BAB I

PENDAHULUAN

1.1 Latar Belakang

Teknologi visi komputer telah banyak diterapkan dalam lingkungan pendidikan. Salah satu topik penelitian yang paling banyak diteliti adalah sistem absensi otomatis berbasis *face recognition*. Implementasi sistem absensi otomatis berbasis *face recognition* terbukti mampu meningkatkan efisiensi dan akurasi pencatatan kehadiran. Dalam penelitian yang mengombinasikan FaceNet dan VGG-16 pada skenario dunia nyata yang melibatkan 32 mahasiswa; sistem ini mencatat rata-rata akurasi pengenalan multi-wajah sebesar 75%, sementara untuk satu wajah FaceNet mencapai 99,20% (Autade dkk., 2023). Keberhasilan ini mempertegas bahwa teknologi pengenalan wajah dapat menggantikan proses administratif yang sebelumnya bergantung pada observasi manusia. Namun, penelitian untuk membangun sistem tersebut hanya berfokus pada tahap pengembangan model visi komputer. Penelitian tersebut belum mencakup koordinasi yang diperlukan untuk komponen dan infrastruktur sistem *machine learning*, yang sering kali rumit, termasuk peran yang diperlukan untuk mengotomatisasi dan mengoperasikan sistem *machine learning* di dunia nyata. Sebagai contoh, di banyak aplikasi industri, *data scientist* masih mengelola alur kerja *machine learning* secara manual, yang mengakibatkan banyak masalah selama pengoperasian solusi *machine learning* (Kreuzberger dkk., 2023).

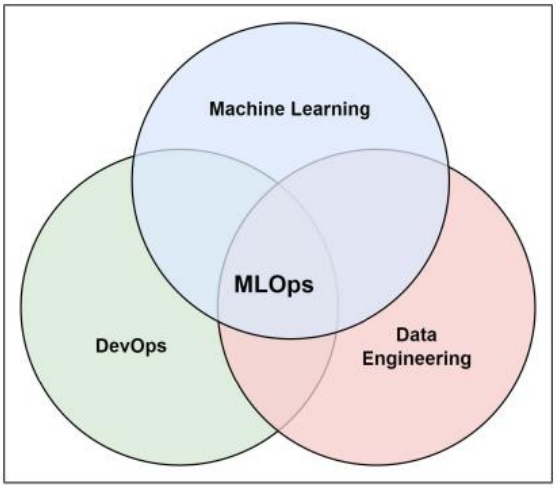
Proses *deployment* model visi komputer ke lingkungan produksi menghadapi berbagai tantangan yang signifikan. Salah satu masalah utama adalah integrasi model dengan sistem yang sudah ada, di mana sering kali diperlukan penyesuaian format data, protokol komunikasi, dan kompatibilitas teknologi, yang dapat memperumit proses implementasi. Selain itu, pemantauan dan pemeliharaan model setelah *deployment* menjadi krusial untuk mendeteksi degradasi performa akibat perubahan data atau lingkungan operasional. Sayangnya, banyak yang belum sepenuhnya mengadopsi prinsip *Machine Learning Operations* (MLOps) yang mencakup otomatisasi proses *deployment*, dan pembaruan model secara berkala. Tanpa MLOps, proses ini sering dilakukan secara manual, yang dapat meningkatkan risiko kesalahan dan inefisiensi. Untuk mengatasi hal tersebut, praktik *Machine Learning Operations* (MLOps) muncul sebagai pendekatan untuk menjamin proses *deployment* dan pemeliharaan model ML secara andal dan efisien di lingkungan operasional. MLOps memperluas konsep DevOps dengan menambahkan kapabilitas khusus *machine learning* seperti pelatihan model secara kontinu, pemantauan kinerja model, serta *versioning* data dan model (Amrit & Narayanappa, 2025). Otomatisasi alur kerja melalui prinsip-prinsip MLOps memungkinkan model visi komputer yang dikembangkan lebih mudah diintegrasikan ke sistem yang ada dan diperbarui secara berkala tanpa intervensi manual. Pada konteks penelitian ini, penambahan data siswa baru di sekolah tiap tahunnya mengharuskan pelatihan ulang model untuk menjaga agar model yang telah di-*deploy* tetap *up to dat*e walaupun terdapat penambahan atau perubahan pada *dataset*.

Oleh karena itu, penerapan MLOps dalam *deployment* model visi komputer tersebut menjadi solusi yang dapat digunakan untuk mengotomatisasi proses *retraining*

model dengan *dataset* baru secara berkala. Dengan demikian, sistem pengawasan yang berbasis visi komputer dapat terus beradaptasi dengan perubahan dataset dan memastikan modelnya berjalan dengan baik di tahap *production*.

1.2 Landasan Teori

1.2.1 *Machine Learning Operations* (MLOps)

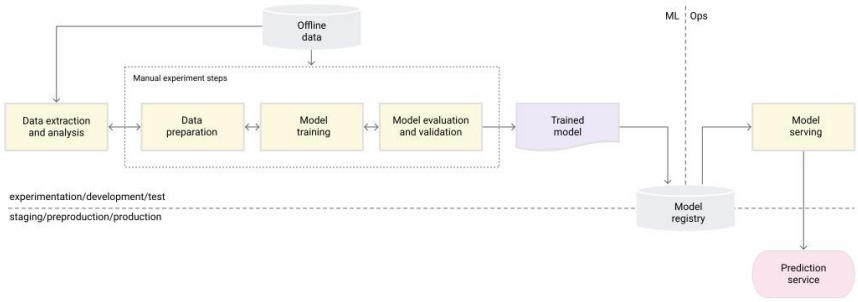


Gambar 1. Lingkup MLOps

Kecerdasan buatan yang didorong oleh kemajuan dalam *machine learning* sedang merevolusi berbagai sektor. Namun, penerapan dan pemeliharaan model AI dalam produksi menghadirkan tantangan yang signifikan. Hal ini memerlukan pendekatan pengembangan yang kokoh dan efisien, di mana MLOps berperan penting (Zarour dkk., 2025). MLOps adalah pendekatan terintegrasi yang mengombinasikan praktik *machine learning* guna mengotomatisasi serta meningkatkan efisiensi proses pengembangan hingga *deployment* model pada lingkungan produksi (Kreuzberger dkk., 2022). MLOps merupakan adaptasi dari disiplin DevOps, yang diciptakan untuk mengatasi masalah *deployment* berkelanjutan yang serupa pada perangkat lunak. Berbeda dengan DevOps, MLOps berusaha mengatasi masalah yang spesifik pada *machine learning*, seperti pelatihan berkelanjutan, pemantauan model, pengujian, dan versi data dan model (Amrit & Narayanappa, 2025).

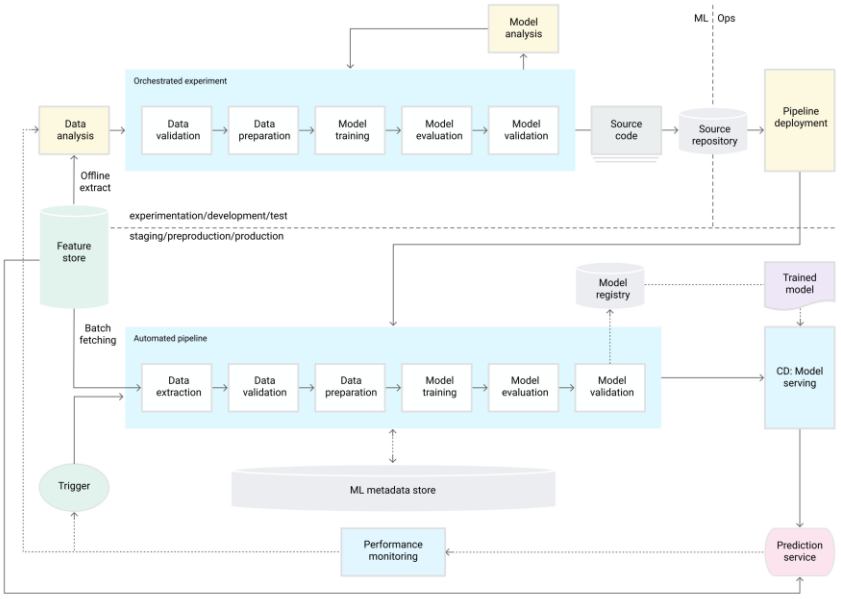
Penelitian menunjukkan bahwa implementasi yang efektif dari sistem yang didukung oleh ML bergantung pada penerapan praktik, proses, dan teknologi MLOps yang kokoh. Survei yang dilakukan oleh Databricks pada tahun 2023 menunjukkan bahwa, meskipun persentase model ML yang mencapai tahap produksi masih rendah, telah terjadi peningkatan yang signifikan seiring dengan kemajuan pesat dalam solusi MLOps misalnya MLFlow, Hugging Face, SageMaker, Google Vertex AI (Eken dkk., 2025). Berdasarkan dokumentasi Google Cloud (Kazmierczak dkk., 2024) memperkenalkan tiga tingkat MLOps yang membantu organisasi memahami sejauh mana proses ML mereka telah diotomatisasi dan distandardisasi.

MLOps level 0: *Manual process.* Pada MLOps Level 0, alur kerja *machine learning* masih dilakukan secara manual, meliputi persiapan data, pelatihan model, evaluasi, hingga *deployment* yang dijalankan tanpa *pipeline* otomatis. Proses seperti ini umumnya bersifat kurang reprodusibel dan rentan terhadap inkonsistensi ketika model perlu diperbarui atau dijalankan kembali. Level ini banyak ditemukan pada tahap eksperimen atau pengembangan awal, namun kurang cocok untuk kebutuhan produksi berskala besar.



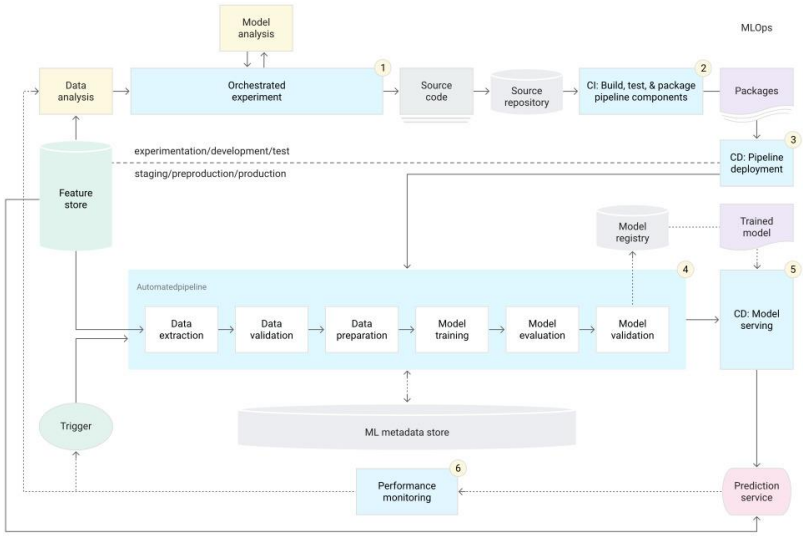
Gambar 2. Arsitektur MLOps level 0: *Manual process*

MLOps level 1: ML pipeline automation. Pada MLOps Level 1, organisasi mulai menerapkan pipeline otomatis untuk tahap pelatihan dan *deployment* model. Pendekatan ini memungkinkan *continuous training* dan *continuous delivery*, di mana model dapat diperbarui secara berkala berdasarkan perubahan data. Pipeline yang terdefinisi dengan baik membantu mengurangi intervensi manual, meningkatkan reprodusibilitas, dan mempercepat siklus iterasi eksperimen.



Gambar 3. Arsitektur MLOps level 1: ML *pipeline automation*

MLOps level 2: CI/CD pipeline automation. Pada tingkatan tertinggi MLOps Level 2, seluruh siklus hidup *machine learning* telah sepenuhnya diotomatisasi, mulai dari pengambilan data, validasi, pelatihan, evaluasi, *deployment*, hingga pemantauan model secara *real-time*. Sistem pada level ini mampu mendeteksi *data drift* atau penurunan performa dan dapat memicu pelatihan ulang secara otomatis melalui mekanisme *CI/CD* untuk model ML. Level ini sangat ideal untuk aplikasi AI skala industri yang membutuhkan stabilitas, reliabilitas, serta respons cepat terhadap perubahan pola data di lapangan.



Gambar 4. Arsitektur MLOps level 2: CI/CD *pipeline automation*

1.2.2 *Computer Vision*

Visi komputer adalah cabang dari kecerdasan buatan yang berfokus pada kemampuan mesin untuk menafsirkan dan memahami informasi visual dari gambar atau video digital (Cao dkk., 2025). Tujuan utamanya adalah memampukan komputer “melihat”, mengenali, dan memahami dunia visual sebagaimana penglihatan manusia (Manakitsa dkk., 2024). Teknologi kunci dalam visi komputer modern adalah *deep learning*, khususnya jaringan saraf tiruan konvolusional (CNN), yang mampu mempelajari fitur visual dari data mentah secara bertahap dan otomatis. CNN bekerja dengan beberapa lapisan (*layer*) yang mentransformasikan representasi data dari informasi mentah menjadi fitur tingkat tinggi yang siap digunakan untuk klasifikasi atau tugas lainnya (Zhao dkk., 2024). Berkat pendekatan ini, metode *deep learning* secara bertahap menggantikan teknik *machine learning* tradisional dengan akurasi yang jauh lebih tinggi.

Secara keseluruhan, kemajuan teknik *deep learning* (terutama CNN) telah memungkinkan komputer belajar langsung dari data visual tanpa rekayasa fitur manual (Manakitsa dkk., 2024). Pendekatan ini mendorong terobosan dalam berbagai tugas visi komputer seperti klasifikasi, deteksi objek, dan segmentasi citra . Seiring dengan peningkatan kapasitas komputasi dan ketersediaan data skala besar, integrasi visi komputer dengan model *deep learning* diperkirakan akan terus mendorong inovasi lintas sektor dalam beberapa tahun ke depan.

1.2.3 *Containerization*

Kontainerisasi merupakan bentuk virtualisasi tingkat sistem operasi (OS) yang membungkus aplikasi beserta seluruh ketergantungannya (misalnya pustaka dan file konfigurasi) ke dalam satu paket ringan "kontainer" sehingga aplikasi tersebut dapat dijalankan secara konsisten di berbagai lingkungan komputasi (Sroor dkk., 2025). Teknologi ini menempatkan aplikasi dalam proses terisolasi yang berbagi kernel sistem operasi *host*, sehingga konsumsi sumber daya menjadi lebih efisien (Sturley dkk., 2024). Hasilnya, aplikasi dalam kontainer dapat dipindahkan dengan mudah ke server fisik, server virtual, atau lingkungan *cloud* manapun tanpa mengkhawatirkan perbedaan konfigurasi lingkungan. Pendekatan ini menjamin kompatibilitas dan konsistensi eksekusi aplikasi di berbagai tahap pengembangan, pengujian, maupun produksi (Urblik dkk., 2024). Kontainerisasi juga mendukung arsitektur *microservices*, di mana fungsi-fungsi aplikasi dipecah menjadi modul-modul kecil yang mandiri. Dengan demikian pengembang dapat memelihara dan menskalakan tiap komponen secara independen, meningkatkan fleksibilitas pengembangan dan kecepatan iterasi aplikasi (Sroor dkk., 2025).

Keunggulan penggunaan kontainer meliputi efisiensi penggunaan sumber daya, performa yang baik, portabilitas tinggi, konsumsi energi lebih rendah, serta waktu *startup* dan *deploy* yang cepat (Fu dkk., 2025). Karena kontainer hanya membutuhkan kernel *host* dan tidak perlu menjalankan sistem operasi lengkap seperti pada mesin virtual, kapasitas penyimpanan dan memori yang dipakai lebih sedikit. Ini membuatnya lebih hemat biaya dan energi dibandingkan metode virtualisasi konvensional. Selain itu, pengembang dapat membangun lingkungan pengembangan, pengujian, dan produksi yang identik dengan menggunakan image kontainer yang sama, sehingga risiko ketidaksesuaian dapat diminimalkan (Urblik dkk., 2024). Dengan sifat portabel dan terstandarisasi ini, kontainer sangat cocok untuk model pengiriman perangkat lunak modern seperti *continuous integration/deployment* dan infrastruktur *cloud-native*.

Dibandingkan dengan virtualisasi tradisional, perbedaan utama kontainerisasi terletak pada lapisan abstraksi sumber daya. Virtualisasi konvensional menggunakan *hypervisor* untuk menjalankan mesin virtual (VM) lengkap dengan sistem operasinya sendiri di atas satu server fisik (Sturley dkk., 2024). Pendekatan ini memberikan isolasi yang sangat kuat antar VM, namun memerlukan waktu inisialisasi yang lebih lama dan sumber daya (CPU, memori, penyimpanan) lebih besar karena tiap VM memuat sistem operasi penuh. Sebaliknya, kontainer hanya menjalankan aplikasi dalam proses terisolasi yang berbagi *kernel host* (Fu dkk., 2025). Oleh karena itu, kontainerisasi jauh lebih ringan dan memungkinkan penambahan atau penghapusan kontainer secara cepat (*autoscaling*) dengan *overhead* minimal (Sturley dkk., 2024). Konsep inilah yang membuat kontainerisasi sangat efektif untuk meningkatkan portabilitas dan kecepatan *deployment* aplikasi di lingkungan *cloud* atau *edge*.

1.2.4 *Cloud Computing*

Cloud computing merupakan paradigma TI yang menyediakan layanan komputasi seperti *server*, penyimpanan, aplikasi, dan basis data melalui internet dengan sifat *on-demand* dan akses jaringan luas. Menurut definisi NIST (National Institute of Standards

and Technology), komputasi awan adalah model yang memungkinkan “akses jaringan yang ada di mana-mana dan sesuai permintaan ke kumpulan sumber daya komputasi yang dapat dikonfigurasi, seperti penyimpanan, aplikasi, server, jaringan, dan layanan, yang dapat disediakan dan dirilis dengan cepat dengan sedikit manajemen atau interaksi dengan penyedia layanan”. Model layanan komputasi awan meliputi layanan perangkat lunak (SaaS), platform (PaaS), dan infrastruktur (IaaS), serta empat model penerapan seperti publik, privat, komunitas, dan hibrid. Karakteristik utama *cloud computing* antara lain layanan mandiri sesuai permintaan, akses jaringan luas, pemusatan sumber daya (resource pooling), elastisitas cepat, dan layanan terukur (Bejarano dkk., 2024). Secara khusus, komputasi awan ditandai dengan kemampuan skalabilitas dan elastisitas yang tinggi. Skalabilitas berarti sistem dapat menambah atau mengurangi sumber daya secara fleksibel sesuai dengan perubahan beban kerja, sementara elastisitas menekankan penyesuaian otomatis sumber daya secara cepat mengikuti fluktuasi permintaan (Bejarano dkk., 2024). Dengan karakteristik ini, organisasi dapat dengan mudah mengalokasikan sumber daya baru ketika beban meningkat dan melepaskannya saat tidak dibutuhkan lagi, tanpa perlu investasi infrastruktur permanen. *Cloud computing* memberikan fleksibilitas tinggi pada infrastruktur, sekaligus efisiensi manajemen, karena penyedia layanan *cloud* menangani layer infrastruktur yang kompleks.

Di sisi manfaat, *cloud computing* menawarkan efisiensi biaya dan agilitas bagi organisasi. *Cloud computing* menghilangkan kebutuhan investasi awal besar untuk server dan peralatan penyimpanan lokal serta mengurangi biaya pemeliharaan infrastruktur. Selain itu, *cloud computing* mendukung kolaborasi dan akses global, karena data dan aplikasi tersimpan di internet sehingga bisa diakses kapan saja dan dari mana saja oleh tim yang terdistribusi. Fitur kolaboratif ini memungkinkan kerja jarak jauh dan koordinasi *real-time* antar anggota tim meski berpangkal di lokasi berbeda. Manfaat lain yang sering dikutip adalah peningkatan fleksibilitas operasional dan kecepatan layanan; misalnya organisasi dapat dengan cepat mengerahkan aplikasi atau layanan baru tanpa harus menunggu penyediaan server fisik. Penyedia cloud umumnya juga menerapkan mekanisme keamanan canggih (seperti enkripsi dan kontrol akses) sehingga data organisasi dapat lebih terlindungi dibanding pengelolaan sendiri (Joe, 2023). Dengan demikian, penggunaan *cloud computing* memungkinkan organisasi mengoptimalkan sumber daya.

1.2.5 FastApi

FastAPI merupakan *framework* web Python modern yang berperforma tinggi untuk membangun antarmuka pemrograman aplikasi (API) berbasis REST (Chen, 2023). Salah satu karakteristik utama FastAPI adalah dukungan penuh terhadap operasi *asynchronous* yang memungkinkan penanganan banyak permintaan sekaligus secara efisien. Kombinasi kecepatan tinggi dan kemudahan tersebut membuat FastAPI populer untuk membangun layanan web modern yang skalabel (Mabotha dkk., 2025).

Dalam konteks deployment model pembelajaran mesin, FastAPI sering digunakan untuk menyajikan model ML sebagai layanan RESTful. Misalnya, dalam arsitektur komputasi berperforma tinggi untuk inferensi model Large Language Model (LLM), FastAPI diintegrasikan sebagai endpoint RESTful berbasis microservices yang mendukung skalabilitas tinggi (Luiz dkk., 2025). Dukungan *asynchronous* FastAPI

memungkinkan eksekusi inferensi (prediksi) model secara simultan dengan latensi rendah, sehingga aplikasi AI dapat memberikan prediksi secara *real-time*.

1.2.6 Metrik Evaluasi Kinerja

Evaluasi kinerja sistem digunakan untuk menganalisis sistem MLOps yang telah dibuat. Pengujian sistem ini dilakukan pada dua bagian yaitu, *retraining* model yang menggunakan metrik *retraining duration* dan *resource utilization*, serta pada *prediction service* yang menggunakan metrik *response time* dan *throughput*.

***Retraining Duration* (Durasi Pelatihan Ulang).** *Retraining duration* merupakan metrik yang digunakan untuk mengukur total waktu yang dibutuhkan sistem untuk menjalankan proses pelatihan ulang model secara *end-to-end*. Pengukuran dimulai dari tahap persiapan data, pemuatan bobot model, proses pelatihan, evaluasi, hingga penyimpanan artefak model ke *repository*. Metrik ini sangat penting dalam konteks MLOps karena mencerminkan seberapa cepat sistem dapat memperbarui model ketika terdapat data baru. Durasi pelatihan yang lebih singkat tidak hanya memungkinkan model untuk beradaptasi lebih cepat terhadap perubahan data, tetapi juga dapat mengurangi biaya operasional terutama ketika pelatihan dilakukan menggunakan GPU di lingkungan *cloud*. Dengan demikian, *retraining duration* menjadi indikator utama efisiensi *pipeline* MLOps yang dibangun.

$$Retraining\ Duration = t_{end} - t_{start} \quad (1)$$

Keterangan:
t_{end} = waktu *training* selesai
t_{start} = waktu *training* dimulai

***Resource Utilization* (Utilisasi Sumber Daya)**. *Resource utilization* digunakan untuk menilai seberapa efisien sistem memanfaatkan sumber daya komputasi seperti CPU dan GPU selama proses pelatihan model. Metrik ini membantu mengidentifikasi apakah sumber daya yang digunakan sudah optimal atau justru berlebihan. CPU *utilization* umumnya mencerminkan intensitas komputasi selama proses *data loading*, *preprocessing*, dan manajemen *pipeline*, sedangkan GPU *utilization* menunjukkan beban komputasi pada proses pelatihan model.

$$Resource\ Utilization = \frac{\sum_{i=1}^{n}(average\ resource\ utilization)_i}{n} \quad (2)$$

Keterangan:
n = durasi *training*
i = menit ke-i

***Response Time* (Waktu Respon)**. *Response time* adalah metrik yang mengukur berapa lama sistem membutuhkan waktu untuk memproses suatu permintaan hingga menghasilkan keluaran penuh. Pada sistem berbasis visi

komputer, metrik ini mencakup waktu pemrosesan mulai dari penerimaan input, deteksi objek, pelacakan, pengenalan wajah, hingga penyimpanan hasil. *Response time* merupakan indikator utama performa layanan inferensi

$$Average\ Response\ Time = \frac{\sum_{i=1}^{n}(t_{end} - t_{start})_i}{n} \quad (3)$$

Keterangan:
n = jumlah total *request*
i = request ke-i
t_{end} = waktu respon diterima
t_{start} = waktu *request* dikirim

Throughput. *Throughput* mengukur kapasitas sistem dalam memproses sejumlah unit data dalam satuan waktu tertentu. Dalam konteks inferensi video, *throughput* biasanya dinyatakan dalam *frame per second*, yaitu berapa banyak *frame* yang dapat diproses oleh sistem dalam satu detik

$$Throughput = \frac{Total\ Frame\ yang\ Diproses}{Waktu\ Pemrosesan} \quad (4)$$

1.3 Rumusan Masalah

Adapun permasalahan utama yang akan diangkat dalam penelitian ini adalah:

1. Bagaimana implementasi MLOps *Pipeline Automation* pada *deployment* model visi komputer?
2. Bagaimana kinerja dari implementasi MLOps *Pipeline Automation* dalam deployment model visi komputer?

1.4 Tujuan dan Manfaat Penelitian

1.4.1 Tujuan

Adapun tujuan dari penelitian ini adalah:

1. Mengimplementasikan MLOps *Pipeline Automation* pada *deployment* model visi komputer.
2. Menganalisis hasil kinerja dari implementasi MLOps *Pipeline Automation* pada *deployment* model visi komputer.

1.4.2 Manfaat

Adapun manfaat dari penelitian ini adalah:

1. Membantu sekolah dalam melakukan pencatatan kepulangan siswa secara otomatis dan akurat, sehingga mengurangi ketergantungan pada pengawasan manual.
2. Memberikan kontribusi pada pengembangan *environment* untuk *deployment* model visi komputer.

1.5 Batasan Masalah

Adapun batasan masalah dalam penelitian ini adalah sebagai berikut:

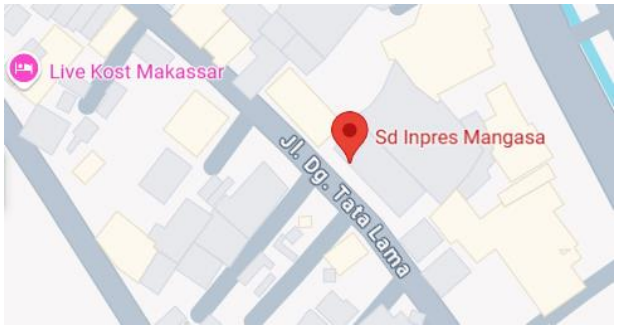
1. MLOps yang dibuat terbatas pada MLOps level 1: ML *pipeline automation*.
2. MLOps *Pipeline Automation* yang akan dibuat terbatas untuk *deployment* model visi komputer dalam sistem *monitoring* data log kepulangan siswa di area penjemputan sekolah.
3. Metrik performa yang akan diukur terbatas pada *retraining duration*, *resource utilization*, *response time*, dan *throughput.*

BAB II

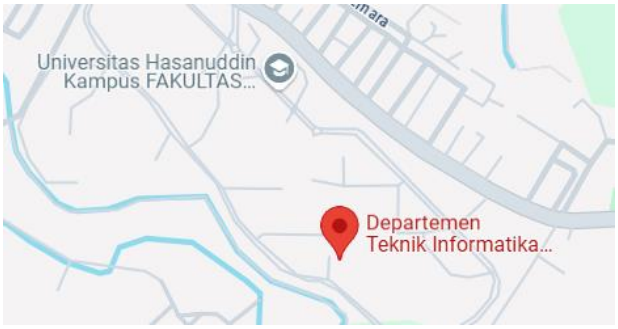
METODE PENELITIAN

2.1 Tempat dan Waktu Penelitian

Penelitian ini dilaksanakan sejak disetujuinya judul penelitian pada bulan Maret hingga November 2025. Lokasi pengambilan data dilakukan di SD Inpres Mangasa, Jl. Dg. Tata Lama, Kelurahan Pandang-Pandang, Kecamatan Somba Opu, Kabupaten Gowa. Adapun proses penelitian dilakukan di Laboratorium *Artificial Intelligence* (AI), Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.



(a)



(b)

Gambar 5. Lokasi penelitian: (a) SD Inpres Mangasa; (b) Laboratorium *Artificial Intelligence* (AI)

2.2 Instrumen Penelitian

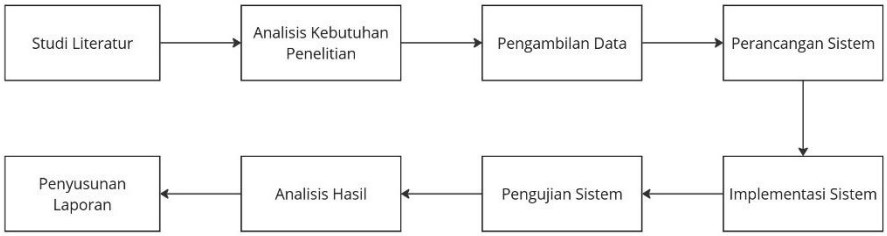
Berikut instrumen atau alat yang digunakan dalam penelitian ini:

1. Perangkat lunak:
   a. Visual Studio Code
   b. Google Cloud Platform
   c. Microsoft Word
   d. Microsoft PowerPoint
   e. Microsoft Edge
   f. *Operating System* Windows 11 64-bit
   g. Zotero

2. Perangkat keras:
   a. Asus ROG Strix G531GT, *Processor* Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz, *Graphic Card* NVIDIA® GeForce GTX™ 1650, RAM 16GB.
3. Bahasa pemrograman:
   a. Python v3.11.13
4. Algoritma:
   a. YOLOv11n
   b. BoT-SORT
   c. MTCNN
   d. FaceNet
5. *Library*:
   a. Pandas v2.3.0, digunakan untuk memproses data, termasuk manipulasi dan analisis data.
   b. Matplotlib v3.10.3, digunakan untuk mengubah data menjadi informasi visual, seperti grafik atau diagram, guna memudahkan evaluasi dan interpretasi hasil.
   c. Numpy v1.26.4, digunakan untuk melakukan berbagai perhitungan numerik.
   d. Opencv-python v4.11.0.86, digunakan untuk pemrosesan citra dan video, seperti membaca frame dari video CCTV, menggambar *bounding-box*, serta ekstraksi wajah dari *frame*.
   e. Ultralytics v8.3.156, digunakan untuk memanggil dan menjalankan model YOLOv11n untuk *object detection* dan BoT-SORT untuk *object tracking.*
   f. Roboflow v1.1.66, digunakan untuk mengakses *dataset* yang telah dilabeli ke dalam *pipeline* pelatihan model.
   g. Motmetrics v1.4.0, digunakan untuk mengevaluasi performa sistem *tracking*, dengan menghitung metrik seperti IDF1, MOTA, dan MOTP.
   h. Imgaug v0.4.0, digunakan untuk melakukan augmentasi pada *dataset* wajah.
   i. Facenet-pytorch v2.6.0, digunakan untuk memanggil model MTCNN (*face detection*).
   j. Tensorflow v2.13.1, digunakan sebagai *framework deep learning* untuk menjalankan model-model *neural network*.
   k. Torch v2.7.1+cu128, digunakan sebagai *framework* utama untuk menjalankan model berbasis PyTorch.
   l. FastApi v0.115.12, digunakan untuk membangun layanan inferensi berbasis REST API yang menyediakan endpoint prediksi secara *real-time*.
6. *Services*:
   a. Artifact Registry, biaya penyimpanan $0.10 per GB / *month*.
   b. Cloud Storage, biaya penyimpanan $0.02/1 *gibibyte month*.
   c. Cloud Workflows, biaya $0.01 per *increment* of 1,000 *steps*.
   d. Eventarc.
   e. Vertex AI Custom Jobs, biaya *machine type* g2-standard-8 dan 1 *accelerators* NVIDIA_TESLA_L4 $1.053116256 / 1 *hour*.
   f. Cloud Run Services, biaya CPU (per vCPU-*second*) $0.0000216 dan *memory* (per GiB-*second*) $0.0000024.

g. Cloud Firestore, biaya *document reads* $0.0369, *document writes* 20,000, *document deletes* 20,000.

2.3 Tahapan Penelitian



Gambar 6. Tahapan penelitian

Terdapat beberapa tahapan dalam proses penelitian ini sebagaimana ditunjukkan pada Gambar 6. Tahapan penelitian. Penelitian diawali dengan studi literatur untuk mengkaji referensi dari berbagai sumber yang relevan guna memperkuat landasan teori. Selanjutnya dilakukan analisis kebutuhan untuk merumuskan permasalahan, tujuan penelitian, serta data dan metode yang akan digunakan. Setelah itu, dilakukan proses pengambilan data sebagai bahan utama dalam pengembangan sistem. Data yang telah dikumpulkan digunakan dalam tahap perancangan sistem, yang meliputi penyusunan alur kerja dan penentuan komponen yang dibutuhkan. Sistem yang telah dirancang kemudian diimplementasikan dan diuji untuk mengetahui kinerja dan tingkat keakuratannya. Hasil dari pengujian dianalisis untuk mengevaluasi efektivitas sistem, dan seluruh rangkaian proses tersebut disusun secara sistematis dalam bentuk laporan penelitian.

2.4 Teknik Pengambilan Data

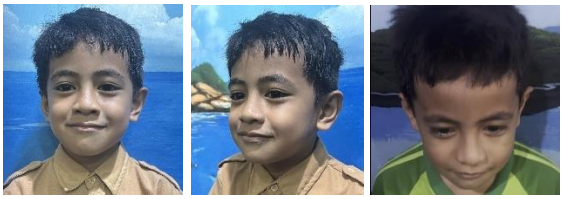
2.4.1 Data Primer

Keseluruhan data yang digunakan dalam penelitian ini merupakan data primer yang diperoleh langsung dari lokasi pengambilan data. Data tersebut terbagi menjadi dua jenis, yaitu *dataset* untuk pelatihan model *object detection* dan *dataset* untuk pelatihan model *face recognition*.

Dataset object detection. *Dataset* pertama, yaitu *dataset object detection*, berisi kumpulan gambar atau *frame* yang diambil dari hasil ekstraksi video yang merekam aktivitas di area gerbang sekolah. Pengambilan data dilakukan menggunakan kamera CCTV Vivotek IP9165-LPC yang diarahkan langsung ke gerbang sekolah.

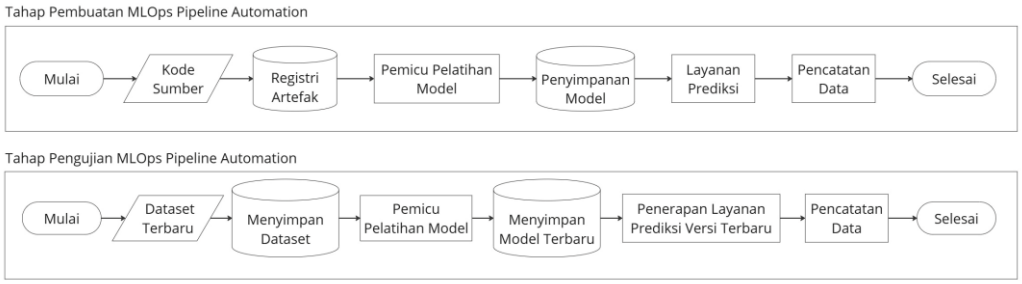
Gambar 7. Contoh *dataset object detection*

Dataset face recognition. *Dataset* kedua adalah *dataset face recognition* yang terdiri dari kumpulan foto wajah siswa yang dikumpulkan secara langsung di lokasi seperti yang ditunjukkan pada Gambar 14.

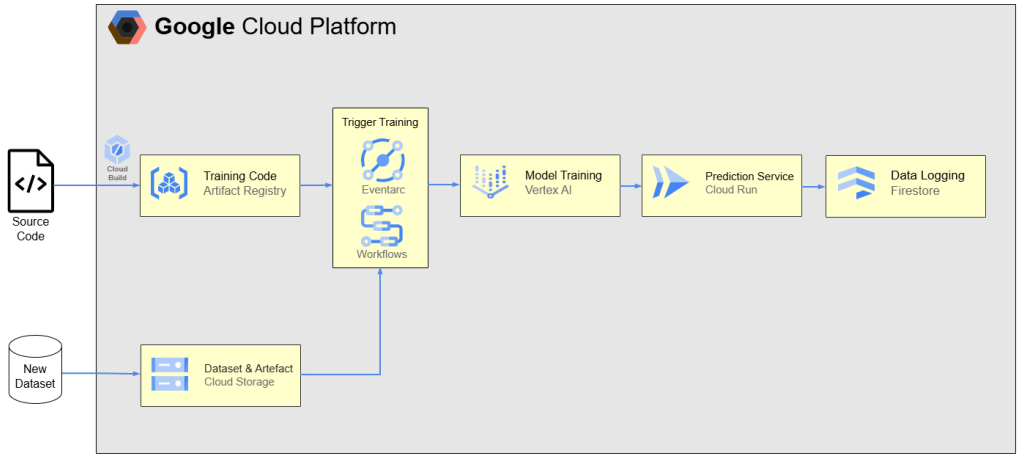


Gambar 8. Contoh *dataset face recognition*

2.5 Perancangan dan Implementasi Sistem



Gambar 9. Perancangan alur sistem

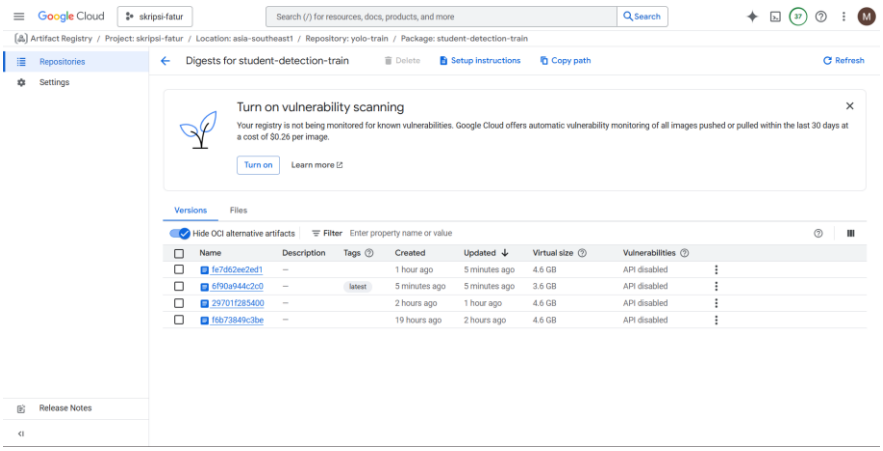


Gambar 10. Topologi Google Cloud Platform *services*

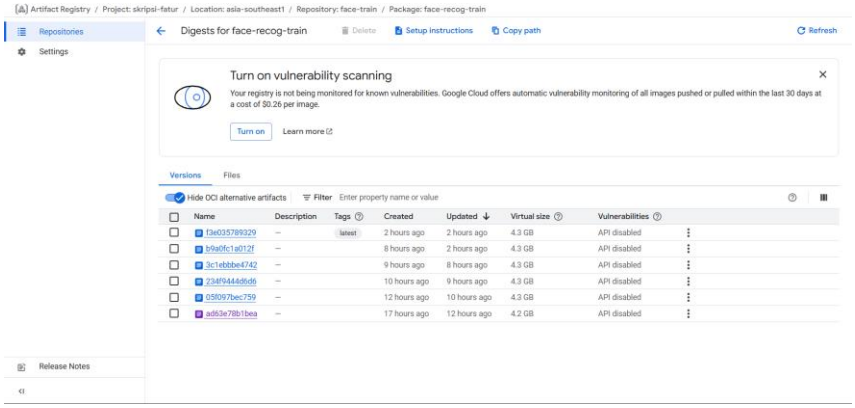
2.5.1 Tahap Pembuatan MLOps Pipeline Automation

1. *Input Source Code*
   Tahap ini merupakan awal dalam pembuatan sistem MLOps ini. Pada tahap ini, seluruh kode yang digunakan untuk proses pelatihan, evaluasi, dan *deployment* model visi komputer disiapkan oleh pengembang *machine learning* yang sebelumnya telah mengembangkan dan menguji model tersebut. Kode yang dihasilkan mencakup skrip *training*, konfigurasi model, *preprocessing*, serta komponen yang diperlukan untuk menjalankan layanan prediksi. *Source code* kemudian dikemas dan diintegrasikan ke dalam *pipeline* MLOps untuk mendukung proses otomatisasi *retraining* dan *deployment* model.

2. Registri Artefak
   Pada tahap ini, registri artefak digunakan untuk menyimpan *container images* hasil *build* dari *source code* yang akan dijalankan pada tahap *model training* maupun *model serving*. Layanan ini mendukung *versioning* sehingga setiap artefak yang dihasilkan dapat dilacak berdasarkan versi dan riwayat pembuatannya. Selain itu, dengan adanya registri artefak, proses *deployment* menjadi lebih aman, konsisten, dan dapat direplikasi di berbagai lingkungan.



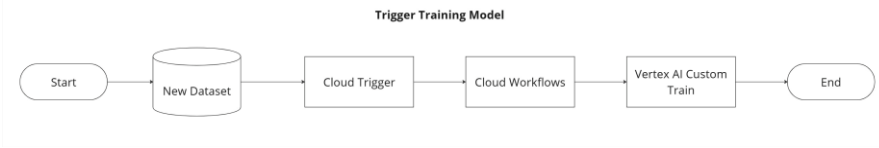
(a)



(b)

Gambar 11. *Artifact registry:* (a) *object detection model*; (b) *face recognition model*

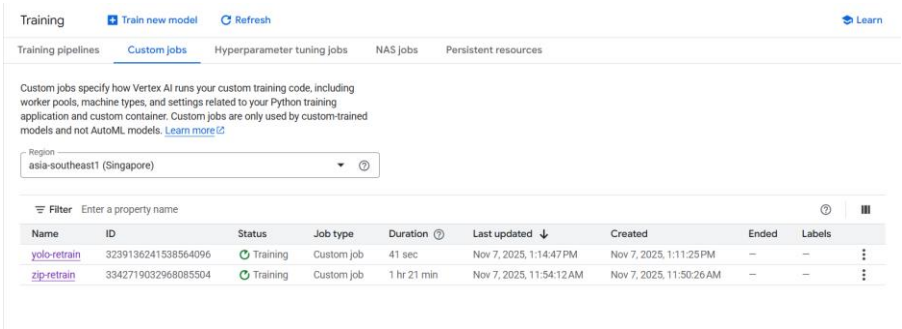
3. Pemicu Pelatihan Model



Gambar 12. *Flowchart pemicu pelatihan model*

Tahapan ini merupakan komponen inti dalam mekanisme otomatisasi pipeline MLOps yang mengatur proses *retraining* model berbasis *event-driven*. Pada penelitian ini, proses pemicu pelatihan ulang diimplementasikan menggunakan kombinasi Google Cloud Workflows dan Cloud Trigger. Ketika sistem mendeteksi adanya *dataset* baru yang diunggah ke Google Cloud Storage (GCS), maka Cloud Trigger akan mengeksekusi alur kerja yang telah didefinisikan di Cloud Workflows. Alur kerja ini kemudian memanggil API Custom Train untuk memulai proses pelatihan ulang model di Vertex AI.

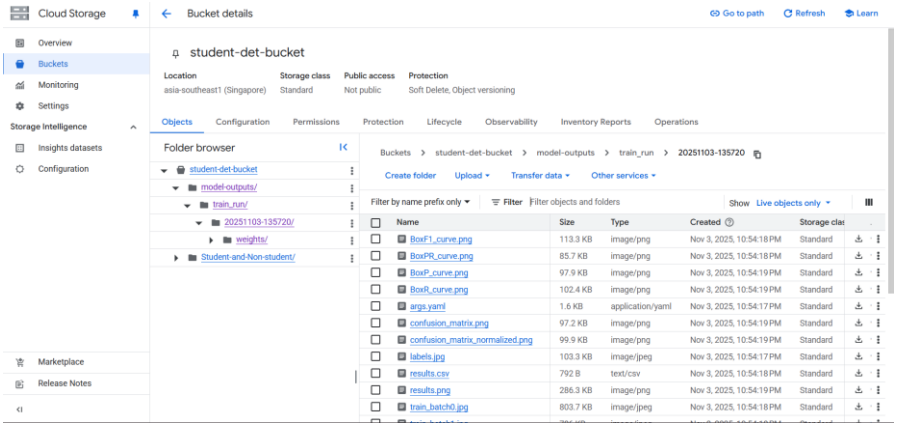
Kode pelatihan yang digunakan dalam proses ini disimpan dalam bentuk docker *image di* Google Cloud Artifact Registry. Pendekatan ini memungkinkan proses retraining berjalan secara otomatis dan konsisten tanpa intervensi manual. Dengan arsitektur *event-driven* seperti ini, sistem menjadi adaptif terhadap perubahan data di lapangan seperti penambahan dataset sehingga model visi komputer selalu diperbarui dengan versi terbaru yang lebih relevan dan akurat untuk digunakan dalam sistem pengawasan *real-time*.



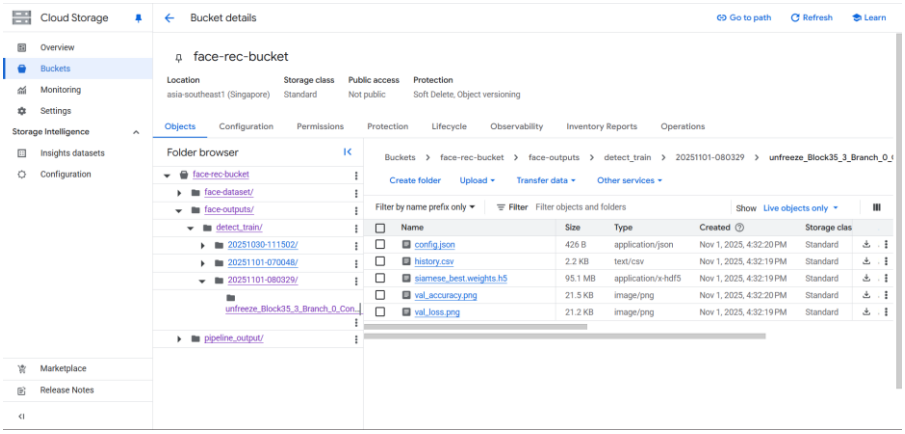
Gambar 13. Vertex AI Custom Jobs

4. Penyimpanan Model

Tahapan ini berfungsi sebagai proses penyimpanan terpusat untuk seluruh artefak yang dihasilkan selama pelatihan model, termasuk file bobot model (best.pt), log evaluasi, serta konfigurasi yang digunakan dalam proses training. Pada penelitian ini, Google Cloud Storage digunakan sebagai media penyimpanan utama untuk seluruh artefak tersebut. Setiap kali proses pelatihan selesai, *pipeline* secara otomatis mengunggah hasil *training* ke GCS dan memperbarui *file* latest.json, yang memuat informasi mengenai *path* model terbaru. File ini berperan sebagai pointer dinamis yang mengarahkan sistem *model serving* di Google Cloud Run agar senantiasa menggunakan versi model terbaik dari hasil pelatihan terakhir.



(a)



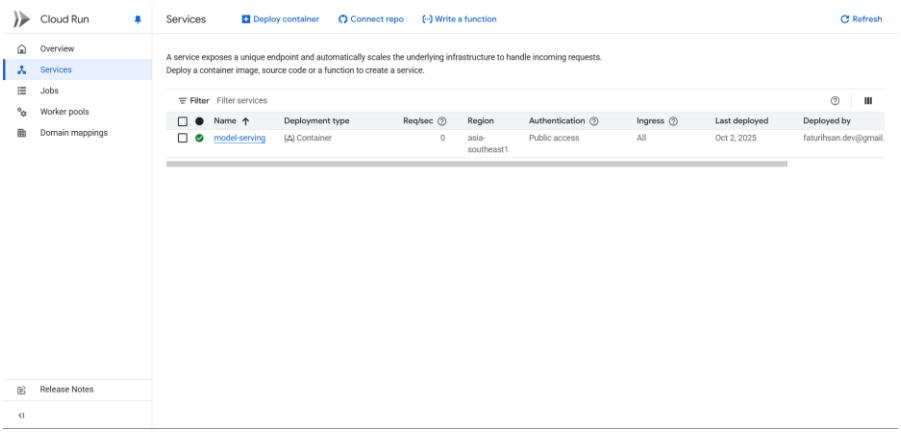
(b)

Gambar 14. Google Cloud Storage: (a) Hasil *training object detection* model; (b) Hasil *training face recognition* model

5. Layanan Prediksi

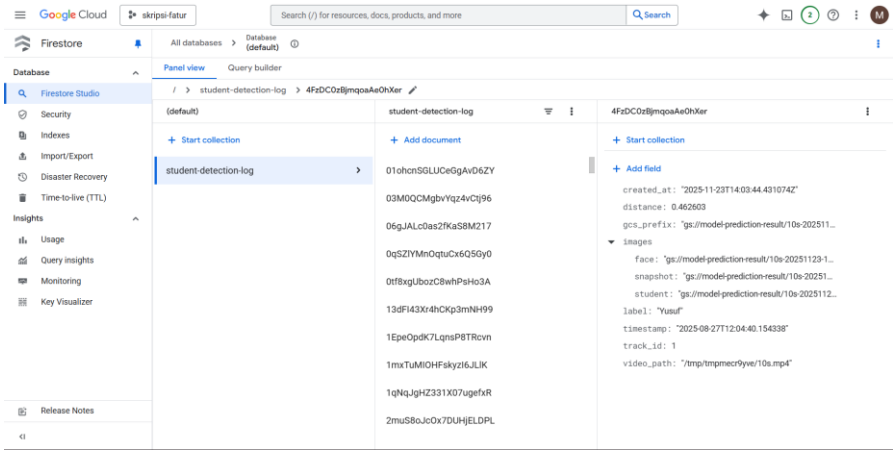
Pada tahap ini, model hasil pelatihan dari tahapan Vertex AI Custom Jobs digunakan untuk membangun layanan inferensi yang dapat diakses. Proses ini diimplementasikan menggunakan FastAPI yang berfungsi menyediakan *endpoint* untuk melakukan prediksi. Aplikasi FastAPI tersebut dikemas ke dalam Docker *image* agar lingkungan eksekusi model tetap konsisten antara proses pengembangan dan *deployment*. Setelah proses *containerization* selesai, *image* hasil *build* diunggah ke Google Artifact Registry. Selanjutnya, *image* tersebut dijalankan pada layanan Google Cloud Run, yang menangani inferensi.

Layanan ini memanfaatkan *file* model hasil pelatihan sebelumnya, yaitu best.pt, yang diambil berdasarkan *path* yang disimpan pada file latest.json di Google Cloud Storage. *File* latest.json berfungsi sebagai *pointer* dinamis untuk memastikan sistem selalu memuat versi model terbaru dengan performa terbaik. Dengan arsitektur ini, proses *deployment* model dapat dilakukan secara otomatis setiap kali model baru dilatih, tanpa perlu menghentikan layanan yang sedang berjalan.



Gambar 15. *Prediction service model*

6. Pencatatan Data
   Tahapan pencatatan data berfungsi untuk mencatat seluruh aktivitas inferensi yang terjadi di layanan prediksi, termasuk *input* data dan hasil prediksi. Data log ini digunakan untuk memantau performa model secara berkelanjutan dan mendeteksi potensi *data drift* yang dapat menyebabkan penurunan akurasi model.



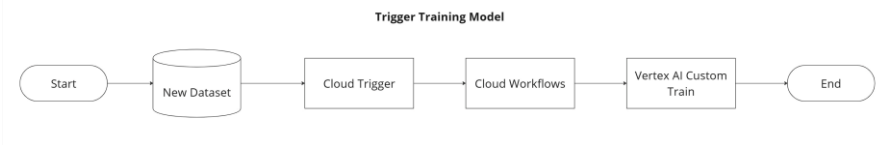
Gambar 16. *Data logging* pada Cloud Firestore

2.5.2 Tahap Pengujian MLOps Pipeline Automation

1. *Input Dataset* Terbaru
   Tahapan ini adalah tahap awal pengumpulan *dataset* baru yang akan digunakan untuk melatih ulang model visi komputer. *Dataset* yang dimasukkan berupa citra siswa dan wajah siswa yang diambil dari kamera sekolah, pendaftaran siswa baru, atau hasil peningkatan kualitas dataset lama.

2. Menyimpan *Dataset*
   Dataset yang dikumpulkan berupa citra wajah siswa dan citra gerbang sekolah yang diperoleh dari berbagai sumber, seperti kamera pengawas di sekolah, proses pendaftaran siswa baru, maupun hasil peningkatan kualitas *dataset* sebelumnya. Seluruh citra wajah tersebut terlebih dahulu dikompilasi ke dalam satu berkas arsip berformat .zip. *File* .zip ini kemudian diunggah ke Google Cloud Storage (GCS) sebagai lokasi penyimpanan utama *dataset* pelatihan. Proses ini menjadi titik awal dalam *pipeline* MLOps, di mana penyimpanan *file dataset* ke GCS secara otomatis memicu *event trigger* untuk menjalankan *pipeline custom jobs* di Vertex AI.

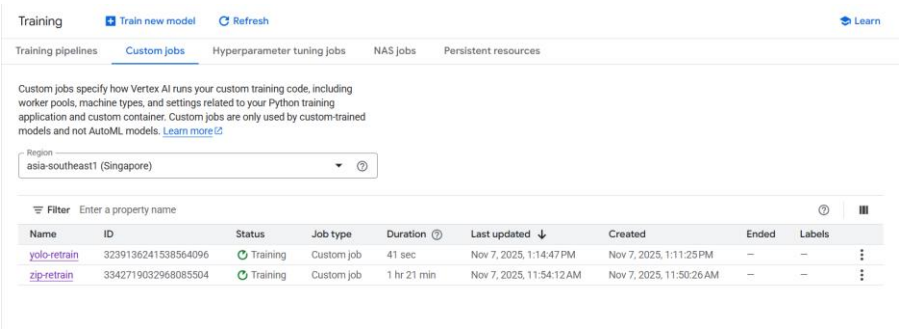
3. Pemicu Pelatihan Model



Gambar 17. *Flowchart trigger training*

Tahapan ini merupakan komponen inti dalam mekanisme otomatisasi pipeline MLOps yang mengatur proses *retraining* model berbasis *event-driven*. Pada penelitian ini, proses pemicu pelatihan ulang diimplementasikan menggunakan kombinasi Google Cloud Workflows dan Cloud Trigger. Ketika sistem mendeteksi adanya *dataset* baru yang diunggah ke Google Cloud Storage (GCS), maka Cloud Trigger akan mengeksekusi alur kerja yang telah didefinisikan di Cloud Workflows. Alur kerja ini kemudian memanggil API Custom Train untuk memulai proses pelatihan ulang model di Vertex AI.

Kode pelatihan yang digunakan dalam proses ini disimpan dalam bentuk docker *image di* Google Cloud Artifact Registry. Pendekatan ini memungkinkan proses retraining berjalan secara otomatis dan konsisten tanpa intervensi manual. Dengan arsitektur *event-driven* seperti ini, sistem menjadi adaptif terhadap perubahan data di lapangan seperti penambahan dataset sehingga model visi komputer selalu diperbarui dengan versi terbaru yang lebih relevan dan akurat untuk digunakan dalam sistem pengawasan *real-time*.



Gambar 18. Vertex AI Custom Jobs

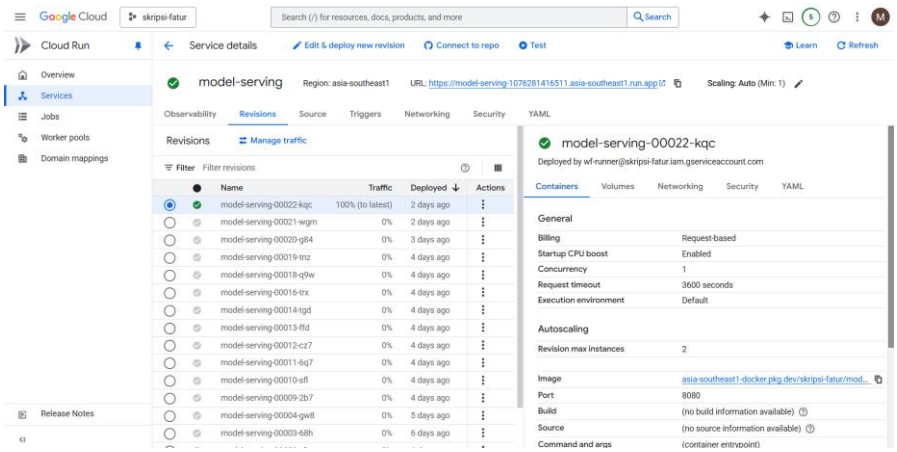
4. Menyimpan Model Terbaru

Tahapan ini merupakan proses penyimpanan hasil pelatihan ulang yang dijalankan setelah *pipeline retraining* dipicu oleh *dataset* baru. Pada tahap ini, seluruh artefak model terbaru termasuk file bobot model (best.pt), log hasil evaluasi, dan konfigurasi *training* disimpan kembali ke Google Cloud Storage (GCS) sebagai repositori utama. Selain itu, *pipeline* secara otomatis memperbarui *file* latest.json, yang berisi *path* terbaru menuju model hasil *retraining*. Pembaruan *file* ini berfungsi sebagai indikator bahwa versi model baru telah tersedia dan siap digunakan oleh layanan prediksi. Dengan mekanisme ini, *prediction service* yang berjalan di Cloud Run selalu dapat memuat model terbaru secara otomatis tanpa intervensi manual.

5. Penerapan Layanan Prediksi Versi Terbaru

   Tahapan ini merupakan proses otomatis untuk memperbarui layanan prediksi agar selalu menggunakan model terbaru yang telah dihasilkan pada tahap *retraining*. Setelah proses pelatihan selesai dan artefak model seperti best.pt serta *file* latest.json diperbarui di Google Cloud Storage, sistem menggunakan kombinasi Cloud Trigger dan Cloud Workflows untuk mendeteksi perubahan tersebut. *File* latest.json berfungsi sebagai penanda versi model terbaru, sehingga setiap kali *file* ini berubah, Cloud Trigger akan memicu *workflow* yang telah didefinisikan untuk melakukan proses pembaruan layanan prediksi. Alur kerja dalam Cloud Workflows kemudian menginisiasi proses *redeploy* Cloud Run service dengan menarik ulang konfigurasi dan memuat path model terbaru yang tercantum dalam latest.json.

   Layanan prediksi yang dijalankan menggunakan FastAPI dan dikemas dalam Docker *image* akan menjalani proses *redeploy service* tanpa menghentikan layanan yang sedang berjalan. Selama proses ini, Google Cloud Run akan memuat ulang model dengan mengambil best.pt dari lokasi terbaru di GCS sehingga layanan inferensi selalu menggunakan model terbaru yang paling akurat. Dengan pendekatan ini, sistem mampu mempertahankan kualitas prediksi secara konsisten dan memastikan setiap pembaruan model hasil *retraining* dapat langsung digunakan di lingkungan produksi tanpa intervensi manual. Proses ini merupakan penerapan penting dalam *MLOps Pipeline Automation*, karena memungkinkan *continuous deployment* model secara cepat, aman, dan terukur.

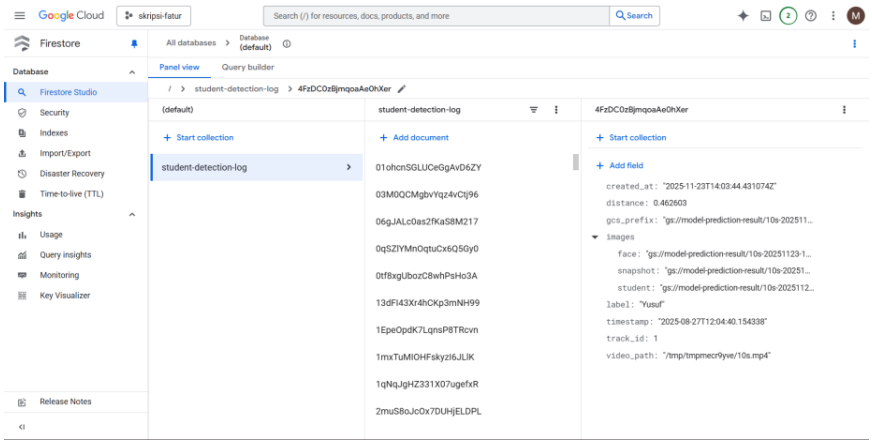


Gambar 19. *Deploying* versi terbaru *prediction service*

6. Pencatatan Data

   Tahapan *data logging* mencatat hasil inferensi yang dilakukan oleh sistem terhadap *input* video. Informasi yang dicatat mencakup waktu deteksi, identitas siswa yang dikenali, dan tingkat kepercayaan model. Data log ini kemudian digunakan untuk mengukur kualitas layanan inferensi dan mendeteksi adanya *model drift* atau *data drift* yang dapat menurunkan akurasi model seiring waktu.

Proses pencatatan ini juga menjadi dasar evaluasi keberhasilan *pipeline* MLOps dalam menjaga kestabilan performa sistem selama beroperasi.



Gambar 20. Data *logging* cloud firestore