan metode yang digunakan. Nilai *threshold RAM* dengan metode *schedule* pada model YOLOv4-Tiny sebesar 21%, YOLOv4 sebesar 68%, YOLOv5s sebesar 24%, YOLOv5m sebesar 26%, YOLOv8s sebesar 23%, dan YOLOv8m sebesar 27%. Sedangkan nilai *threshold RAM* dengan metode paralel pada model YOLOv4-Tiny sebesar 30%, YOLOv5s sebesar 29%, YOLOv5m sebesar 32%, YOLOv8s sebesar 28%, dan YOLOv8m sebesar 36%. Berikut adalah tabel perbandingan hasil pengujian skenario empat, yaitu deteksi objek beberapa model YOLO menggunakan *Resource-aware Framework* pada dua kamera dengan dua metode.

Tabel 4.9 Perbandingan Hasil Pengujian Skenario 4 pada Beberapa Model YOLO

No	Model - Metode	Rata-rata CPU (%)	Rata-rata RAM (%)	Waktu Disk Naik 1% sebelum Threshold (s)	Waktu Disk Naik 1% setelah Threshold (s)	Selisih Waktu (s)	Rata-rata Jumlah deteksi /kamera
1	YOLOv4-Tiny - <i>Schedule</i>	24,74	21,23	194,23	1035,45	1 – 2	89,6
2	YOLOv4 - <i>Schedule</i>	69,85	68,56	219,25	1172,36	10–50	74,2
3	YOLOv5s - <i>Schedule</i>	32,68	23,94	189,42	1006,42	1 – 2	84,2
4	YOLOv5m - <i>Schedule</i>	38,6	25,19	212,69	1057,34	3 – 5	71,7
5	YOLOv8s - <i>Schedule</i>	36,55	22,95	188,64	1029,24	2 – 5	89
6	YOLOv8m - <i>Schedule</i>	45,84	26,13	187,92	961,12	4 – 8	70
7	YOLOv4-Tiny - Paralel	40,02	29,22	93,65	494,97	1 – 2	89,6
8	YOLOv4 - Paralel	NaN	NaN	NaN	NaN	NaN	NaN
9	YOLOv5s - Paralel	50,04	28,46	93,82	492,89	1 – 2	78,7

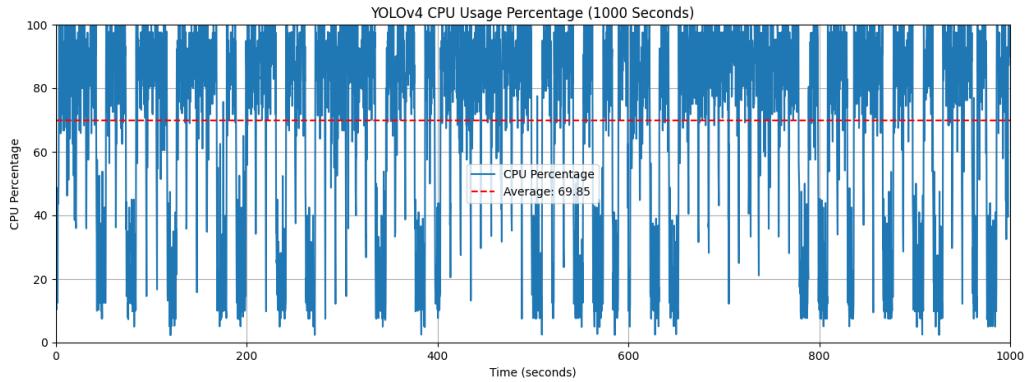
10	YOLOv5m - Paralel	64,95	31,3	103,41	521,64	4 – 7	52,5
11	YOLOv8s - Paralel	58,39	26,04	101,88	501,09	2 – 3	79,6
12	YOLOv8m - Paralel	72,92	35,3	114,72	529,86	5 – 12	51,3

Pada pengujian skenario empat dengan metode *schedule*, hasil yang diperoleh hanya sedikit berbeda dengan skenario kedua karena pada dasarnya hanya satu proses deteksi objek yang berjalan. Rata-rata penggunaan CPU tertinggi pada model *schedule* terdapat pada model YOLOv4, yaitu sebesar 69,85%, sedangkan yang terendah terdapat pada YOLOv4-Tiny, yaitu sebesar 24,74%. Pada metode paralel, penggunaan CPU meningkat signifikan karena harus memproses deteksi objek pada dua gambar sekaligus. Pada saat menggunakan metode paralel, semua model YOLO menunjukkan penggunaan CPU yang lebih tinggi dibandingkan dengan metode *schedule*, dengan rata-rata penggunaan berkisar antara 40% hingga 72%. Meskipun sudah dibantu dengan *Resource-aware Framework*, peningkatan rata-rata penggunaan CPU antara metode *schedule* dan metode paralel mencapai lebih dari 50%. Sebagai contoh, pada model YOLOv4-Tiny, rata-rata penggunaan CPU dengan metode *schedule* adalah 24,74%, sedangkan dengan metode paralel meningkat menjadi 40,02%, yang berarti ada peningkatan sebesar 61,76%. Penggunaan RAM pada metode paralel tentu saja lebih tinggi dibandingkan dengan metode *schedule*. Penggunaan RAM mengalami kenaikan yang bervariasi. Meski demikian, penggunaan RAM tertinggi tercatat pada model YOLOv4 dengan metode *schedule*, yang berada di kisaran 68,56%. Hal ini disebabkan oleh ukuran model yang cukup besar, sehingga membutuhkan kapasitas RAM yang lebih besar saat memuat model tersebut. Menariknya, ketika dilakukan pengujian dengan metode paralel, Raspberry Pi mengalami freeze karena penggunaan RAM melebihi batas (*Out of Memory*).

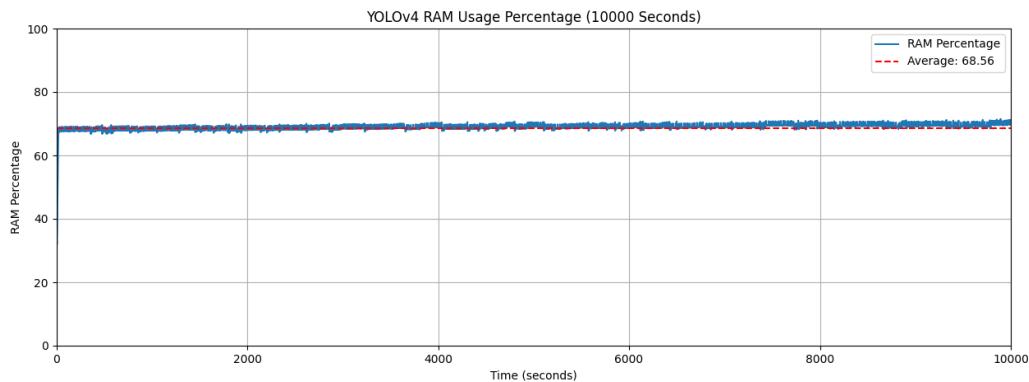
Dalam hal penggunaan disk, metode paralel menunjukkan peningkatan penggunaan disk sebesar 1% sebelum mencapai *threshold* dalam waktu yang lebih singkat, yaitu antara 90 hingga 120 detik. Sebaliknya, metode *schedule* memerlukan waktu lebih lama, yaitu antara 180 hingga 220 detik. Setelah mencapai *threshold*, peningkatan penggunaan disk sebesar 1% untuk kedua metode, *schedule* dan paralel, membutuhkan waktu sekitar 5 kali lebih lama dibandingkan sebelum mencapai *threshold*. Perbedaan ini terjadi karena metode paralel menangkap dua gambar secara bersamaan, yang menyebabkan peningkatan penggunaan disk yang lebih cepat. Ada perbedaan dalam selisih waktu setiap interval 1000 detik antara metode *schedule* dan metode paralel. Perbedaannya terlihat jelas saat metode paralel pertama kali dijalankan, di mana selisih waktunya cukup besar. Namun, setelah itu, perbedaannya mulai stabil. Hal ini disebabkan oleh metode paralel yang menjalankan dua proses berat, yaitu *load model* secara bersamaan. Setelah itu, selisih waktu mulai stabil dan menunjukkan bahwa metode *schedule* sedikit lebih cepat dibandingkan dengan metode paralel. selisih waktu dengan metode *schedule* berkisar antara 3 – 5 detik, sementara dengan metode paralel, selisih waktu meningkat menjadi antara 4 – 7 detik.

Jumlah deteksi objek menggunakan dua kamera antara metode *schedule* dan metode paralel memiliki selisih yang sedikit. Sebagai contoh, model YOLOv5s dengan metode *schedule* memiliki jumlah deteksi berkisar antara 70 hingga 100, sedangkan dengan metode paralel, jumlah deteksinya hanya sedikit berbeda, yaitu antara 60 hingga 90. Namun, jika hasil dari dua

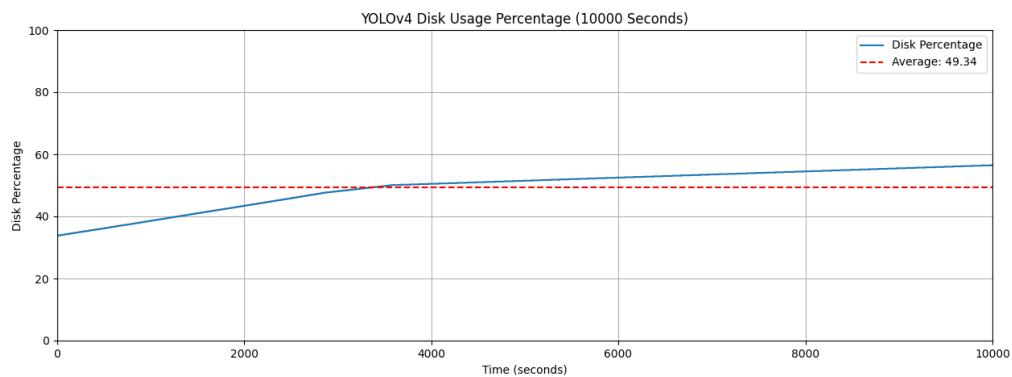
kamera digabungkan, perbandingan jumlah deteksi antara kedua metode tersebut menunjukkan peningkatan dua kali lipat. Ini menunjukkan bahwa penggunaan *Resource-aware Framework* dengan dua kamera yang berjalan bersamaan lebih efisien dan menguntungkan dibandingkan dengan metode *schedule*. Berikut merupakan grafik dan tabel dari hasil pengujian pada model YOLOv4 dengan metode *schedule* dan model YOLOv8s dengan metode paralel. Grafik untuk model lainnya dapat dilihat pada Lampiran 4.



Gambar 4.13 Persentase Penggunaan CPU YOLOv4 dengan Metode *Schedule* pada Skenario 4



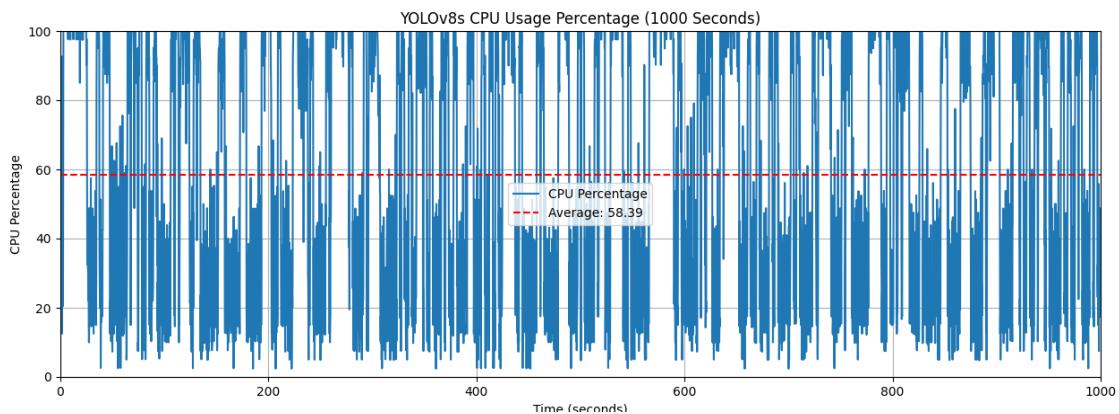
Gambar 4.14 Persentase Penggunaan RAM YOLOv4 dengan Metode *Schedule* pada Skenario 4



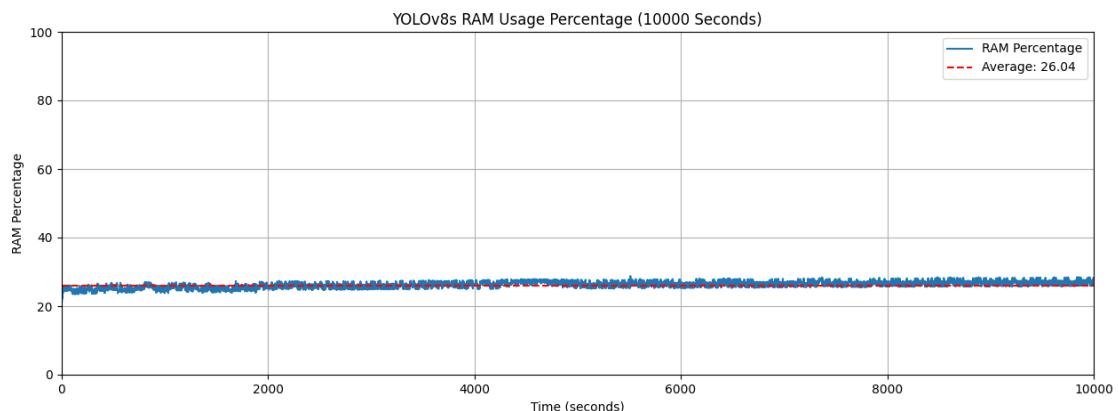
Gambar 4.15 Persentase Penggunaan Disk YOLOv4 dengan Metode *Schedule* pada Skenario 4

Tabel 4.10 Selisih Waktu Proses Deteksi YOLOv4 dengan Metode *Schedule* pada Skenario 4 dalam Interval 1000 Detik

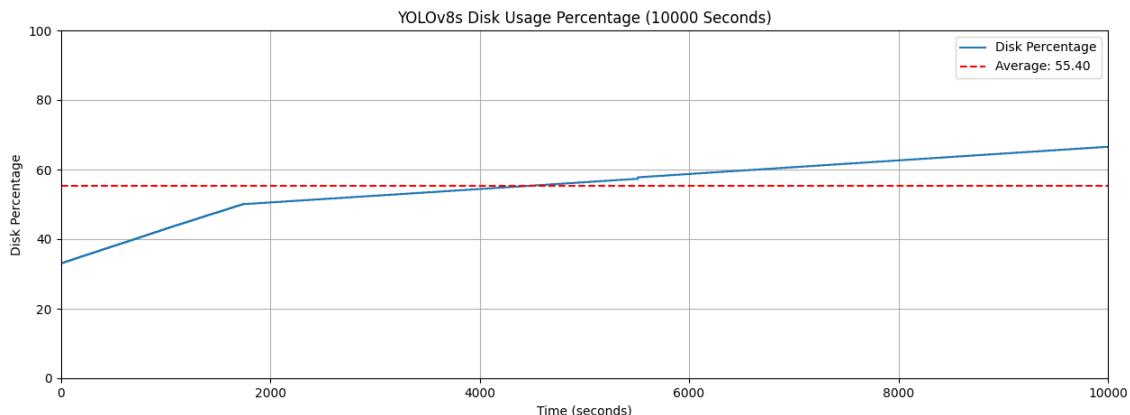
Waktu Berlalu (s)	Waktu Proses Deteksi	Waktu Pengambilan Gambar	Selisih Waktu (s)	Jumlah Deteksi
0	18:39:24	18:39:05	19	1
998	18:56:02	18:55:51	11	73
1998	19:12:42	19:11:58	44	81
2995	19:29:19	19:29:09	10	78
3996	19:46:00	19:45:16	44	78
5005	20:02:49	20:02:39	10	76
5998	20:19:22	20:19:04	18	68
6995	20:35:59	20:35:41	18	67
7999	20:52:43	20:52:06	37	73
9002	21:09:26	21:09:16	10	72
10001	21:26:05	21:25:43	22	76



Gambar 4.16 Persentase Penggunaan CPU YOLOv8m dengan Metode Paralel pada Skenario 4



Gambar 4.17 Persentase Penggunaan RAM YOLOv8m dengan Metode Paralel pada Skenario 4



Gambar 4.18 Persentase Penggunaan Disk YOLOv8m dengan Metode Paralel pada Skenario 4

Tabel 4.11 Selisih Waktu Kamera 1 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 4 dalam Interval 1000 Detik

Waktu Berlalu (s)	Waktu Proses Deteksi	Waktu Pengambilan Gambar	Selisih Waktu (s)	Jumlah Deteksi
0	15:39:53	15:39:25	28	1
1002	15:56:36	15:56:30	6	51
1986	16:13:00	16:12:55	5	56
2980	16:29:34	16:29:28	6	51
4019	16:46:53	16:46:47	6	47
4983	17:02:57	17:02:51	6	46
5988	17:19:42	17:19:35	7	57
7017	17:36:51	17:36:42	9	48
7999	17:53:13	17:53:01	12	47
9005	18:09:59	18:09:51	8	44
10007	18:26:41	18:26:36	5	50

Tabel 4.12 Selisih Waktu Kamera 2 Proses Deteksi YOLOv8m dengan Metode Paralel pada Skenario 4 dalam Interval 1000 Detik

Waktu Berlalu (s)	Waktu Proses Deteksi	Waktu Pengambilan Gambar	Selisih Waktu (s)	Jumlah Deteksi
0	15:39:54	15:39:25	29	1
990	15:56:24	15:56:16	8	61
2019	16:13:33	16:13:21	12	59
2999	16:29:53	16:29:48	5	52

4002	16:46:36	16:46:30	6	54
4997	17:03:11	17:03:05	6	46
5996	17:19:50	17:19:44	6	52
7013	17:36:47	17:36:39	8	45
8000	17:53:14	17:53:04	10	55
9010	18:10:04	18:09:53	11	48
9994	18:26:28	18:26:20	8	57

## 4.2 Pembahasan

Pada bagian ini akan menjelaskan kinerja pada beberapa model YOLO serta beberapa sumber daya, seperti CPU, RAM dan *disk storage*. Hasil dari pengamatan kinerja tersebut dibahas pada sub bab berikut.

### 4.2.1 Model YOLO

Pada pengujian di atas, berbagai model digunakan untuk mencari tingkat akurasi yang tinggi dan kemampuan mendeteksi orang sesuai jumlahnya. Berdasarkan penelitian lain, saat menggunakan YOLOv4-Tiny dengan Raspberry Pi, tingkat akurasi mencapai 99,21% untuk mendeteksi keberadaan orang. Namun, ketepatan dalam mendeteksi jumlah orang tidaklah akurat, hanya benar sebesar 43,65%, sedangkan kesalahan dalam mendeteksi jumlah orang mencapai 56,34% (Shiddiqi, Yogatama, & Navastara, 2023). Berikut ini merupakan hasil akurasi deteksi objek pada beberapa model.

Tabel 4.13 Perbandingan Tingkat Akurasi pada Beberapa Model YOLO

No	Model	Akurasi Deteksi (%)	Akurasi Jumlah Orang (%)
1	YOLOv4-Tiny	94,26	30,33
2	YOLOv4	97,54	43,44
3	YOLOv5s	98,36	47,54
4	YOLOv5m	98,77	59,43
5	YOLOv8s	98,77	56,97
6	YOLOv8m	98,77	68,03

Hasil pengujian diatas menggunakan dataset gambar yang sudah ter-*capture* dari pengujian sebelumnya. Setelah itu, semua model digunakan untuk mendeteksi gambar-gambar tersebut dan menentukan tingkat akurasi masing-masing model. Dari hasil tersebut, terlihat bahwa YOLOv8m memiliki performa terbaik dengan akurasi deteksi objek sebesar 98,77% dan akurasi jumlah orang sebesar 68,03%. Di sisi lain, YOLOv4-Tiny, meskipun memiliki akurasi deteksi yang cukup tinggi sebesar 94,26%, hanya mencapai akurasi jumlah orang sebesar 30,33%. Model YOLOv5s, YOLOv5m, dan YOLOv8s juga menunjukkan kinerja yang mengesankan dengan akurasi deteksi yang lebih dari 97% dan peningkatan bertahap dalam akurasi jumlah orang. Rata-rata jumlah deteksi yang dihasilkan oleh setiap model YOLO menunjukkan variasi yang signifikan. Hal ini mencerminkan kinerja masing-masing model yang bervariasi. Skenario tanpa *Resource-aware Framework* (Skenario 1 dan 3) menghasilkan deteksi lebih banyak dibandingkan skenario dengan *Resource-aware Framework* (Skenario 2

dan 4). Perbedaan ini disebabkan oleh mekanisme *Resource-aware Framework* yang menyesuaikan sumber daya saat mencapai ambang batas tertentu. Penyesuaian ini bertujuan untuk meminimalkan penggunaan sumber daya dengan mengorbankan kecepatan proses deteksi objek, yang pada akhirnya berakibat pada penurunan jumlah deteksi. Berikut merupakan tabel perbandingan rata-rata jumlah deteksi per kamera antar model YOLO.

Tabel 4.14 Perbandingan Rata-rata Jumlah Deteksi Antar Model YOLO

No	Model	Rata-rata Jumlah Deteksi per Kamera					
		Skenario 1	Skenario 2	Skenario 3 (schedule)	Skenario 3 (paralel)	Skenario 4 (schedule)	Skenario 4 (paralel)
1	YOLOv4-Tiny	826,1	97,7	686,1	379,6	89,6	84,8
2	YOLOv4	103,4	72,5	90,8	NaN	74,2	NaN
3	YOLOv5s	565,1	99,7	456,4	260,1	84,2	78,7
4	YOLOv5m	253,5	70,7	212,2	122,1	71,7	52,5
5	YOLOv8s	362,6	102,2	318,5	169,1	89	79,6
6	YOLOv8m	191,5	76,3	164,5	84,8	70	51,3

Penggunaan satu kamera dan dua kamera menunjukkan perbedaan signifikan dalam rata-rata jumlah deteksi, sebagaimana terlihat pada skenario 1 dan 2 (satu kamera) dibandingkan dengan skenario 3 dan 4 (dua kamera). Secara umum, penggunaan satu kamera menghasilkan jumlah deteksi yang lebih banyak dibandingkan dengan dua kamera. Perbedaan ini dapat dijelaskan dengan mempertimbangkan metode yang digunakan untuk mengoperasikan dua kamera. Pada metode *schedule*, proses deteksi dilakukan secara bergantian pada setiap kamera. Kamera sebelumnya dilepas terlebih dahulu sebelum kamera berikutnya disambungkan. Hal ini dapat menyebabkan waktu *idle* pada kamera yang tidak digunakan, sehingga mengurangi total waktu deteksi dan berakibat pada jumlah deteksi yang lebih sedikit. Sedangkan pada metode paralel, dua kamera berjalan bersamaan sehingga beban sumber daya juga terbagi. Contoh yang mencolok adalah model YOLOv4-Tiny. Tanpa *Resource-aware Framework* (skenario 1 dan 3), penggunaan satu kamera menghasilkan rata-rata jumlah deteksi yang lebih banyak dibandingkan dengan dua kamera. Secara rinci, pada skenario 1 (satu kamera) didapatkan rata-rata jumlah deteksi sebanyak 826,1. Sedangkan pada skenario 3 (dua kamera) didapatkan rata-rata jumlah deteksi sebanyak 686,1.

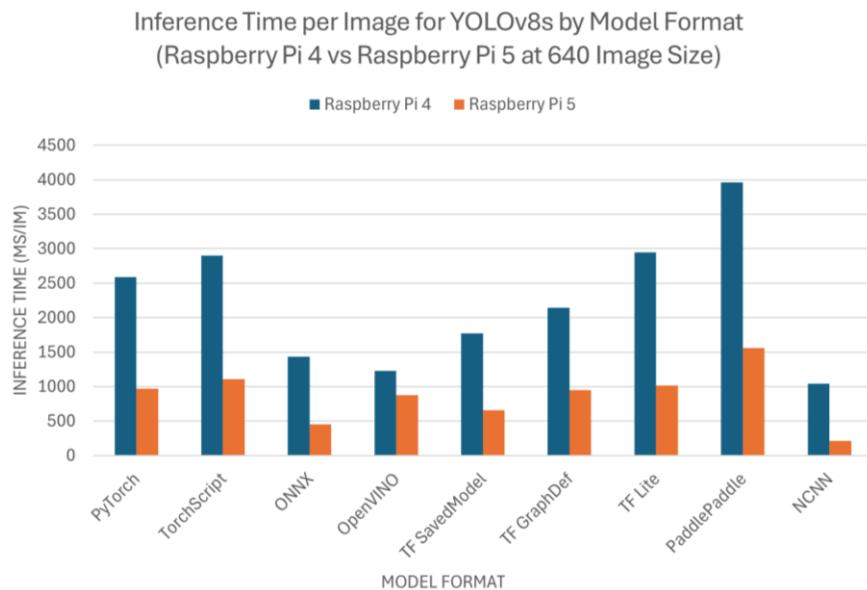
Penggunaan *Resource-aware Framework* menunjukkan hasil yang berbeda. Perhatikan pada skenario 2 dan 4 di mana rata-rata jumlah deteksi per kamera menunjukkan selisih yang tipis. Dalam skenario ini, satu kamera masih menghasilkan deteksi yang lebih banyak per kamera (skenario 2: 97,7 dan skenario 4: 84,4). Namun, jika total deteksi dari dua kamera dengan metode paralel dijumlahkan, maka jumlah deteksi keseluruhan menjadi lebih banyak dibandingkan dengan satu kamera. Pada skenario 4 dengan metode paralel, didapatkan rata-rata jumlah deteksi total 168,8 (hasil penjumlahan dari deteksi kedua kamera), sedangkan pada skenario 2 (satu kamera) didapatkan rata-rata jumlah deteksi sebesar 97,7. Perbedaan performa deteksi antar model YOLO tanpa *Resource-aware Framework* dikaitkan erat dengan *inference time* masing-masing model. *inference time* ini mengacu pada waktu yang dibutuhkan oleh

model YOLO untuk memproses satu frame gambar atau video dan menghasilkan output deteksi objek. Faktor *inference time* ini sangatlah penting dalam menentukan performa deteksi objek. Model YOLO dengan waktu inferensi yang lebih cepat umumnya menghasilkan deteksi objek yang lebih banyak dibandingkan model YOLO dengan waktu inferensi yang lebih lambat. *Inference time* adalah waktu yang diperlukan oleh model untuk memproses satu frame gambar atau video dan menghasilkan output deteksi objek. *Inference time* ini mempengaruhi jumlah deteksi yang dihasilkan, jika proses deteksi memakan waktu lama, maka jumlah deteksi yang dihasilkan juga akan lebih sedikit. Berikut adalah perbandingan *inference time* dari beberapa model YOLO berdasarkan hasil pengujian di atas.

Tabel 4.15 Perbandingan *Inference Time* pada Hasil Pengujian Beberapa Model YOLO

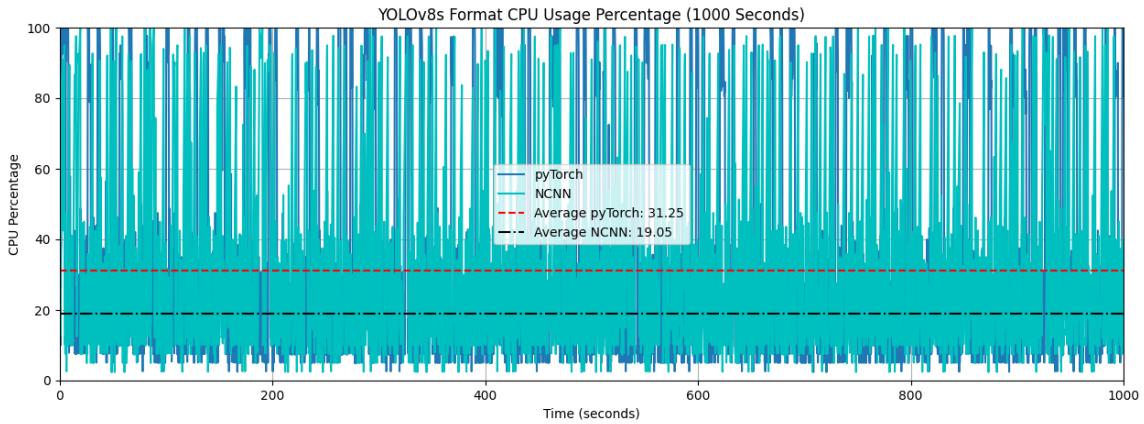
No	Model	<i>Inference Time (s)</i>
1	YOLOv4-Tiny	0,995
2	YOLOv4	8,753
3	YOLOv5s	1,657
4	YOLOv5m	3,984
5	YOLOv8s	2,357
6	YOLOv8m	5,282

*Inference time* juga dipengaruhi oleh format model yang digunakan. Format yang dipakai dalam seluruh pengujian merupakan default awal dari masing-masing pengembang, namun dapat diganti dengan format yang lebih cepat, seperti ONNX atau NCNN. Pada YOLOv8, misalnya, disarankan menggunakan format NCNN untuk perangkat keras seperti Raspberry Pi. Penggunaan format yang lebih efisien dapat membantu mengurangi *inference time* dan meningkatkan jumlah deteksi yang dihasilkan (Jocher, Chaurasia, & Qiu, 2023).



Gambar 4.19 Perbandingan Format pada Model YOLOv8s (Jocher, Chaurasia, & Qiu, 2023)

Pada pengujian kali ini juga melakukan uji coba pada YOLOv8s menggunakan format NCNN dengan memanfaatkan *Resource-aware Framework*. Perbandingan hasil grafiknya dengan YOLOv8s yang menggunakan format pytorch sebagai berikut.



Gambar 4.20 Perbandingan Penggunaan CPU YOLOv8s Format NCNN dan PyTorch

Dari grafik tersebut, terlihat bahwa penggunaan format NCNN lebih efisien dengan rata-rata penggunaan CPU yang rendah, yakni sekitar 19,05%. Model ini mampu menghasilkan jumlah deteksi antara 130 hingga 160, dengan selisih waktu antara proses deteksi dan pengambilan gambar hanya sekitar 1 hingga 2 detik. Di sisi lain, format pyTorch menunjukkan rata-rata penggunaan CPU sebesar 31,25%, dengan jumlah deteksi yang dihasilkan antara 80 hingga 120, dan selisih waktu yang sedikit lebih besar, yakni sekitar 2 hingga 3 detik.

#### 4.2.2 CPU

Penggunaan CPU sebagai sumber daya utama dalam komputasi program deteksi objek menjadi salah satu aspek penting yang perlu diperhatikan. Hal ini dikarenakan konsumsi CPU yang berlebihan dapat berakibat pada penurunan performa dan menyebabkan *overheat* pada Raspberry Pi. Dengan penggunaan *Resource-aware Framework* terbukti mampu menekan penggunaan CPU hingga 60%. Berikut merupakan tabel perbandingan rata-rata penggunaan CPU pada semua skenario.

Tabel 4.16 Perbandingan Rata-rata Penggunaan CPU pada Semua Skenario

No	Model	Rata-rata Penggunaan CPU (%)					
		Skenario 1	Skenario 2	Skenario 3 (schedule)	Skenario 3 (parallel)	Skenario 4 (schedule)	Skenario 4 (parallel)
1	YOLOv4-Tiny	80	25,1	79,48	74,74	24,74	40,02
2	YOLOv4	86,97	70,91	85,61	NaN	69,85	NaN
3	YOLOv5s	92,52	30,68	93,42	99,06	32,68	50,04
4	YOLOv5m	93,56	41,27	94,98	99,47	38,6	64,95
5	YOLOv8s	92,78	31,69	93,95	99,27	36,55	58,39
6	YOLOv8m	94,25	44,63	94,27	99,52	45,84	72,92

Perbandingan penggunaan rata-rata CPU dengan dan tanpa *Resource-aware Framework* dapat dilihat pada perbandingan skenario 1 dengan 2, serta skenario 3 dengan 4. Pada perbandingan ini, penggunaan CPU bisa berkurang hingga 68%. Sebagai contoh, pada model YOLOv4-Tiny, skenario 1 (tanpa *Resource-aware Framework*) menunjukkan rata-rata penggunaan CPU sebesar 80%, sedangkan pada skenario 2 (dengan *Resource-aware Framework*), rata-rata penggunaan CPU turun menjadi 25,1%, menunjukkan penurunan sebesar 68,62%. Hal ini menunjukkan bahwa penggunaan *Resource-aware Framework* dapat menekan penggunaan CPU. Detail perbandingan dapat dilihat pada tabel berikut.

Tabel 4.17 Persentase Penurunan Penggunaan CPU dengan menggunakan *Resource-aware Framework*

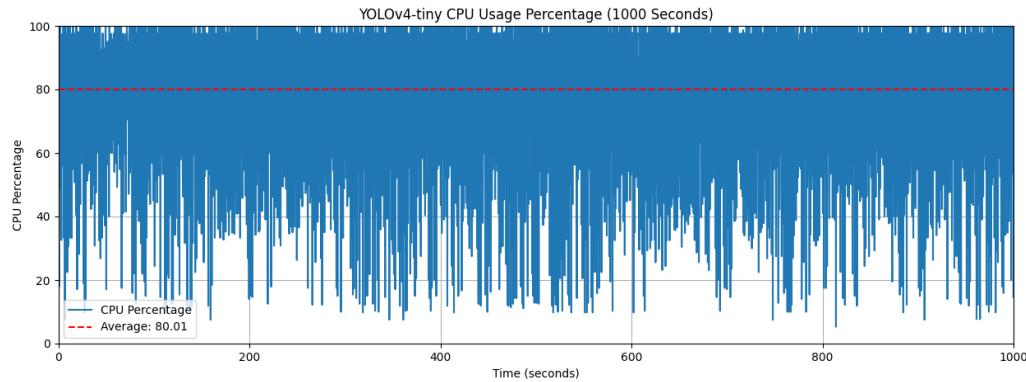
No	Model	Penurunan Penggunaan CPU (%)		
		Skenario 1 & 2	Skenario 3 & 4 (schedule)	Skenario 3 & 4 (parallel)
1	YOLOv4-Tiny	68,63	68,87	46,45
2	YOLOv4	18,47	18,41	NaN
3	YOLOv5s	66,84	65,02	49,49
4	YOLOv5m	55,89	59,36	34,70
5	YOLOv8s	65,84	61,10	41,18
6	YOLOv8m	52,65	51,37	26,73

Rata-rata penggunaan CPU dengan satu kamera dan dua kamera menunjukkan perbedaan yang tergantung pada metode yang digunakan untuk dua kamera. Perbandingan antara satu kamera dan dua kamera dapat dilihat pada skenario 1 dan 3, serta skenario 2 dan 4. Pada metode *schedule*, perbandingan antara satu kamera dan dua kamera menunjukkan perbedaan kecil dalam penggunaan CPU, bahkan bisa menunjukkan penurunan penggunaan CPU karena adanya proses *idle* saat pergantian kamera. Sebaliknya, pada metode parallel, penggunaan CPU meningkat saat menggunakan dua kamera sekaligus dibandingkan dengan satu kamera. Detail perbandingan dapat dilihat di bawah ini.

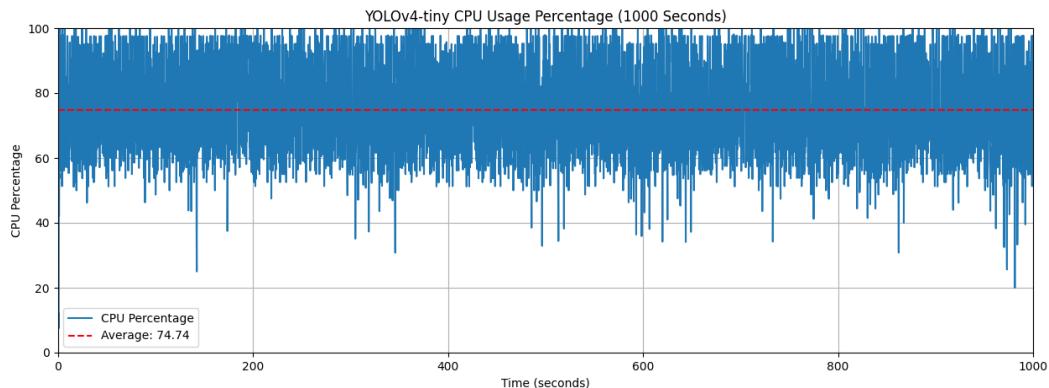
Tabel 4.18 Persentase Peningkatan Penggunaan CPU dengan Perbandingan Penggunaan Satu Kamera dan Dua Kamera

No	Model	Peningkatan Penggunaan CPU (%)			
		Skenario 1 & 3 (schedule)	Skenario 1 & 3 (parallel)	Skenario 2 & 4 (schedule)	Skenario 2 & 4 (parallel)
1	YOLOv4-Tiny	-0,65	-6,58	-1,43	59,44
2	YOLOv4	-1,56	NaN	-1,49	NaN
3	YOLOv5s	0,97	7,07	6,52	63,10
4	YOLOv5m	1,52	6,32	-6,47	57,38
5	YOLOv8s	1,26	7,00	5,34	84,25
6	YOLOv8m	0,02	5,59	2,71	63,39

Namun, ada satu model, yaitu YOLOv4-Tiny, yang justru mengalami penurunan penggunaan CPU dalam metode paralel dibandingkan dengan penggunaan satu kamera. Meskipun demikian, penggunaan CPU pada YOLOv4-Tiny dengan metode paralel tetap stabil di angka yang cukup tinggi, jarang turun di bawah 50%. Grafik yang menunjukkan hasil tersebut dapat dilihat pada gambar berikut.

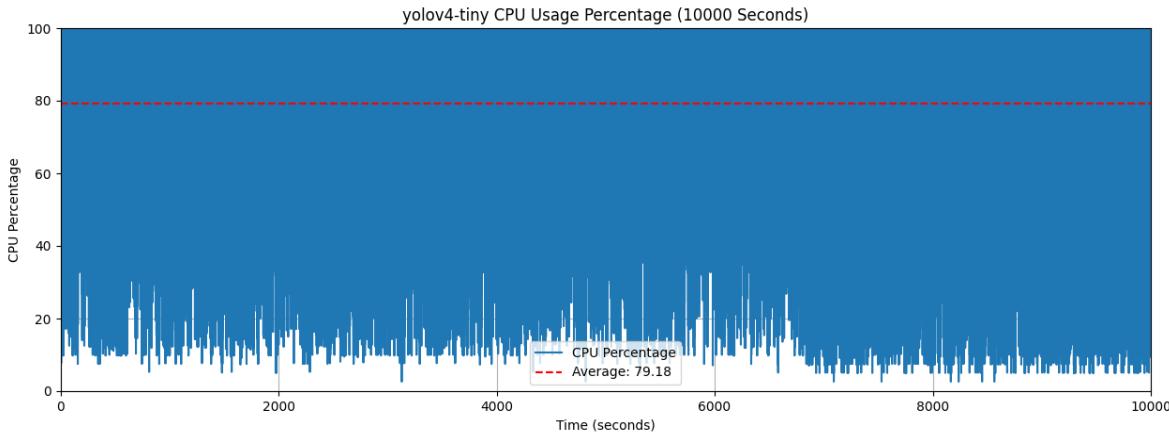


Gambar 4.21 Persentase Penggunaan CPU YOLOv4-tiny pada Satu Kamera di Pengujian Skenario 1



Gambar 4.22 Persentase Penggunaan CPU YOLOv4-tiny pada Dua Kamera dengan Metode Paralel di Pengujian Skenario 3

Pada subbab sebelumnya, data penggunaan CPU disajikan dalam bentuk grafik dengan rentang waktu hanya 1000 detik. Hal ini dilakukan agar grafik terlihat lebih jelas dan dapat menunjukkan bagian mana yang dominan dalam penggunaan CPU. Jika grafik dibuat dengan rentang waktu yang sama seperti untuk RAM dan Disk, yaitu selama 10000 detik, hasilnya akan terlihat terlalu padat dan menumpuk, seperti yang dapat dilihat pada gambar 4.23. Meski grafik penggunaan CPU hanya dibuat dengan rentang waktu 1000 detik, ini sudah cukup untuk memberikan gambaran yang akurat tentang penggunaan CPU selama program dijalankan. Rata-rata penggunaan CPU selama 1000 detik hampir sama dengan rata-rata penggunaan selama 10000 detik, seperti yang ditunjukkan pada grafik di bawah ini. Hal ini berarti bahwa pola penggunaan CPU yang diamati dalam 1000 detik pertama dapat digunakan sebagai representasi penggunaan CPU untuk periode yang lebih panjang.



Gambar 4.23 Persentase Penggunaan CPU dalam Rentang Waktu 10.000 Detik

Dengan adanya *Resource-aware Framework*, penggunaan CPU dapat ditekan hingga 68%. Grafik penggunaan CPU dengan *Resource-aware Framework* menunjukkan pola yang naik turun secara drastis, berbeda dengan pengujian tanpa menggunakan *framework* di mana rata-rata penggunaan CPU hampir selalu di atas 90%. Tingkat penggunaan CPU yang tinggi seperti ini dapat mengakibatkan beban yang besar pada perangkat dan meningkatkan risiko *overheat* yang dapat merusak Raspberry Pi atau perangkat lainnya. Penurunan sesekali dalam penggunaan CPU terjadi ketika terjadi kesalahan dalam proses pengambilan gambar secara baik (Shiddiqi, Yogatama, & Navastara, 2023).

#### 4.2.3 RAM

Dalam pengujian penggunaan RAM, pada seluruh skenario penggunaan RAM tidak mengalami perbedaan yang besar. Rata-rata penggunaan RAM tertinggi ketika model yang digunakan yaitu Model YOLOv4. Hampir pada seluruh percobaan di angka kurang lebih 68%. Sedangkan rata-rata penggunaan RAM terkecil terjadi ketika menggunakan model YOLOv4-Tiny yang menggunakan satu kamera dan dua kamera dengan metode schedule. Berikut merupakan tabel perbandingan rata-rata penggunaan RAM pada semua skenario.

Tabel 4.19 Perbandingan Rata-rata Penggunaan RAM pada Semua Skenario

No	Model	Rata-rata Penggunaan RAM (%)					
		Skenario 1	Skenario 2	Skenario 3 (schedule)	Skenario 3 (paralel)	Skenario 4 (schedule)	Skenario 4 (paralel)
1	YOLOv4-Tiny	21,13	21,13	21,35	33,77	21,23	29,22
2	YOLOv4	68,29	68,35	68,94	NaN	68,56	NaN
3	YOLOv5s	23,72	23,41	25,22	27,48	23,94	28,46
4	YOLOv5m	26,81	25,22	27,07	31,48	25,19	31,3
5	YOLOv8s	23,29	21,71	23,8	26,57	22,95	26,04
6	YOLOv8m	25,89	26,15	27,09	34,95	26,13	35,3

Penggunaan Resource-aware Framework dengan tanpa menggunakan framework itu, pada penggunaan RAM tidak memiliki selisih yang banyak. Dapat dibuktikan dengan melihat perbandingan skenario 1 (tanpa menggunakan Resource-aware Framework) dan 2 (menggunakan Resource-aware Framework), serta antara skenario 3 (tanpa menggunakan Resource-aware Framework) dan 4 (menggunakan Resource-aware Framework). Sebagai contoh penggunaan RAM pada model YOLOv4-Tiny, yang menggunakan Resource-aware Framework maupun tidak memakai, menunjukkan rata-rata penggunaan yang sama di angka 21,13%. Ini menunjukkan bahwa tidak ada peningkatan ataupun penurunan pada model YOLOv4-Tiny. Untuk model YOLO lain dapat dilihat pada tabel dibawah ini.

Tabel 4.20 Persentase Penurunan Penggunaan RAM dengan menggunakan *Resource-aware Framework*

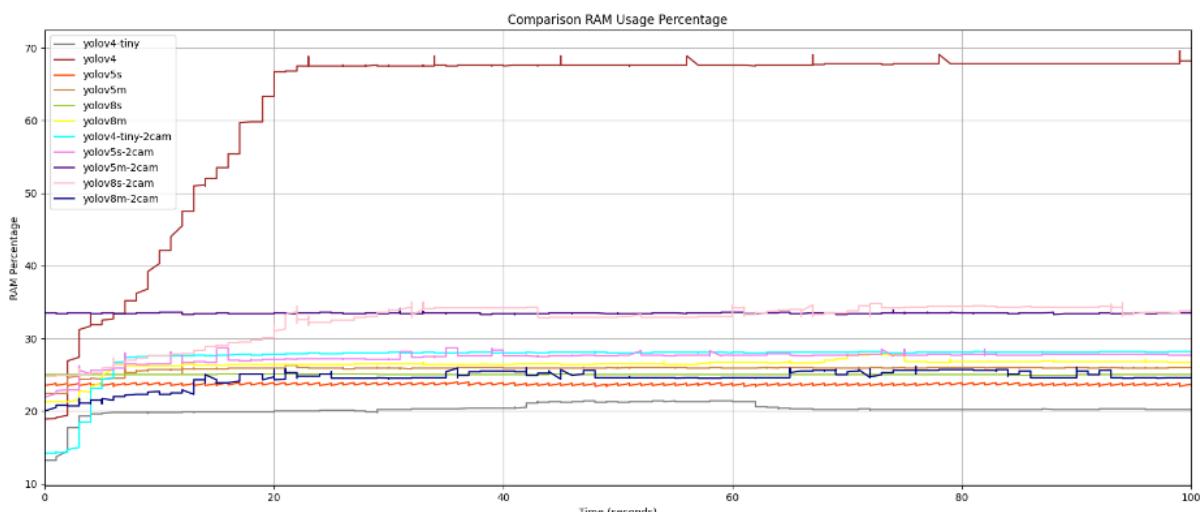
No	Model	Penurunan Penggunaan RAM (%)		
		Skenario 1 & 2	Skenario 3 & 4 (schedule)	Skenario 3 & 4 (paralel)
1	YOLOv4-Tiny	0,00	0,56	13,47
2	YOLOv4	-0,09	0,55	NaN
3	YOLOv5s	1,31	5,08	-3,57
4	YOLOv5m	5,93	6,94	0,57
5	YOLOv8s	6,78	3,57	1,99
6	YOLOv8m	-1,00	3,54	-1,00

Persentase penurunan penggunaan RAM terbesar tercatat pada model YOLOv4-Tiny dengan penggunaan dua kamera menggunakan metode paralel, yaitu sebesar 13,47%, dari 33,77% menjadi 29,22%. Namun, ada beberapa model dalam skenario lain yang justru menunjukkan peningkatan penggunaan RAM saat menggunakan Resource-aware Framework, yang ditandai dengan angka minus pada tabel 4.20. Hal ini menunjukkan bahwa Resource-aware Framework hanya mampu mengurangi penggunaan RAM hingga maksimal 13,47%. Sebaliknya, saat diterapkan pada penggunaan CPU, framework ini mampu mengurangi persentase penggunaan CPU hingga 68%. Perbedaan yang mencolok terjadi pada penggunaan metode paralel dengan dua kamera dibandingkan dengan satu kamera, dengan menambahkan dua kamera dan dua proses deteksi, penggunaan RAM meningkat. Peningkatan penggunaan RAM tertinggi sebesar 59,82%. Meskipun demikian, penambahan satu kamera dan proses deteksi tidak sampai meningkatkan penggunaan RAM hingga dua kali lipat. Hal ini menunjukkan bahwa Raspberry Pi mampu menjalankan deteksi objek dengan menggunakan beberapa kamera secara bersamaan.

Tabel 4.21 Persentase Peningkatan Penggunaan RAM dengan Perbandingan Penggunaan Satu Kamera dan Dua Kamera

No	Model	Peningkatan Penggunaan RAM (%)			
		Skenario 1 & 3 (schedule)	Skenario 1 & 3 (paralel)	Skenario 2 & 4 (schedule)	Skenario 2 & 4 (paralel)
1	YOLOv4-Tiny	1,04	59,82	0,47	38,29
2	YOLOv4	0,95	NaN	0,31	NaN
3	YOLOv5s	6,32	15,85	2,26	21,57
4	YOLOv5m	0,97	17,42	-0,12	24,11
5	YOLOv8s	2,19	14,08	5,71	19,94
6	YOLOv8m	4,63	34,99	-0,08	34,99

Penggunaan RAM dipengaruhi oleh ukuran model. Semakin besar model, semakin banyak RAM yang dibutuhkan. Sebagai contoh, model YOLOv4 yang berukuran 256MB membuat penggunaan RAM mencapai hampir 70%. Ketika pertama kali dijalankan, penggunaan RAM langsung melonjak karena proses *load model*. Namun, setelah beberapa waktu, penggunaan RAM akan stabil. Lonjakan awal ini menyebabkan selisih waktu antara proses deteksi dan pengambilan gambar menjadi lebih lama. Grafik di bawah ini menunjukkan penggunaan RAM pada 100 detik pertama untuk masing-masing model.



Gambar 4.24 Perbandingan Penggunaan Persentase Penggunaan RAM pada Beberapa Model YOLO dalam Interval 100 Detik

#### 4.2.4 Disk

Dalam penggunaan disk, penting untuk membandingkan waktu yang dibutuhkan untuk mencapai peningkatan 1% penggunaan disk. Hal ini disebabkan oleh perbedaan awal *disk storage* pada semua pengujian, yang berkisar antara 29% hingga 35% dari total *disk storage* sebesar 32 GB. Karena perbedaan ini, tidak memungkinkan untuk menggunakan rata-rata penggunaan disk atau peningkatan persentase disk secara langsung. Oleh karena itu,

perbandingan yang lebih adil dilakukan dengan mengukur berapa lama waktu yang dibutuhkan untuk penggunaan disk naik sebesar 1% dari total kapasitas disk.

Tabel 4.22 Perbandingan Waktu Disk Naik 1% pada Skenario 1 dan 2

No	Model	Waktu Disk Naik 1% (s)		
		Skenario 1	Skenario 2 (sebelum threshold)	Skenario 2 (setelah threshold)
1	YOLOv4-Tiny	181,48	186,77	1045,94
2	YOLOv4	185,52	252,14	1310
3	YOLOv5s	180,18	204,15	1110,38
4	YOLOv5m	182,14	189,11	971,25
5	YOLOv8s	179,85	221,19	1190
6	YOLOv8m	180,83	189,32	968,25

Tabel 4.23 Perbandingan Waktu Disk Naik 1% pada Skenario 3 dan 4

No	Model	Waktu Disk Naik 1% (s)		
		Skenario 3	Skenario 4 (sebelum threshold)	Skenario 4 (setelah threshold)
1	YOLOv4-Tiny – Schedule	165,28	194,23	1035,45
2	YOLOv4 - Schedule	171,23	219,25	1172,36
3	YOLOv5s - Schedule	176,05	189,42	1006,42
4	YOLOv5m - Schedule	174,21	212,69	1057,34
5	YOLOv8s - Schedule	163,93	188,64	1029,24
6	YOLOv8m - Schedule	163,39	187,92	961,12
7	YOLOv4-Tiny – Paralel	111,68	93,65	494,97
8	YOLOv4 - Paralel	NaN	NaN	NaN
9	YOLOv5s - Paralel	121,63	93,82	492,89
10	YOLOv5m - Paralel	128,14	103,41	521,64
11	YOLOv8s - Paralel	122,4	101,88	501,09
12	YOLOv8m - Paralel	129,74	114,72	529,86

Pengujian antara menggunakan *Resource-aware Framework* dan tanpa *Resource-aware Framework* (skenario 1 dan 2 serta skenario 3 dan 4) menunjukkan perbedaan yang signifikan. Hal ini dapat dilihat pada kedua skenario tersebut, ketika penggunaan disk belum mencapai *threshold*, pengambilan gambar dilakukan setiap detik, mirip dengan pendekatan pada

pengujian tanpa *Resource-aware Framework*. Namun, setelah penggunaan disk mencapai *threshold*, sistem mengalami adaptasi yang memperlambat pengambilan gambar hingga enam kali lipat.

Tabel 4.24 Persentase Peningkatan Waktu Disk Naik 1% dengan menggunakan *Resource-aware Framework*

No	Model	Peningkatan Waktu Disk Naik 1% (%)		
		Skenario 1 & 2	Skenario 3 & 4 (schedule)	Skenario 3 & 4 (paralel)
1	YOLOv4-Tiny	476,34	526,48	343,20
2	YOLOv4	606,12	584,67	NaN
3	YOLOv5s	516,26	471,67	305,24
4	YOLOv5m	433,24	506,93	307,09
5	YOLOv8s	561,66	527,85	309,39
6	YOLOv8m	435,45	488,24	308,40

Dalam pengujian dengan dua kamera dibandingkan dengan satu kamera, penggunaan disk pada dua kamera tergantung dari metode yang digunakan. Pada metode *schedule*, hasil yang didapatkan memiliki selisih yang kecil dengan penggunaan satu kamera. Namun, pada metode paralel, penggunaan disk bisa meningkat hingga dua kali lipat lebih cepat daripada dengan satu kamera. Hal ini disebabkan karena pada metode paralel, sistem mengambil dua gambar sekaligus dari dua kamera. Detail perbandingan antara satu kamera dengan dua kamera menggunakan metode *schedule* dan *paralel* dapat dilihat pada tabel berikut.

Tabel 4.25 Persentase Penurunan Waktu Disk Naik 1% pada Penggunaan Dua Kamera dengan Metode Schedule

No	Model	Penurunan Waktu Disk Naik 1% (%)		
		Skenario 1 & 3	Skenario 2 & 4 (sebelum <i>threshold</i> )	Skenario 2 & 4 (setelah <i>threshold</i> )
1	YOLOv4-Tiny	8,93	3,84	-1,00
2	YOLOv4	7,70	-15,00	-10,51
3	YOLOv5s	2,29	-7,78	-9,36
4	YOLOv5m	4,35	11,09	8,86
5	YOLOv8s	8,85	-17,26	-13,51
6	YOLOv8m	9,64	-0,74	-0,74

Tabel 4.26 Persentase Penurunan Waktu Disk Naik 1% pada Penggunaan Dua Kamera dengan Metode Paralel

No	Model	Penurunan Waktu Disk Naik 1% (%)		
		Skenario 1 & 3	Skenario 2 & 4 (sebelum <i>threshold</i> )	Skenario 2 & 4 (setelah <i>threshold</i> )
1	YOLOv4-Tiny	38,46	49,86	52,68
2	YOLOv4	NaN	NaN	NaN
3	YOLOv5s	32,50	54,04	55,61
4	YOLOv5m	29,65	45,32	46,29
5	YOLOv8s	31,94	53,94	57,89
6	YOLOv8m	28,25	39,40	45,28

Jika terjadi penurunan waktu penggunaan disk, maka peningkatan disk sebesar 1% akan lebih cepat, karena semakin kecil waktu yang diperlukan, semakin cepat pula kenaikan 1% dalam persentase penggunaan disk. Berdasarkan tabel 4.24, penurunan waktu yang dibutuhkan untuk disk naik 1% mencapai 57%. Ini berarti penggunaan disk pada dua kamera dengan metode paralel lebih cepat hingga dua kali lipat dibandingkan dengan penggunaan satu kamera.

*(Halaman ini sengaja dikosongkan)*

## BAB V KESIMPULAN DAN SARAN

### 5.1 Kesimpulan

Pengujian dan analisis dalam studi ini menghasilkan simpulan yang memberikan jawaban terhadap permasalahan yang telah dibahas. Beberapa kesimpulan yang dapat ditarik dari penelitian ini meliputi:

1. Implementasi dua kamera dapat dilakukan menggunakan dua metode, yaitu metode *schedule* dan metode paralel. Pada metode *schedule*, penggunaan sumber daya memiliki peningkatan yang sedikit dibandingkan dengan penggunaan satu kamera, yaitu penggunaan CPU dan RAM hanya mengalami peningkatan sekitar 0 – 6%. Di sisi lain, metode paralel mengalami peningkatan yang lebih besar, terutama pada CPU yang meningkat hingga 84% dan penggunaan RAM yang meningkat hingga 60%. Oleh karena itu, implementasi dua kamera dengan metode paralel membutuhkan sumber daya yang lebih besar dibandingkan dengan metode *schedule* maupun penggunaan satu kamera.
2. Penggunaan *Resource-aware Framework* dapat menekan penggunaan sumber daya pada satu kamera maupun dua kamera. Dengan menggunakan *Resource-aware Framework*, persentase rata-rata penggunaan CPU dapat diturunkan hingga mencapai 68%. Selain itu, penggunaan disk juga dapat diperlambat hingga enam kali lipat. Untuk penggunaan RAM, *framework* ini dapat menurunkan persentase rata-rata penggunaan hingga 13%. *Framework* ini membantu dalam mengelola alokasi sumber daya seperti CPU, RAM, dan Disk secara dinamis berdasarkan kebutuhan. Dengan pengaturan yang tepat, *Resource-aware Framework* dapat mengoptimalkan penggunaan sumber daya.
3. Evaluasi kinerja beberapa varian Algoritma YOLO seperti YOLOv4-Tiny, YOLOv4, YOLOv5s, YOLOv8s, YOLOv5m, dan YOLOv8m menunjukkan hasil yang bervariasi:
  - YOLOv4-Tiny: Cepat dalam mendeteksi objek dengan *inference time* sebesar 0,995s dan Tingkat akurasi yang mencapai 94,26%, akan tetapi tingkat akurasi kesesuaian jumlah orang yang cukup rendah, yaitu hanya 30,33%.
  - YOLOv4: Memiliki *inference time* yang lebih lama, yaitu 8,753 detik. Meski demikian, tingkat akurasi mencapai 97,54% dengan tingkat akurasi kesesuaian jumlah orang sebesar 43,44%.
  - YOLOv5s: Tingkat akurasi yang lebih baik dibandingkan YOLOv4-Tiny dan YOLOv4, yaitu 98,36% dengan tingkat akurasi kesesuaian jumlah orang sebesar 47,54%. Model ini lebih cepat dalam mendeteksi objek dibandingkan model lain, dengan *inference time* sebesar 1,657 detik, namun masih lebih lambat dibandingkan YOLOv4-Tiny.
  - YOLOv5m: Memiliki tingkat akurasi yang tinggi, yaitu 98,77%, serta tingkat akurasi kesesuaian jumlah orang sebesar 59,43%. *Inference time* yang lebih tinggi daripada YOLOv5s yaitu 3,984s.
  - YOLOv8s: Memiliki tingkat akurasi yang sama dengan YOLOv5m, yaitu 98,77%, dan tingkat akurasi kesesuaian jumlah orang sebesar 56,97%. *Inference time* sebesar 2,357s.
  - YOLOv8m: Sama dengan YOLOv5m dan YOLOv8s dalam hal tingkat akurasi, yaitu 98,77%, namun memiliki tingkat akurasi kesesuaian jumlah orang yang paling

tinggi, yaitu 68,03%. *Inference time* lebih lama dibandingkan YOLOv8s, yaitu 5,282 detik.

Pemilihan varian algoritma dan format model seperti NCNN pada Raspberry Pi juga mempengaruhi kinerja sistem secara keseluruhan. Dengan mempertimbangkan keseimbangan antara kecepatan, akurasi, dan penggunaan sumber daya, pemantauan multi-kamera dapat diimplementasikan dengan efektif sesuai dengan kebutuhan dan kondisi lingkungan yang ada.

## 5.2 Saran

Penelitian ini membutuhkan tahapan eksperimen lanjutan yang melibatkan implementasi langsung pada produk-produk IoT atau perangkat elektronik lainnya. Hal ini diperlukan untuk menguji dan mengoptimalkan aplikasi dari hasil penelitian dalam skenario praktis yang lebih kompleks dan beragam. Hasil penelitian dapat diverifikasi keandalannya dalam kondisi nyata dan kemudian dikembangkan lebih lanjut untuk memenuhi kebutuhan dan tantangan spesifik yang mungkin muncul dalam implementasi di lapangan.

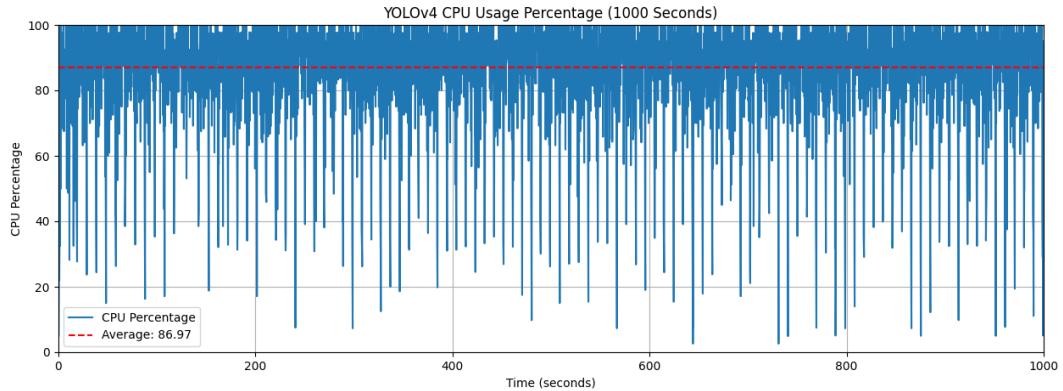
## DAFTAR PUSTAKA

- Alieksieiev, V., & Gaiduchok, O. (2017). *ABOUT THE PROBLEM OF DATA LOSSES IN REAL-TIME IOT BASED MONITORING SYSTEMS*.
- Alothali, E., Alashwal, H., & Harous, S. (2019). Data stream mining techniques: A review. *Telkomnika (Telecommunication Computing Electronics and Control)*, 17(2), 728–737. <https://doi.org/10.12928/TELKOMNIKA.v17i2.11752>
- Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1). <https://doi.org/10.1186/s40537-021-00444-8>
- Chavdarova, T., & Fleuret, F. (2017). Deep multi-camera people detection. *Proceedings - 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, 2017-December*, 848–853. <https://doi.org/10.1109/ICMLA.2017.00-50>
- Cuzzocrea, A., Gaber, M. M., & Shiddiqi, A. M. (2015). Distributed classification of data streams: An adaptive technique. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9263, 296–309. [https://doi.org/10.1007/978-3-319-22729-0\\_23](https://doi.org/10.1007/978-3-319-22729-0_23)
- Fang, W., Wang, L., & Ren, P. (2020). Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments. *IEEE Access*, 8, 1935–1944. <https://doi.org/10.1109/ACCESS.2019.2961959>
- Gaber, M. M., & Yu, P. S. (2006). A Holistic Approach for Resource-aware Adaptive Data Stream Mining. *New Generation Computing*, 25(1), 95–115. <https://doi.org/10.1007/s00354-006-0005-1>
- Jakhar, D., & Kaur, I. (2020). Artificial intelligence, machine learning and deep learning: definitions and differences. *Clinical and Experimental Dermatology*, 45(1), 131–132. <https://doi.org/10.1111/ced.14029>
- James, W. (2020, June 19). *What is Hikvision ColorVu and How Does it Work?* <Https://Icamsecurity.Com.Au/Blogs/News/What-Is-Hikvision-Colovu-and-How-Does-It-Work>.
- Jiang, Z., Zhao, L., Li, S., & Jia, Y. (2020). Real-time object detection method based on improved YOLOv4-tiny. *CoRR, abs/2011.04244*. <https://arxiv.org/abs/2011.04244>
- Jocher, G. (2020). *YOLOv5 by Ultralytics*. <https://doi.org/10.5281/zenodo.3908559>
- Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLO*. <https://github.com/ultralytics/ultralytics>
- Khoshrou, S., Cardoso, J. S., & Teixeira, L. F. (2014). Active learning from video streams in a multi-camera scenario. *Proceedings - International Conference on Pattern Recognition*, 1248–1253. <https://doi.org/10.1109/ICPR.2014.224>
- Liu, Q., Fan, X., Xi, Z., Yin, Z., & Yang, Z. (2022). Object detection based on Yolov4-Tiny and Improved Bidirectional feature pyramid network. *Journal of Physics: Conference Series*, 2209(1). <https://doi.org/10.1088/1742-6596/2209/1/012023>
- Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under Concept Drift: A Review. In *IEEE Transactions on Knowledge and Data Engineering* (Vol. 31, Issue 12, pp. 2346–2363). IEEE Computer Society. <https://doi.org/10.1109/TKDE.2018.2876857>
- Mishra, C., & Gupta, D. L. (2017). Deep Machine Learning and Neural Networks: An Overview. *IAES International Journal of Artificial Intelligence (IJ-AI)*, 6(2), 66. <https://doi.org/10.11591/ijai.v6.i2.pp66-73>

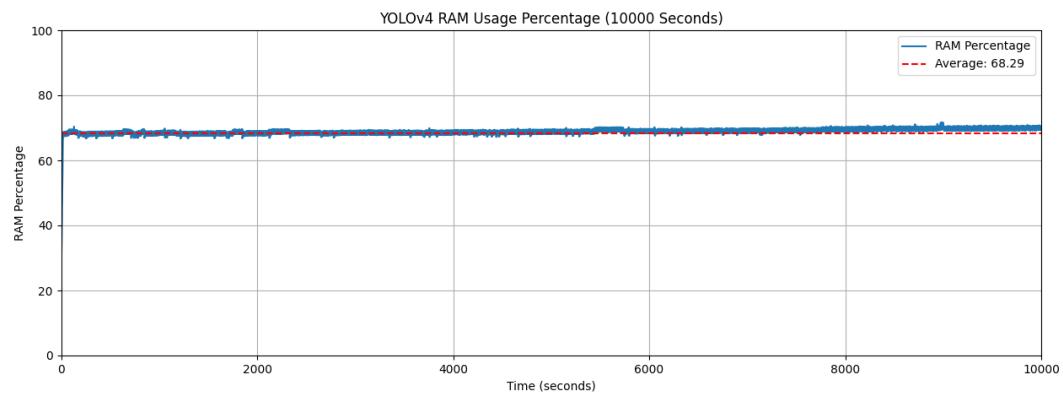
- Mohammed, H. K., & Soliman, A. F. (n.d.). *Data Stream Mining*.  
<https://www.researchgate.net/publication/259647558>
- Moreau, L., & Queinnec, C. (2005). Resource Aware Programming. *ACM Trans. Program. Lang. Syst.*, 27(3), 441–476. <https://doi.org/10.1145/1065887.1065891>
- Patil, D. D., & Wadhai, V. M. (2012). Adaptive Real Time Data Mining Methodology for Wireless Body Area Network Based Healthcare Applications. *Advanced Computing: An International Journal*, 3(4), 59–70.  
<https://doi.org/10.5121/acij.2012.3408>
- Pesapane, F., Codari, M., & Sardanelli, F. (2018). Artificial intelligence in medical imaging: threat or opportunity? Radiologists again at the forefront of innovation in medicine. In *European Radiology Experimental* (Vol. 2, Issue 1). Springer.  
<https://doi.org/10.1186/s41747-018-0061-6>
- Raspberry Pi Foundation. (n.d.). *Tech, Learn, and Make with the Raspberry Pi Foundation*. <Https://Www.Raspberrypi.Org/>.
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. <http://pjreddie.com/yolo/>
- Sharma, A. (2022, April 4). *Introduction to the YOLO family*.  
<Https://Pyimagesearch.Com/2022/04/04/Introduction-to-the-Yolo-Family/>.
- Shiddiqi, A. M., Yogatama, E. D., & Navastara, D. A. (2023). Resource-aware video streaming (RAViS) framework for object detection system using deep learning algorithm. *MethodsX*, 11. <https://doi.org/10.1016/j.mex.2023.102285>
- Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–6.  
<https://doi.org/10.1109/ICCUBEA.2018.8697857>
- Strbac, B., Gostovic, M., Lukac, Z., & Samardzija, D. (2020). YOLO Multi-Camera Object Detection and Distance Estimation. In *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*.
- Terven, J., & Cordova-Esparza, D. (2023). *A Comprehensive Review of YOLO: From YOLOv1 and Beyond*. <http://arxiv.org/abs/2304.00501>
- Wai, C. W., Jegatheesan, J., & Loon, S. C. (2015). Exploring IOT Application Using Raspberry Pi. *International Journal of Computer Networks and Applications*, 2(1).  
<http://www.digi.com>
- Wares, S., Isaacs, J., & Elyan, E. (2019). Data stream mining: methods and challenges for handling concept drift. In *SN Applied Sciences* (Vol. 1, Issue 11). Springer Nature. <https://doi.org/10.1007/s42452-019-1433-0>
- Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2019). *Object Detection in 20 Years: A Survey*. <http://arxiv.org/abs/1905.05055>

## LAMPIRAN-LAMPIRAN ATAU APPENDIKS

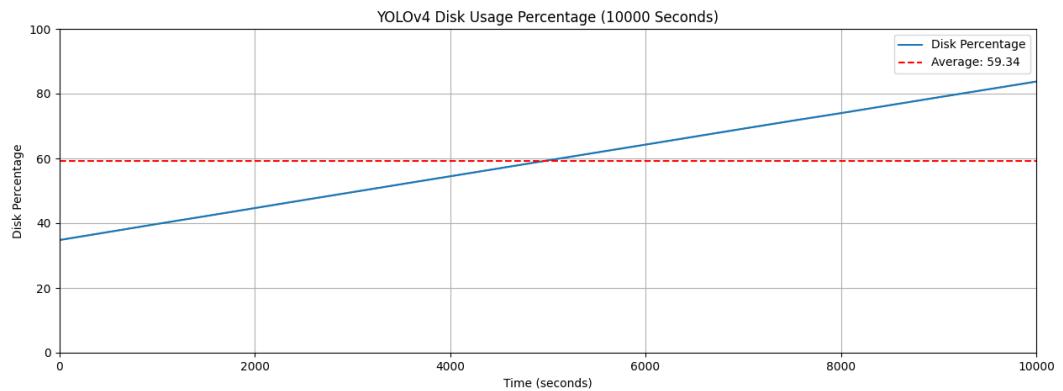
Lampiran 1. Gambar Penggunaan Sumber Daya pada Skenario 1



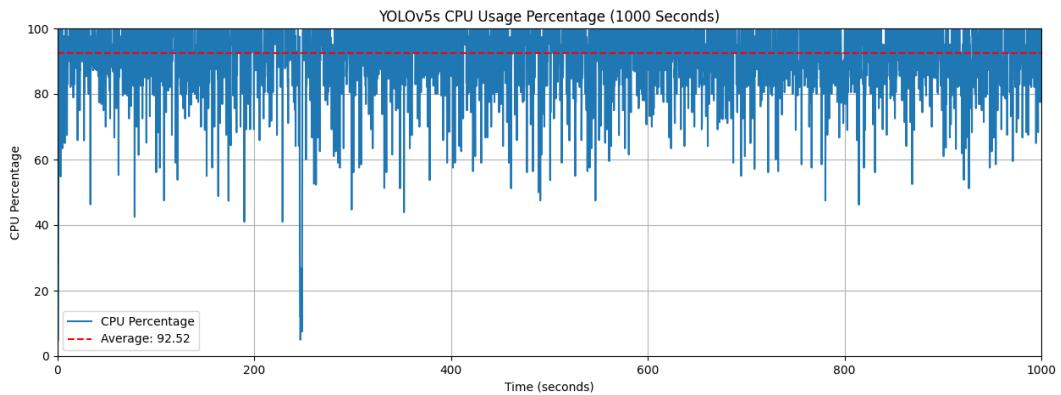
Gambar Persentase Penggunaan CPU YOLOv4 tanpa *Resource-aware Framework*



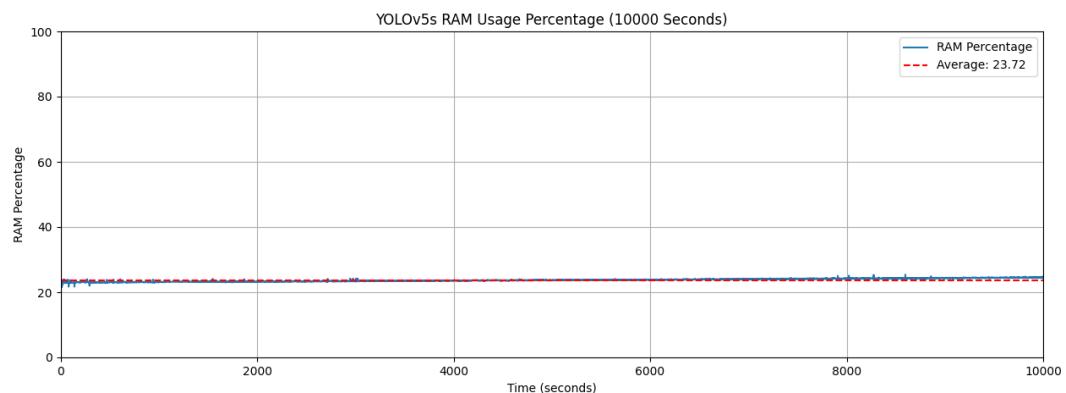
Gambar Persentase Penggunaan RAM YOLOv4 tanpa *Resource-aware Framework*



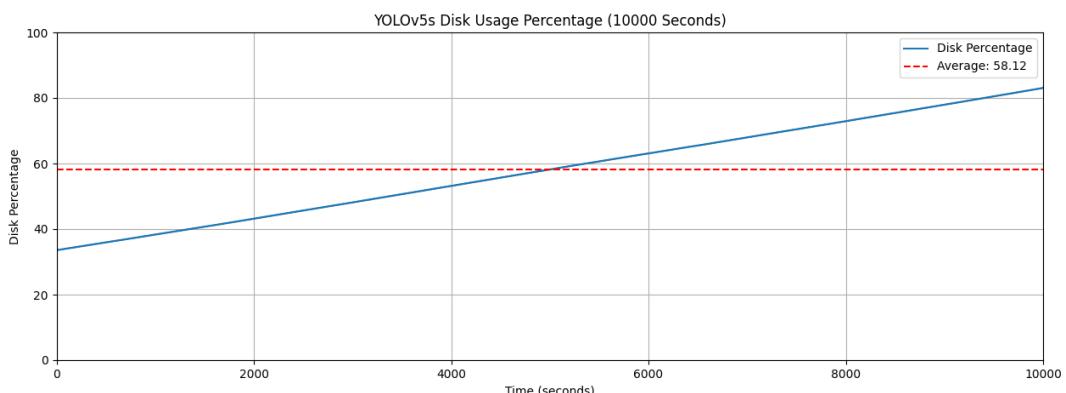
Gambar Persentase Penggunaan Disk YOLOv4 tanpa *Resource-aware Framework*



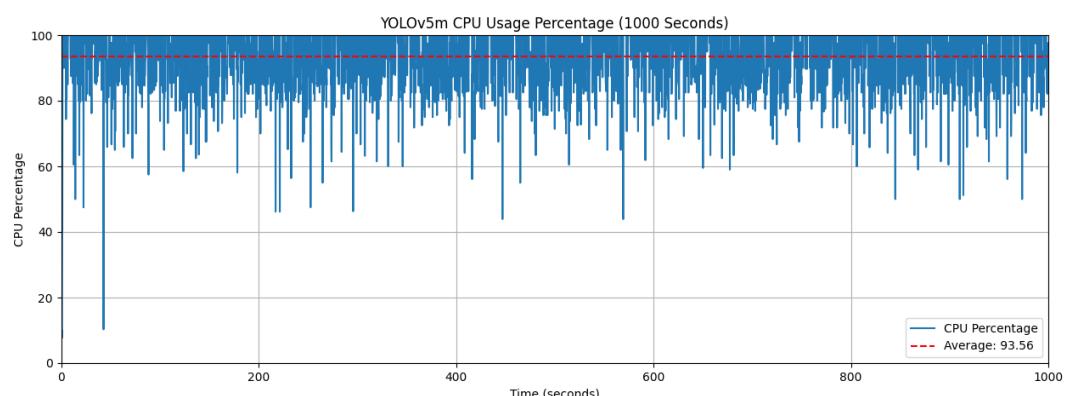
Gambar Persentase Penggunaan CPU YOLOv5s tanpa *Resource-aware Framework*



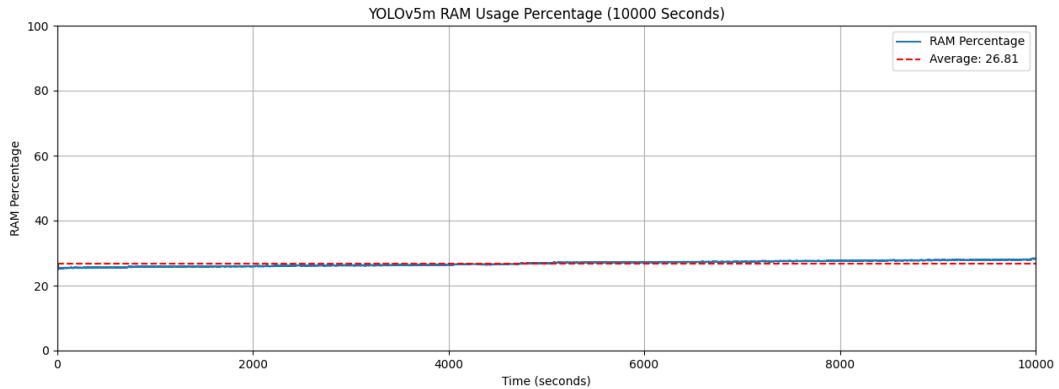
Gambar Persentase Penggunaan RAM YOLOv5s tanpa *Resource-aware Framework*



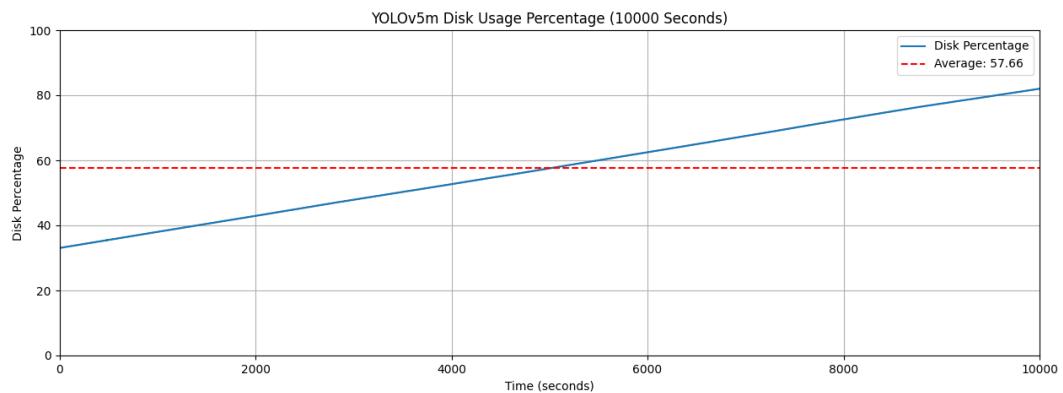
Gambar Persentase Penggunaan Disk YOLOv5s tanpa *Resource-aware Framework*



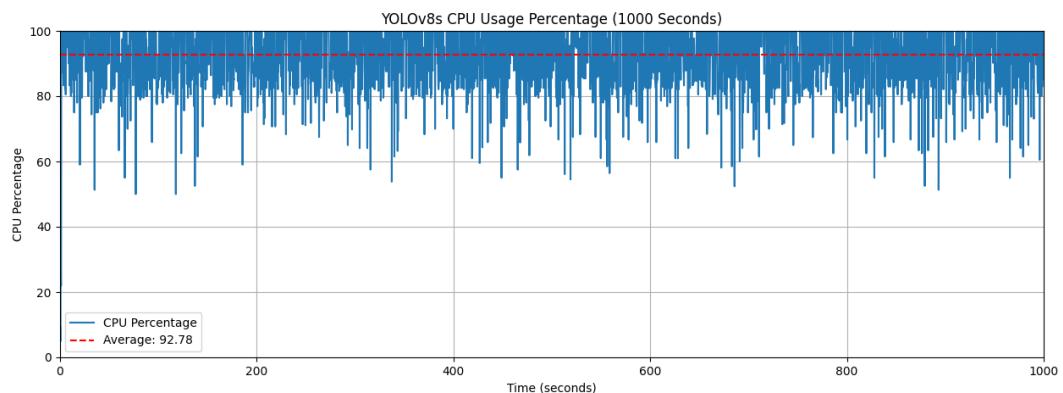
Gambar Persentase Penggunaan CPU YOLOv5m tanpa *Resource-aware Framework*



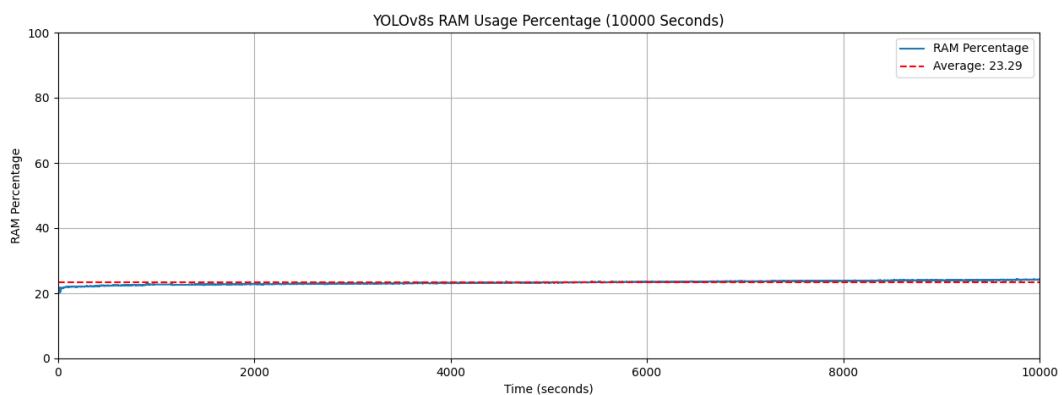
Gambar Persentase Penggunaan RAM YOLOv5m tanpa *Resource-aware Framework*



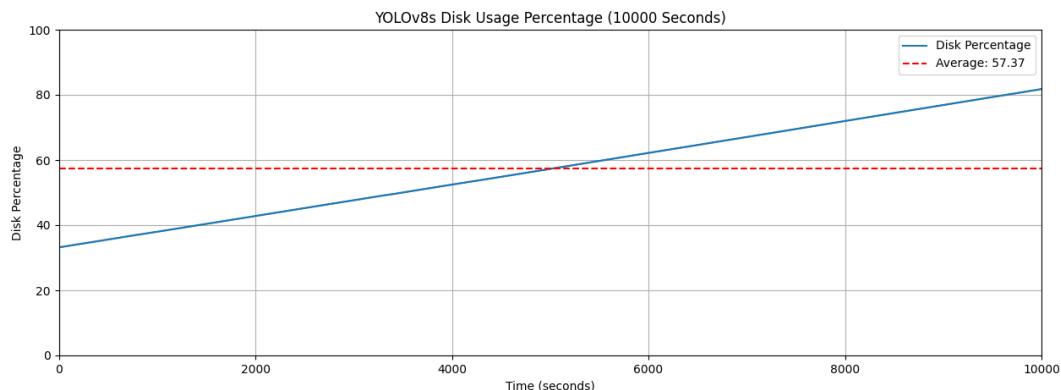
Gambar Persentase Penggunaan Disk YOLOv5m tanpa *Resource-aware Framework*



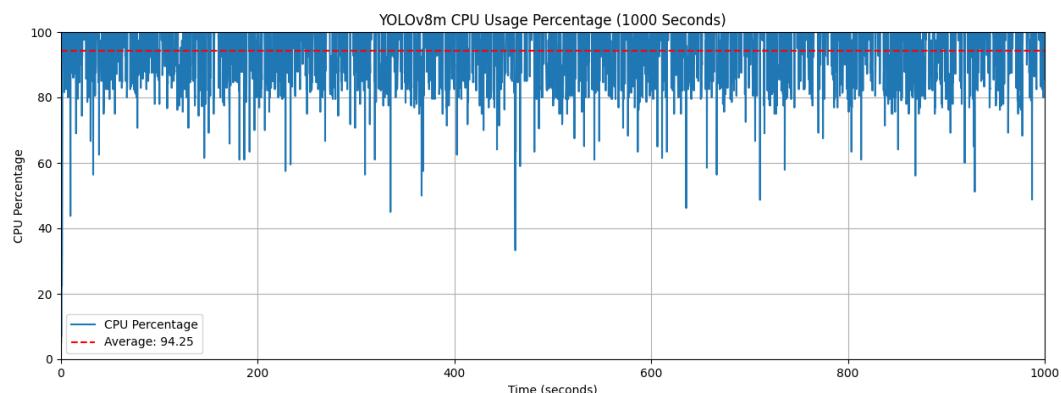
Gambar Persentase Penggunaan CPU YOLOv8s tanpa *Resource-aware Framework*



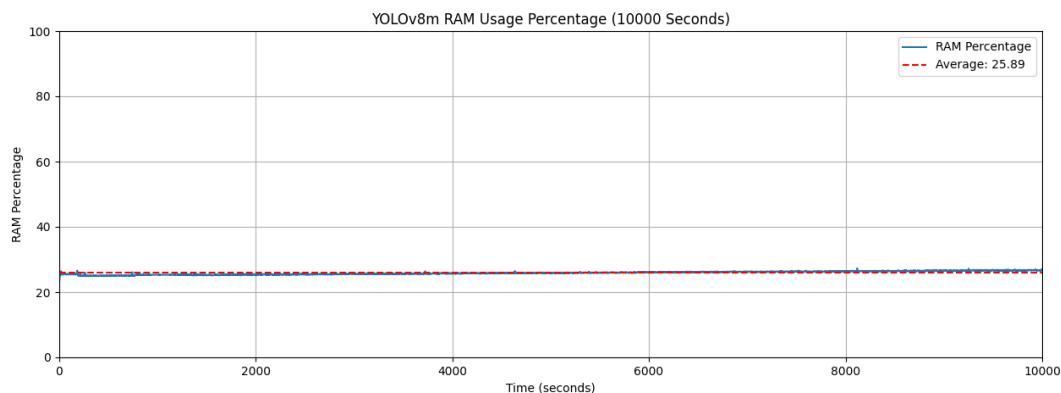
Gambar Persentase Penggunaan RAM YOLOv8s tanpa *Resource-aware Framework*



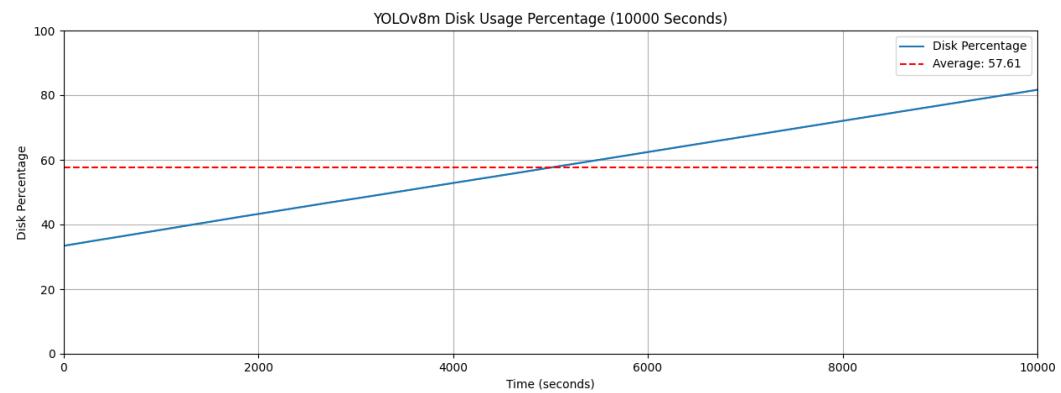
Gambar Persentase Penggunaan Disk YOLOv8s tanpa *Resource-aware Framework*



Gambar Persentase Penggunaan CPU YOLOv8m tanpa *Resource-aware Framework*

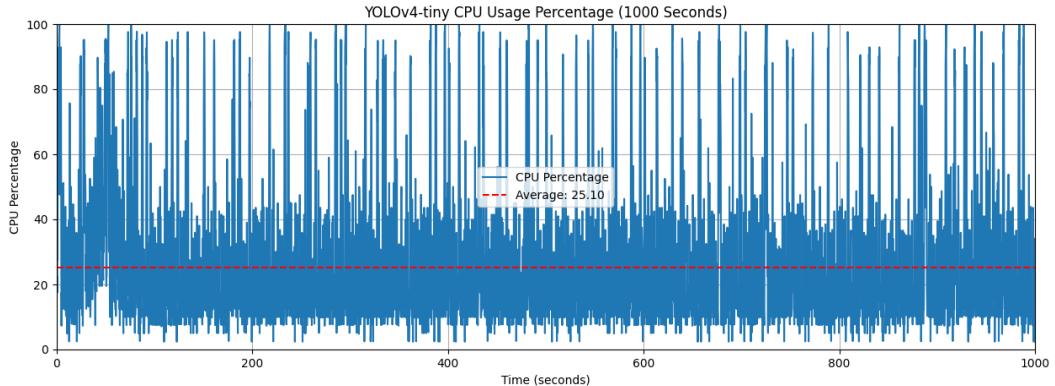


Gambar Persentase Penggunaan RAM YOLOv8m tanpa *Resource-aware Framework*

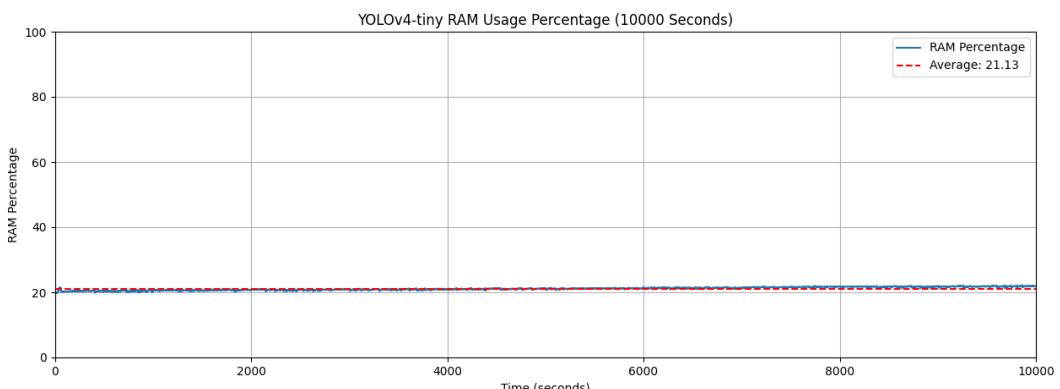


Gambar Persentase Penggunaan Disk YOLOv8m tanpa *Resource-aware Framework*

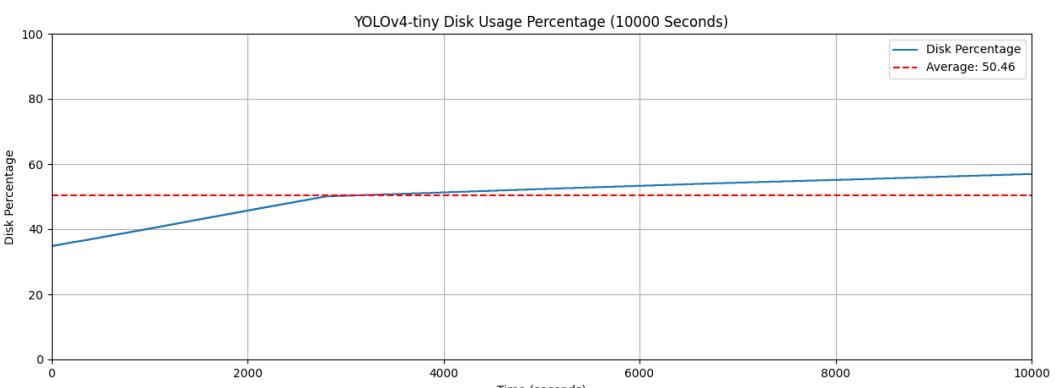
## Lampiran 2. Gambar Penggunaan Sumber Daya pada Skenario 2



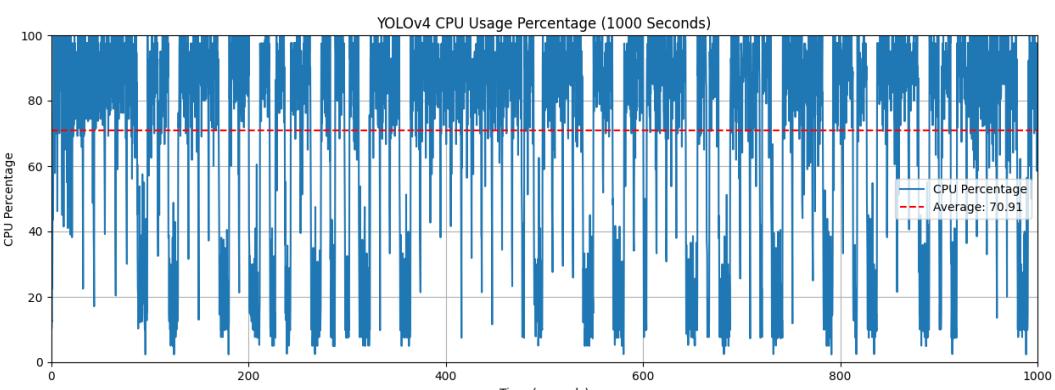
Gambar Persentase Penggunaan CPU YOLOv4-tiny dengan *Resource-aware Framework*



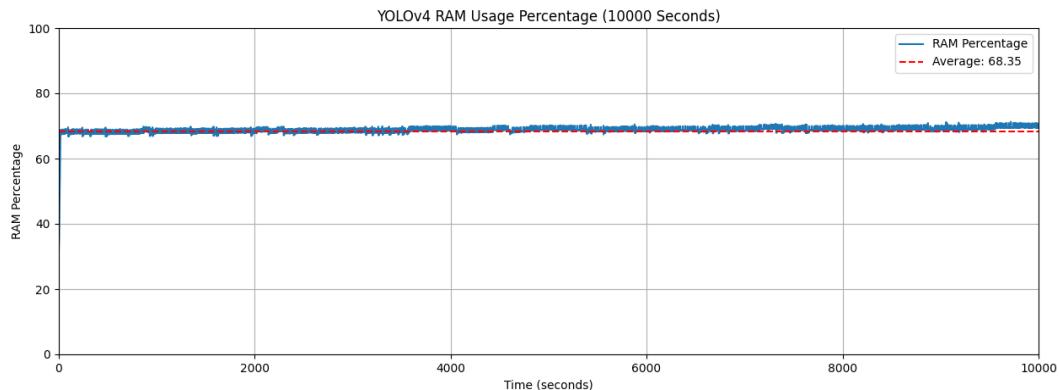
Gambar Persentase Penggunaan RAM YOLOv4-tiny dengan *Resource-aware Framework*



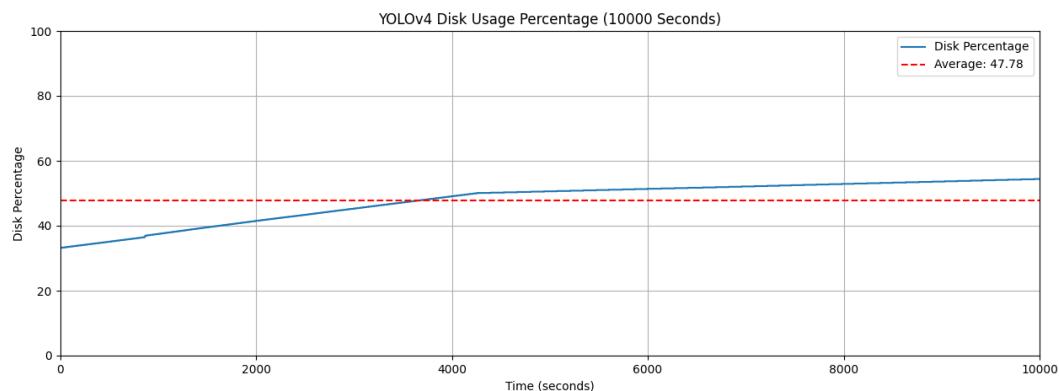
Gambar Persentase Penggunaan Disk YOLOv4-tiny dengan *Resource-aware Framework*



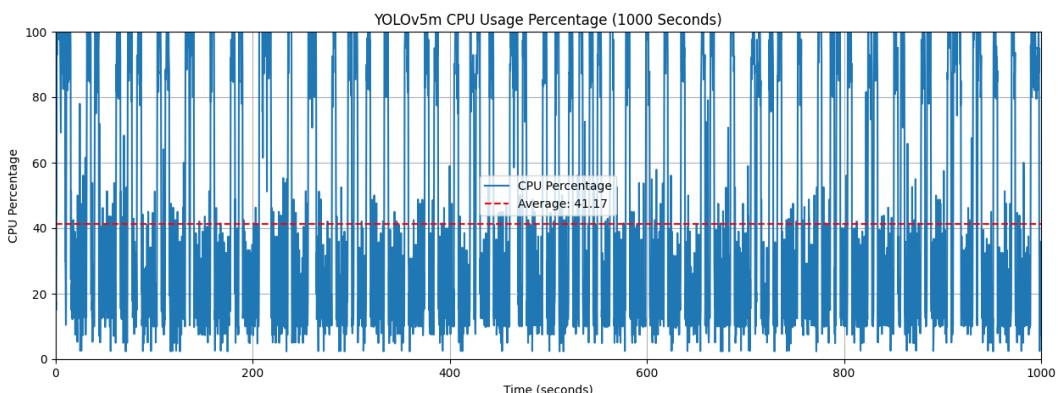
Gambar Persentase Penggunaan CPU YOLOv4 dengan *Resource-aware Framework*



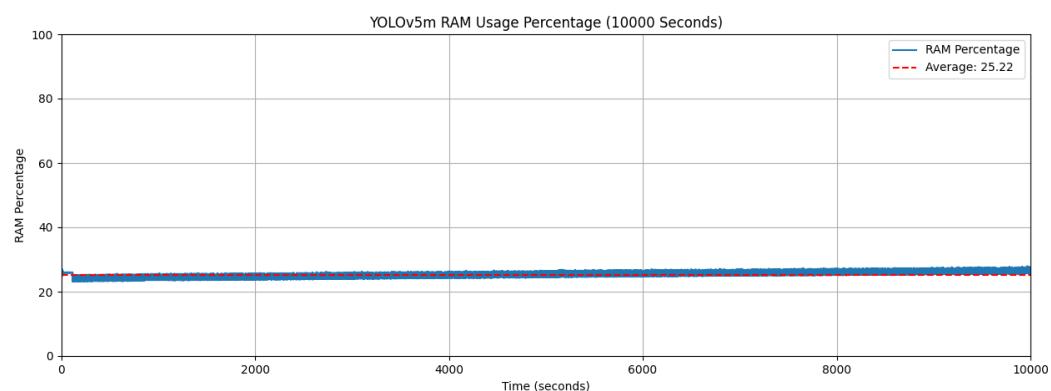
Gambar Persentase Penggunaan RAM YOLOv4 dengan *Resource-aware Framework*



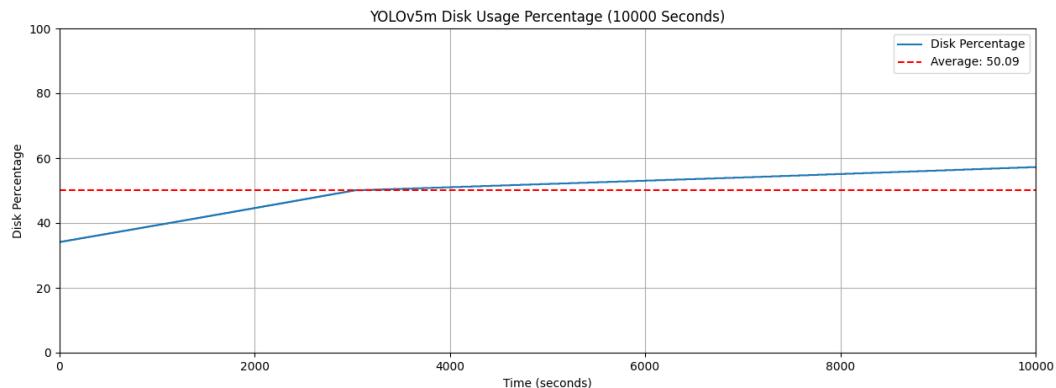
Gambar Persentase Penggunaan Disk YOLOv4 dengan *Resource-aware Framework*



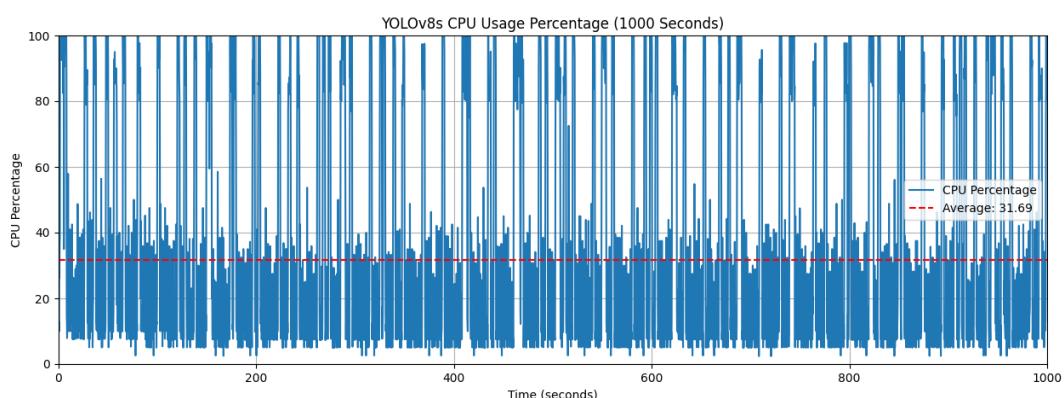
Gambar Persentase Penggunaan CPU YOLOv5m dengan *Resource-aware Framework*



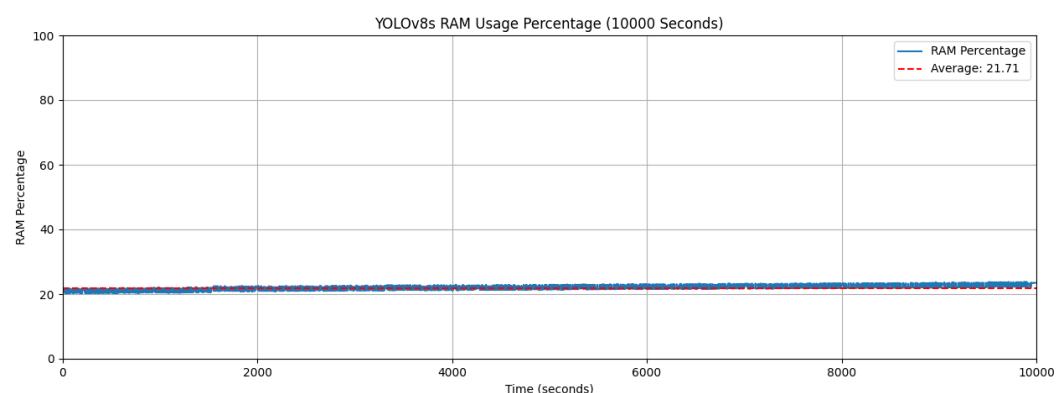
Gambar Persentase Penggunaan RAM YOLOv5m dengan *Resource-aware Framework*



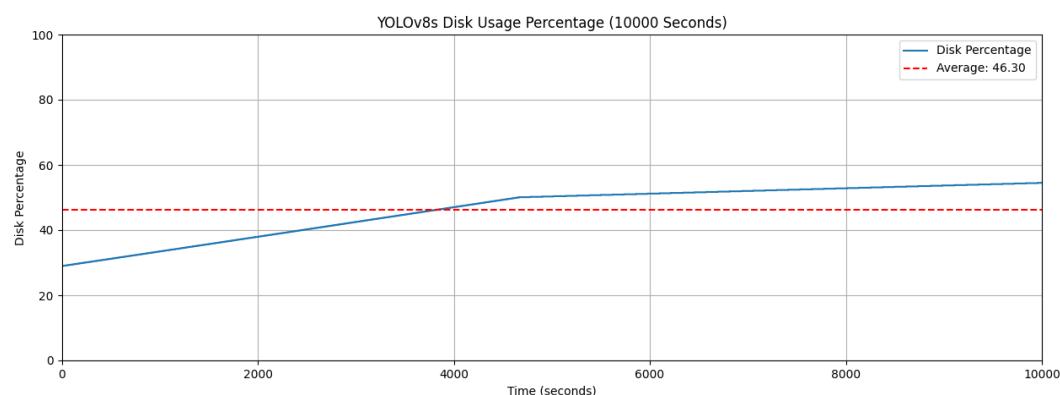
Gambar Persentase Penggunaan Disk YOLOv5s dengan *Resource-aware Framework*



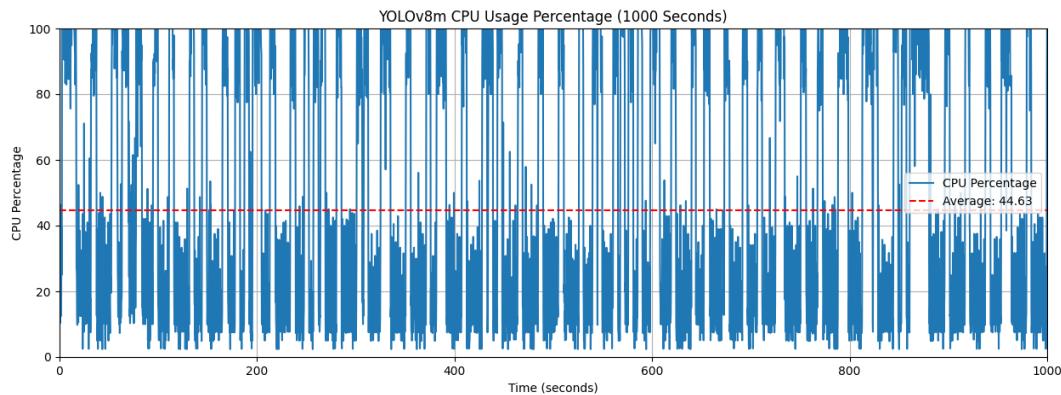
Gambar Persentase Penggunaan CPU YOLOv8s dengan *Resource-aware Framework*



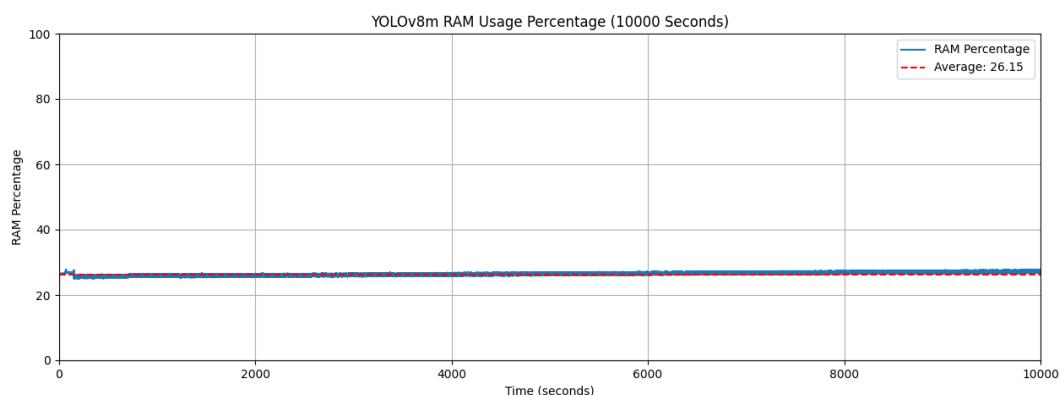
Gambar Persentase Penggunaan RAM YOLOv8s dengan *Resource-aware Framework*



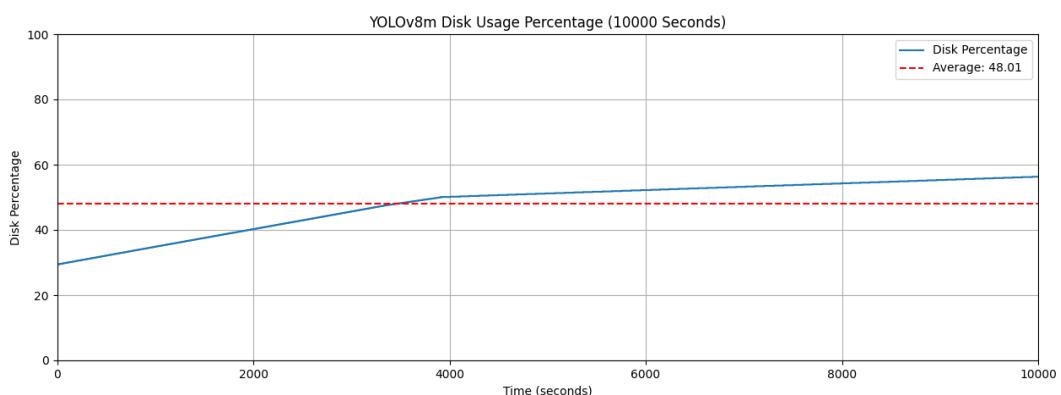
Gambar Persentase Penggunaan Disk YOLOv8s dengan *Resource-aware Framework*



Gambar Persentase Penggunaan CPU YOLOv8m dengan *Resource-aware Framework*

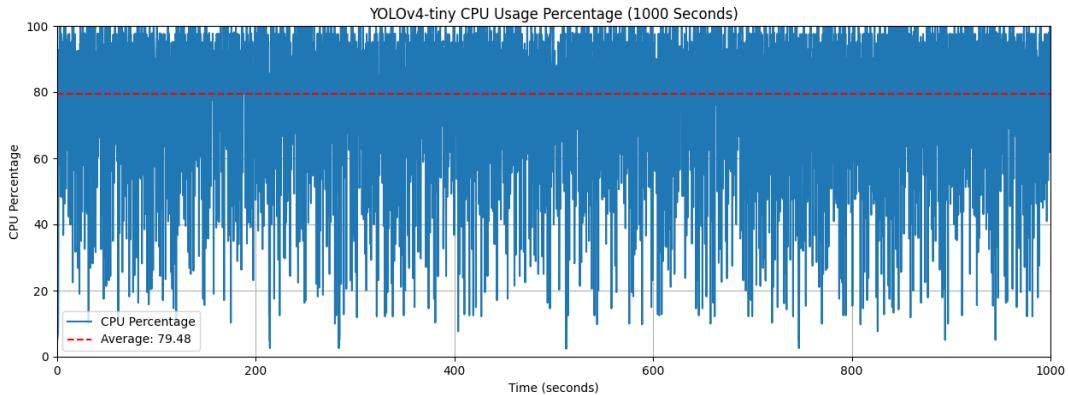


Gambar Persentase Penggunaan RAM YOLOv8m dengan *Resource-aware Framework*

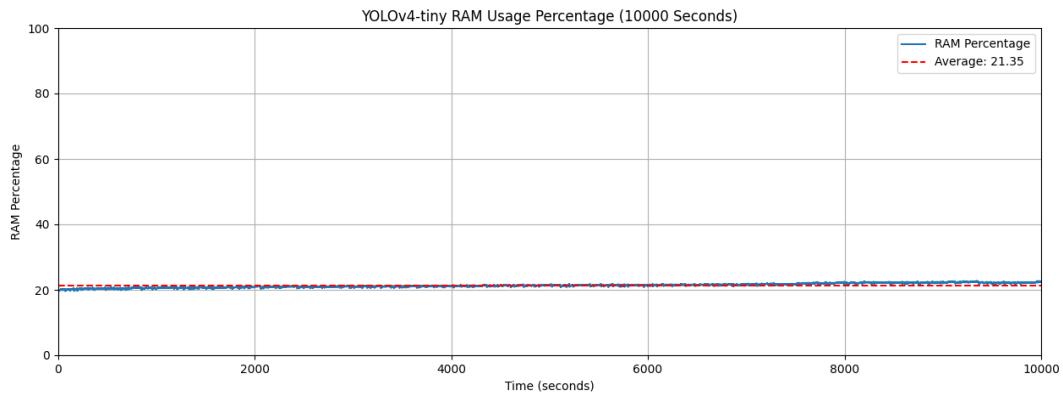


Gambar Persentase Penggunaan Disk YOLOv8m dengan *Resource-aware Framework*

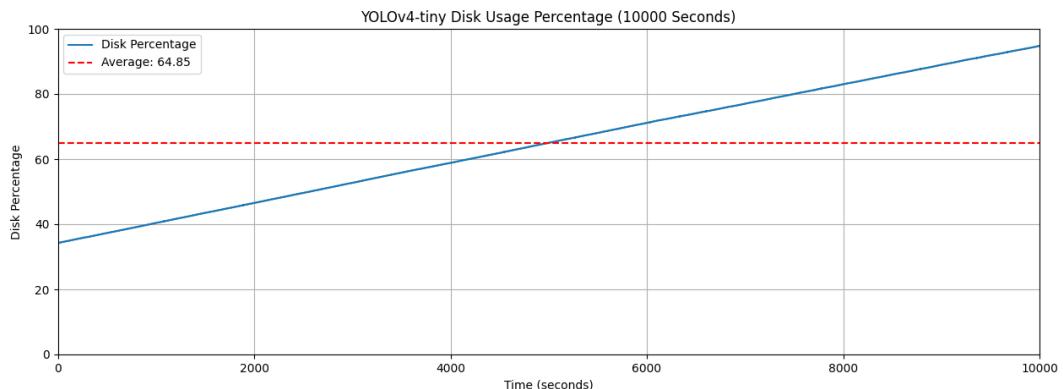
### Lampiran 3. Gambar Penggunaan Sumber Daya pada Skenario 3



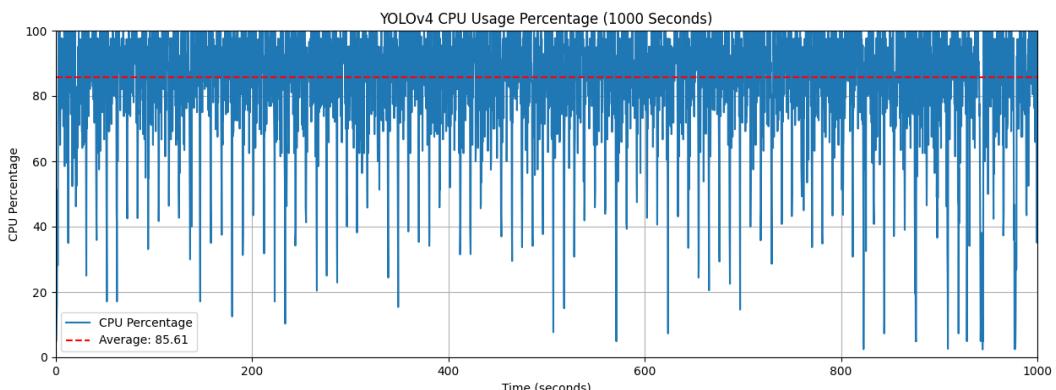
Gambar Persentase Penggunaan CPU YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dalam Metode *Schedule*



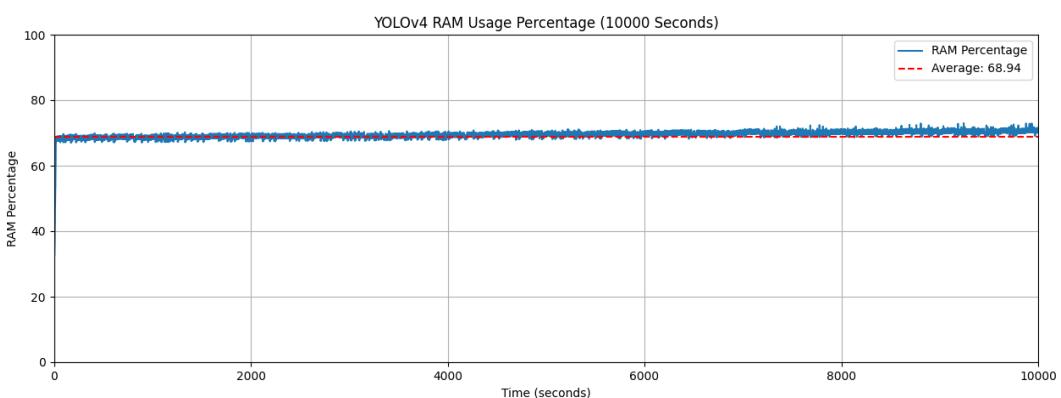
Gambar Persentase Penggunaan RAM YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



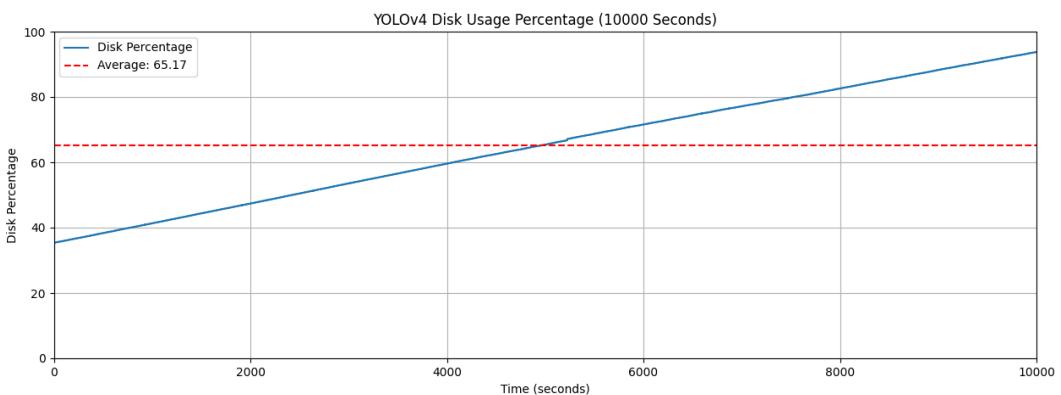
Gambar Persentase Penggunaan disk YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



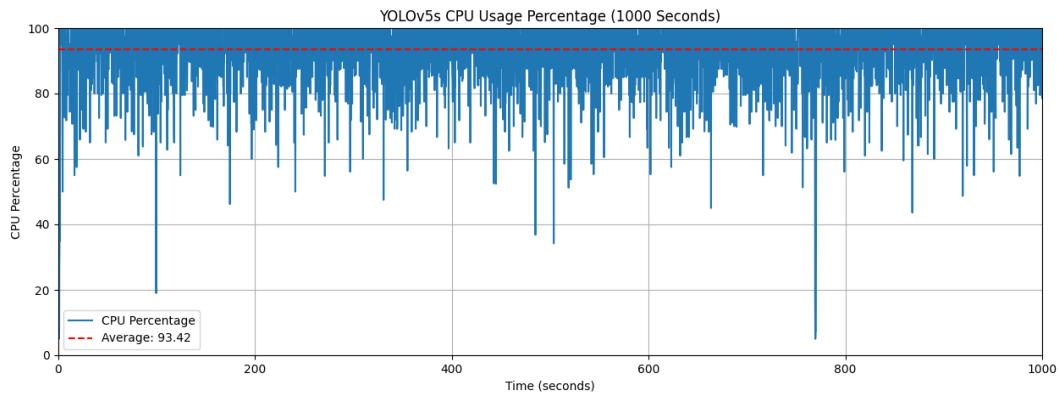
Gambar Persentase Penggunaan CPU YOLOv4 tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



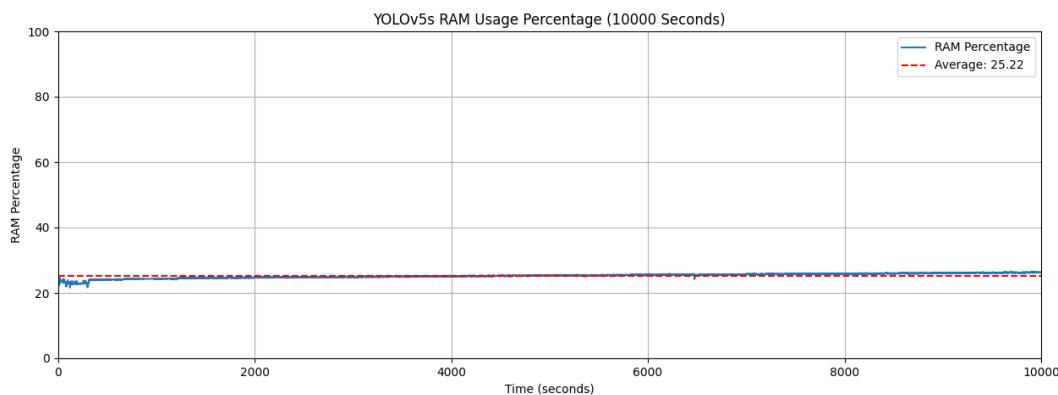
Gambar Persentase Penggunaan RAM YOLOv4 tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



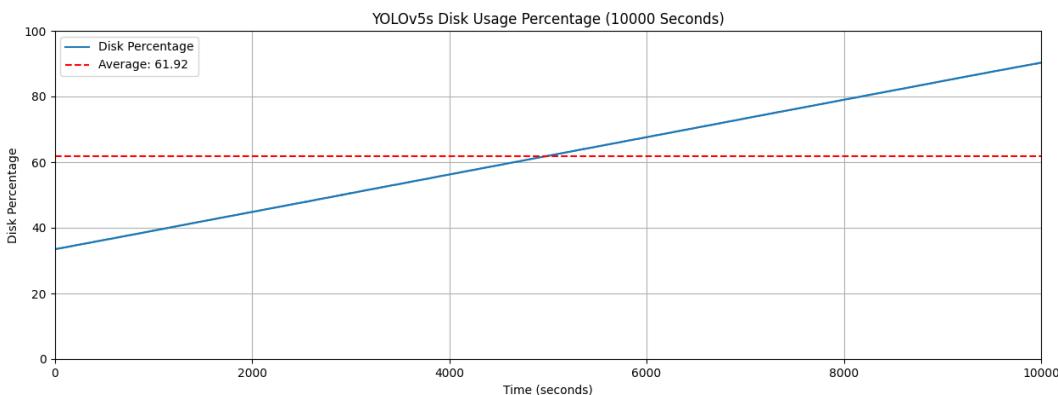
Gambar Persentase Penggunaan disk YOLOv4 tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



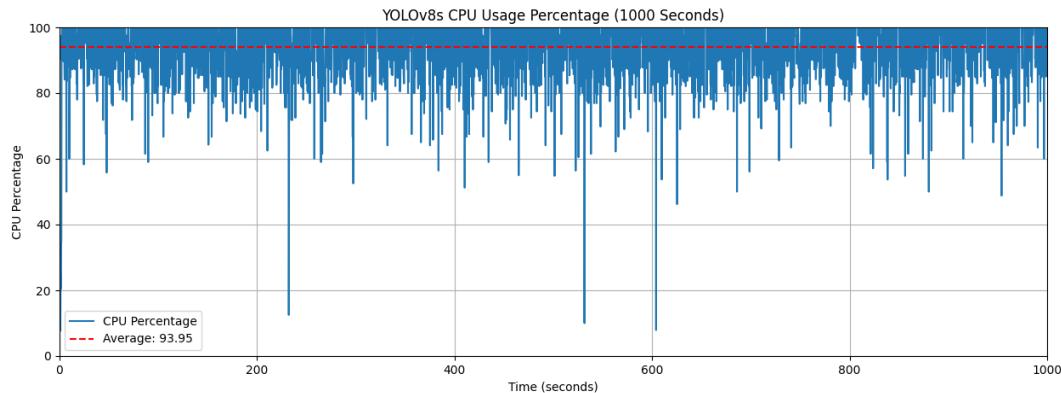
Gambar Persentase Penggunaan CPU YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



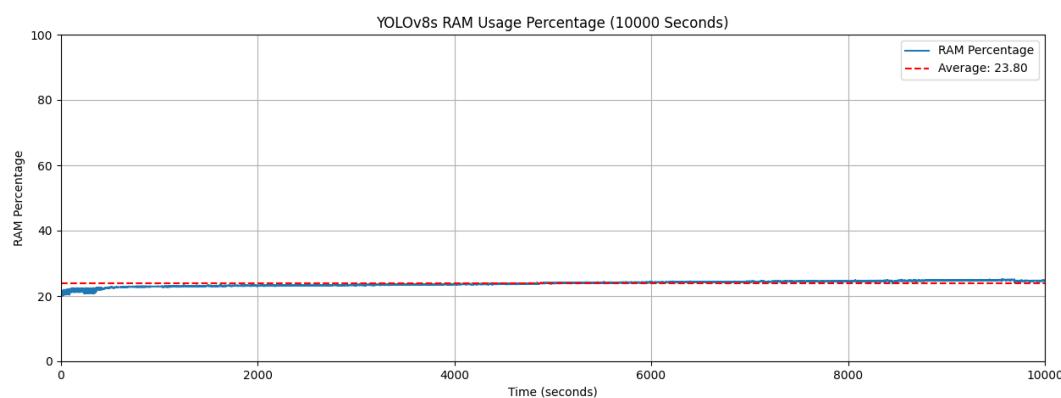
Gambar Persentase Penggunaan RAM YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



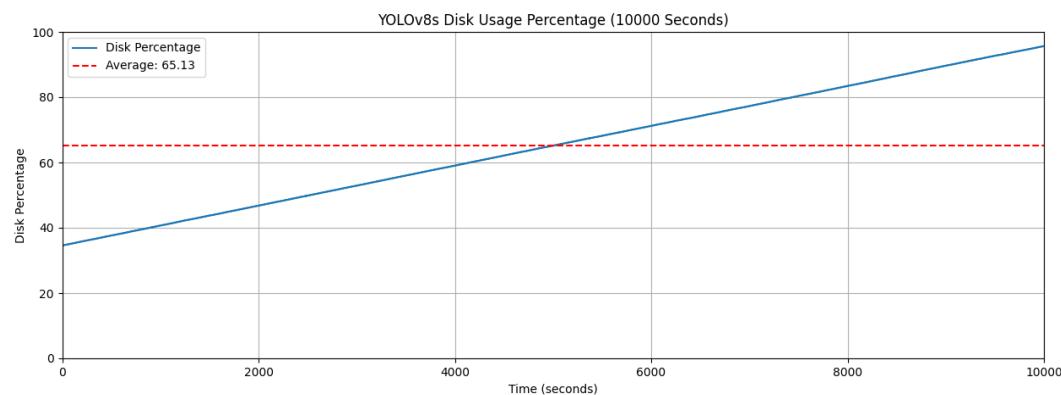
Gambar Persentase Penggunaan Disk YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



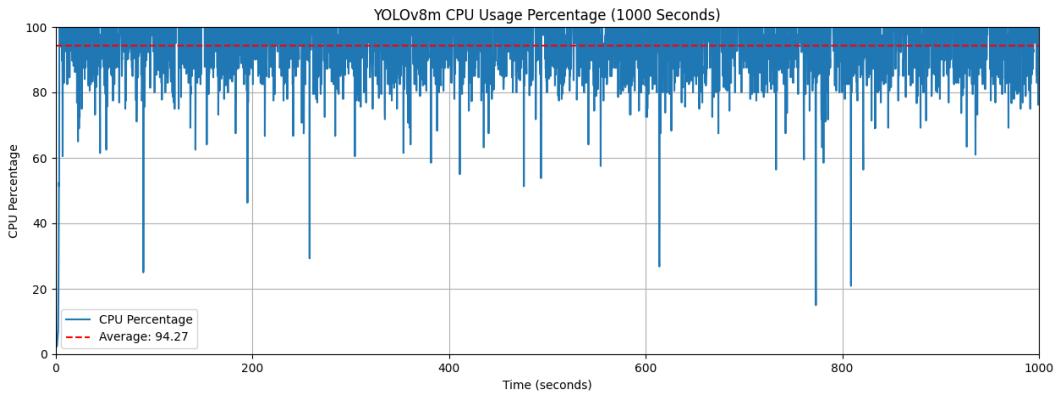
Gambar Persentase Penggunaan CPU YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



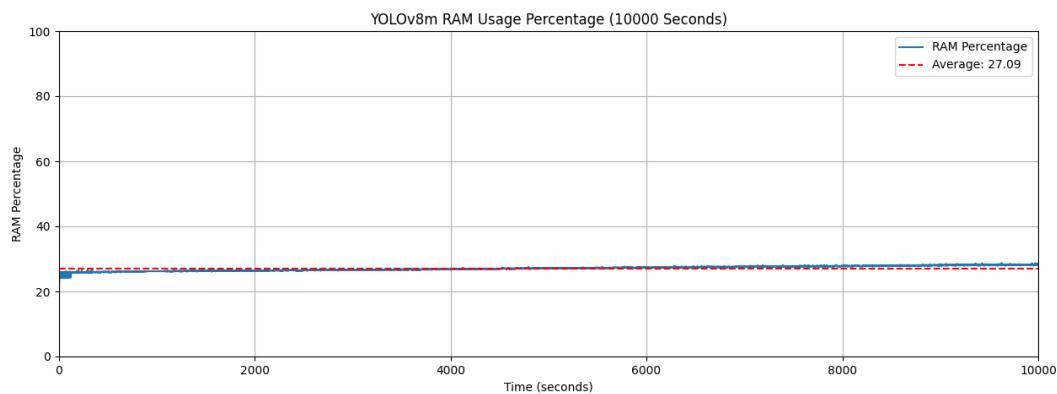
Gambar Persentase Penggunaan RAM YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



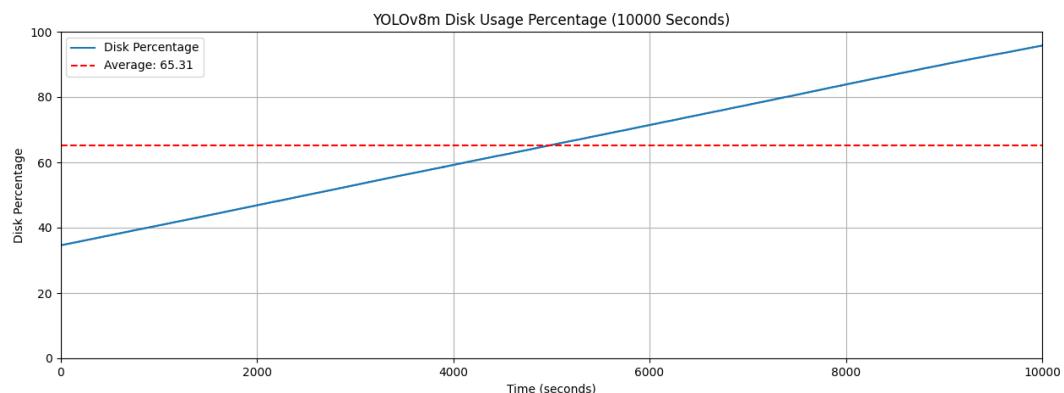
Gambar Persentase Penggunaan Disk YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



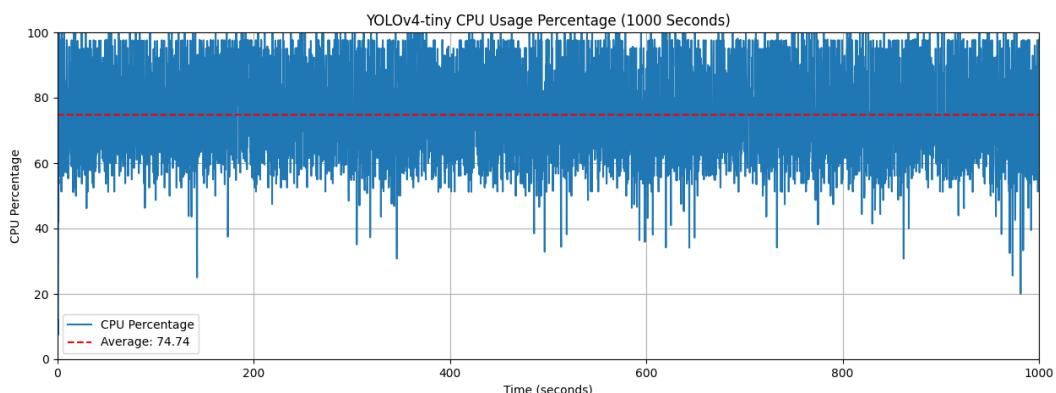
Gambar Persentase Penggunaan CPU YOLOv8m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



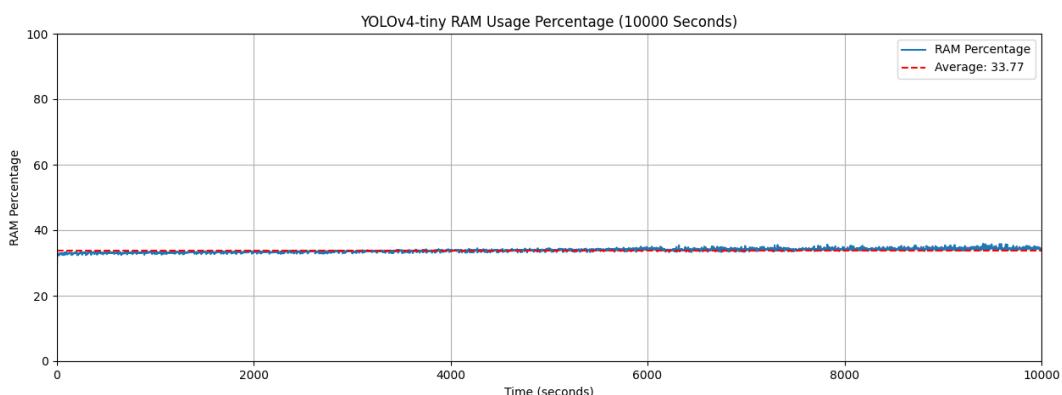
Gambar Persentase Penggunaan RAM YOLOv8m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



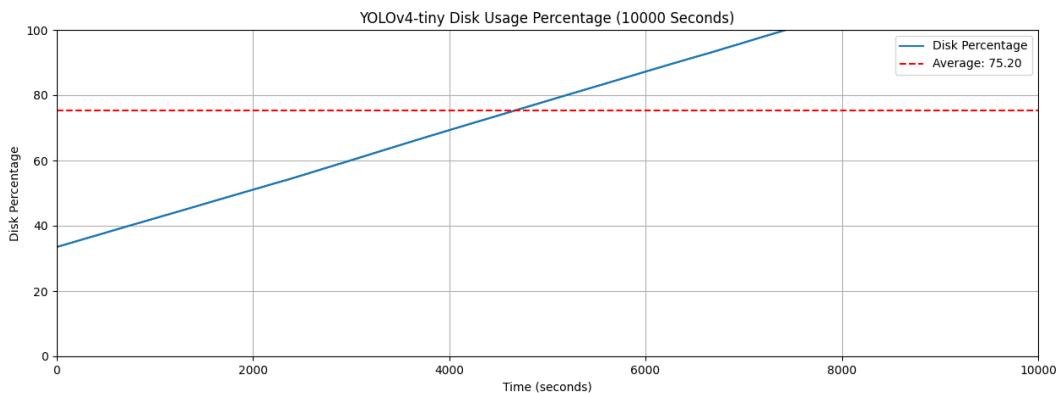
Gambar Persentase Penggunaan Disk YOLOv8m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



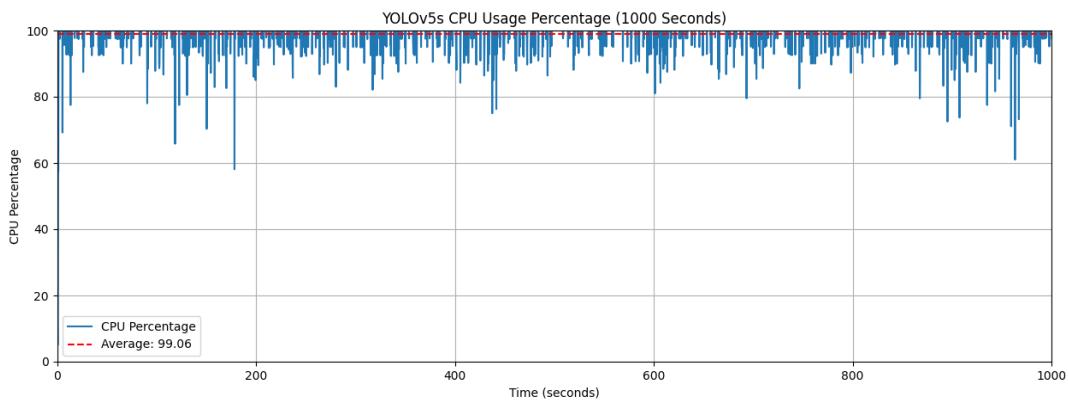
Gambar Persentase Penggunaan CPU YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



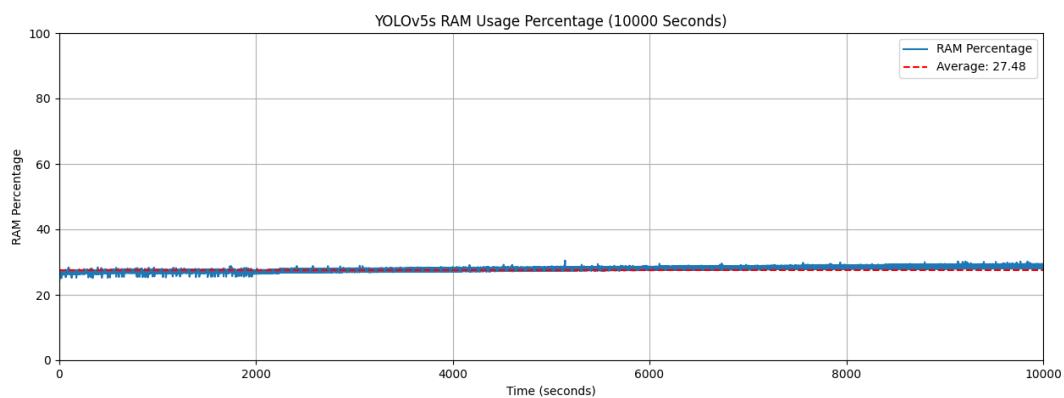
Gambar Persentase Penggunaan RAM YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



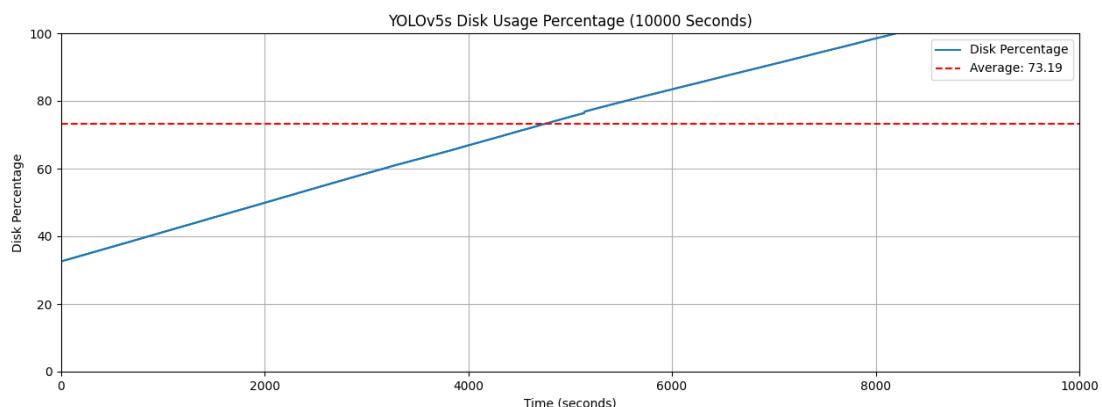
Gambar Persentase Penggunaan Disk YOLOv4-tiny tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



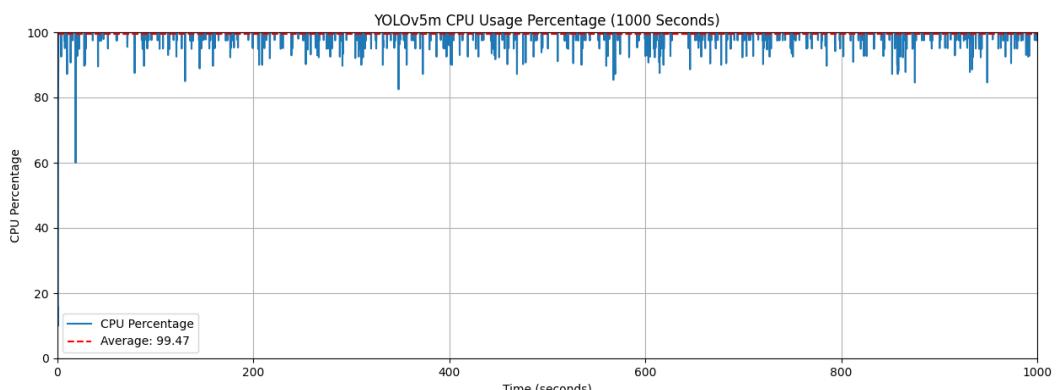
Gambar Persentase Penggunaan CPU YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



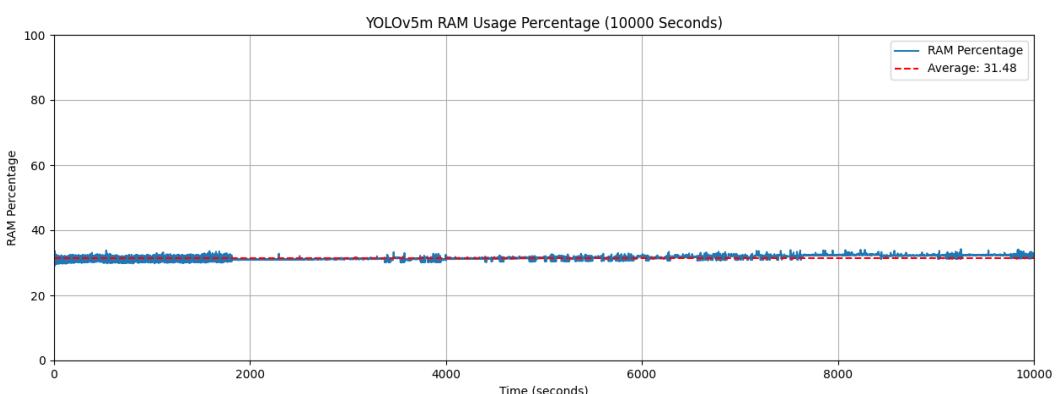
Gambar Persentase Penggunaan RAM YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



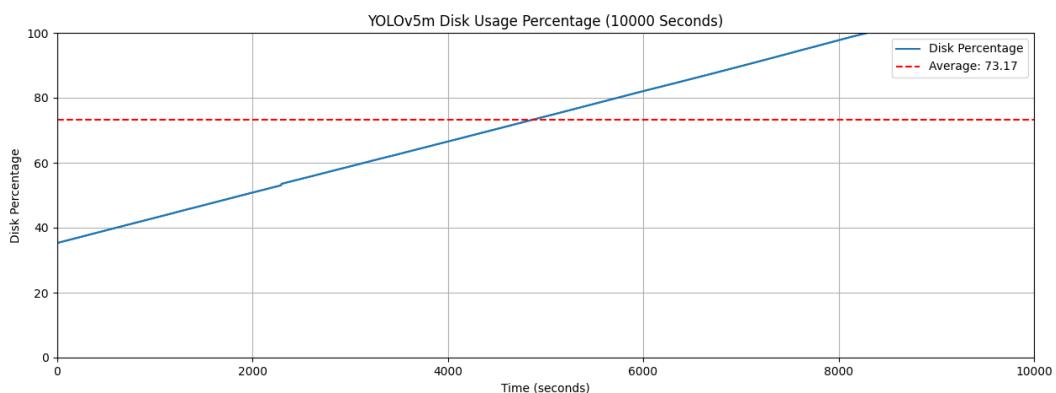
Gambar Persentase Penggunaan Disk YOLOv5s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



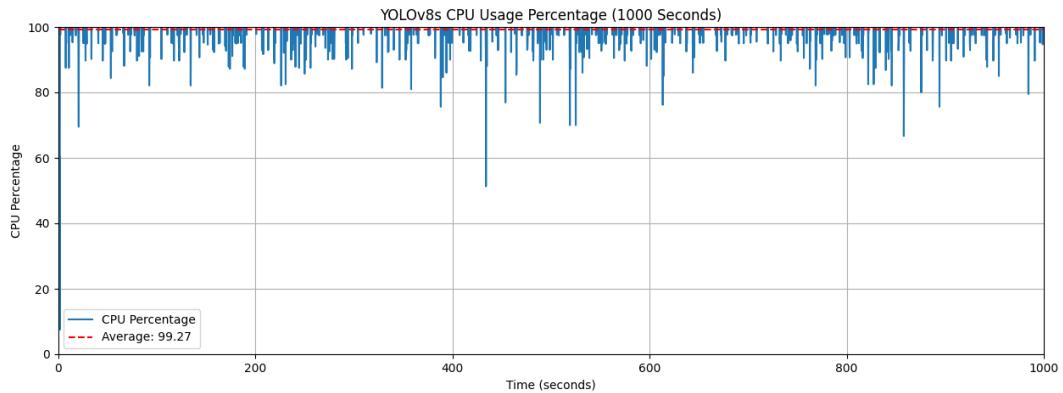
Gambar Persentase Penggunaan CPU YOLOv5m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



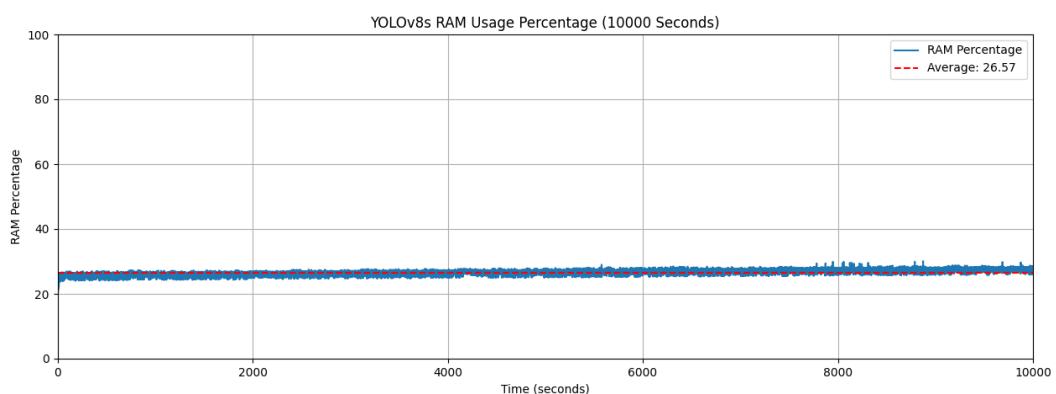
Gambar Persentase Penggunaan RAM YOLOv5m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



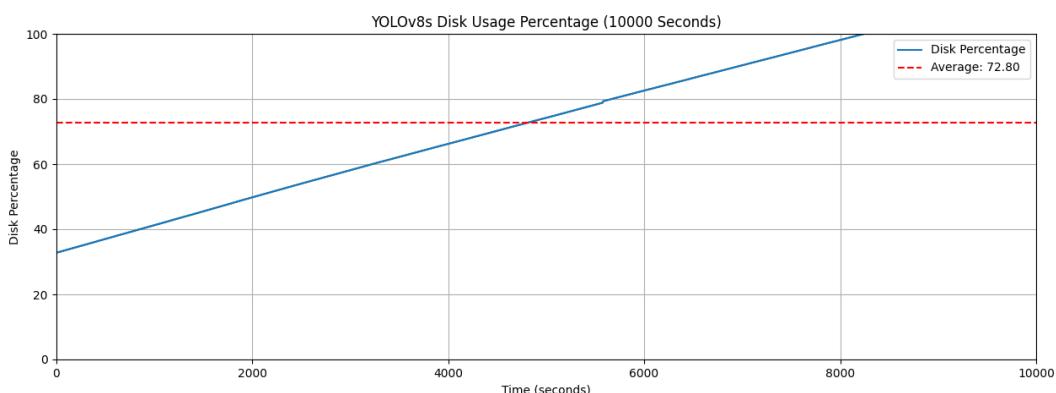
Gambar Persentase Penggunaan Disk YOLOv5m tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



Gambar Persentase Penggunaan CPU YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel

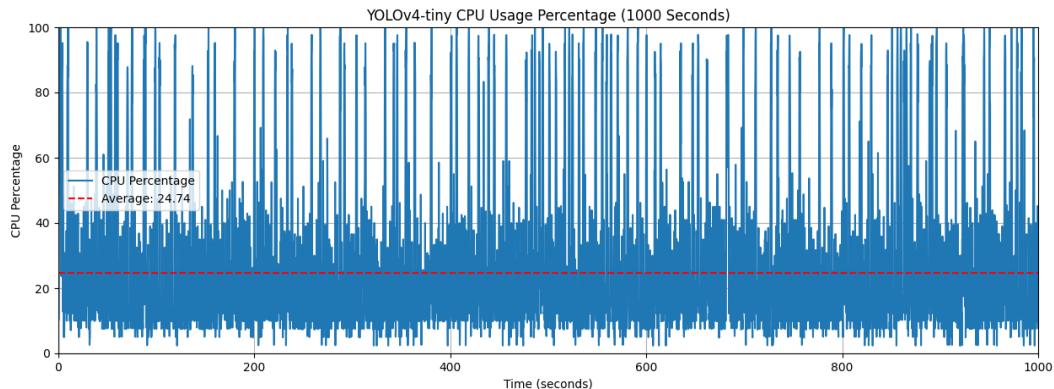


Gambar Persentase Penggunaan RAM YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel

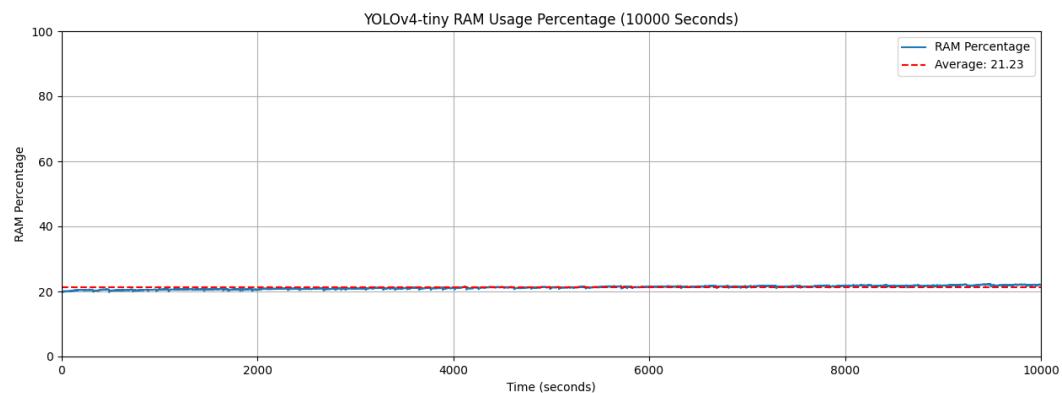


Gambar Persentase Penggunaan Disk YOLOv8s tanpa *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel

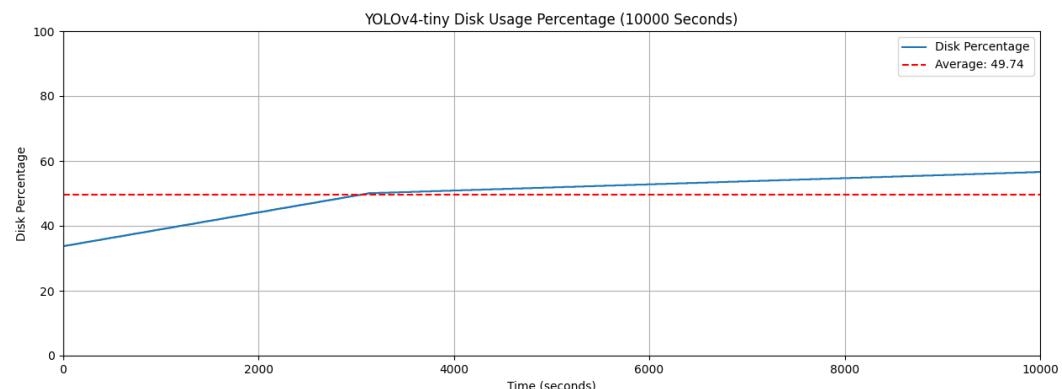
#### Lampiran 4. Gambar Penggunaan Sumber Daya pada Skenario 4



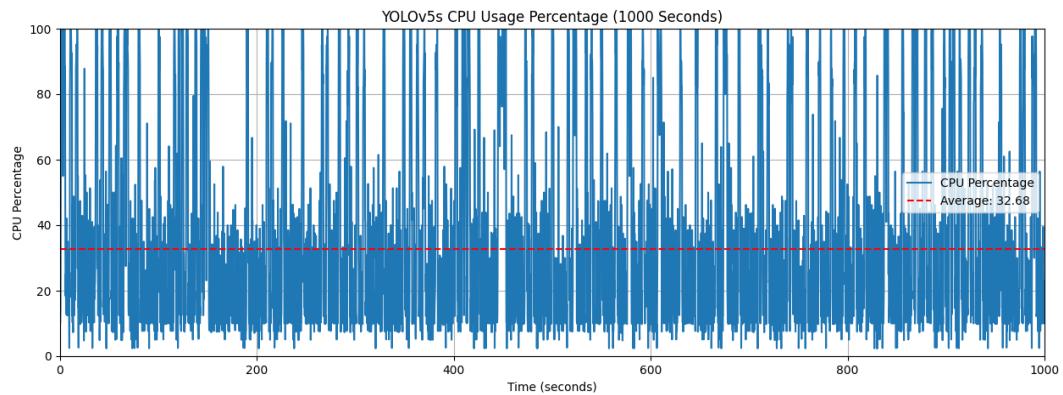
Gambar Persentase Penggunaan CPU YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



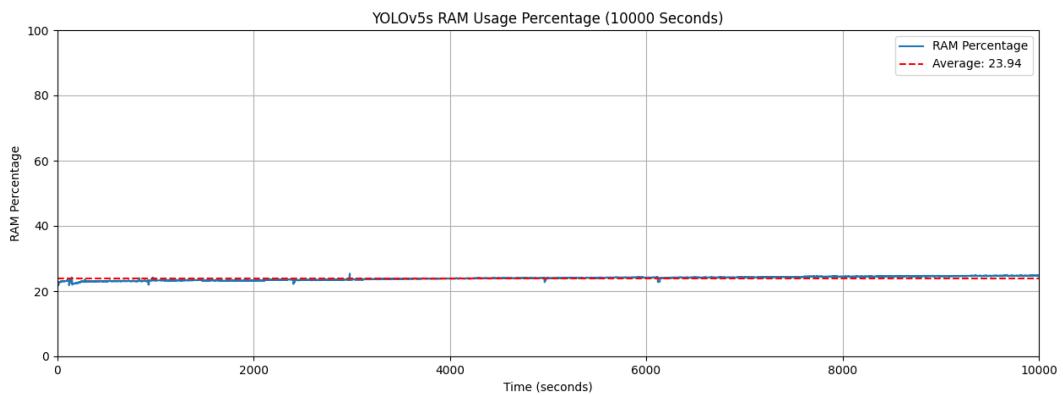
Gambar Persentase Penggunaan RAM YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



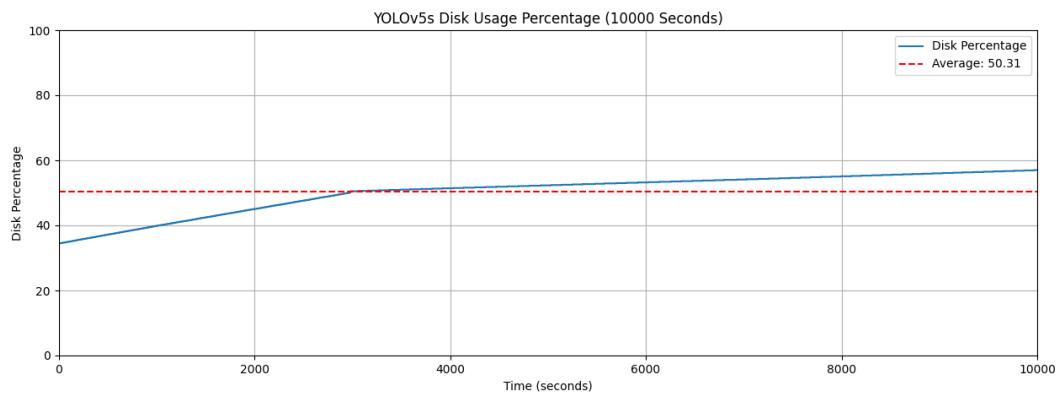
Gambar Persentase Penggunaan Disk YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



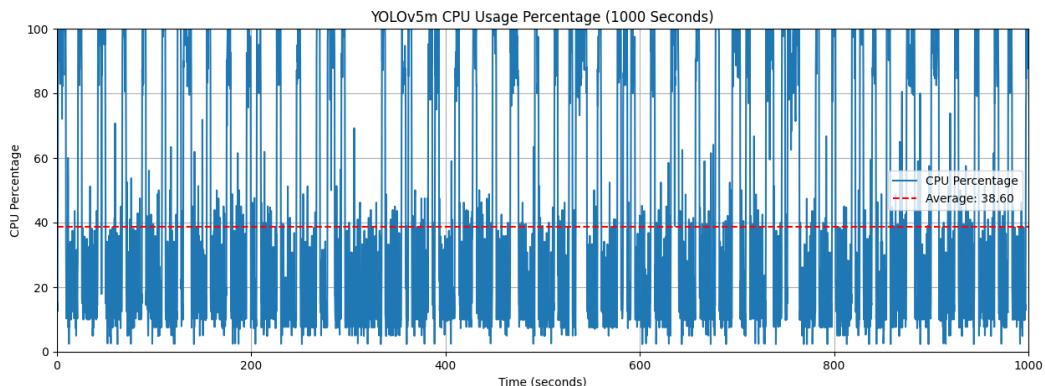
Gambar Persentase Penggunaan CPU YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



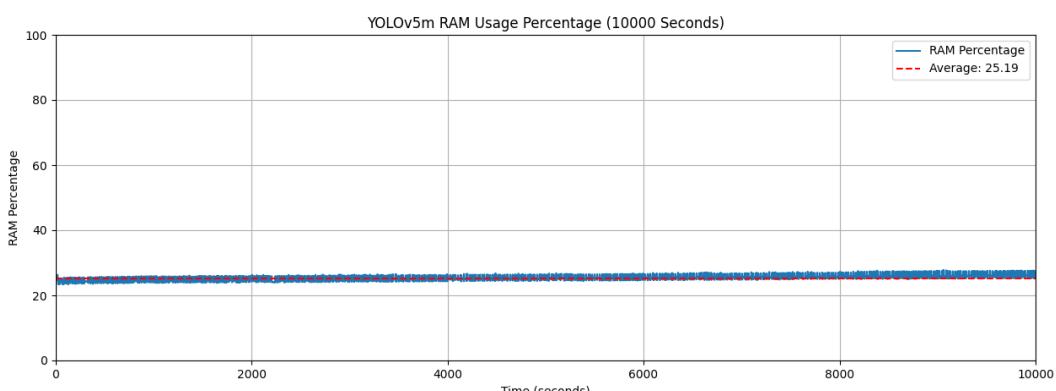
Gambar Persentase Penggunaan RAM YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



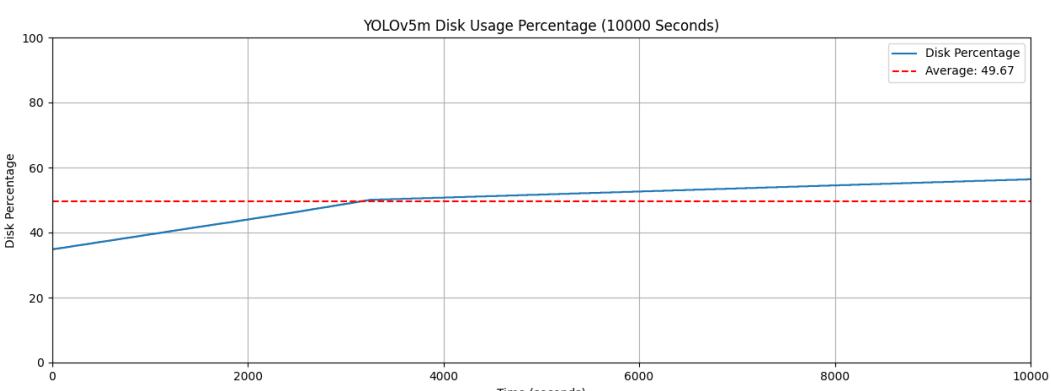
Gambar Persentase Penggunaan Disk YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



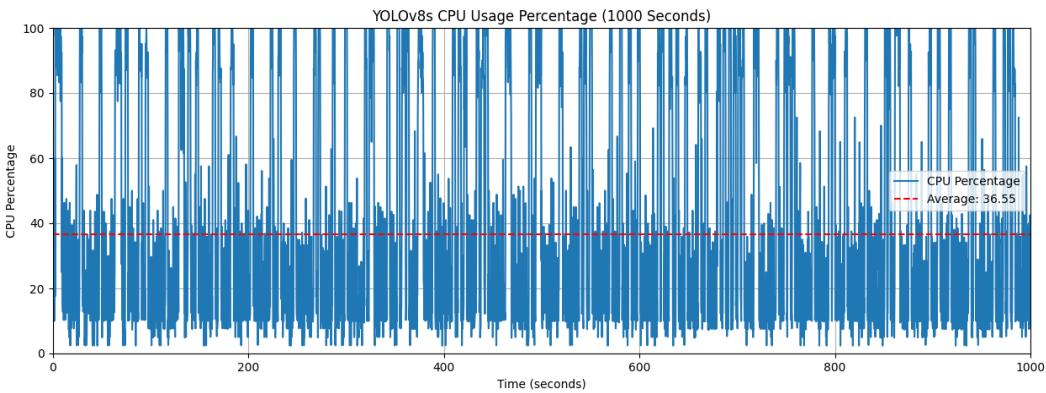
Gambar Persentase Penggunaan CPU YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



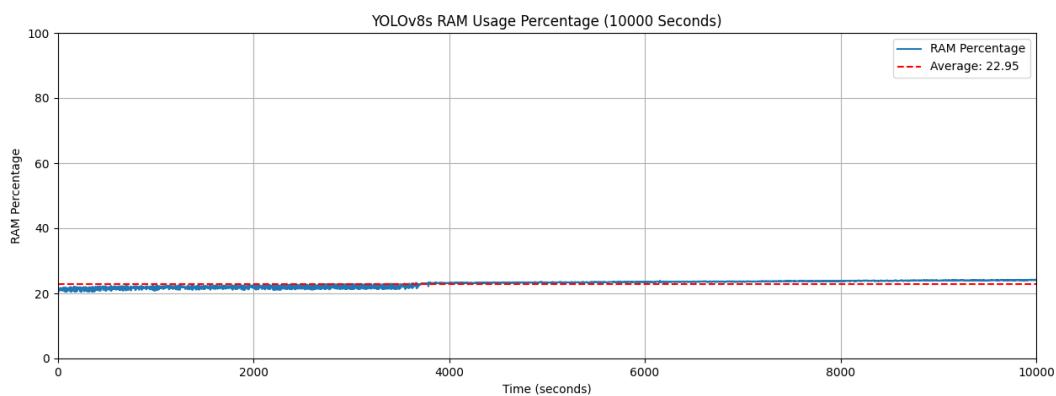
Gambar Persentase Penggunaan RAM YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



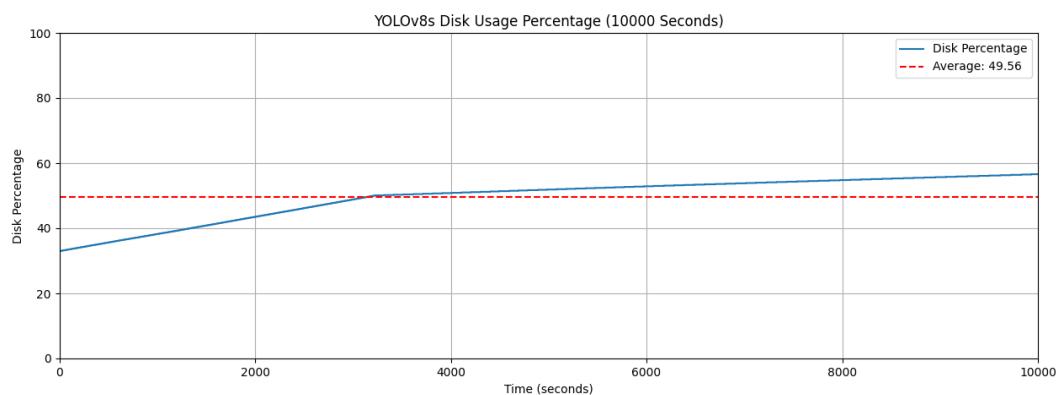
Gambar Persentase Penggunaan Disk YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



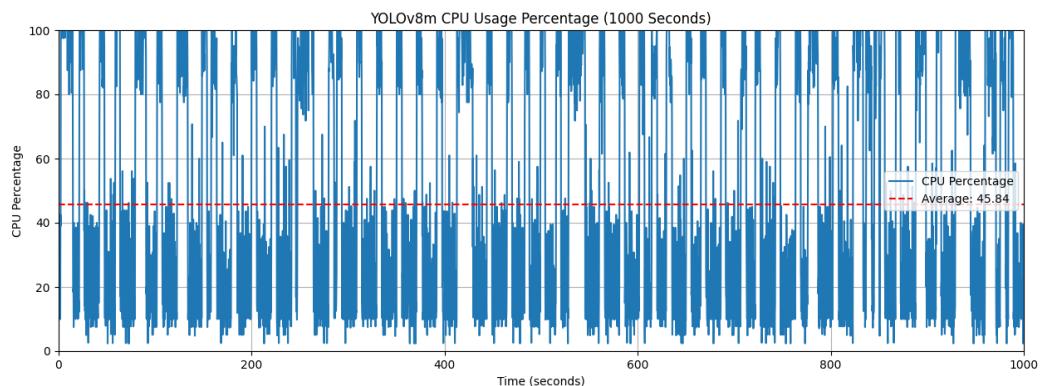
Gambar Persentase Penggunaan CPU YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



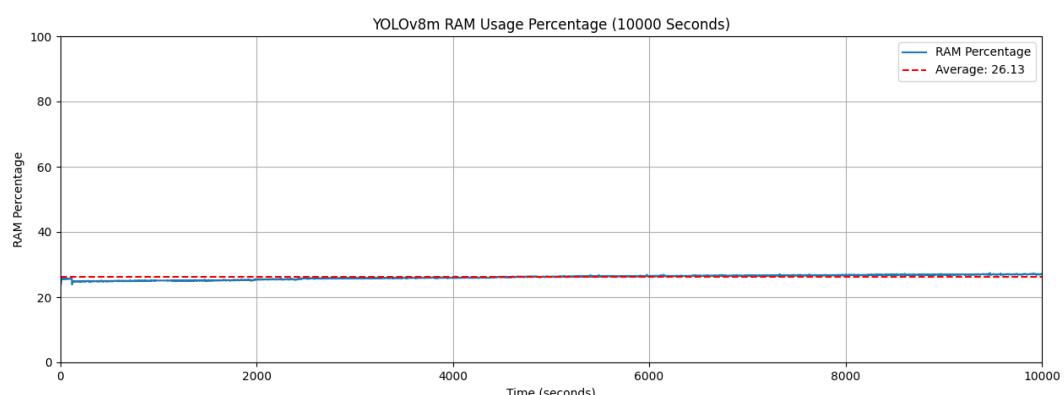
Gambar Persentase Penggunaan RAM YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



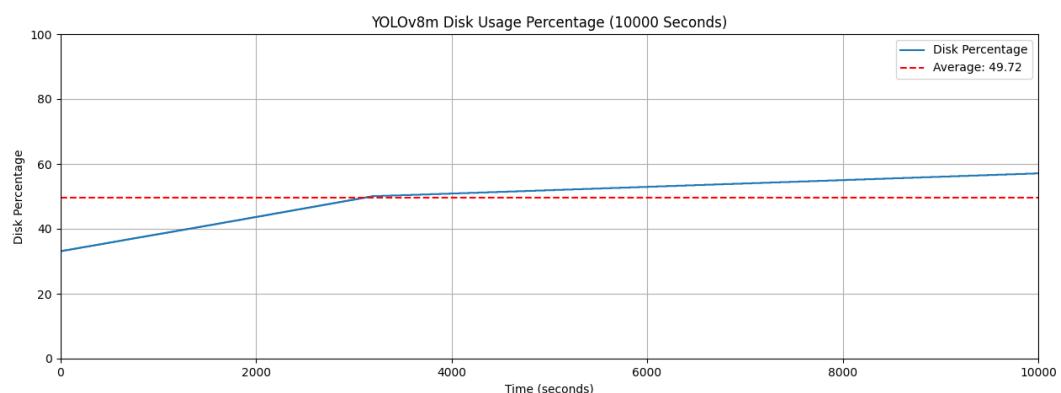
Gambar Persentase Penggunaan Disk YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



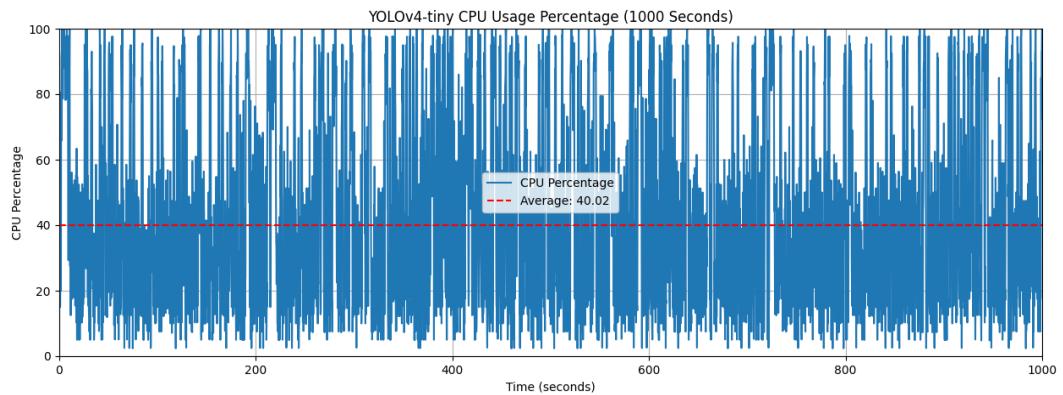
Gambar Persentase Penggunaan CPU YOLOv8m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



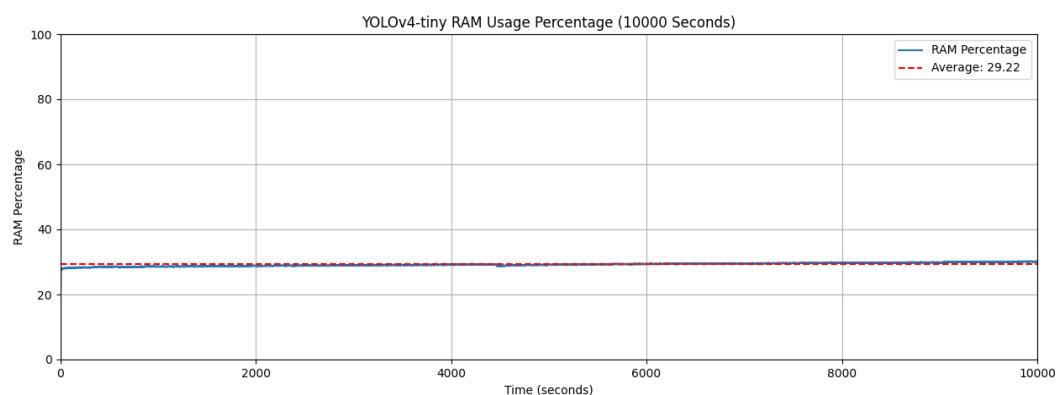
Gambar Persentase Penggunaan RAM YOLOv8m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



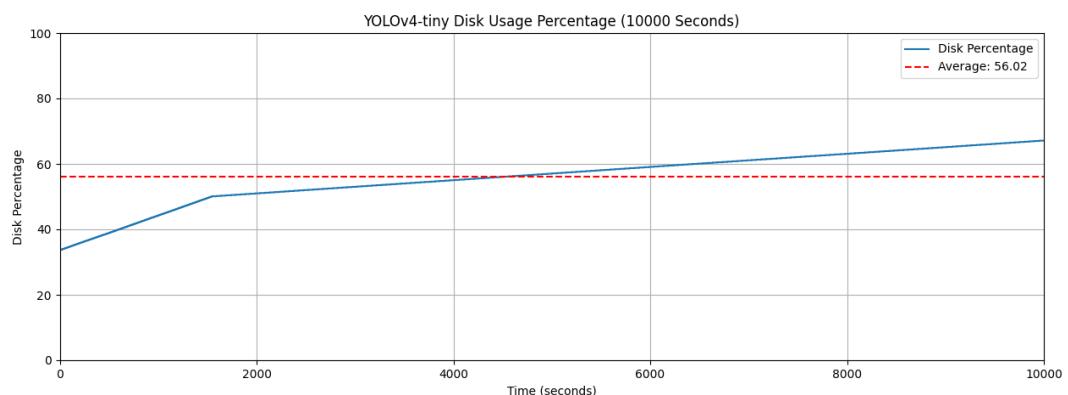
Gambar Persentase Penggunaan Disk YOLOv8m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode *Schedule*



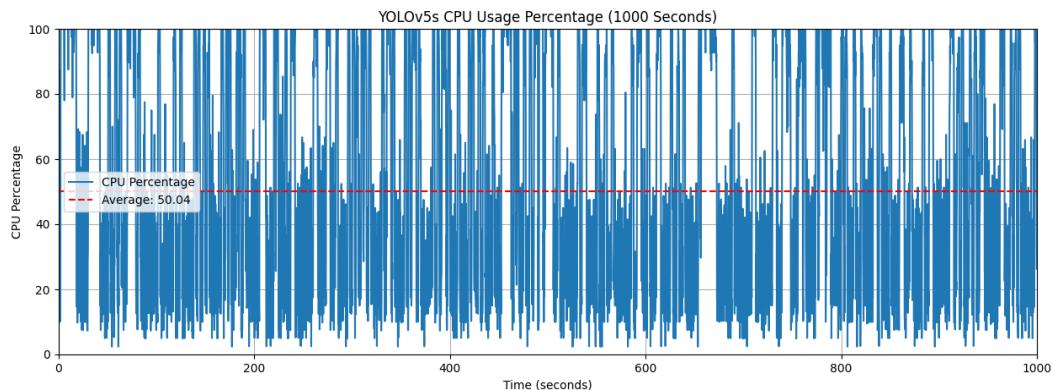
Gambar Persentase Penggunaan CPU YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



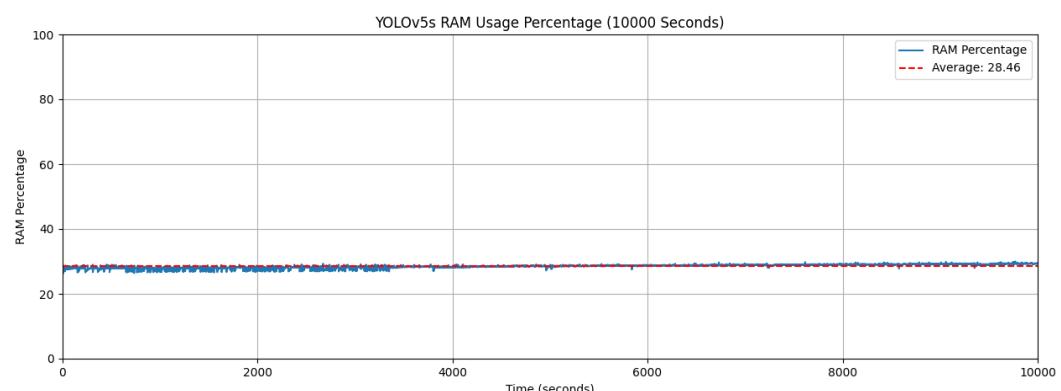
Gambar Persentase Penggunaan RAM YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



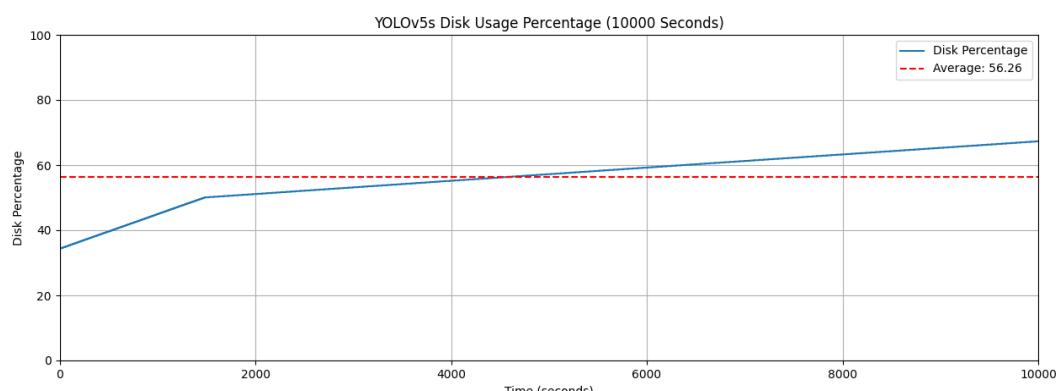
Gambar Persentase Penggunaan Disk YOLOv4-Tiny dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



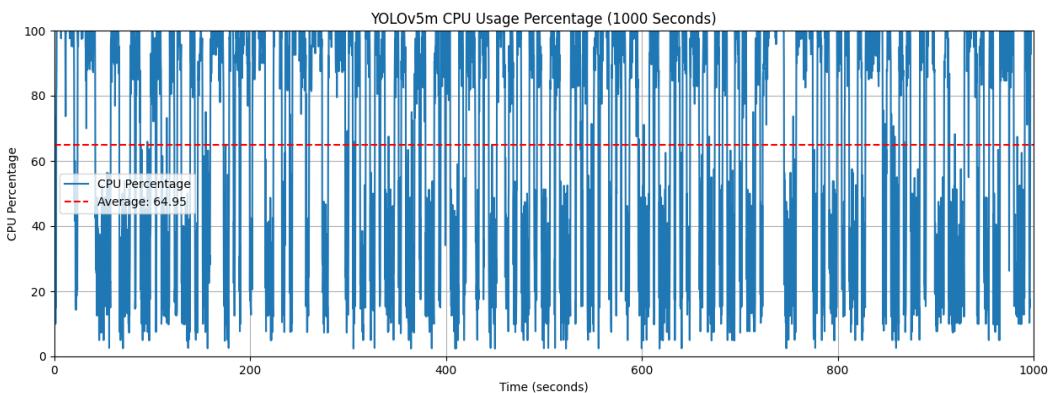
Gambar Persentase Penggunaan CPU YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



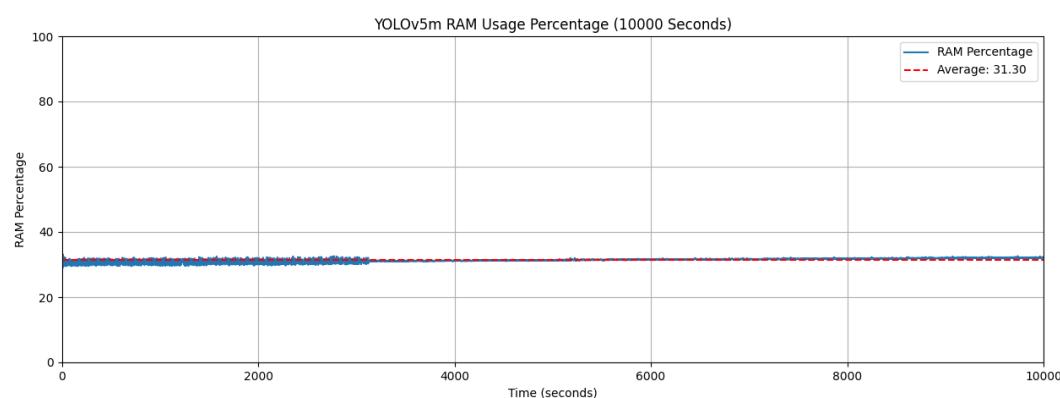
Gambar Persentase Penggunaan RAM YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



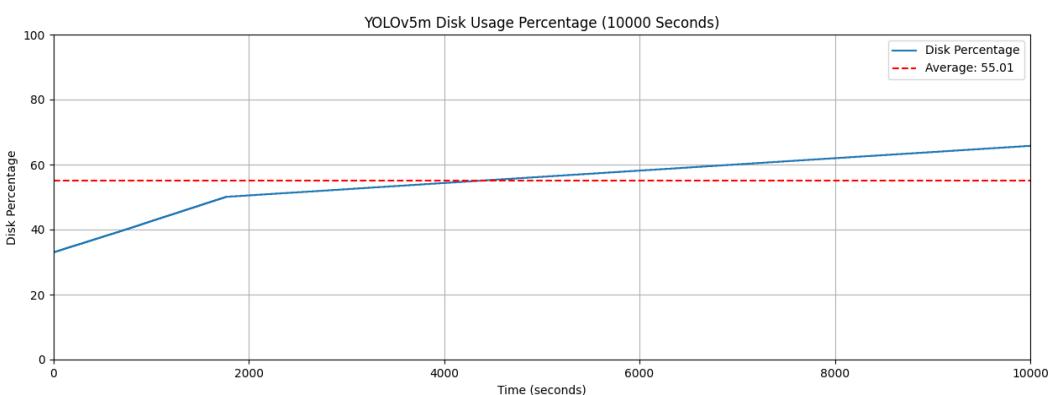
Gambar Persentase Penggunaan Disk YOLOv5s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



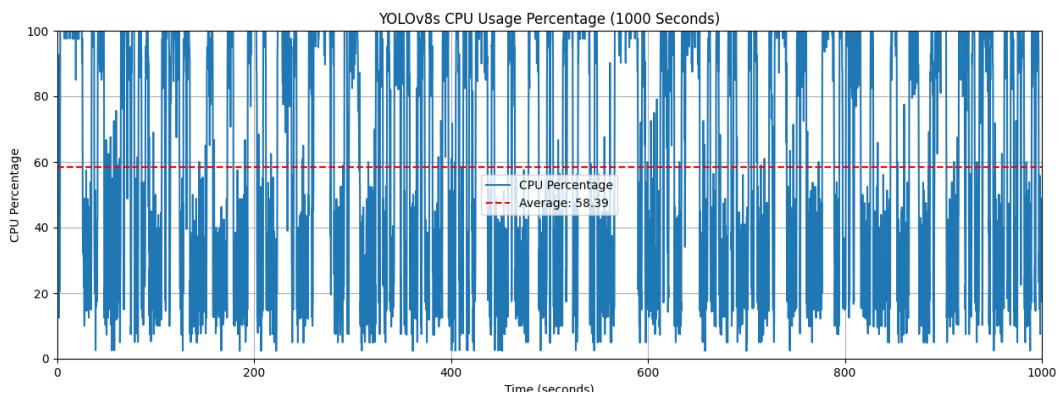
Gambar Persentase Penggunaan CPU YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



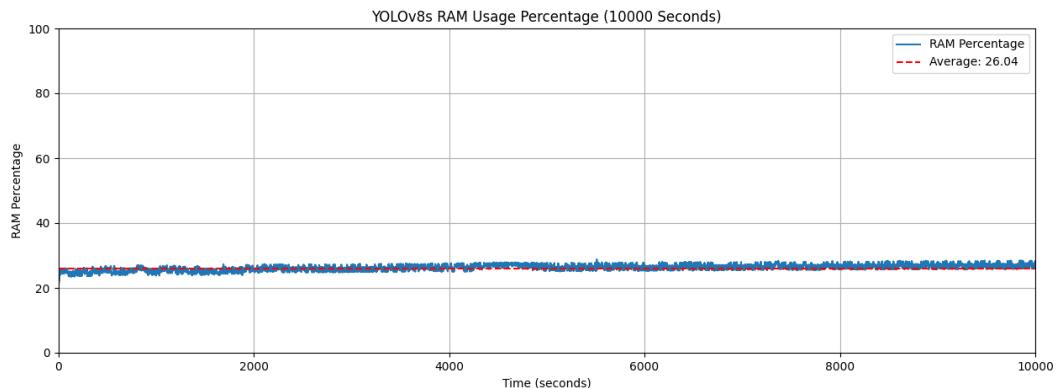
Gambar Persentase Penggunaan RAM YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



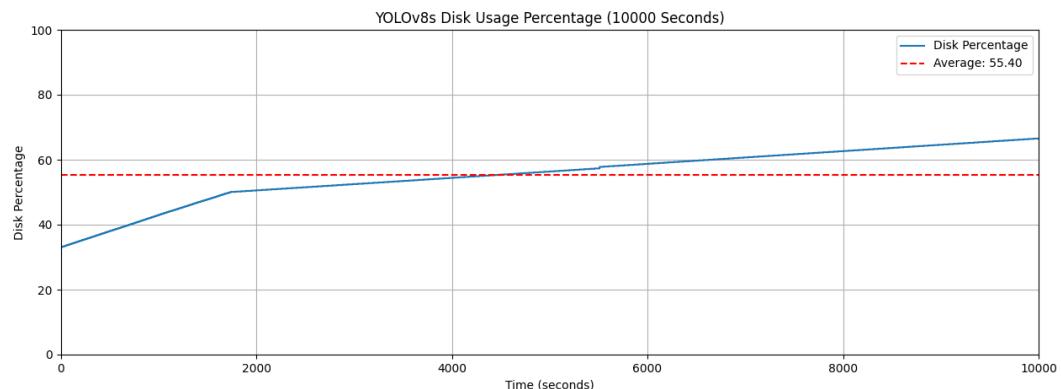
Gambar Persentase Penggunaan Disk YOLOv5m dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



Gambar Persentase Penggunaan CPU YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



Gambar Persentase Penggunaan RAM YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel



Gambar Persentase Penggunaan Disk YOLOv8s dengan *Resource-aware Framework* pada Dua Kamera dengan Metode Paralel

## Lampiran 5. Data Hasil *Running*

### a. CPU.csv

```
1. id,Time,Time_Elapsed,CPU_Percentage
2. 0,18:39:03,0.1080482006073,79.5
3. 1,18:39:03,0.309329986572266,15
4. 2,18:39:03,0.510409832000732,12.8
5. 3,18:39:03,0.711220264434814,10.5
6. 4,18:39:03,0.914565324783325,10.3
7. 5,18:39:04,1.11604619026184,48.7
8. 6,18:39:04,1.31665778160095,12.8
9. 7,18:39:04,1.51737999916077,12.5
10. 8,18:39:04,1.71808362007141,14.6
11. 9,18:39:04,1.91877818107605,12.5
12. 10,18:39:05,2.11964964866638,15
13. 11,18:39:05,2.32057929039001,43.6
14. 12,18:39:05,2.52746272087097,45.7
15. 13,18:39:05,2.7292640209198,43.6
16. 14,18:39:05,2.9310154914856,45
17. 15,18:39:06,3.13622736930847,51.3
18. 16,18:39:06,3.33951210975647,65.8
19. 17,18:39:06,3.54731345176697,100
20. 18,18:39:06,3.74838829040527,100
21. 19,18:39:06,3.95265650749207,97.6
22. 20,18:39:07,4.15431499481201,82.1
23. 21,18:39:07,4.36369276046753,100
24. 22,18:39:07,4.57254028320313,95
25. 23,18:39:07,4.77440547943115,89.7
26. 24,18:39:07,4.97948122024536,100
27. 25,18:39:08,5.18129134178162,65.8
28. 26,18:39:08,5.38417029380798,71.8
29. 27,18:39:08,5.59257650375366,87.5
30. 28,18:39:08,5.79413771629334,95.1
31. 29,18:39:08,5.99975633621216,83.3
32. 30,18:39:09,6.20332479476929,87.8
33. 31,18:39:09,6.4117193220459,90.2
34. 32,18:39:09,6.61381554603577,85
35. 33,18:39:09,6.8151969909668,90
36. 34,18:39:09,7.01967978477478,85.4
37. 35,18:39:10,7.22141361236572,100
38. 36,18:39:10,7.42418146133423,71.8
39. 37,18:39:10,7.62913513183594,84.6
40. 38,18:39:10,7.83089876174927,75
41. 39,18:39:10,8.03282308578491,66.7
42. 40,18:39:11,8.2366099357605,85
43. 41,18:39:11,8.44388103485107,90.7
44. 42,18:39:11,8.6557629108429,76.2
45. 43,18:39:11,8.86409854888916,66.7
46. 44,18:39:12,9.06857919692993,87.2
47. 45,18:39:12,9.26965618133545,94.7
48. 46,18:39:12,9.47523140907288,78
49. 47,18:39:12,9.67671918869019,100
50. 48,18:39:12,9.87929129600525,84.6
51. 49,18:39:13,10.0876314640045,97.6
52. 50,18:39:13,10.2892036437988,70
53. 51,18:39:13,10.495810508728,100
54. 52,18:39:13,10.6972639560699,70
55. 53,18:39:13,10.9034383296967,100
56. 54,18:39:14,11.1077659130096,97.6
57. 55,18:39:14,11.3134543895721,67.5
58. 56,18:39:14,11.5183701515198,100
59. 57,18:39:14,11.7257816791534,92.5
```

## b. RAM.csv

```
1. id,Time,Time_Elapsed,RAM_Precentage
2. 0,18:39:02,0.000896215438842773,18.4
3. 1,18:39:03,0.101896047592163,18.7
4. 2,18:39:03,0.202592611312866,18.8
5. 3,18:39:03,0.303234338760376,18.8
6. 4,18:39:03,0.403690338134766,18.9
7. 5,18:39:03,0.504300355911255,18.9
8. 6,18:39:03,0.60485053062439,19
9. 7,18:39:03,0.705401182174683,19
10. 8,18:39:03,0.807115316390991,19
11. 9,18:39:03,0.907585144042969,19
12. 10,18:39:03,1.00795531272888,19
13. 11,18:39:04,1.10838270187378,19
14. 12,18:39:04,1.20885825157166,19
15. 13,18:39:04,1.30932855606079,18.9
16. 14,18:39:04,1.40979337692261,19
17. 15,18:39:04,1.51027178764343,19
18. 16,18:39:04,1.61076402664185,19
19. 17,18:39:04,1.71118783950806,19
20. 18,18:39:04,1.81169700622559,19
21. 19,18:39:04,1.91233801841736,19
22. 20,18:39:04,2.01278948783875,19
23. 21,18:39:05,2.1132652759552,19
24. 22,18:39:05,2.21372532844543,19
25. 23,18:39:05,2.31524109840393,19.1
26. 24,18:39:05,2.41578388214111,19.3
27. 25,18:39:05,2.51647734642029,20.5
28. 26,18:39:05,2.61726808547974,21.5
29. 27,18:39:05,2.71859622001648,22.5
30. 28,18:39:05,2.82324576377869,23.2
31. 29,18:39:05,2.92410588264465,24.2
32. 30,18:39:05,3.02601456642151,24.9
33. 31,18:39:06,3.13139796257019,25.7
34. 32,18:39:06,3.23224830627441,26.7
35. 33,18:39:06,3.33466911315918,27.3
36. 34,18:39:06,3.4356415271759,27.9
37. 35,18:39:06,3.53908467292786,28.7
38. 36,18:39:06,3.63973355293274,29.2
39. 37,18:39:06,3.74032688140869,29.5
40. 38,18:39:06,3.84165406227112,29.8
41. 39,18:39:06,3.94711542129517,30.5
42. 40,18:39:07,4.05116248130798,30.6
43. 41,18:39:07,4.1517128944397,31.1
44. 42,18:39:07,4.25371098518372,31.7
45. 43,18:39:07,4.35462379455566,32.3
46. 44,18:39:07,4.45573711395264,32.2
47. 45,18:39:07,4.5592315196991,32.5
48. 46,18:39:07,4.66742086410522,32.7
49. 47,18:39:07,4.76805138587952,33.1
50. 48,18:39:07,4.86864113807678,31.8
51. 49,18:39:07,4.96930408477783,31.8
52. 50,18:39:08,5.06997895240784,31.8
53. 51,18:39:08,5.17322874069214,31.8
54. 52,18:39:08,5.27463245391846,31.9
55. 53,18:39:08,5.37902307510376,32
56. 54,18:39:08,5.47949004173279,32
57. 55,18:39:08,5.58311820030212,32
58. 56,18:39:08,5.68395566940308,32
59. 57,18:39:08,5.78456592559814,32
```

### c. Disk.csv

```
1. id,Time,Time_Elapsed,Disk_Precentage
2. 0,18:39:02,0.000172615051269531,33.8
3. 1,18:39:03,0.105672121047974,33.8
4. 2,18:39:03,0.205962657928467,33.8
5. 3,18:39:03,0.306141376495361,33.8
6. 4,18:39:03,0.406456708908081,33.8
7. 5,18:39:03,0.506643533706665,33.8
8. 6,18:39:03,0.607038021087647,33.8
9. 7,18:39:03,0.70732307434082,33.8
10. 8,18:39:03,0.809322595596314,33.8
11. 9,18:39:03,0.913130044937134,33.8
12. 10,18:39:03,1.01357674598694,33.8
13. 11,18:39:04,1.11450743675232,33.8
14. 12,18:39:04,1.21464228630066,33.8
15. 13,18:39:04,1.31477570533752,33.8
16. 14,18:39:04,1.41504979133606,33.8
17. 15,18:39:04,1.5152063369751,33.8
18. 16,18:39:04,1.61544537544251,33.8
19. 17,18:39:04,1.71571755409241,33.8
20. 18,18:39:04,1.81598520278931,33.8
21. 19,18:39:04,1.91622638702393,33.8
22. 20,18:39:04,2.01683974266052,33.8
23. 21,18:39:05,2.11755990982056,33.8
24. 22,18:39:05,2.21783423423767,33.8
25. 23,18:39:05,2.31811809539795,33.8
26. 24,18:39:05,2.42065525054932,33.8
27. 25,18:39:05,2.52093291282654,33.8
28. 26,18:39:05,2.62152600288391,33.8
29. 27,18:39:05,2.72212076187134,33.8
30. 28,18:39:05,2.82282304763794,33.8
31. 29,18:39:05,2.92319512367249,33.8
32. 30,18:39:05,3.02391767501831,33.8
33. 31,18:39:06,3.12460780143738,33.8
34. 32,18:39:06,3.2261528968811,33.8
35. 33,18:39:06,3.32738208770752,33.8
36. 34,18:39:06,3.42882823944092,33.8
37. 35,18:39:06,3.52982616424561,33.8
38. 36,18:39:06,3.63017988204956,33.8
39. 37,18:39:06,3.7334258556366,33.8
40. 38,18:39:06,3.83374857902527,33.8
41. 39,18:39:06,3.93735766410828,33.8
42. 40,18:39:06,4.03866696357727,33.8
43. 41,18:39:07,4.13978862762451,33.8
44. 42,18:39:07,4.2407853603363,33.8
45. 43,18:39:07,4.34136652946472,33.8
46. 44,18:39:07,4.44561910629272,33.8
47. 45,18:39:07,4.54639744758606,33.8
48. 46,18:39:07,4.65517234802246,33.8
49. 47,18:39:07,4.75744271278381,33.8
50. 48,18:39:07,4.85780191421509,33.8
51. 49,18:39:07,4.96126937866211,33.8
52. 50,18:39:08,5.06156754493713,33.8
53. 51,18:39:08,5.16185092926025,33.8
54. 52,18:39:08,5.26259613037109,33.8
55. 53,18:39:08,5.36312580108643,33.8
56. 54,18:39:08,5.4642550945282,33.8
57. 55,18:39:08,5.56973028182983,33.8
58. 56,18:39:08,5.67016267776489,33.8
59. 57,18:39:08,5.77065205574036,33.8
```

#### d. YOLODetect.csv

```
1. timeProcessed,filename,score,detect
2. 28-05-2024_18:39:24,./image/cam0/28-05-2024_18:39:05.png,0.47019875546296436,1
3. 28-05-2024_18:39:35,./image/cam0/28-05-2024_18:39:06.png,0,0
4. 28-05-2024_18:39:46,./image/cam0/28-05-2024_18:39:07.png,0.48339517414569855,2
5. 28-05-2024_18:40:06,./image/cam0/28-05-2024_18:39:55.png,0,0
6. 28-05-2024_18:40:16,./image/cam0/28-05-2024_18:39:56.png,0,0
7. 28-05-2024_18:40:38,./image/cam0/28-05-2024_18:40:27.png,0,0
8. 28-05-2024_18:40:49,./image/cam0/28-05-2024_18:40:28.png,0,0
9. 28-05-2024_18:41:00,./image/cam0/28-05-2024_18:40:29.png,0,0
10. 28-05-2024_18:41:20,./image/cam1/28-05-2024_18:41:09.png,0.7424754165112972,2
11. 28-05-2024_18:41:30,./image/cam1/28-05-2024_18:41:10.png,0.725902279218038,2
12. 28-05-2024_18:41:41,./image/cam1/28-05-2024_18:41:11.png,0.7959823787212372,2
13. 28-05-2024_18:41:52,./image/cam1/28-05-2024_18:41:12.png,0.7448158883131467,2
14. 28-05-2024_18:42:12,./image/cam1/28-05-2024_18:42:01.png,0.729939067363739,2
15. 28-05-2024_18:42:32,./image/cam1/28-05-2024_18:42:22.png,0.5413668372414329,2
16. 28-05-2024_18:42:43,./image/cam1/28-05-2024_18:42:23.png,0.5554490579026086,3
17. 28-05-2024_18:42:54,./image/cam1/28-05-2024_18:42:24.png,0.6398716992453525,2
18. 28-05-2024_18:43:14,./image/cam1/28-05-2024_18:43:04.png,0.6808458933463464,2
19. 28-05-2024_18:43:25,./image/cam1/28-05-2024_18:43:05.png,0.6551367663420163,1
20. 28-05-2024_18:43:45,./image/cam0/28-05-2024_18:43:34.png,0,0
21. 28-05-2024_18:43:56,./image/cam0/28-05-2024_18:43:35.png,0,0
22. 28-05-2024_18:44:07,./image/cam0/28-05-2024_18:43:37.png,0,0
23. 28-05-2024_18:44:17,./image/cam0/28-05-2024_18:43:38.png,0,0
24. 28-05-2024_18:44:28,./image/cam0/28-05-2024_18:43:39.png,0,0
25. 28-05-2024_18:44:38,./image/cam0/28-05-2024_18:43:41.png,0,0
26. 28-05-2024_18:44:58,./image/cam0/28-05-2024_18:44:48.png,0,0
27. 28-05-2024_18:45:09,./image/cam0/28-05-2024_18:44:50.png,0,0
28. 28-05-2024_18:45:19,./image/cam0/28-05-2024_18:44:51.png,0.47222432494163513,1
29. 28-05-2024_18:45:40,./image/cam1/28-05-2024_18:45:29.png,0.5053518638014793,1
30. 28-05-2024_18:45:56,./image/cam1/28-05-2024_18:45:45.png,0.5694164216518403,2
31. 28-05-2024_18:46:07,./image/cam1/28-05-2024_18:45:47.png,0.5713474787771702,2
32. 28-05-2024_18:46:17,./image/cam1/28-05-2024_18:45:48.png,0.5860234243529183,2
33. 28-05-2024_18:46:28,./image/cam1/28-05-2024_18:45:49.png,0.5574007391929626,2
34. 28-05-2024_18:46:39,./image/cam1/28-05-2024_18:45:50.png,0.7951462902128696,2
35. 28-05-2024_18:46:50,./image/cam1/28-05-2024_18:45:52.png,0.6789927507440249,2
36. 28-05-2024_18:47:00,./image/cam1/28-05-2024_18:45:53.png,0.5795012282000648,1
37. 28-05-2024_18:47:11,./image/cam1/28-05-2024_18:45:54.png,0.5653949469327927,1
38. 28-05-2024_18:47:21,./image/cam1/28-05-2024_18:45:56.png,0.6694804267449812,1
39. 28-05-2024_18:47:42,./image/cam0/28-05-2024_18:47:32.png,0.6379505395889282,2
40. 28-05-2024_18:47:55,./image/cam0/28-05-2024_18:47:44.png,0.7265098501335491,1
41. 28-05-2024_18:48:05,./image/cam0/28-05-2024_18:47:46.png,0.697754599831321,1
42. 28-05-2024_18:48:25,./image/cam0/28-05-2024_18:48:15.png,0.7586300894618034,2
43. 28-05-2024_18:48:46,./image/cam0/28-05-2024_18:48:36.png,0.7319306011001269,1
44. 28-05-2024_18:49:02,./image/cam0/28-05-2024_18:48:52.png,0.7639993835579265,1
45. 28-05-2024_18:49:15,./image/cam0/28-05-2024_18:49:04.png,0.7188797295093536,2
46. 28-05-2024_18:49:25,./image/cam0/28-05-2024_18:49:06.png,0.755570822291904,2
47. 28-05-2024_18:49:45,./image/cam1/28-05-2024_18:49:35.png,0.795032627052731,2
48. 28-05-2024_18:50:06,./image/cam1/28-05-2024_18:49:55.png,0.6964936435222626,1
49. 28-05-2024_18:50:16,./image/cam1/28-05-2024_18:49:56.png,0.6732533145695925,1
50. 28-05-2024_18:50:27,./image/cam1/28-05-2024_18:49:57.png,0.8032692313194275,1
51. 28-05-2024_18:50:37,./image/cam1/28-05-2024_18:49:58.png,0.7803998708724975,1
52. 28-05-2024_18:50:48,./image/cam1/28-05-2024_18:50:00.png,0.7803998708724975,1
53. 28-05-2024_18:50:58,./image/cam1/28-05-2024_18:50:01.png,0.6759494915604591,1
54. 28-05-2024_18:51:09,./image/cam1/28-05-2024_18:50:02.png,0.8138377785682678,1
55. 28-05-2024_18:51:19,./image/cam1/28-05-2024_18:50:04.png,0.728725066408515,1
56. 28-05-2024_18:51:30,./image/cam1/28-05-2024_18:50:05.png,0.7067754715681076,1
57. 28-05-2024_18:51:40,./image/cam1/28-05-2024_18:50:07.png,0.7099599912762642,1
```

## **BIODATA PENULIS**



Penulis dilahirkan di Kediri, 15 April 2002, merupakan anak kedua dari 6 bersaudara. Penulis telah menempuh pendidikan formal yaitu di SDN Burengan 3 Kota Kediri, SMPN 1 Kota Kediri dan SMAN 2 Kota Kediri. Setelah lulus dari SMAN tahun 2020, Penulis mengikuti SNMPTN dan diterima di Departemen Teknik Informatika FTEIC - ITS pada tahun 2020 dan terdaftar dengan NRP 5025201051.

Di Departemen Teknik Informatika Penulis sempat aktif di beberapa kegiatan Seminar yang diselenggarakan oleh Departemen, Kegiatan yang diadakan Himpunan Mahasiswa Teknik Computer – Informatika (HMTC). Selain itu, penulis memiliki pengalaman studi independen kampus merdeka di Alterra Academy sebagai Backend Developer dan magang kampus merdeka di PT Telkom Indonesia sebagai Backend Developer.