PENGEMBANGAN SISTEM KLASIFIKASI BAHASA ISYARAT BISINDO SECARA REAL-TIME DENGAN RASPBERRY PI

TUGAS AKHIR

OLFAT HARITS ALATAS

312210018

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG
MALANG

2025

LEMBAR PENGESAHAN

TUGAS AKHIR

PENGEMBANGAN SISTEM KLASIFIKASI BAHASA ISYARAT BISINDO SECARA REAL-TIME DENGAN RASPBERRY PI

Oleh:

OLFAT HARITS ALATAS

NIM. 312210018

dari:

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI DAN DESAIN

UNIVERSITAS MA CHUNG

Telah dinyatakan lulus dalam melaksanakan Tugas Akhir sebagai syarat kelulusan dan berhak mendapatkan gelar Sarjana Komputer (S.Kom)

Dosen Pembimbing I	Dosen Pembimbing II
Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.	**Mochamad Subianto, S.Kom., M.Cs.**
NIP. 20070035	**NIP. 20100002**

Dekan Fakultas Teknologi dan Desain

Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.

NIP. 20070035

PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan dibawah ini :

Nama : Olfat Harits Alatas

NIM : 312210018

Program Studi : Teknik Informatika

Perguruan Tinggi : Universitas Ma Chung

Dengan ini menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**PENGEMBANGAN SISTEM KLASIFIKASI BAHASA ISYARAT BISINDO SECARA REAL-TIME DENGAN RASPBERRY PI**"adalah asli (orisinil) atau tidak plagiat dan benar hasil karya intelektual mandiri, dan belum pernah diterbitkan/dipublikasikan dimanapun dan dalam bentuk apapun.

Surat penyataan ini saya buat dengan sebenar-benarnya dengan kesaran sendiri dan tanpa ada paksaan dari pihak manapun. Apabila dikemudian hari diduga kuat ada ketidaksesuai antara fakta dengan dokumen pernyataan ini, saya bersedia diproses oleh Universitas Ma Chung, dengan sanksi terberat berupa pembatalan kelulusan atau pencabutan sarjana.

Malang, 14 Januari 2026

Olfat Harits Alatas

NIM. 312210018

KATA PENGANTAR

Puji dan syukur Penulis panjatkan ke hadirat Allah SWT, oleh karena anugerah-Nya yang melimpah, kemurahan dan kasih sayang-Nya yang besar, senantiasa menjadi penolong bagi Penulis sepanjang hidupnya. Hanya karena kebaikan dan ridho-Nya lah yang menuntun Penulis dalam mengerjakan laporan proyek tugas akhir dengan judul "PENGEMBANGAN SISTEM KLASIFIKASI BAHASA ISYARAT BISINDO SECARA REAL-TIME DENGAN RASPBERRY PI", sehingga dapat diselesaikan dengan baik.

Laporan proyek tugas akhir ini disusun untuk memenuhi salah satu syarat guna memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika Universitas Ma Chung. Penulis berharap dengan dibuatnya laporan ini dapat memperluas pengetahuan dan wawasan bagi para pembaca.

Penulis menyadari sepenuhnya bahwa usulan proposal penelitian ini masih jauh dari kesempurnaan karena segala keterbatasan yang ada. Dengan tersusunnya laporan proyek tugas akhir ini, Penulis menyampaikan terima kasih kepada para pihak yang telah memberi dukungan kepada Penulis baik secara moril maupun materil, di antaranya yang terhormat:

1. Bapak Dr. Ir. Stefanus Yufra Manahen Taneo, M.S., M.Sc., selaku Rektor Universitas Ma Chung.
2. Bapak Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd., selaku Dekan Fakultas Teknologi dan Desain Universitas Ma Chung serta sekaligus sebagai Dosen Pembimbing 1, yang memberikan dukungan yang luar biasa, saran dan masukan dengan sangat baik dan bijaksana dalam menyempurnakan Laporan Tugas Akhir ini.
3. Bapak Hendry Setiawan, S.T., M.Kom, selaku Ketua Program Studi Teknik Informatika Universitas Ma Chung.

4. Bapak Mochamad Subianto, S.Kom., M.Cs. selaku Dosen Pembimbing 2, yang memberikan saran dan masukan dengan sangat baik dan bijaksana dalam menyempurnakan Laporan Tugas Akhir ini.
5. Bapak Windra Swastika, S.Kom., MT., Ph.D. selaku Dosen Penguji yang telah memberikan saran terhadap Laporan Tugas Akhir ini.
6. Keluarga Penulis yang telah memberikan doa, kasih sayang, dorongan, semangat, serta motivasi kepada Penulis dalam berbagai pihak.
7. Devinda Gusmananda, S.Farm, terima kasih karena telah menjadi *support system* terbaik, yang senantiasa mendampingi, mendengarkan keluh kesah, serta memberikan dukungan moril dan semangat tanpa henti di samping Penulis selama proses pengerjaan Tugas Akhir ini.
8. Seluruh Partisipan Penelitian yang tidak dapat Penulis sebutkan namanya satu per satu. Terima kasih yang sebesar-besarnya atas kesediaan waktu dan kerjasamanya dalam proses pengambilan data, yang menjadi pondasi utama dalam pembentukan *dataset* penelitian ini.
9. Teman-teman angkatan 2022 yang telah menemani Penulis dalam lika-liku perkuliahan.

Semoga Allah SWT senantiasa memberikan rahmat dan berkah-Nya atas ketulusan dan kebaikan yang telah diberikan kepada Penulis. Demikian dan semoga Laporan Tugas Akhir ini dapat bermanfaat bagi pembaca.

Malang, 9 Januari 2026

Penulis

(Olfat Harits Alatas)

PENGEMBANGAN SISTEM KLASIFIKASI BAHASA ISYARAT BISINDO SECARA REAL-TIME DENGAN RASPBERRY PI

Olfat Harits Alatas, Romy Budhi, Mochamad Subianto

Universitas Ma Chung

ABSTRAK

Komunikasi merupakan kebutuhan fundamental manusia, namun kesenjangan komunikasi masih sering terjadi antara masyarakat dengar dan komunitas Tuli yang menggunakan Bahasa Isyarat Indonesia (BISINDO). Solusi teknologi yang ada saat ini seringkali bergantung pada perangkat keras mahal atau komputasi cloud yang tidak praktis untuk penggunaan sehari-hari. Penelitian ini bertujuan untuk mengembangkan sistem penerjemah bahasa isyarat portable dan *real-time* berbasis embedded system menggunakan *Raspberry Pi 5*.

Penelitian ini menerapkan pendekatan *hybrid* dalam mengklasifikasikan gestur. Untuk gestur statis (huruf, angka dan kata statis), digunakan algoritma *Random Forest* karena efisiensinya pada data tabular. Sedangkan untuk gestur dinamis, dilakukan studi perbandingan antara arsitektur *Long Short-Term Memory* (LSTM) dan *Transformer* untuk menangkap dependensi spasio-temporal. Ekstraksi fitur dilakukan menggunakan *MediaPipe Hands* yang menghasilkan 21 titik koordinat kerangka tangan (*landmarks*).

Hasil pengujian membuktikan bahwa pendekatan *hybrid* sangat efektif. Pada gestur statis, *Random Forest* mencatatkan performa sempurna dengan akurasi 100%. Temuan signifikan terlihat pada klasifikasi gestur dinamis, di mana arsitektur *Transformer* berhasil mengungguli LSTM dengan akurasi uji 98,57% berbanding 94,50%. Keunggulan ini semakin teruji pada validasi *real-time*, di mana *Transformer* mampu mempertahankan stabilitas prediksi dengan akurasi 93%, jauh melampaui LSTM yang hanya mencapai 86%. Hal ini menunjukkan bahwa mekanisme *Self-Attention* pada *Transformer* lebih efektif dalam menangkap konteks spasio-temporal jangka panjang dibandingkan gerbang memori LSTM, menjadikan sistem ini solusi yang lebih andal untuk implementasi di dunia nyata.

Kata Kunci: BISINDO, *Raspberry Pi*, *Random Forest*, LSTM, *Transformer*, *Computer Vision*.

DEVELOPMENT OF A REAL-TIME BISINDO SIGN LANGUAGE CLASSIFICATION SYSTEM USING RASPBERRY PI

Olfat Harits Alatas, Romy Budhi, Mochamad Subianto

Universitas Ma Chung

ABSTRACT

Communication is a fundamental human need, however, a significant communication gap persists between the hearing society and the Deaf community who utilize Indonesian Sign Language (BISINDO). Current technological solutions often rely on expensive hardware or cloud computing, rendering them impractical for daily usage. This study aims to develop a portable, real-time sign language translation system based on an embedded system utilizing Raspberry Pi 5.

This research implements a hybrid approach for gesture classification. For static gestures (letters, numbers, and static words), the Random Forest algorithm is employed due to its efficiency with tabular data. Meanwhile, for dynamic gestures, a comparative study is conducted between Long Short-Term Memory (LSTM) and Transformer architectures to capture spatiotemporal dependencies. Feature extraction utilizes MediaPipe Hands, generating 21 hand skeletal coordinate points (landmarks).

Experimental results demonstrate that the hybrid approach is highly effective. In static gesture classification, Random Forest achieved perfect performance with 100% accuracy. Significant findings emerged in dynamic gesture classification, where the Transformer architecture successfully outperformed LSTM with a test accuracy of 98.57% compared to 94.50%. This superiority was further validated in real-time testing, where Transformer maintained prediction stability with 93% accuracy, significantly surpassing LSTM which only reached 86%. These results indicate that the Self-Attention mechanism in the Transformer is more effective in capturing long-term spatiotemporal contexts compared to LSTM's memory gates, rendering this system a more reliable solution for real-world implementation.

Keywords: BISINDO, *Embedded System*, *Random Forest*, LSTM, *Transformer*, *Spatiotemporal*.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR PERSAMAAN

Bab I
Pendahuluan

1.1. Latar Belakang

Bahasa Isyarat Indonesia (BISINDO) merupakan media komunikasi utama bagi penyandang tunarungu dan tunawicara di Indonesia. Sebagai bahasa visual, BISINDO memanfaatkan kombinasi gerakan tangan, ekspresi wajah, dan posisi tubuh, sehingga sering kali sulit dipahami oleh masyarakat umum yang tidak terbiasa menggunakannya. Kondisi ini kerap menjadi hambatan komunikasi antara penyandang disabilitas dengan lingkungan sosialnya. Di Indonesia, dikenal dua sistem bahasa isyarat, yaitu Sistem Isyarat Bahasa Indonesia (SIBI) dan BISINDO. Namun, BISINDO lebih banyak digunakan karena berkembang secara alami di komunitas tunarungu, bersifat kultural, serta tidak terikat pada struktur bahasa Indonesia formal. Mengingat cakupan penggunaannya yang lebih luas dan praktis, penelitian ini memilih BISINDO sebagai fokus utama dalam pengembangan sistem penerjemah bahasa isyarat otomatis untuk mendukung komunikasi yang lebih inklusif.

Kemajuan teknologi kecerdasan buatan, khususnya machine learning, telah memungkinkan komputer mengenali pola dari data secara efektif. Salah satu penerapannya adalah pada sistem pengenalan bahasa isyarat berbasis citra. Sejumlah studi menunjukkan keberhasilan algoritma machine learning seperti Random Forest dan Long Short-Term Memory (LSTM) dalam mengklasifikasikan gerakan tangan. Penelitian Fadlilah et al. (2022) berhasil membangun sistem pengenal isyarat dasar BISINDO menggunakan kamera dan Raspberry Pi yang mampu menerjemahkan huruf dan angka menjadi teks secara *real-time*.

Berbagai pendekatan teknologi telah dikembangkan untuk menjembatani kesenjangan komunikasi ini. Salah satu metode yang umum digunakan adalah pendekatan berbasis sensor (*wearable devices*). Sebagai contoh, penelitian yang dilakukan oleh Wungow dkk (2022) mengembangkan sarung tangan penerjemah untuk Sistem Bahasa Isyarat Indonesia (SIBI) menggunakan *flex sensor* yang dikombinasikan dengan metode *K-Nearest Neighbors* (KNN) dan *Artificial Neural Network* (ANN).

Meskipun penelitian tersebut berhasil mencapai akurasi tinggi hingga 99% untuk klasifikasi statis, penggunaan perangkat keras yang harus dipasang pada tubuh pengguna dinilai kurang praktis untuk komunikasi sehari-hari. Selain itu, akurasi pada metode berbasis sensor sangat bergantung pada kesesuaian dimensi tangan pengguna dengan alat, di mana perbedaan ukuran tangan dapat menyebabkan sensor tidak menekuk secara maksimal. Mengatasi keterbatasan tersebut, penelitian ini mengusulkan pendekatan berbasis *computer vision* yang lebih fleksibel tanpa memerlukan alat tambahan pada tubuh pengguna. Berbeda dengan penelitian sebelumnya yang berfokus pada SIBI, penelitian ini akan berfokus pada BISINDO yang lebih umum digunakan oleh komunitas Tuli, dengan memanfaatkan arsitektur *Deep Learning.*

Bahasa isyarat tidak hanya terdiri dari gestur statis, tetapi juga gestur dinamis yang membentuk kata atau kalimat. Untuk mengenali urutan gerakan dalam dimensi waktu, dibutuhkan model yang mampu menangani data sekuensial. LSTM, sebagai bagian dari arsitektur *Recurrent Neural Network* (RNN), terbukti efektif untuk memproses data urutan. Penelitian Aljabar dan Suharjito (2020) menggabungkan CNN dan LSTM untuk pengenalan BISINDO *real-time* berbasis *desktop*, dengan capaian akurasi hingga 96% dalam pengenalan kata tertentu.

Penelitian Kothadiya et al. (2022) mengusulkan sistem pengenalan bahasa isyarat berbasis *deep learning* dengan mengombinasikan LSTM dan GRU. Dataset yang digunakan, IISL2020, berisi sebelas kelas kata dengan ribuan sampel video yang direkam dalam kondisi alami tanpa sensor tambahan. Fitur citra diekstraksi menggunakan InceptionResNetV2, kemudian diproses oleh LSTM untuk memahami pola urutan gerakan dan dilanjutkan ke GRU guna menyaring informasi sebelum diklasifikasikan dengan *softmax*. Model ini dijalankan pada desktop dan mencapai akurasi sekitar 97%, meskipun masih terbatas pada pengenalan gestur terisolasi.

Raspberry Pi menjadi platform yang menarik untuk pengembangan sistem ini karena memiliki keunggulan dari sisi biaya, ukuran yang ringkas, dan portabilitas. Integrasinya dengan kamera CSI (PiCamera) memungkinkan pengambilan citra berkecepatan tinggi dengan latensi rendah, yang sangat penting untuk aplikasi *real-time*. Abed dan Rahman (2017) menunjukkan bahwa Raspberry Pi dengan modul kamera

dapat digunakan untuk sistem pengenalan gestur tangan berbasis visi komputer dengan akurasi hingga 98%, sekaligus menawarkan efisiensi yang tinggi untuk aplikasi lapangan.

Hasil penelitian Alexander dkk. (2023) mengungkapkan bahwa algoritma *Random Forest* menunjukkan performa terbaik dibandingkan metode klasifikasi lain, seperti *K-Nearest Neigh*bor (KNN), *Support Vector Machine* (SVM), dan *Decision Tree*, dalam mengidentifikasi gestur bahasa isyarat BISINDO. Pada pengujian data, *Random Forest* mampu mencapai akurasi, presisi, f1-score, dan recall sebesar 97,9%, sedangkan pada pengujian *real-time* memperoleh presisi 84%. Keunggulan lain yang dicapai adalah waktu klasifikasi tercepat di antara model yang diuji, yaitu 34,6 kata per menit, sehingga menjadikannya pilihan yang tepat untuk aplikasi berbasis kamera dengan kebutuhan *real-time*.

Selain itu, optimalisasi model menggunakan *TensorFlow Lite* pada perangkat *edge computing* seperti Raspberry Pi telah terbukti dapat meningkatkan efisiensi proses inferensi sekaligus mempertahankan kinerja model untuk aplikasi *real-time*. Toyib et al. (2025) mengimplementasikan *TensorFlow Lite* pada aplikasi pengenalan Sistem Isyarat Bahasa Indonesia (SIBI) secara *real-time*, dan hasilnya menunjukkan performa sistem yang responsif dengan konsumsi sumber daya yang rendah.

Penelitian oleh Alatas & Widodo (2025) mengembangkan sistem penerjemah bahasa isyarat BISINDO menggunakan kombinasi algoritma *Random Forest* untuk gestur statis dan LSTM untuk gestur dinamis pada *Raspberry Pi*. Hasil penelitian tersebut menunjukkan bahwa sistem mampu berjalan pada perangkat *edge* dengan tingkat akurasi yang baik, namun akurasi *real-time* pada model LSTM masih dapat ditingkatkan. Hal ini menjadi landasan utama pengembangan penelitian ini untuk mengoptimalkan akurasi *real-time* melalui pemanfaatan *TensorFlow Lite* dan pengambilan citra menggunakan *PiCamera*.

Penelitian oleh Hoque et al. (2018) mengembangkan sistem deteksi bahasa isyarat Bangladeshi (BdSL) menggunakan algoritma Faster R-CNN dengan backbone Inception V2. Sistem dilatih pada dataset BdSLImset yang berisi sepuluh kelas huruf isyarat dengan variasi latar belakang dan kondisi pencahayaan. Hasil pengujian menunjukkan bahwa model mampu mencapai akurasi rata-rata sebesar 98,2% dengan

waktu deteksi sekitar 90 milidetik per citra, sehingga dapat berjalan secara *real-time*. Meskipun demikian, penelitian ini mencatat adanya kendala pada huruf dengan bentuk isyarat yang mirip, sehingga akurasi pada kondisi tertentu masih dapat ditingkatkan.

Penelitian oleh Shin et al. (2023) mengembangkan sistem pengenalan bahasa isyarat Korea (KSL) menggunakan pendekatan *deep learning* berbasis transformer ringan. Model yang diusulkan memadukan keunggulan CNN untuk ekstraksi fitur lokal dan *Convolutional Layer-Based* Transformer dengan *Lightweight Multi-Head Self Attention* (LMHSA) untuk menangkap ketergantungan global, serta diperkuat dengan *grain module* yang berfungsi menggantikan mekanisme patch tradisional pada *Vision Transformer* agar representasi awal lebih efektif dan efisien. Sistem dilatih pada dataset KSL dengan 77 kelas serta dataset tambahan berisi 20 kata penting dalam KSL. Hasil pengujian menunjukkan bahwa model mencapai akurasi 89,0% pada dataset KSL dan 98,3% pada dataset usulan, melampaui performa metode sebelumnya. Meskipun demikian, penelitian ini mencatat masih adanya keterbatasan pada ukuran dataset dan kompleksitas komputasi, sehingga pengembangan lebih lanjut diperlukan untuk meningkatkan generalisasi dan efisiensi sistem.

Penelitian oleh Chaudhary et al (2022) melalui model SignNet II menunjukkan kinerja yang unggul secara kuantitatif dibandingkan pendekatan berbasis RNN dan LSTM. Pada tugas *sign-to-text translation*, model ini mampu mencapai skor BLEU hingga sekitar 23, lebih tinggi 4–6 poin dibanding baseline sebelumnya yang hanya berkisar 15–18. Sementara pada arah *text-to-sign translation*, SignNet II menghasilkan peningkatan akurasi urutan gloss sebesar 4% hingga 7% dibandingkan metode terdahulu. Selain itu, penggunaan mekanisme *shared representation* menjadikan model ini lebih efisien, dengan jumlah parameter dan waktu pelatihan yang lebih hemat sekitar 10–15% dibandingkan jika melatih dua model terpisah. Hasil ini menegaskan bahwa arsitektur Transformer tidak hanya lebih akurat, tetapi juga lebih efisien untuk tugas penerjemahan dua arah bahasa isyarat, meskipun tantangan terkait keterbatasan data dan kebutuhan evaluasi lebih luas masih perlu ditangani dalam penelitian lanjutan.

Berdasarkan temuan-temuan tersebut, penelitian ini mengembangkan sistem klasifikasi bahasa isyarat BISINDO *real-time* berbasis Raspberry Pi dengan

memanfaatkan kombinasi algoritma *Random Forest* untuk gestur statis, serta LSTM dan *Transformer* yang dioptimasi menggunakan *TensorFlow Lite* untuk gestur dinamis. Penggunaan *Pi Camera* memungkinkan pengambilan citra berkecepatan tinggi, sementara kombinasi kedua algoritma diharapkan dapat meningkatkan akurasi pengenalan baik pada gestur statis maupun dinamis dalam kondisi penggunaan di lapangan. Selain itu, penelitian ini juga menghadirkan pembaruan berupa penerapan metode *Transformer* khusus untuk pengenalan gestur dinamis, sehingga dapat dibandingkan dengan LSTM dalam mendeteksi urutan gerakan bahasa isyarat serta memberikan evaluasi terhadap efektivitas *Transformer* sebagai alternatif metode terkini. Pemilihan *Transformer* dibandingkan GRU dan CNN didasarkan pada kemampuannya dalam menangkap ketergantungan jangka panjang secara lebih efektif melalui mekanisme *self-attention*, tanpa terjebak pada keterbatasan memori yang sering muncul pada RNN maupun variannya. Selain itu, Transformer memungkinkan pemrosesan paralel yang lebih cepat dibandingkan model sekuensial seperti GRU, sekaligus mampu mengintegrasikan informasi spasial dan temporal lebih baik daripada CNN yang umumnya berfokus pada pola lokal. Dengan demikian, *Transformer* dipandang sebagai alternatif yang menjanjikan untuk meningkatkan akurasi dan efisiensi dalam pengenalan gestur dinamis BISINDO secara *real-time*.

Arsitektur *Deep Learning* seperti *Convolutional Neural Networks* (CNN) dan *Recurrent Neural Networks* (RNN), khususnya LSTM, telah lama menjadi pilihan utama untuk tugas-tugas klasifikasi dan prediksi pada data sekuensial. Namun, kemunculan arsitektur *Transformer* telah mengubah lanskap secara signifikan dan menjadi fokus utama dalam pengembangan model pemrosesan data, termasuk untuk data *audio*. Sebuah studi komparatif dilakukan untuk menguji kinerja ketiga arsitektur ini (CNN, RNN-LSTM, dan Transformer) dalam tugas klasifikasi genre musik menggunakan *Mel-Frequency Cepstrum Coefficients* (MFCCs) sebagai fitur masukan (Ali, 2024).

Studi tersebut menyoroti perbedaan kinerja yang signifikan antar model ketika dihadapkan pada keterbatasan sumber daya komputasi. Model RNN-LSTM menunjukkan kinerja terendah, dengan akurasi yang stagnan (*flat line*) di sekitar 66% dan tidak menunjukkan tanda-tanda perbaikan meski epoch pelatihan ditambah. Model

CNN, di sisi lain, terbukti paling efektif untuk skenario waktu atau sumber daya yang terbatas, dengan cepat mencapai akurasi puncak sekitar 75% sebelum akhirnya juga mengalami stagnasi. Berbeda dari keduanya, model *Transformer* menunjukkan karakteristik sebagai "pembelajar yang lambat namun konsisten". Meskipun akurasi awalnya lebih rendah, model *Transformer* menunjukkan pertumbuhan performa yang terus-menerus dan stabil tanpa mengalami plateau, bahkan ketika durasi pelatihan diperpanjang. Temuan ini mendukung kesimpulan bahwa dengan sumber daya dan data yang memadai, arsitektur Transformer memiliki potensi untuk melampaui kinerja model-model pendahulunya secara signifikan (Ali, 2024)

Arsitektur *Transformer* tidak hanya menggantikan RNN dalam tugas pemrosesan bahasa alami, tetapi juga telah diuji secara intensif untuk aplikasi pemrosesan ucapan (*speech applications*), seperti *Automatic Speech Recognition* (ASR), *Speech Translation* (ST), dan *Text-to-Speech* (TTS). Perbedaan fundamentalnya adalah bahwa *Transformer* mempelajari informasi sekuensial melalui mekanisme *self-attention*, sedangkan RNN bergantung pada koneksi rekuren (Karita et al., 2019).

Dalam sebuah studi komparatif berskala besar yang membandingkan kedua arsitektur pada 15 *benchmark* ASR yang berbeda, *Transformer* menunjukkan superioritas yang mengejutkan dengan mengungguli RNN pada 13 dari 15 *benchmark* tersebut. Keunggulan ini juga terlihat jelas dalam efisiensi pelatihan. Karena tidak bergantung pada operasi sekuensial yang iteratif seperti RNN, pelatihan *Transformer* dapat diparalelkan sepenuhnya. Hasilnya, pada salah satu *benchmark* (LibriSpeech), model *Transformer* mampu mencapai tingkat akurasi terbaik yang dihasilkan RNN, namun dengan waktu pelatihan delapan kali lebih cepat (Karita et al., 2019).

Studi ini juga mencatat bahwa strategi optimalisasi untuk kedua model ini berbeda. Kinerja *Transformer* sangat diuntungkan oleh penggunaan *minibatch* berukuran besar, yang secara simultan meningkatkan akurasi dan kecepatan pelatihan. Sebaliknya, peningkatan ukuran *minibatch* tidak memberikan manfaat yang sama pada model RNN. Selain itu, penggunaan *dropout* terbukti esensial untuk mencegah *overfitting* pada *Transformer*, sementara teknik yang sama tidak menunjukkan peningkatan signifikan pada RNN. Temuan ini menggarisbawahi bahwa *Transformer* tidak hanya unggul dalam

performa, tetapi juga memperkenalkan dinamika pelatihan yang berbeda dari arsitektur berbasis rekuren (Karita et al., 2019).

Evolusi model *sequence-to-sequence* menunjukkan lintasan yang jelas, dimulai dari *Recurrent Neural Networks* (RNNs) sebagai fondasi, beralih ke *Long Short-Term Memory* (LSTM) untuk mengatasi kelemahan RNN, dan akhirnya mencapai pergeseran paradigma dengan hadirnya arsitektur *Transformer*. Model RNN tradisional, meskipun fundamental, memiliki keterbatasan dalam menangani sekuens panjang akibat masalah *vanishing gradient*. Model LSTM dan variannya (seperti GRU) secara khusus dirancang untuk mengatasi masalah ini, sehingga unggul dalam tugas-tugas yang menuntut memori dependensi jangka panjang dan pemahaman struktur sintaktis yang kompleks (Zhu, 2023).

Di sisi lain, arsitektur *Transformer* membawa kemajuan baru dengan berfokus pada mekanisme attention. *Transformer* terbukti lebih unggul untuk tugas-tugas yang membutuhkan pemahaman konteks yang mendalam dan hubungan antara bagian-bagian teks yang tidak berdekatan. Meskipun demikian, model LSTM masih mempertahankan relevansinya untuk skenario spesifik. Ketika sebuah tugas sangat bergantung pada analisis sintaktis yang mendalam atau membutuhkan interpretasi model yang lebih transparan, arsitektur LSTM seringkali masih menjadi pilihan yang valid. Pada akhirnya, pemilihan antara arsitektur berbasis rekuren (LSTM) dan berbasis *attention* (*Transformer*) sangat bergantung pada kebutuhan spesifik dari tugas yang dihadapi (Zhu, 2023).

Arsitektur *Transformer*, yang diperkenalkan oleh Vaswani et al. pada tahun 2017, telah menjadi model fundamental dalam *deep learning*, khususnya untuk data sekuensial. Perbedaan utamanya dengan model-model klasik seperti *Recurrent Neural Network* (RNN) adalah pada metode pemrosesan sekuens. Model RNN bersifat sekuensial dan iteratif, yang menyebabkan proses pelatihannya memakan waktu lama. Sebaliknya, *Transformer* mengganti mekanisme rekuren tersebut sepenuhnya dengan *self-attention*, yang memungkinkan pelatihan dilakukan secara paralel. Kemampuan ini secara signifikan meningkatkan efisiensi komputasi dan mempersingkat waktu yang dibutuhkan untuk melatih model (Shiri et al., 2024).

Keunggulan ini telah dibuktikan secara empiris dalam sebuah studi komparatif. Pada tugas analisis sentimen (dataset IMDB), model *Transformer* tidak hanya berhasil mencapai akurasi klasifikasi tertinggi, mengungguli varian RNN seperti LSTM dan GRU, tetapi juga memiliki waktu pelatihan yang jauh lebih singkat dibandingkan model RNN berkinerja terbaik lainnya (Bi-GRU) (Shiri et al., 2024). Performa superior ini juga terkonfirmasi pada jenis data sekuensial yang berbeda, yaitu data sensor untuk pengenalan aktivitas manusia (dataset ARAS). Pada dataset tersebut, *Transformer* kembali mengungguli model RNN lainnya dalam hal akurasi, *recall*, dan *F1-score*, sekaligus menunjukkan kurva pelatihan yang lebih cepat stabil (Shiri et al., 2024). Studi ini menyimpulkan bahwa mekanisme *attention* pada *Transformer* menjadikannya model yang berkinerja lebih baik daripada model-model berbasis RNN klasik, terutama untuk tugas analisis teks (Shiri et al., 2024).

1.2. Identifikasi Masalah

Permasalahan yang diangkat dalam penelitian ini meliputi rendahnya pemahaman masyarakat terhadap Bahasa Isyarat Indonesia (BISINDO), belum tersedianya sistem penerjemah bahasa isyarat yang praktis dan *real-time* pada perangkat portabel seperti *Raspberry Pi*, serta perlunya penerapan metode klasifikasi terpisah untuk gestur statis dan gestur dinamis menggunakan algoritma yang sesuai.

1.3. Batasan Masalah

a) Sistem difokuskan untuk mengenali gestur Bahasa Isyarat Indonesia (BISINDO) berupa huruf, angka, dan kosakata tertentu sesuai dataset penelitian.
b) Proses pengambilan data menggunakan webcam *Logitech C270* dan pengujian dilakukan menggunakan *PiCamera v1* yang terhubung ke *Raspberry Pi 5* sebagai perangkat utama.
c) Model klasifikasi yang digunakan terdiri dari *Random Forest* untuk gestur statis serta LSTM dan *Transformer* untuk gestur dinamis yang dibandingkan kinerjanya.

d) Dataset berupa citra dan data *landmark* tangan 3D (x, y, z) diperoleh menggunakan *MediaPipe Hands* dan digunakan untuk pelatihan serta pengujian sistem.
e) Sistem dirancang beroperasi secara *offline* pada *Raspberry Pi*, dengan seluruh proses inferensi dan klasifikasi dilakukan secara lokal tanpa koneksi internet.

1.4. Rumusan Masalah

Bagaimana merancang bangun dan menganalisis kinerja sistem klasifikasi bahasa isyarat BISINDO secara *real-time* pada *Raspberry Pi 5* yang mengintegrasikan algoritma *Random Forest* untuk gestur statis serta membandingkan performa LSTM dan Transformer untuk gestur dinamis guna mencapai akurasi dan stabilitas sistem yang optimal?

1.5. Tujuan Penelitian

a) Membangun sistem klasifikasi bahasa isyarat BISINDO berbasis kamera CSI *PiCamera v1* yang berjalan pada perangkat *Raspberry Pi 5* RAM 16 GB.
b) Menerapkan algoritma *Random Forest* untuk klasifikasi gestur statis bahasa isyarat BISINDO.
c) Membandingkan kinerja algoritma LSTM dengan metode *Transformer* dalam klasifikasi gestur dinamis bahasa isyarat BISINDO.
d) Menguji kemampuan sistem dalam melakukan klasifikasi bahasa isyarat BISINDO secara *real-time* pada perangkat *Raspberry Pi 5* RAM 16 GB dengan hasil yang akurat.

1.6. Luaran Penelitian

Luaran yang diharapkan dari penelitian ini adalah:

1. Rancangan dan implementasi sistem klasifikasi bahasa isyarat BISINDO berbasis *Raspberry Pi 5* dengan kamera CSI PiCamera v1 yang mampu melakukan deteksi dan klasifikasi gestur secara *real-time*.
2. Model klasifikasi gestur statis menggunakan algoritma *Random Forest* yang terlatih pada dataset BISINDO.

3. Perbandingan kinerja antara algoritma LSTM dan metode *Transformer* dalam mendeteksi gestur dinamis bahasa isyarat BISINDO secara *real-time* pada perangkat *edge*.
4. Dataset hasil ekstraksi landmark tangan dari *MediaPipe Hands* yang difokuskan pada deteksi tangan, digunakan sebagai data pelatihan dan pengujian sistem.
5. Hasil pengujian performa sistem yang mencakup pengukuran akurasi, presisi, *recall*, dan *f1-score*, baik untuk pengujian *offline* maupun *real-time*.
6. Analisis kinerja sistem pada perangkat *Raspberry Pi*, meliputi kecepatan pemrosesan, tingkat akurasi, dan efisiensi penggunaan sumber daya.

1.7. Manfaat

a) Memberikan solusi teknologi yang dapat membantu komunikasi antara penyandang tunarungu dan masyarakat umum melalui sistem penerjemah BISINDO *real-time* berbasis *Raspberry Pi*.
b) Menjadi referensi dalam pengembangan sistem pengenalan bahasa isyarat berbasis *Raspberry Pi* dan kamera CSI *PiCamera* dengan optimasi model menggunakan *TensorFlow Lite*.
c) Memberikan informasi dan perbandingan performa antara algoritma *Random Forest* untuk gestur statis, LSTM dan Transformer untuk gestur dinamis.
d) Mendorong pemanfaatan model *deep learning* berbasis sekuensial (LSTM) maupun *Transformer* pada aplikasi pengenalan gestur dinamis yang dijalankan di perangkat *edge computing*.

1.8. Sistematika Penulisan

Berikut ini adalah sistematika penulisan pada penelitian ini:

1. BAB I Pendahuluan

   Berisi uraian pendahuluan penelitian yang mencakup latar belakang, identifikasi masalah, batasan masalah, rumusan masalah, tujuan penelitian,

luaran penelitian, manfaat penelitian, sistematika penulisan, dan jadwal penelitian.

2. BAB II Tinjauan Pustaka

   Berisi teori-teori dasar dan ulasan penelitian terdahulu yang relevan dengan pengembangan sistem klasifikasi bahasa isyarat BISINDO secara *real-time* menggunakan *Raspberry Pi* dan kamera CSI *PiCamera v1*. Dalam bab ini juga dibahas literatur terkait penerapan algoritma *Random Forest* untuk gestur statis, LSTM dan *Transformer* yang dioptimasi dengan *TensorFlow Lite* untuk gestur dinamis sebagai pembaruan metode untuk pengenalan gestur dinamis. Selain itu, ditinjau pula penelitian-penelitian sebelumnya dalam bidang pengenalan bahasa isyarat untuk melihat posisi dan kebaruan penelitian ini.

3. BAB III Rancangan Sistem

   Berisi tahapan perancangan sistem mulai dari metode pengumpulan data, proses pengolahan data menggunakan *MediaPipe Hands*, perancangan model klasifikasi, optimasi model dengan *TensorFlow Lite*, serta integrasi sistem pada perangkat *Raspberry Pi*.

4. BAB IV Hasil dan Pembahasan

   Berisi hasil implementasi sistem dan pembahasan mendalam terkait performa algoritma, akurasi, presisi, *recall*, *f1-score*, serta evaluasi kinerja *real-time* pada *Raspberry Pi*.

5. BAB V Simpulan dan Saran

   Berisi simpulan menyeluruh dari hasil penelitian, kesesuaian dengan tujuan penelitian, serta saran-saran untuk pengembangan lebih lanjut agar sistem dapat digunakan secara optimal di lapangan.

Bab II
Tinjauan Pustaka

2.1. Bahasa Isyarat

Bahasa isyarat merupakan sistem komunikasi visual-manual yang digunakan oleh komunitas Tuli untuk menyampaikan informasi dan menjalin interaksi sosial. Bahasa ini memiliki struktur linguistik yang kompleks yang mencakup fonologi, morfologi, sintaksis, serta semantik, dan tidak bersifat universal karena tiap negara memiliki bahasa isyaratnya masing-masing (Pujiati, 2019). Di Indonesia, terdapat dua sistem yang dikenal luas, yaitu Bahasa Isyarat Indonesia (BISINDO) dan Sistem Isyarat Bahasa Indonesia (SIBI), yang memiliki perbedaan mendasar dalam struktur dan penggunaannya (Pujiati, 2019).

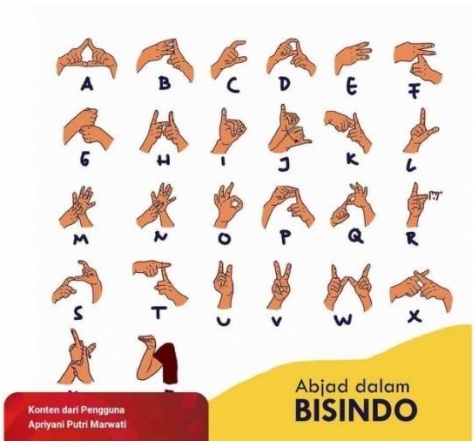
Selain sebagai alat komunikasi, bahasa isyarat juga merupakan bagian dari identitas budaya komunitas Tuli. Penggunaan bahasa isyarat memberikan akses yang lebih luas terhadap pendidikan, pekerjaan, dan kehidupan sosial secara umum. Namun, pengakuan resmi terhadap bahasa isyarat sebagai bahasa nasional dan penerapannya dalam kebijakan publik masih menjadi tantangan di berbagai negara, termasuk Indonesia (Wijaya, 2018).

Dalam konteks pendidikan, pemahaman dan penguasaan bahasa isyarat memiliki peranan penting dalam menciptakan lingkungan belajar yang inklusif bagi siswa Tuli. Implementasi pembelajaran dengan pendekatan bilingual menggunakan bahasa isyarat dan bahasa tulis mampu meningkatkan prestasi belajar dan perkembangan kognitif siswa Tuli (Murni et al., 2024). Selain itu, pelatihan bahasa isyarat bagi guru, teman sebaya, dan keluarga turut memperkuat dukungan sosial yang mereka terima.

Seiring dengan kemajuan teknologi, banyak inovasi digital yang mendukung pembelajaran bahasa isyarat, seperti aplikasi interaktif berbasis multimedia. Aplikasi ini dirancang untuk mengajarkan kosakata bahasa isyarat melalui animasi, audio, dan latihan visual, sehingga menarik minat pengguna dan meningkatkan efektivitas pembelajaran (Assa et al., 2021).

2.1.1. BISINDO (Bahasa Isyarat Indonesia)

BISINDO adalah bahasa yang didorong pengembangannya oleh Gerakan Kesejahteraan Tunarungu Indonesia (Gerkatin) dan dikembangkan secara mandiri oleh komunitas tunarungu. Oleh karena itu, BISINDO menjadi sistem komunikasi yang praktis dan efisien bagi penyandang tunarungu di Indonesia karena bahasa ini memang lahir dari kebutuhan dan pengalaman langsung para tunarungu itu sendiri (Borman et al., 2017). Karena berasal dari komunitas tunarungu itu sendiri, BISINDO lebih mudah dipahami dan diterima dalam interaksi sehari-hari. Dibandingkan dengan SIBI yang muncul belakangan, BISINDO telah digunakan lebih lama dan mencerminkan cara berkomunikasi teman tuli secara alami. Dalam penggunaannya, BISINDO menekankan ekspresi wajah dan gerakan mulut sebagai bagian penting dari makna isyarat. Selain itu, struktur bahasa ini terdiri dari lima unsur utama, yaitu lokasi isyarat, bentuk tangan, orientasi, gerakan tangan, dan ekspresi non-manual (Nugraheni et al., 2021). Visualisasi dari beberapa contoh gestur BISINDO dapat dilihat pada Gambar 2.1, yang memperlihatkan bentuk tangan dan ekspresi khas yang digunakan dalam komunikasi sehari-hari.



Gambar 2.1 Abjad Dalam BISINDO

(Sumber: Kompasiana, 2023)

2.1.2. SIBI (Sistem Isyarat Bahasa Indonesia)

SIBI (Sistem Isyarat Bahasa Indonesia) merupakan alat bantu komunikasi bagi individu tunarungu yang menggabungkan unsur bahasa lisan, gerakan isyarat, ekspresi wajah, dan gerak tubuh lainnya. Pemerintah menetapkan SIBI sebagai bahasa isyarat resmi yang digunakan di Sekolah Luar Biasa (SLB). Namun, banyak penyandang tunarungu merasa bahwa SIBI tidak sepenuhnya mewakili cara berkomunikasi mereka, karena SIBI menggunakan aturan isyarat yang cenderung menyesuaikan dengan struktur bahasa lisan dalam menyampaikan kosakata (Nugraheni et al., 2021). Gambar 2.2 menyajikan visualisasi dari beberapa contoh gestur dalam SIBI.

Gambar 2.2 Abjad Dalam SIBI

(Sumber: Yayasan Peduli Kasih ABK, 2018)

2.2. Python

Python adalah bahasa pemrograman tingkat tinggi dan serbaguna yang dikembangkan oleh Guido van Rossum pada akhir 1980-an dan pertama kali dirilis pada 1991. Filosofi desainnya menekankan keterbacaan kode dengan penggunaan indentasi yang signifikan, bersifat dinamis dalam pengecekan tipe data, dan mendukung berbagai paradigma pemrograman seperti terstruktur, berorientasi objek, dan fungsional. Sering disebut sebagai bahasa *"batteries included"* karena memiliki pustaka standar yang komprehensif, Python telah berkembang melalui beberapa versi

utama (*Python* 2 dan *Python* 3) dan secara konsisten menduduki peringkat sebagai salah satu bahasa pemrograman terpopuler, terutama dalam komunitas machine learning. Nama "*Python*" diambil dari serial komedi Inggris *"Monty Python's Flying Circus"*, dan kepemimpinan proyeknya beralih dari Van Rossum (yang dijuluki "*benevolent dictator for life*") kepada *Steering Council* beranggotakan lima orang pada 2019 (Van Rossum, G., 2007).

2.2.1. Open CV

Open Source Computer Vision Library (*OpenCV*) merupakan pustaka *open-source* yang dikembangkan oleh Intel pada tahun 1999 sebagai bagian dari inisiatif pengembangan aplikasi pemrosesan visual *real-time* yang efisien. Pustaka ini ditulis dalam bahasa pemrograman C dan dioptimalkan untuk mendukung arsitektur prosesor *multicore*, menjadikannya sangat cocok untuk aplikasi yang membutuhkan kecepatan pemrosesan tinggi (Bradski dan Kaehler, 2008; Kaehler dan Bradski, 2016). *OpenCV* kini telah berkembang menjadi salah satu pustaka paling populer dalam pengembangan sistem *computer vision* karena sifatnya yang fleksibel dan dapat dijalankan di berbagai platform.

Menurut Dawson-Howe (2019), *OpenCV* memiliki beragam fungsi penting, mulai dari pengolahan citra dasar seperti *filtering*, konversi warna, dan deteksi tepi, hingga pemrosesan citra lanjutan seperti deteksi wajah, pelacakan objek, kalibrasi kamera, serta pengenalan pola berbasis *machine learning*. Selain itu, *OpenCV* juga banyak digunakan untuk pengolahan video, rekonstruksi 3D, dan implementasi *augmented reality* serta sistem bantuan pengemudi. Kelengkapan dan kemudahan integrasi pustaka ini membuat *OpenCV* menjadi alat yang sangat bermanfaat untuk riset maupun pengembangan industri.

2.2.2. Numpy

NumPy (*Numerical* Python) adalah pustaka *open-source* dalam Python yang digunakan untuk komputasi numerik, terutama pada array dan matriks berdimensi banyak. Pustaka ini menyediakan fungsi-fungsi efisien untuk operasi matematika, statistik, aljabar linier, transformasi *Fourier*, dan bilangan acak. *NumPy* dibangun di atas kode C yang dioptimalkan, sehingga menawarkan kecepatan tinggi dengan sintaks

Python yang sederhana. Meskipun tidak menyediakan fungsi statistik lanjutan, NumPy menjadi dasar penting bagi pustaka lain seperti *pandas*, terutama dalam pengolahan data tabular (Gupta, P., dan Bagchi, A., 2024).

2.2.3. Pandas

Pandas adalah pustaka *open-source* dalam *Python* yang dirancang untuk keperluan analisis dan manipulasi data, terutama data dalam bentuk tabel (tabular) seperti *spreadsheet* dan *database*. Pustaka ini memungkinkan *Python* untuk melakukan operasi pemrosesan data secara cepat dan efisien, seperti membaca, membersihkan, menyusun ulang, menyatukan, hingga memodelkan dan menganalisis data. *Pandas* menawarkan struktur data utama berupa *Series* (data satu dimensi) dan *DataFrame* (data dua dimensi), yang memudahkan pengguna dalam mengelola data kompleks.

Pandas pertama kali dikembangkan oleh Wes McKinney pada tahun 2008, dan dibangun di atas *NumPy*. Meskipun tidak menggantikan *NumPy*, *Pandas* memperluas kemampuannya dengan menyediakan alat-alat analisis data yang lebih ekspresif dan terstruktur. Pandas mendukung berbagai format data seperti CSV, Excel, JSON, dan SQL, serta dapat menangani data heterogen, data waktu (*time series*), dan data tidak lengkap (Gupta, P., dan Bagchi, A., 2024).

2.2.4. Scikit-learn

Scikit-learn adalah pustaka *open-source* untuk *machine learning* yang ditulis dalam bahasa *Python*. Pustaka ini memungkinkan integrasi metode *machine learning* secara cepat dan mudah ke dalam kode Python. *Scikit-learn* menyediakan berbagai metode seperti klasifikasi, regresi, estimasi *matriks kovarian*, reduksi dimensi, praproses data, hingga pembuatan dataset uji.

Pustaka ini dapat digunakan di berbagai sistem operasi dan terus dikembangkan secara aktif. *Scikit-learn* banyak digunakan dalam aplikasi komersial, penelitian akademik, dan publikasi ilmiah. Untuk meningkatkan efisiensi, beberapa algoritma dalam *scikit-learn* ditulis dalam bahasa C dan diintegrasikan melalui *Cython*, yang memungkinkan kompilasi *Python* secara lebih cepat. Selain itu, metode seperti SVM

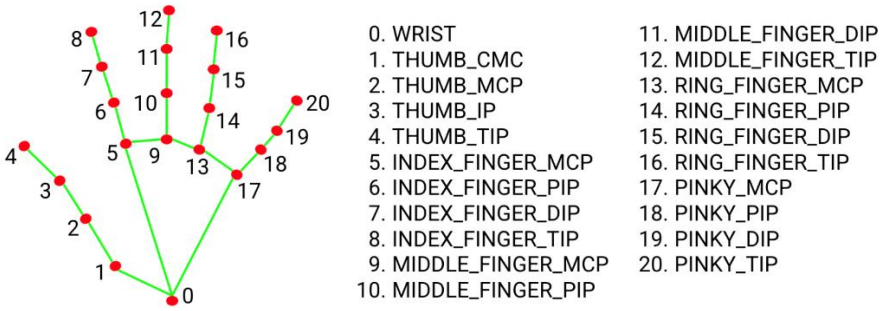
dan *logistic regression* dalam *scikit-learn* menggunakan pustaka LIBSVM dan LIBLINEAR sebagai dasar algoritmanya (Kramer, O., dan Kramer, O., 2016).

2.2.5. Mediapipe

MediaPipe adalah *framework open-source* yang dikembangkan oleh *Google* untuk membangun *pipeline* pemrosesan data sensorik seperti video, audio, dan input dari sensor lainnya secara modular dan *real-time*. *Framework* ini dirancang untuk membantu pengembang dalam membangun sistem pemrosesan visual yang kompleks dengan menyusun proses ke dalam bentuk *graph* kalkulator (*calculator graph*)—yaitu kumpulan komponen modular yang dapat digunakan kembali dan dikombinasikan sesuai kebutuhan.

MediaPipe mendukung berbagai platform, termasuk desktop, mobile (Android/iOS), dan bahkan perangkat *embedded* seperti Raspberry Pi, menjadikannya cocok untuk pengembangan sistem portabel. *Framework* ini dapat berjalan baik di CPU maupun GPU, serta dilengkapi dengan alat bantu seperti *Tracer* dan *Visualizer* untuk memantau performa *pipeline*. Keunggulan lainnya adalah kemampuannya untuk melakukan sinkronisasi waktu yang presisi antar stream data, serta dukungan penuh terhadap integrasi dengan model *machine learning* eksternal.

Salah satu komponen yang paling sering digunakan dalam *MediaPipe* adalah modul *hand tracking*, yang dapat mendeteksi dan melacak 21 titik kunci (*landmark*) pada tangan secara *real-time*. Titik-titik ini mencakup sendi dan ujung jari serta pusat telapak tangan, dan direpresentasikan dalam koordinat tiga dimensi. Informasi landmark ini menjadi sangat penting untuk mengenali bentuk dan pola gerakan tangan, sehingga sangat relevan dalam konteks pengenalan bahasa isyarat, khususnya BISINDO. Gambar 2.3 memperlihatkan peta visual dari 21 titik *landmark* tangan versi *MediaPipe*, yang menjadi acuan dalam proses ekstraksi fitur gerakan untuk klasifikasi gestur secara otomatis



Gambar 2.3 Peta *Mediapipe Hand Landmarks* (Sumber: AI Google Dev)

Dalam penelitian ini, *MediaPipe* digunakan untuk mengekstraksi fitur berupa posisi tangan dari input video yang ditangkap oleh kamera. Data koordinat landmark kemudian digunakan sebagai fitur masukan (*input features*) untuk algoritma *machine learning* seperti *Random Forest* dalam mengenali isyarat statis, serta *Long Short-Term Memory* (LSTM) untuk isyarat dinamis. Arsitektur *MediaPipe* yang ringan namun akurat menjadikannya solusi ideal untuk dikombinasikan dengan *Raspberry Pi*, menciptakan sistem penerjemah bahasa isyarat BISINDO yang portabel, efisien, dan *real-time* (Lugaresi et al., 2019).

2.2.6. Tensorflow

TensorFlow merupakan *framework open-source* dari Google yang dirancang untuk mendukung komputasi numerik berskala besar dalam konteks *machine learning*. *Framework* ini memodelkan perhitungan sebagai struktur data berbasis *computational graph*, yang memungkinkan optimasi eksekusi melalui pemetaan otomatis *node-node* graf ke berbagai unit komputasi, baik itu CPU, GPU, maupun distribusi antar *node* dalam sebuah *cluster* (Shukla, N., dan Fricklas, K., 2018).

2.2.7. Joblib

Joblib adalah pustaka *Python* yang menyediakan mekanisme pemrosesan terstruktur secara efisien melalui konsep *pipelining* yang ringan. Pustaka ini mendukung *caching* fungsi secara otomatis ke disk untuk menghindari komputasi ulang, serta memungkinkan eksekusi paralel yang sederhana, sehingga cocok digunakan dalam *workflow machine learning* dan pemrosesan data berskala besar (Faouzi, J., dan Janati, H., 2020).

2.2.8. Collections

Modul *collections* menyediakan berbagai tipe data kontainer khusus yang dirancang sebagai alternatif dari struktur data bawaan *Python* seperti *dict, list, set*, dan *tuple*. Tipe-tipe ini menawarkan fungsionalitas tambahan dan efisiensi yang lebih tinggi dalam kasus penggunaan tertentu, seperti pengurutan, penghitungan frekuensi, atau struktur data berorientasi *queue* dan *mapping* (Van Rossum, G. and Drake, F.L., 1995).

2.2.9. Pickle

Pickle adalah pustaka standar *Python* yang digunakan untuk melakukan serialisasi dan deserialisasi objek *Python* ke dalam format biner. Dengan kata lain, modul ini memungkinkan objek *Python* seperti *list, dictionary*, atau bahkan model *machine learning* disimpan ke dalam file dan dimuat kembali di lain waktu tanpa kehilangan struktur dan nilainya. Fitur ini sangat berguna dalam proses penyimpanan model, *caching data*, atau transfer objek antar sistem, terutama dalam *workflow machine learning* di mana hasil pelatihan model sering kali perlu disimpan dan digunakan kembali tanpa perlu dilatih ulang (Rostami et al., 2024).

2.2.10. Matplotlib

Matplotlib adalah pustaka grafik dalam *Python* yang digunakan untuk visualisasi data, dan merupakan bagian penting dalam ekosistem *data science Python*. *Library* ini memungkinkan pembuatan berbagai jenis grafik seperti *line chart, bar chart, scatter plot*, dan lain-lain dengan fleksibilitas tinggi. *Matplotlib* terintegrasi dengan baik bersama pustaka lain seperti *NumPy*, *Pandas*, dan pustaka ilmiah lainnya, sehingga memudahkan proses eksplorasi, analisis, dan presentasi data dalam bentuk visual yang informatif dan interaktif (Sial et al., 2021).

2.2.11. Regex

Regular expression (atau *regex*) adalah pola yang digunakan untuk merepresentasikan sekumpulan *string* yang sesuai dengan kriteria tertentu. Modul *re* dalam *Python* menyediakan berbagai fungsi untuk memeriksa apakah suatu *string* cocok dengan pola *regex* yang diberikan, atau sebaliknya, apakah suatu pola *regex* cocok dengan *string* tertentu yang secara konsep merupakan hal yang sama. *Regular*

expression sangat berguna dalam pemrosesan teks, seperti pencarian pola, validasi format *data*, hingga ekstraksi informasi dari teks secara efisien (Van Rossum, G. and Drake, F.L., 1995).

2.2.12. OS (Operating System)

Modul *os* menyediakan antarmuka yang portabel untuk mengakses fungsionalitas yang bergantung pada sistem operasi. Melalui modul ini, pengguna dapat melakukan berbagai operasi sistem seperti mengelola file, direktori, dan variabel lingkungan, tanpa harus bergantung pada detail sistem operasi tertentu.

Jika hanya ingin membaca atau menulis file, bisa langsung menggunakan fungsi *open()*. Untuk manipulasi *path*, tersedia submodul *os.path*. Jika perlu membaca seluruh baris dari banyak *file* yang diberikan melalui *command line*, modul *fileinput* lebih sesuai. Untuk membuat *file* atau direktori sementara, dapat menggunakan modul *tempfile*, dan untuk operasi tingkat tinggi seperti menyalin atau memindahkan file dan folder, disarankan menggunakan modul *shutil* (Van Rossum, G. and Drake, F.L., 1995).

2.2.13. Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) merupakan pengembangan dari arsitektur *Recurrent Neural Network* (RNN) yang dirancang untuk mengatasi kelemahan utama RNN standar dalam memproses data sekuensial, yaitu masalah *vanishing gradient* di mana informasi dari input awal cenderung hilang atau memudar saat jaringan memproses urutan yang panjang (Graves, A., dan Graves, A,. 2012). Masalah ini menyebabkan jaringan kesulitan dalam mempertahankan konteks jangka panjang, sehingga tidak mampu mengenali pola yang membutuhkan informasi historis yang lebih jauh.

LSTM mengatasi masalah ini dengan memperkenalkan blok memori yang terdiri dari sel memori dan tiga jenis gerbang (*gates*):

1. *Input gate* (mengontrol kapan informasi baru dapat disimpan ke dalam memori.).

$$i^{(t)} = \sigma\left(W_i x^{(t)} + R_i y^{(t-1)} + p_i \odot c^{(t-1)} + b_i\right) \quad (1)$$

2. *Forget gate* (memutuskan informasi mana dari memori sebelumnya yang perlu dilupakan).

$$f^{(t)} = \sigma\left(W_f x^{(t)} + R_f y^{(t-1)} + p_f \odot c^{(t-1)} + b_f\right) \quad (2)$$

3. *Output gate* (mengatur kapan informasi dalam sel memori digunakan sebagai keluaran).

$$o^{(t)} = \sigma\left(W_o x^{(t)} + R_o y^{(t-1)} + p_o \odot c^{(t)} + b_o\right) \quad (3)$$

Ketiga gerbang ini bekerja secara bersamaan untuk memungkinkan jaringan menyimpan, mempertahankan, dan membuang informasi secara selektif, sehingga membuat LSTM sangat efektif dalam mempelajari ketergantungan jangka panjang dalam data sekuensial.

Dalam ulasan komprehensif yang dilakukan oleh Van Houdt et al. (2020), LSTM terbukti sangat efektif dan telah menjadi arsitektur utama dalam berbagai aplikasi, termasuk pengenalan suara, terjemahan otomatis, hingga sistem interaktif berbasis *AI* seperti *Google Translate* dan *Amazon Alexa*. LSTM juga banyak diadopsi dalam *domain computer vision* untuk mengenali pola dalam data visual sekuensial seperti video, gerakan tubuh, dan *gesture recognition*.

Dengan kemampuannya tersebut, LSTM menjadi arsitektur yang sangat tepat untuk digunakan dalam penelitian ini, khususnya dalam mengenali gestur dinamis dalam Bahasa Isyarat Indonesia (BISINDO). Untuk mendukung pengenalan gestur statis, digunakan algoritma *Random Forest*, yang menurut penelitian oleh Alexander dkk. (2023), menunjukkan performa akurasi yang tinggi dan waktu klasifikasi yang efisien dibandingkan dengan algoritma lainnya seperti KNN, SVM, dan *Decision Tree*. Oleh karena itu, dalam penelitian ini dikembangkan pendekatan gabungan *Random Forest* dan LSTM, dengan tujuan mengoptimalkan akurasi sistem penerjemah bahasa isyarat berbasis kamera yang mampu berjalan secara *real-time* di perangkat Raspberry Pi.

2.2.14. Random Forest

Random Forest merupakan metode *ensemble learning* berbasis pohon keputusan yang pertama kali diperkenalkan oleh Breiman (2001). Metode ini membentuk sekumpulan (*forest*) pohon keputusan yang dilatih menggunakan teknik

bootstrap aggregating (*bagging*), di mana setiap pohon dibangun dari *subset* data pelatihan yang diambil secara acak dengan pengembalian. Selain itu, pada setiap percabangan (*split*) dalam pohon, hanya sebagian acak dari fitur yang dipertimbangkan, guna menciptakan keragaman struktural antar pohon dalam *ensemble*. Tujuan utama pendekatan ini adalah untuk mengurangi *variance* tanpa meningkatkan bias, sehingga menghasilkan model yang lebih stabil dan memiliki performa yang lebih tinggi dibandingkan model *decision tree*.

Secara formal, misalkan $\mathcal{D}_n = \big((X_1, Y_1), \dots, (X_n, Y_n)\big)$ adalah data pelatihan dengan $X_i \in \mathbb{R}^p$ sebagai vektor fitur dan Y_i sebagai label target (untuk klasifikasi), maka *estimator Random Forest* didefinisikan sebagai rata-rata dari prediksi seluruh pohon:

$$m_{M,n}\big(x;\ominus_1, \dots, \ominus_M, \mathcal{D}_n\big) = \frac{1}{M}\sum_{j=1}^{M} m_n\big(x;\ominus_j, \mathcal{D}_n\big) \tag{4}$$

Biau dan Scornet (2016) menjelaskan bahwa model ini dapat dikaji lebih lanjut melalui versi yang disederhanakan, seperti *purely random trees*, untuk memahami karakteristik bias dan konvergensinya secara teoritis. Dalam varian ini, pemisahan dilakukan tanpa mempertimbangkan label data, yang memungkinkan analisis sifat konsistensi model terhadap fungsi target dalam regresi.

Keunggulan lain dari *Random Forest* adalah kemampuannya untuk mengevaluasi performa model secara internal menggunakan metode *Out-of-Bag* (OOB) *error*. Teknik ini memanfaatkan data yang tidak terambil selama proses *bootstrap* untuk menguji akurasi model, sehingga tidak memerlukan pembagian eksplisit antara data latih dan data uji. Selain itu, *Random Forest* juga menyediakan metrik *feature importance* yang memungkinkan interpretasi terhadap pengaruh relatif masing-masing fitur dalam proses prediksi. Dua metrik umum yang digunakan adalah *Mean Decrease in Impurity* (MDI) dan *Mean Decrease in Accuracy* (MDA), yang dapat memberikan wawasan penting dalam analisis variabel.

Biau dan Scornet (2016) juga membandingkan *Random Forest* dengan

algoritma populer lainnya seperti SVM dan KNN. Mereka menekankan bahwa meskipun SVM unggul dalam masalah klasifikasi *margin* sempit dan KNN menawarkan pendekatan berbasis tetangga terdekat yang intuitif, *Random Forest* lebih unggul dalam hal ketahanan terhadap *noise*, efisiensi pada data berdimensi tinggi, serta kemampuannya menangani fitur numerik dan kategorikal secara bersamaan tanpa perlu normalisasi.

Secara keseluruhan, karakteristik non-parametrik, ketahanan terhadap *overfitting*, serta kemampuannya dalam memberikan estimasi generalisasi dan interpretabilitas menjadikan *Random Forest* sebagai model yang sangat sesuai untuk aplikasi klasifikasi dalam berbagai *domain*, termasuk dalam sistem pengenalan gestur bahasa isyarat berbasis *landmark* tangan.

2.2.15. Transformer

Arsitektur Transformer yang diperkenalkan oleh Vaswani et al. (2017) melalui makalah *Attention Is All You Need* menjadi terobosan besar dalam pemodelan data sekuensial. Model ini menggantikan *Recurrent Neural Networks* (RNN) dan *Convolutional Neural Networks* (CNN) dengan sepenuhnya mengandalkan mekanisme *self-attention*. Mekanisme inti yang digunakan adalah *Scaled Dot-Product Attention*, yang diformulasikan sebagai:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (5)$$

Dengan Q (*query*), K (*key*), V (*value*), dan d_k dimensi kunci. Rumus ini memungkinkan transformer untuk secara efisien menangkap ketergantungan jangka panjang antar elemen dalam sebuah urutan, sekaligus memfasilitasi paralelisasi sehingga pelatihan menjadi lebih cepat dibanding arsitektur berbasis RNN.

Kemampuan transformer ini kemudian diadaptasi pada domain pengenalan bahasa isyarat oleh Camgoz et al. (2020) yang mengusulkan *Sign Language Transformer* (SLT). Mereka membangun model end-to-end yang dapat melakukan *Continuous Sign Language Recognition* (CSLR) dan *Sign Language Translation* (SLT) secara bersamaan. Dengan memanfaatkan encoder–decoder berbasis transformer,

model ini terbukti mencapai hasil *state-of-the-art* pada dataset RWTH-PHOENIX-Weather-2014T.

Selanjutnya, Chaudhary et al. (2022) mengembangkan SignNet II, yaitu model berbasis transformer untuk penerjemahan dua arah bahasa isyarat (*sign-to-text* dan *text-to-sign*). Melalui mekanisme *dual learning* dan *metric embedding learning*, mereka berhasil meningkatkan kualitas translasi secara signifikan, khususnya dalam mempertahankan kesamaan antar tanda pada representasi pose. Hasil tersebut semakin memperkuat posisi transformer sebagai arsitektur yang fleksibel dan efektif dalam pemrosesan multimodal.

Dalam konteks penelitian ini, pengembangan sistem klasifikasi Bahasa Isyarat Indonesia (BISINDO) secara *real-time* dengan *Raspberry Pi 5* RAM 16 GB dilakukan dengan menghadirkan transformer sebagai metode pembaruan dari penelitian sebelumnya. Penyisipan transformer diharapkan mampu memberikan peningkatan performa, khususnya dalam menangkap pola *spasio-temporal* yang kompleks pada data gerakan bahasa isyarat, sekaligus menjadi kontribusi yang selaras dengan tren penelitian terkini.

2.2.16. Tensorflow Lite

TensorFlow Lite (TFLite) merupakan kerangka kerja *open-source* yang dikembangkan oleh Google untuk menjalankan model pembelajaran mesin pada perangkat dengan sumber daya terbatas seperti ponsel, IoT, dan mikrokontroler. Framework ini hadir sebagai solusi dari kebutuhan *Tiny Machine Learning* (TinyML) yang memungkinkan model *deep learning* dijalankan secara langsung pada perangkat *edge* dengan keterbatasan daya, memori, dan komputasi (David et al., 2021).

Cara kerja TFLite dimulai dari konversi model yang telah dilatih di *TensorFlow* ke dalam format *FlatBuffer* (.tflite). Proses konversi ini sering disertai dengan berbagai optimasi, antara lain *quantization* (mengubah representasi bobot dari 32-bit float ke 8-bit integer), *operator fusion*, dan *constant folding*. Optimasi tersebut membuat ukuran model lebih kecil, konsumsi memori lebih rendah, serta kecepatan inferensi meningkat tanpa mengurangi akurasi secara signifikan. Setelah itu, model dijalankan

menggunakan *TFLite Interpreter*, yaitu komponen ringan yang dirancang untuk memfasilitasi eksekusi model di berbagai platform *edge* (David et al., 2021).

Keunggulan utama TFLite adalah efisiensi dan portabilitasnya. Penelitian Coffen & Mahmud (2021) menunjukkan bahwa model LSTM untuk pengenalan gesture yang awalnya berukuran 2.8 MB dapat diperkecil dengan teknik konversi dan kuantisasi sehingga mampu dijalankan langsung pada perangkat *edge*. Dengan demikian, konsumsi daya berkurang dan sistem tidak perlu lagi bergantung pada komunikasi data ke server eksternal. Penelitian lain oleh Konaite et al. (2021) memperlihatkan bahwa Raspberry Pi 4 yang menjalankan model SSD MobileNet v2 melalui TFLite dapat mendeteksi objek secara *real-time* dengan kecepatan rata-rata 5 frame per detik. Hal ini menegaskan bahwa TFLite mampu menjembatani kebutuhan *deep learning* di perangkat dengan keterbatasan sumber daya, sekaligus tetap mempertahankan performa yang baik.

Dengan karakteristik tersebut, TFLite menjadi komponen penting dalam pengembangan aplikasi *machine learning* modern yang bersifat portabel, hemat energi, dan dapat beroperasi secara *real-time*. Hal ini membuatnya relevan dalam berbagai bidang, mulai dari sistem pengenalan gesture hingga perangkat wearable yang membutuhkan pemrosesan data langsung di perangkat.

2.2.17. Augmentasi Citra

Augmentasi citra merupakan strategi fundamental dalam pengembangan model *Deep Learning*, khususnya pada bidang visi komputer, yang bertujuan untuk meningkatkan kuantitas dan variabilitas data latih secara artifisial tanpa melalui proses akuisisi data baru yang memakan biaya. Urgensi penerapan metode ini didasari oleh kebutuhan model dengan parameter besar terhadap volume data yang masif guna mencapai kinerja yang kompetitif dan mencegah terjadinya *overfitting* atau fenomena di mana model mengingat data spesifik akibat keterbatasan sampel pelatihan. Secara teoritis, mekanisme augmentasi citra bekerja dengan memanipulasi distribusi data melalui konsep *vicinity distribution*. Konsep ini mengasumsikan bahwa distribusi probabilitas data tidak hanya terpaku pada titik tunggal sampel asli, melainkan dapat diperluas ke area sekitarnya melalui modifikasi visual yang tetap mempertahankan

label semantiknya, sehingga model dapat mempelajari fitur yang lebih general dan tangguh (*robust*).

Berdasarkan jurnal yang dipaparkan oleh Xu et al. (2023), metode augmentasi citra diklasifikasikan ke dalam beberapa kategori utama, di mana pendekatan yang diterapkan dalam penelitian ini tergolong sebagai *Model-free Single-image Augmentation*. Kategori ini memanfaatkan teknik pengolahan citra konvensional pada citra tunggal untuk menghasilkan variasi baru, dengan fokus utama pada transformasi geometris. Transformasi geometris bertujuan untuk memodifikasi hubungan spasial antar piksel guna mensimulasikan variasi kondisi fisik objek di dunia nyata. Teknik-teknik spesifik yang termasuk dalam domain ini meliputi translasi untuk memvariasikan posisi objek dalam bingkai , rotasi untuk mengubah perspektif sudut pandang objek , serta penskalaan yang berfungsi meniru variasi jarak atau ukuran objek. Penerapan kombinasi transformasi ini terbukti efektif dalam memperkaya distribusi data latih, khususnya pada dataset dengan tantangan variasi deformasi dan posisi objek.

2.3. Raspberry Pi

Raspberry Pi adalah komputer mini berukuran sebesar kartu kredit yang dikembangkan oleh *Raspberry Pi Foundation* di Inggris, dengan tujuan menyediakan perangkat komputasi murah dan portabel untuk pendidikan dan pengembangan teknologi. Raspberry Pi mendukung berbagai bahasa pemrograman seperti *Python*, C, C++, *Java*, dan lainnya, serta dapat menjalankan sistem operasi berbasis *Linux* seperti *Raspbian* (sekarang *Raspberry Pi OS*), *Debian*, dan lainnya.

Seiring perkembangan teknologinya, Raspberry Pi telah dirilis dalam berbagai varian model dengan spesifikasi dan fungsi yang disesuaikan untuk kebutuhan yang berbeda. Beberapa di antaranya dirancang untuk penggunaan umum dan edukasi, seperti *Raspberry Pi* 3 dan 4, sementara model lain seperti *Raspberry Pi Zero* dan *Compute Module* lebih ditujukan untuk proyek *embedded* dan aplikasi industri. Tiap model memiliki perbedaan signifikan dalam hal kapasitas RAM, kecepatan prosesor, jumlah *port*, serta dukungan terhadap fitur seperti *Wi-Fi*, *Bluetooth*, dan antarmuka video. Tabel 2.1. berikut menyajikan ringkasan spesifikasi dari berbagai versi Raspberry Pi yang telah dirilis hingga saat ini.

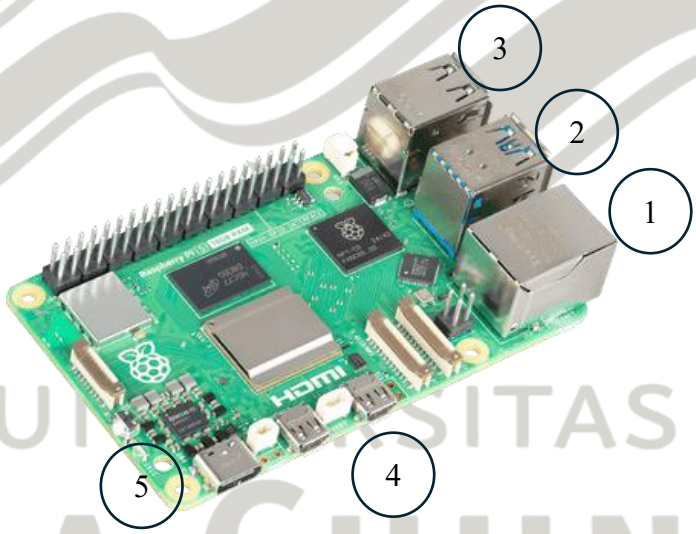
Tabel 2.1 Versi Raspberry Pi

Model	RAM	CPU	Port USB	HDMI / Display	Konektivitas
Raspberry Pi 5	4/8/16 GB	4 × 2.4 GHz	4 (2×USB 3.0)	2×micro-HDMI	WiFi & Bluetooth
Raspberry Pi 400	4 GB	4 × 1.8 GHz	4 (2×USB 3.0)	Ya	WiFi & Bluetooth
Raspberry Pi 4 B	1/2/4/8 GB	4 × 1.5 GHz	4 (2×USB 3.0)	2×micro-HDMI	WiFi & Bluetooth
Compute Module 4	1/2/4/8 GB	4 × 1.5 GHz	Tidak ada	Tidak ada	Opsional
Raspberry Pi 3 B+	1 GB	4 × 1.4 GHz	4	Ya	WiFi & Bluetooth
Raspberry Pi 3 A+	512 MB	4 × 1.4 GHz	1	Tidak	WiFi & Bluetooth
Raspberry Pi Zero 2 W	512 MB	4 × 1 GHz	1 (micro)	Tidak	WiFi & Bluetooth
Raspberry Pi Zero W	512 MB	1 × 1 GHz	1 (micro)	Tidak	WiFi & Bluetooth
Raspberry Pi Zero	512 MB	1 × 1 GHz	1 (micro)	Tidak	Tidak ada
Raspberry Pi 2 B	1 GB	4 × 900 MHz	4	Ya	Tidak ada

Versi terbaru saat ini, *Raspberry Pi 5*, menawarkan peningkatan spesifikasi yang signifikan dengan *prosesor quad-core Arm Cortex-A76* berkecepatan 2.4GHz dan kapasitas RAM hingga 16GB. Model ini juga dilengkapi dengan konektivitas yang lebih canggih, termasuk dua *port* USB 3.0 dengan *bandwidth* simultan penuh, *Wi-Fi*, *Bluetooth* 5.0, *port micro*-HDMI ganda, serta tambahan antarmuka *PCIe* 2.0 untuk

periferal berkecepatan tinggi. Dengan lonjakan performa tersebut, *Raspberry Pi 5* sangat mumpuni untuk menangani beban komputasi menengah hingga berat, menjadikannya *platform* yang ideal untuk menjalankan algoritma *machine learning* dan sistem *computer vision* yang kompleks secara lebih responsif.

Meskipun begitu, Monk (2023) juga mengingatkan bahwa Raspberry Pi memiliki keterbatasan, terutama dalam hal manajemen suhu. *Raspberry Pi 4*, misalnya, rentan terhadap *overheating* saat digunakan untuk komputasi intensif dalam jangka waktu lama. Untuk mengatasi hal ini, disarankan penggunaan sistem pendingin aktif seperti kipas (*fan*), *heatsink*, atau casing berpendingin. Selain itu, pemilihan *SD card* berkecepatan tinggi (minimal *Class 10*) dan *power supply* 5V 3A juga penting untuk menjamin kestabilan sistem.



Gambar 2.4 Raspberry Pi 5 (Sumber: Raspberry Pi Ltd, 2025)

Gambar 2.4. memperlihatkan tampilan fisik Raspberry Pi yang digunakan dalam penelitian ini. Pada gambar tersebut, beberapa komponen utama diberi penomoran untuk mempermudah identifikasi dan penjelasan masing-masing fungsinya. Berikut ini adalah uraian dari komponen-komponen penting yang ditandai:

1. Gigabit Ethernet

   Port ini digunakan untuk konektivitas jaringan kabel (LAN) dengan kecepatan hingga 1 Gbps. Gigabit Ethernet sangat berguna untuk keperluan

transfer data berkecepatan tinggi, komunikasi antarmuka dengan *server*, atau ketika Raspberry Pi digunakan dalam jaringan lokal yang stabil dan responsif.

USB 3.0

2. Port USB 3.0 (berwarna biru) menawarkan kecepatan transfer data yang jauh lebih tinggi dibandingkan USB 2.0, hingga 5 Gbps. Dalam penelitian ini, USB 3.0 dapat dimanfaatkan untuk menghubungkan kamera eksternal (*webcam*), *flashdisk*, atau perangkat penyimpanan lainnya untuk mempercepat pemrosesan dan penyimpanan data.
3. USB 2.0

   Port USB 2.0 digunakan untuk koneksi perangkat seperti *mouse, keyboard*, atau perangkat input-output lainnya yang tidak membutuhkan kecepatan transfer tinggi. Keberadaan USB 2.0 memungkinkan Raspberry Pi berfungsi layaknya komputer desktop dalam skala mini.1
4. Micro HDMI

   Raspberry Pi 5 dilengkapi dua port micro HDMI yang dapat digunakan untuk menampilkan output grafis ke layar monitor atau TV. Port ini mendukung output video hingga resolusi 4K. Dalam konteks pengujian sistem deteksi *gesture*, micro HDMI digunakan untuk menampilkan antarmuka visual dan hasil klasifikasi secara langsung di layar eksternal.
5. USB Type-C (*Power Supply*)

   Port ini berfungsi sebagai jalur utama untuk memasok daya ke Raspberry Pi. Dibandingkan versi sebelumnya yang menggunakan *micro* USB, *port* USB *Type-C* mampu menyediakan arus listrik yang lebih stabil dan cukup untuk mendukung komponen-komponen tambahan seperti kamera, sensor, atau layar tambahan.

Raspberry Pi sangat populer dalam pengembangan sistem tertanam (*embedded systems*), terutama karena harganya yang murah, ukuran kecil, dan fleksibilitas tinggi. Keunggulannya antara lain: dukungan terhadap banyak sensor dan perangkat eksternal melalui GPIO, kompatibilitas dengan berbagai jenis kode, serta dapat difungsikan

sebagai komputer portabel. Dalam penelitian ini, *Raspberry Pi* digunakan sebagai platform utama untuk menjalankan sistem klasifikasi bahasa isyarat BISINDO secara *real-time*, karena kemampuannya yang cukup untuk memproses input dari kamera dan menjalankan model *machine learning* secara efisien.

Namun, Raspberry Pi juga memiliki keterbatasan, seperti tidak adanya penyimpanan internal (mengandalkan SD *card*), kinerja grafis yang terbatas, serta potensi *overheating* jika digunakan dalam waktu lama tanpa pendingin tambahan. Meskipun demikian, kombinasi fleksibilitas, portabilitas, dan kemudahan pemrograman menjadikan Raspberry Pi sangat ideal untuk proyek-proyek inovatif berbasis *AI* dan pengolahan citra (Ghael et al., 2020).

Kemampuan *Raspberry Pi* sebagai unit pemrosesan *computer vis*ion telah dibuktikan dalam berbagai implementasi praktis. Sebagai contoh, *Raspberry Pi* telah digunakan sebagai inti dari sistem visi berbasis *Deep Neural Network* (DNN) untuk tugas-tugas keselamatan gedung. Dalam studi tersebut, sistem yang dikembangkan mampu mendeteksi aliran asap sekaligus melakukan estimasi kepadatan manusia di dalam ruangan secara *real-time* selama simulasi insiden kebakaran. Platform yang sama juga telah diusulkan untuk sistem penghitungan manusia dalam sebuah adegan dengan memanfaatkan deteksi kepala (Birajdar et al., 2021).

Selain itu, *Raspberry Pi* juga populer digunakan sebagai unit pemrosesan utama dalam sistem tertanam (*embedded systems*) yang lebih kompleks, seperti robotika otonom. Sebuah tinjauan teknologi pada robot pemetik buah dan sayuran menyoroti peran krusial modul visi yang seringkali ditenagai oleh *single-board computer*. Dalam aplikasi tersebut, sistem visi menggunakan kamera untuk menangkap citra, yang kemudian diproses menggunakan algoritma *machine learning* untuk mengidentifikasi secara akurat jenis, lokasi, dan bahkan tingkat kematangan dari hasil panen. Data hasil pemrosesan visual ini kemudian digunakan oleh robot untuk menentukan lokasinya secara presisi dan merencanakan aktuasi lengan robotik untuk melakukan pemetikan (Chen et al, 2024).

2.3.1. Pi Camera

Kamera Raspberry Pi atau *Pi Camera* merupakan modul kamera yang dirancang khusus untuk papan pengembangan Raspberry Pi. Modul ini terhubung melalui *Camera Serial Interface* (CSI) dengan kabel pita 15-pin sehingga mampu berkomunikasi langsung dengan GPU Raspberry Pi. Keunggulan ini membuat *Pi Camera* dapat melakukan pemrosesan gambar dengan cepat tanpa membebani CPU, serta mendukung perekaman video berkualitas tinggi seperti HD video, *time-lapse*, maupun *slow-motion* (Symon et al., 2017). Fitur tersebut menjadikan *Pi Camera* banyak dimanfaatkan dalam aplikasi berbasis visi komputer, sistem pemantauan, maupun pengembangan sistem cerdas yang membutuhkan pengolahan citra secara *real-time*.

Dalam penelitian Symon et al. (2017), *Pi Camera* digunakan sebagai sensor visual utama pada sistem pemantauan bayi berbasis *Raspberry Pi*. Kamera ini bekerja secara terintegrasi dengan sensor lain, seperti PIR sensor untuk mendeteksi gerakan dan mikrofon untuk mendeteksi suara tangisan bayi. Hasil tangkapan kamera ditampilkan secara *real-time* melalui LCD display, sementara buzzer digunakan sebagai alarm jika bayi terdeteksi bergerak atau menangis. Penelitian ini menunjukkan bahwa Pi Camera tidak hanya berfungsi sebagai perangkat pengambil gambar, tetapi juga sebagai komponen penting dalam *embedded system* yang memerlukan monitoring visual secara langsung.

Seiring perkembangannya, *Pi Camera* telah hadir dalam berbagai versi dengan peningkatan resolusi, kualitas sensor, dan fitur tambahan. Perbandingan versi *Pi Camera* dapat dilihat pada Tabel 2.2.

Tabel 2.2 Versi Pi Camera

Versi	Sensor	Resolusi Foto	Resolusi Video	Tahun Rilis
V1	OmniVision OV5647	5 MP (2592×1944)	1080p @30fps, 720p @60fps, 480p @90fps	2013
V2	Sony IMX219	8 MP (3280×2464)	1080p @30fps, 720p @60fps, 480p @90fps	2016
HQ Camera	Sony IMX477	12.3 MP (4056×3040)	4K @30fps	2020
Camera Module 3	Sony IMX708	12 MP	4608×2592 @60fps	2023

Berdasarkan tabel tersebut, dapat dilihat bahwa Raspberry Pi Camera telah mengalami perkembangan signifikan dari versi 1 hingga *Camera Module* 3, baik dari segi resolusi, sensor, maupun fitur tambahan. Perkembangan ini memperluas cakupan aplikasi *Pi Camera*, mulai dari penelitian akademik, sistem pengenalan citra, hingga penggunaan profesional dalam bidang industri dan IoT.

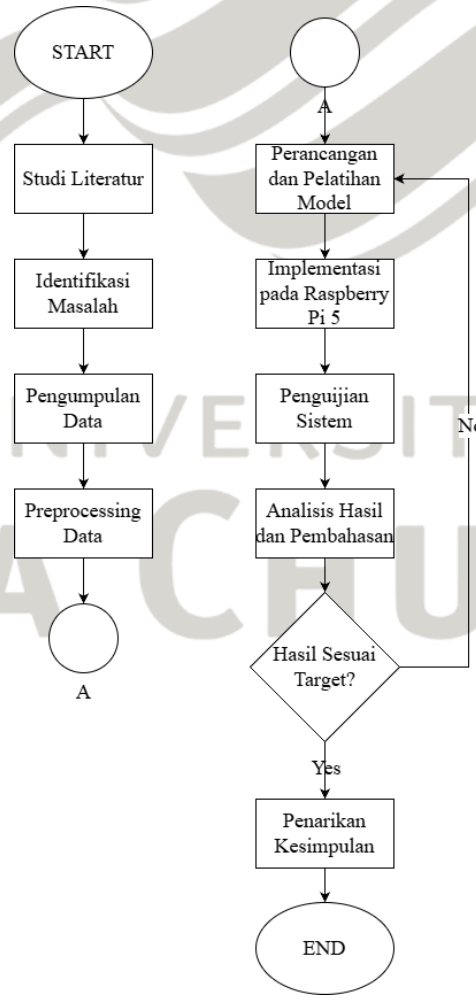
Pada penelitian ini, penulis menggunakan *Pi Camera Module v1* yang dilengkapi sensor *OmniVision OV5647* dengan resolusi 5 megapiksel. Modul ini mampu merekam video hingga 1080p pada 30 fps, sehingga cukup mendukung kebutuhan pengolahan citra untuk sistem pengenalan bahasa isyarat berbasis Raspberry Pi. Walaupun spesifikasi modul v1 masih terbatas dibanding generasi terbaru, penggunaannya tetap relevan karena konsumsi daya yang rendah, kemudahan integrasi, serta ketersediaannya yang luas di komunitas pengembang. Hal ini menjadikan Pi Camera v1 pilihan yang tepat untuk tahap awal pengembangan sistem portabel berbasis Raspberry Pi. Bentuk fisik dari modul kamera ini dapat dilihat pada Gambar 2.5.

Gambar 2.5 Pi Camera v1 5MP (Sumber: Pomaska, 2019)

Bab III
Analisis dan Perancangan Sistem

3.1. Metode Penelitian

Untuk mencapai tujuan penelitian yang telah dirumuskan, penelitian ini dilaksanakan melalui serangkaian tahapan yang sistematis dan terstruktur. Metode penelitian yang diterapkan menggunakan pendekatan eksperimental untuk mengembangkan sistem klasifikasi bahasa isyarat BISINDO pada perangkat *embedded*. Secara garis besar, alur tahapan penelitian ini digambarkan dalam *Flowchart* pada Gambar 3.1.



Gambar 3.1 Flowchart Metode Penelitian

Berdasarkan Gambar 3.1, tahapan penelitian dijelaskan secara rinci sebagai berikut:

1. **Studi Literatur**: Tahap awal dilakukan dengan mengkaji teori terkait BISINDO, pengolahan citra digital, arsitektur *Deep Learning* (LSTM dan *Transformer*), serta implementasi pada *Raspberry Pi*.
2. **Identifikasi Masalah**: Merumuskan permasalahan terkait kendala komunikasi teman Tuli dan kebutuhan sistem penerjemah yang portabel dan *real-time*.
3. **Pengumpulan Data**: Melakukan pengambilan data citra gestur statis dan *sequence* gestur dinamis menggunakan kamera, yang kemudian diproses menggunakan *MediaPipe* untuk mendapatkan fitur *landmark* tangan.
4. **Perancangan Sistem**: Merancang arsitektur perangkat keras menggunakan *Raspberry Pi 5* dan *Pi Camera v1*, serta merancang arsitektur model *Random Forest*, LSTM, dan *Transformer*.
5. **Implementasi Model**: Melatih model klasifikasi menggunakan *dataset* yang telah dikumpulkan, melakukan optimasi *hyperparameter*, dan mengonversi model ke format *TensorFlow Lite*.
6. **Pengujian dan Evaluasi**: Tahap ini dilakukan untuk memvalidasi kinerja sistem melalui dua skenario pengujian utama:
    - **Evaluasi Model**: Mengukur performa model klasifikasi menggunakan data uji yang telah dipisahkan. Evaluasi dilakukan menggunakan metrik *Confusion Matrix*, Akurasi, Presisi, *Recall*, dan *F1-Score* untuk mengetahui kemampuan model dalam mengenali gestur sebelum diimplementasikan ke perangkat keras.
    - **Evaluasi Sistem *Real-Time***: Menguji integrasi sistem secara langsung pada perangkat *Raspberry Pi 5*. Pengujian ini mencakup akurasi deteksi gestur secara *real-time* oleh pengguna, serta pengukuran kinerja komputasi yang meliputi kecepatan inferensi (*latency*), *Frame Rate* (FPS), penggunaan CPU, dan stabilitas suhu perangkat.

7. **Analisis dan Kesimpulan**: Menganalisis hasil pengujian untuk membandingkan kinerja antara metode LSTM dan *Transformer*, serta menarik kesimpulan dari penelitian yang dilakukan.

3.2. Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk merumuskan spesifikasi sistem yang akan dikembangkan agar sesuai dengan tujuan penelitian. Sistem yang dirancang adalah pengembangan sistem klasifikasi Bahasa Isyarat Indonesia (BISINDO) secara *real-time* berbasis Raspberry Pi, dengan pembaruan berupa integrasi metode *Random Forest* untuk gestur statis, serta perbandingan antara *Long Short-Term Memory* (LSTM) dan *Transformer* dalam pengenalan gestur dinamis. Analisis ini menjadi landasan dalam merancang arsitektur, algoritma, dan implementasi sistem yang handal, portabel, serta dapat bekerja pada perangkat dengan keterbatasan sumber daya.

3.2.1. Kebutuhan Fungsionalitas

Kebutuhan fungsional menjelaskan fungsi utama yang harus dimiliki sistem agar dapat berjalan sesuai dengan tujuan penelitian, yaitu:

1. Sistem mampu menangkap gerakan tangan pengguna melalui kamera yang terpasang pada Raspberry Pi.
2. Sistem dapat mendeteksi dan mengekstraksi landmark tangan secara *real-time* menggunakan pustaka *MediaPipe*.
3. Sistem mengklasifikasikan gestur statis BISINDO menggunakan algoritma *Random Forest*.
4. Sistem mengklasifikasikan gestur dinamis dengan menggunakan dua pendekatan berbeda, yaitu LSTM dan *Transformer*, untuk kemudian dilakukan analisis perbandingan performa.
5. Sistem dapat menampilkan hasil prediksi gestur dalam bentuk label teks secara *real-time*.

3.2.2. Kebutuhan Non-Fungsional

Selain fungsi utama, sistem juga harus memenuhi aspek non-fungsional agar dapat digunakan secara efektif, yaitu:

1. Kinerja: sistem harus dapat memproses input video secara *real-time* dengan latensi rendah.
2. Portabilitas: sistem dijalankan pada perangkat Raspberry Pi sehingga dapat digunakan secara mandiri tanpa ketergantungan pada komputer eksternal.
3. Efisiensi: penggunaan daya dan memori harus dioptimalkan mengingat keterbatasan perangkat keras.
4. Kemudahan penggunaan: antarmuka sistem dirancang sederhana agar dapat dipahami dan digunakan tanpa kesulitan oleh pengguna awam.

3.2.3. Kebutuhan Data

Sistem memerlukan dataset yang terdiri dari dua jenis gestur, yaitu gestur statis dan gestur dinamis.

- Gestur statis: dikumpulkan dalam bentuk citra tunggal tangan pada posisi tertentu yang mewakili huruf, angka, atau kosakata BISINDO. Data ini digunakan untuk pelatihan model *Random Forest*.
- Gestur dinamis: dikumpulkan dalam bentuk rangkaian *frame* (*sequence*) berisi koordinat *landmark* tangan. Dataset ini digunakan untuk pelatihan model LSTM dan *Transformer*. Setiap kelas gestur dinamis dikumpulkan minimal 105 *sequence* dengan panjang 20 *frame per sequence*, sehingga cukup mewakili variasi antar-subjek.

3.3. Pengumpulan Data

Jenis gestur yang digunakan dalam penelitian ini diklasifikasikan menjadi dua kategori utama, yaitu gestur statis dan gestur dinamis, berdasarkan karakteristik pergerakannya. Gestur statis merupakan bentuk isyarat yang dapat dikenali dari satu citra tunggal tanpa memerlukan analisis urutan waktu, seperti huruf alfabet, angka, dan beberapa kosakata BISINDO yang tidak melibatkan perubahan posisi tangan secara signifikan. Gestur-gestur tersebut dilatih menggunakan algoritma *Random Forest*, yang efektif dalam mengenali pola visual tetap dari koordinat landmark tangan. Daftar lengkap gestur statis yang digunakan sebagai data pelatihan dan pengujian dalam

penelitian ini disajikan pada Tabel 3.1, yang mencakup seluruh kata statis dalam BISINDO.

Tabel 3.1 Gestur yang dilatih Random Forest

A-Z	BAWAH	MENDENGAR
0-10	BISA	BERDOA
ANDA KAMU	MAKAN	KANAN
ITU MENUNJUK	TIDUR	KIRI
SAYA	BACA	

Sementara itu, gestur dinamis dilatih menggunakan metode LSTM dan *Transformer* karena kedua model tersebut mampu mempelajari pola temporal dari rangkaian gerakan tangan. Gestur dinamis mencakup kosakata yang direpresentasikan dalam bentuk *sequence* pergerakan tangan. Daftar gestur dinamis yang digunakan dalam penelitian ini ditampilkan pada Tabel 3.2.

Tabel 3.2 Gestur kosakata yang dilatih LSTM dan Transformer

AIR	SEPERTI	TAHUN	DARI	TURUN
INI	ATAU	BELAJAR	MINUM	HANYA
ROTI	DALAM	LUAR	ORANG	DAN
ATAS	SIAPA	MILIK PUNYA	DIA	APA
JADI	KERJA	UNTUK	NAIK	NASI
MEREKA	KITA	JIKA KALAU	G	BANGUN
BICARA	TULIS	JALAN	LARI	MEMBELI
MENUTUP	MEMBERI	MENERIMA	MENOLONG	MENUNGGU
DUDUK	BERDIRI	DATANG	MASUK	KELUAR

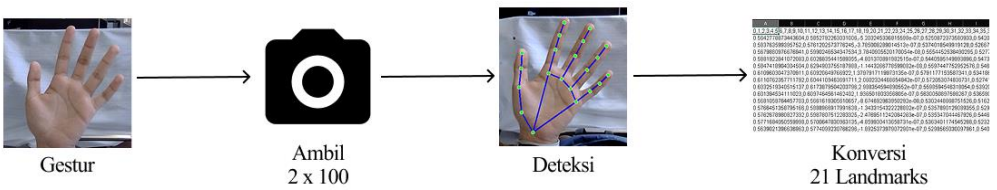
TUTUP (MATA)	MENANGIS	TERTAWA	MENJAWAB	MENANYAKAN
MENYAPU	MENCUCI BAJU	MEMASAK	MENGIRIM	NAMA
BERCERITA	MINTA MAAF	MENYANYI	BERMAIN	DEPAN
ANTARA	DEKAT	JAUH	DI SINI	DI SANA
TIMUR	BARAT	SELATAN	UTARA	R
J	TAPI	TAHU PAHAM	SEMUA	LIHAT
MENJUAL	MEMINTA	PAKAI	MEMBACA AL QURAN	MENERIMA PESAN
BELAKANG	SEBELUM	BERKATA	AMBIL	AKAN
PULANG	PERGI	DENGAR	MEMBUKA	MENONTON
BUKA (MATA)	MENONTON TV	MENIKAH	SAMPING	SESUDAH

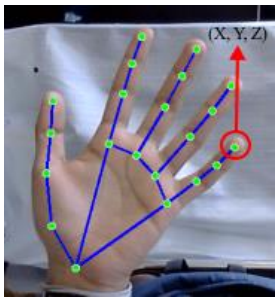
Proses pengumpulan data dalam penelitian ini dibedakan menjadi dua tahap sesuai dengan karakteristik model yang digunakan, yaitu *Random Forest* untuk klasifikasi gestur statis dan LSTM/*Transformer* untuk klasifikasi gestur dinamis.

Seluruh proses akuisisi data dilaksanakan di lingkungan Laboratorium *Human-Machine Interaction* (HMI). Kondisi pencahayaan di area pengambilan data dipantau secara ketat menggunakan alat ukur *Luxmeter*, dengan intensitas cahaya dijaga stabil pada rentang 200 hingga 300 *Lux* untuk memastikan visibilitas fitur tangan tetap optimal bagi sensor kamera. Berbeda dengan pendekatan yang menggunakan latar belakang polos, penelitian ini menerapkan kondisi latar belakang natural yang kompleks yang memuat berbagai objek inventaris laboratorium. Hal ini bertujuan untuk menguji ketahanan model dalam memisahkan objek tangan dari gangguan visual di lingkungan nyata.

Untuk menjaga konsistensi proporsi dan skala citra tangan, jarak antara subjek dan kamera dipertahankan konstan pada rentang 50 hingga 80 cm, baik pada fase pengumpulan data maupun pengujian. Namun, terdapat perbedaan pada spesifikasi perangkat keras akuisisi yang digunakan. Pada tahap pengumpulan *dataset* latih, perekaman dilakukan menggunakan *Webcam Logitech C270* dengan konfigurasi resolusi 1280x720 piksel dan kecepatan 30 fps. Sedangkan pada tahap evaluasi sistem *real-time*, perangkat digantikan oleh modul *PiCamera v1* yang terhubung dengan sistem pemroses, dengan pengaturan jarak dan sudut pandang yang identik serta resolusi input yang disesuaikan untuk menjaga validitas performa model pada *embedded device.*

Pada tahap pertama, data untuk model *Random Forest* diperoleh dalam bentuk citra tunggal dari setiap gestur statis. Proses pengambilan dilakukan menggunakan PC yang terhubung dengan *webcam* sebagai perangkat perekaman. Terdapat 6 partisipan dalam proses pengambilan data, yang seluruhnya merupakan mahasiswa berusia 20–23 tahun. Setiap partisipan diminta memperagakan gestur sesuai kategori yang ditentukan, kemudian direkam sebanyak 200 gambar per gestur dengan bantuan pustaka *OpenCV*. Seluruh citra hasil tangkapan kamera selanjutnya diproses menggunakan *MediaPipe Hands* untuk mengekstraksi titik-titik koordinat tangan dalam bentuk tiga dimensi (x, y, z). Data landmark tersebut digunakan sebagai representasi fitur yang kemudian dilatih dengan model *Random Forest*. Kategori gestur yang dikumpulkan meliputi 104 kosakata harian, huruf alfabet A–Z, serta angka 0–10, sehingga dataset mencakup variasi gerakan yang luas untuk mendukung pengenalan gestur statis BISINDO. Alur proses pengumpulan data ini dapat dilihat pada Gambar 3.2. beserta juga contoh data yang diperoleh pada Gambar 3.3.



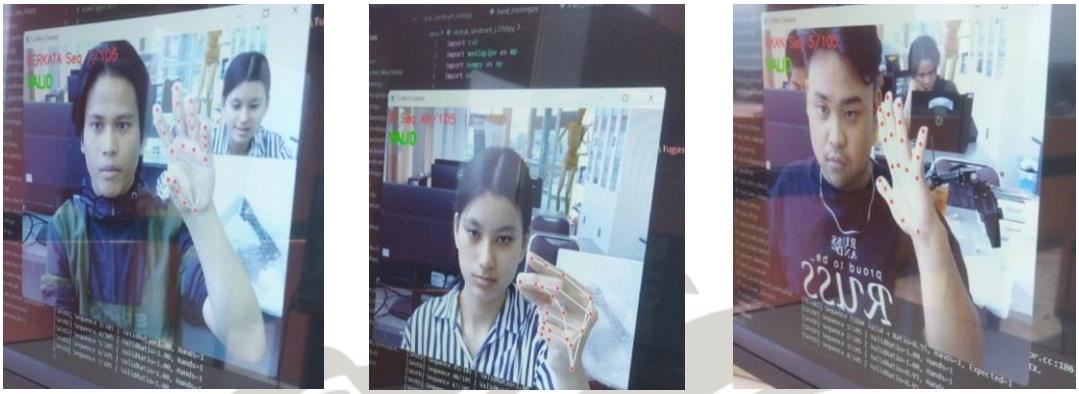


Gambar 3.2 Skema pengumpulan data gestur statis dan titik-titik landmark yang akan dikonversi dalam format file CSV

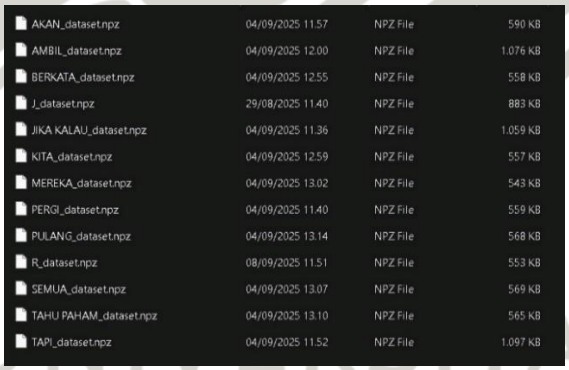
Gambar 3.3 Dataset *Random Forest* Gestur "BAWAH"

Berbeda dengan *Random Forest*, pengumpulan data untuk model LSTM dan *Transformer* dilakukan dalam bentuk rangkaian *sequence* yang merepresentasikan gestur dinamis. Proses pengambilan juga dilakukan menggunakan PC dan *webcam* dengan dukungan pustaka *MediaPipe Hands*. Setiap *sequence* terdiri dari 20 *frame* berurutan, dengan setiap *frame* berisi koordinat tiga dimensi dari 21 titik landmark tangan. Untuk setiap kelas gestur dinamis, dikumpulkan sebanyak 105 *sequence* dari setiap partisipan. Agar partisipan tidak mengalami kelelahan akibat repetisi yang berulang, pengambilan data dirancang dengan memberi jeda istirahat selama 1 menit setiap kali selesai satu kelas gestur. Dataset hasil perekaman kemudian disimpan dalam format .npz, yang berisi array berdimensi (105, 20, 63/126) tergantung jumlah tangan yang terdeteksi, serta metadata seperti identitas partisipan, kategori gestur, dan parameter perekaman. Struktur data ini dipilih karena sesuai dengan kebutuhan input model LSTM dan *Transformer* yang memerlukan representasi sekuensial. Visualisasi

akuisisi *dataset* gestur dinamis ditampilkan pada Gambar 3.4 dan hasil akuisisi *dataset* ditampilkan pada gambar 3.5.



Gambar 3.4 Pengambilan Dataset Gestur Dinamis



Gambar 3.5 Hasil Pengambilan Dataset Gestur Dinamis

Dengan adanya perbedaan metode pengumpulan data ini, sistem dapat menangani dua jenis gestur dengan pendekatan yang sesuai. Gestur statis direpresentasikan dalam bentuk citra tunggal, diekstraksi menjadi landmark tangan menggunakan *MediaPipe Hands*, lalu dilatih menggunakan *Random Forest*. Sementara itu, gestur dinamis direpresentasikan sebagai *sequence frame*, diekstraksi dengan cara yang sama menggunakan *MediaPipe Hands*, lalu digunakan sebagai input untuk melatih model LSTM maupun *Transformer* agar pola temporal pergerakan dapat dipelajari secara lebih mendalam.

Sebelum digunakan dalam pelatihan model, seluruh gestur yang dikumpulkan divalidasi oleh ahli Bahasa Isyarat Indonesia (BISINDO) untuk memastikan kesesuaian bentuk gestur dengan standar bahasa isyarat yang berlaku. Setelah divalidasi, dataset dibagi dengan rasio 80:20 untuk data latih dan data uji. Selain itu, pelatihan model juga menerapkan *5-Fold Cross-Validation* untuk meningkatkan reliabilitas dan generalisasi model terhadap data baru. Dengan adanya perbedaan metode pengumpulan dan pengolahan data ini, sistem mampu mengenali baik gestur statis maupun dinamis secara efektif sesuai karakteristik model masing-masing.

3.4. Pembentukan Model Klasifikasi

Pembentukan model klasifikasi dalam penelitian ini dilakukan dengan tiga pendekatan utama, yaitu *Random Forest*, LSTM (*Long Short-Term Memory*), dan *Transformer*. Seluruh model dilatih menggunakan dataset BISINDO yang telah diproses menjadi representasi koordinat landmark tangan hasil ekstraksi dari *MediaPipe*. Untuk menjamin konsistensi dan reliabilitas model, data dibagi menjadi 80% untuk pelatihan dan validasi, serta 20% untuk pengujian akhir. Selain itu, pada tahap pelatihan diterapkan *5-Fold Cross Validation* pada data latih untuk memperoleh evaluasi yang lebih menyeluruh serta mengurangi potensi bias akibat pemisahan data tunggal.

Dalam proses pelatihan model, penentuan nilai *hyperparameter* memegang peranan krusial terhadap performa akhir sistem. Pemilihan *hyperparameter* pada penelitian ini dilakukan menggunakan pendekatan empiris melalui serangkaian eksperimen iteratif. Nilai parameter tidak ditentukan menggunakan pencarian otomatis, melainkan disesuaikan secara manual berdasarkan observasi terhadap konvergensi *loss* dan akurasi model pada data validasi.

Proses ini dimulai dengan mengadopsi nilai standar yang direkomendasikan dalam literatur terkait, kemudian dilakukan penyesuaian bertahap untuk mendapatkan konfigurasi yang paling optimal bagi karakteristik dataset gestur yang digunakan. Konfigurasi akhir yang dipilih adalah konfigurasi yang menghasilkan keseimbangan terbaik antara akurasi pelatihan dan kemampuan generalisasi guna menghindari *overfitting*.

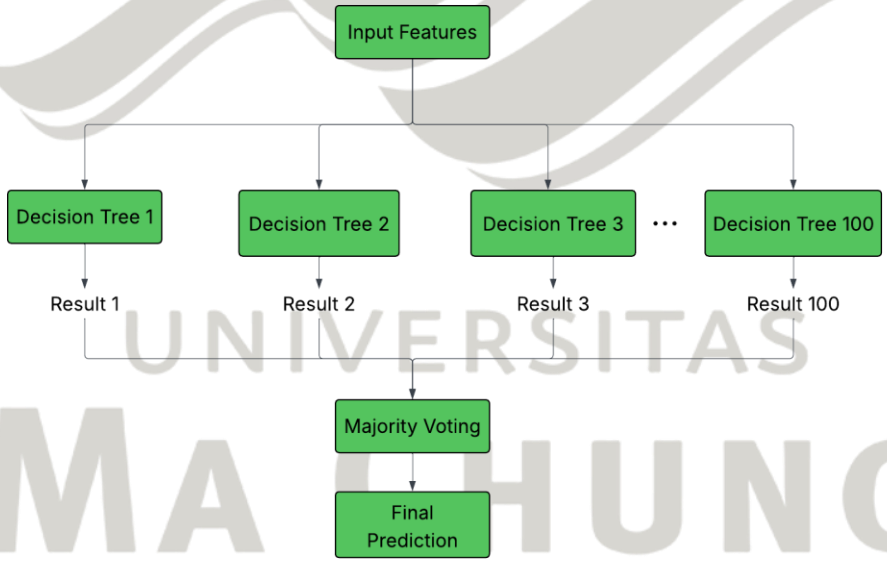
3.4.1. Arsitektur Model Random Forest

Model *Random Forest* dirancang untuk mengklasifikasikan gestur statis, seperti huruf alfabet, angka, dan beberapa kosakata BISINDO. Masukan model berupa vektor fitur tunggal dari koordinat landmark tangan. *Random Forest* dibangun sebagai kumpulan pohon keputusan yang masing-masing dilatih pada subset acak dari data, kemudian hasil prediksi digabungkan menggunakan mekanisme *majority voting*. Pada perancangan ini, parameter utama yang digunakan adalah jumlah pohon (*n_estimators*), fungsi pemisahan (*criterion*), serta kedalaman pohon (*max_depth*), yang ditentukan untuk menyeimbangkan kompleksitas dan kemampuan generalisasi model. Struktur model ini divisualisasikan pada Gambar 3.6, yang menunjukkan hubungan hierarkis antar pohon keputusan dalam menghasilkan prediksi akhir berdasarkan mayoritas hasil klasifikasi. Detail pengaturan hyperparameter yang digunakan pada model *Random Forest* ditampilkan pada Tabel 3.3. Pemilihan parameter dilakukan secara empiris berdasarkan hasil beberapa percobaan awal untuk menyeimbangkan antara akurasi, waktu pelatihan, dan kemampuan generalisasi model terhadap variasi data gestur. Nilai-nilai tersebut menghasilkan performa terbaik pada pengujian *train-test split* (80:20) maupun *5-fold cross-validation*.

Tabel 3.3 Hyperparameter Model Random Forest

Parameter	Nilai yang Digunakan	Keterangan
n_estimators	100	Jumlah pohon keputusan yang digunakan dalam ensemble. Semakin besar nilainya, semakin stabil hasil prediksi.
criterion	Gini	Fungsi pengukuran impurity yang digunakan untuk menentukan pemisahan optimal pada setiap node.

Parameter	Nilai yang Digunakan	Keterangan
max_depth	None	Kedalaman maksimum pohon tidak dibatasi untuk memberi fleksibilitas pada proses pembelajaran.
random_state	42	Nilai seed acak untuk memastikan hasil pelatihan dapat direproduksi.
test_size	0.2	Rasio pembagian dataset antara data latih dan data uji sebesar 80:20.
cross_validation	5-Fold	Validasi silang digunakan untuk mengevaluasi kestabilan dan konsistensi performa model.



Gambar 3.6 Arsitektur Model Random Forest

3.4.2. Arsitektur Model LSTM

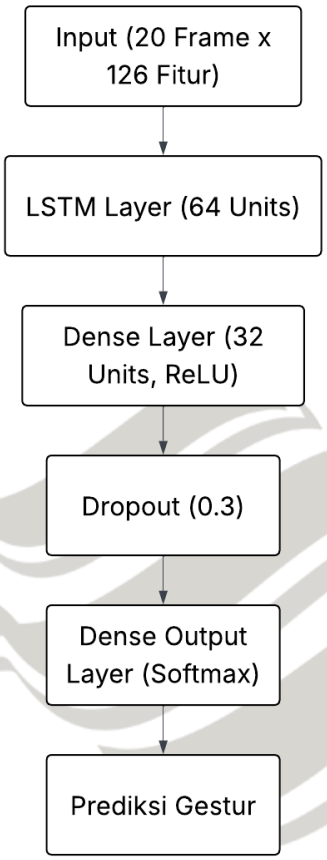
Model LSTM digunakan untuk mengklasifikasikan gestur dinamis, yaitu pola bahasa isyarat yang terdiri dari urutan beberapa *frame* berturut-turut. Dataset sekuensial disimpan dalam format NPZ, yang memuat representasi koordinat landmark

tangan untuk setiap *frame*. Setiap sampel masukan memiliki dimensi 20 *frame* dengan 126 fitur per *frame*, merepresentasikan pergerakan tangan dalam bentuk sekuensial. Arsitektur LSTM terdiri dari beberapa lapisan berurutan yang dirancang untuk menangkap hubungan temporal antar *frame*, dilengkapi dengan *dropout* sebagai regularisasi, serta lapisan *Dense* dengan aktivasi ReLU sebelum menuju lapisan keluaran (*output layer*) yang menggunakan fungsi aktivasi *softmax* untuk klasifikasi multi-kelas. Ilustrasi arsitektur LSTM ditunjukkan pada Gambar 3.7, yang menggambarkan alur pemrosesan data sekuensial dari input hingga output dengan mekanisme memori jangka panjang antar lapisan. Detail konfigurasi arsitektur dan *hyperparameter* model LSTM yang digunakan dalam penelitian ini ditampilkan pada Tabel 3.4. Nilai parameter ditentukan berdasarkan hasil percobaan awal untuk memperoleh keseimbangan antara akurasi, stabilitas pelatihan, serta efisiensi komputasi agar dapat diimplementasikan secara optimal pada perangkat *Raspberry Pi*. Proses pelatihan dilakukan dengan rasio data latih dan uji sebesar 80:20, serta menggunakan validasi silang (*5-Fold Cross Validation*) untuk memastikan model memiliki kemampuan generalisasi yang baik terhadap variasi data gestur dinamis.

Tabel 3.4 Hyperparameter Model LSTM

Parameter	Nilai	Keterangan
Input Shape	(20, 126)	Setiap sampel berisi 20 frame dengan 126 fitur koordinat per frame.
Lapisan LSTM	64 unit, return_sequences=False, unroll=True	Menangkap hubungan temporal antar frame gestur dinamis.
Lapisan Dense	32 neuron, aktivasi ReLU, regularisasi L2(0.001)	Menyaring fitur hasil ekstraksi dari LSTM dan mencegah overfitting.

Parameter	Nilai	Keterangan
Lapisan Dropout	0.3	Mengurangi risiko overfitting dengan menonaktifkan sebagian neuron selama pelatihan.
Lapisan Output	Dense(num_classes, activation='softmax')	Menghasilkan probabilitas untuk setiap kelas gestur.
Optimizer	Adam	Menyesuaikan bobot model secara adaptif untuk mempercepat konvergensi.
Loss Function	Categorical Crossentropy	Digunakan untuk tugas klasifikasi multi-kelas.
Batch Size	16	Jumlah sampel yang diproses dalam satu iterasi pelatihan.
Epochs Maksimal	300	Jumlah iterasi pelatihan dengan early stopping otomatis jika validasi tidak membaik.
Validation Split	5-Fold Cross Validation	Mengukur stabilitas performa model di setiap lipatan data.
Early Stopping	patience=10, restore_best_weights=True	Menghentikan pelatihan lebih awal untuk mencegah overfitting.
Learning Rate Scheduler	ReduceLROnPlateau (factor=0.5, patience=5, min_lr=1e-6)	Menurunkan learning rate secara adaptif ketika validasi stagnan.



Gambar 3.7 Arsitektur Model LSTM

3.4.3. Arsitektur Model Transformer

Sementara itu, model *Transformer* dirancang untuk mendeteksi pola temporal gestur dinamis menggunakan mekanisme *self-attention*. Dataset yang sama dalam format NPZ digunakan untuk melatih model *Transformer*. Proses pelatihan dilakukan dengan membagi data menjadi *mini-batch* sekuensial, kemudian melewati blok *encoder Transformer* yang terdiri dari lapisan *multi-head attention*, normalisasi, dan *feed-forward network*. Lapisan keluaran menggunakan fungsi aktivasi *softmax* dengan jumlah *neuron* sesuai jumlah kelas gestur, sehingga mampu menghasilkan distribusi probabilitas atas setiap kategori. Rancangan arsitektur *Transformer* ini divisualisasikan pada Gambar 3.8, yang menampilkan dua blok *encoder* bertingkat yang memproses informasi temporal secara paralel untuk memperoleh representasi fitur yang

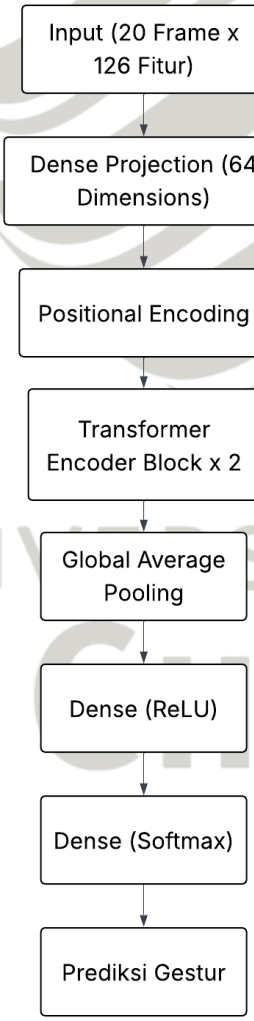
kontekstual. Detail konfigurasi arsitektur dan *hyperparameter* model *Transformer* yang digunakan dalam penelitian ini disajikan pada Tabel 3.5. Nilai-nilai parameter ditentukan melalui proses eksplorasi empiris untuk menyeimbangkan antara kompleksitas model dan efisiensi komputasi, mengingat model ini akan diimplementasikan pada perangkat *Raspberry Pi*. Pelatihan dilakukan menggunakan pembagian data sebesar 80% untuk pelatihan dan validasi serta 20% untuk pengujian, dengan skema *5-Fold Cross Validation* pada data latih. Tujuannya adalah untuk memastikan bahwa model *Transformer* mampu mempelajari hubungan temporal antar *frame* secara efektif melalui mekanisme self-attention, sekaligus menjaga stabilitas generalisasi terhadap variasi gerakan tangan antar partisipan.

Tabel 3.5 Hyperarameter Model Transformer

Parameter	**Nilai**	**Keterangan**
Input Shape	(20,126)	Setiap sampel terdiri dari 20 frame dengan 126 fitur koordinat 3D dari 21 titik tangan per frame.
Dense Projection Layer	64 unit	Mengubah dimensi fitur input menjadi representasi vektor berdimensi tetap (d_model).
Positional Embbeding	Panjang sekuens 20	Menambahkan informasi posisi tiap frame agar model memahami urutan temporal.
Jumlah Blok Encoder	2	Tiap blok terdiri dari Multi-Head Attention, Layer Normalization, dan Feed Forward Network.

Parameter	Nilai	Keterangan
Multi-Head Attention	4 head, key_dim = 64	Menangkap hubungan antar frame dari berbagai perspektif secara paralel.
Feed Forward Network	64 unit, aktivasi ReLU	Mengubah representasi hasil attention menjadi fitur non-linear yang lebih dalam.
Dropout Rate	0.3	Mengurangi risiko overfitting selama pelatihan.
Global Average Pooling	-	Merata-ratakan keluaran temporal menjadi satu vektor global.
Dense Layer	64 → Softmax (jumlah kelas)	Menghasilkan distribusi probabilitas untuk setiap kelas gestur.
Optimizer	Adam (learning rate = 0.001)	Menyesuaikan bobot secara adaptif agar konvergensi cepat dan stabil.
Loss Function	Sparse Categorical Crossentropy	Cocok untuk klasifikasi multi-kelas dengan label integer.
Batch Size	16	Ukuran mini-batch pada proses pelatihan.
Epochs Maksimal	300	Proses pelatihan berhenti lebih awal jika performa validasi tidak meningkat.
Validation Split	5-Fold Cross Validation	Mengukur konsistensi performa pada tiap lipatan data.

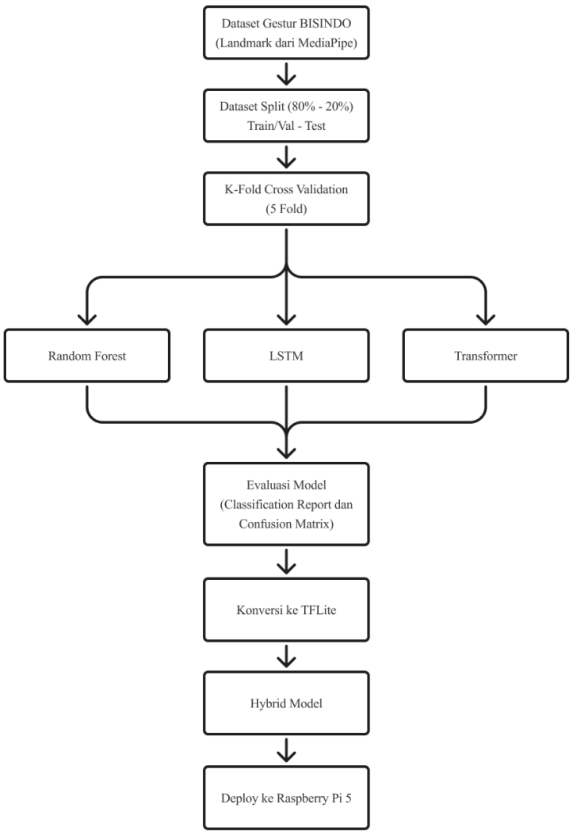
Parameter	Nilai	Keterangan
Early Stopping	patience = 10	Menghentikan pelatihan otomatis untuk mencegah overfitting.
Learning Rate Scheduler	ReduceLROnPlateau (factor=0.5, patience=5)	Menurunkan learning rate secara adaptif ketika validasi stagnan.



Gambar 3.8 Arsitektur Model Transformer

Setelah seluruh model selesai dirancang, hasil pelatihan akan dikonversi ke format *TensorFlow Lite* (TFLite) menggunakan skrip konversi. Format ini dipilih karena lebih ringan dan efisien untuk dijalankan pada perangkat dengan keterbatasan sumber daya. Selanjutnya, model *Random Forest*, LSTM, dan *Transformer* akan diintegrasikan dalam sebuah arsitektur *hybrid*. *Hybrid* model ini memungkinkan perbandingan performa antara LSTM dan *Transformer* dalam mengenali gestur dinamis, sementara *Random Forest* tetap menangani gestur statis. Setelah integrasi, keseluruhan sistem kemudian diimplementasikan pada Raspberry Pi 5 dengan input dari PiCamera v1, sehingga mampu melakukan pengenalan BISINDO secara *real-time* dan portabel.

Alur proses pembentukan model hingga implementasi sistem secara keseluruhan dapat dilihat pada Gambar 3.9.

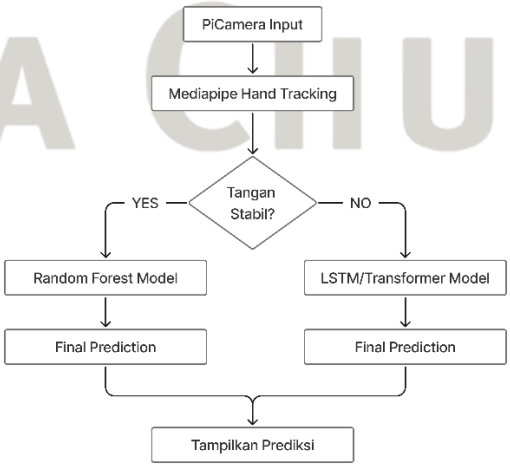


Gambar 3.9 Flowchart Pembuatan Model

3.5. Hybrid Model

Hybrid model dalam penelitian ini dirancang untuk menangani perbedaan karakteristik antara gestur statis dan gestur dinamis pada bahasa isyarat BISINDO. Gestur statis, seperti huruf alfabet dan angka, lebih sesuai diklasifikasikan menggunakan *Random Forest* karena cukup dilihat sebagai snapshot tunggal tanpa pola pergerakan. Sementara itu, gestur dinamis seperti kata atau frasa memerlukan analisis temporal antarframe, sehingga lebih tepat diproses menggunakan model sekuensial seperti LSTM maupun *Transformer*.

Dalam implementasinya, sistem terlebih dahulu mengekstrak landmark tangan dari input kamera menggunakan *MediaPipe*. Jika gerakan tangan terdeteksi stabil pada beberapa *frame*, input dianggap sebagai gestur statis dan diproses oleh *Random Forest*. Sebaliknya, ketika terdapat perubahan posisi tangan yang signifikan dalam urutan *frame*, maka data sekuensial akan diteruskan ke model sekuensial (LSTM atau *Transformer*) untuk mengenali pola gerakan dinamis. Alur proses pengambilan keputusan antara model statis dan dinamis tersebut ditunjukkan pada Gambar 3.10, yang menggambarkan topologi sistem *hybrid* dalam mengenali gestur secara *real-time* pada perangkat *Raspberry Pi*.



Gambar 3.10 Diagram Alur Hybrid Model

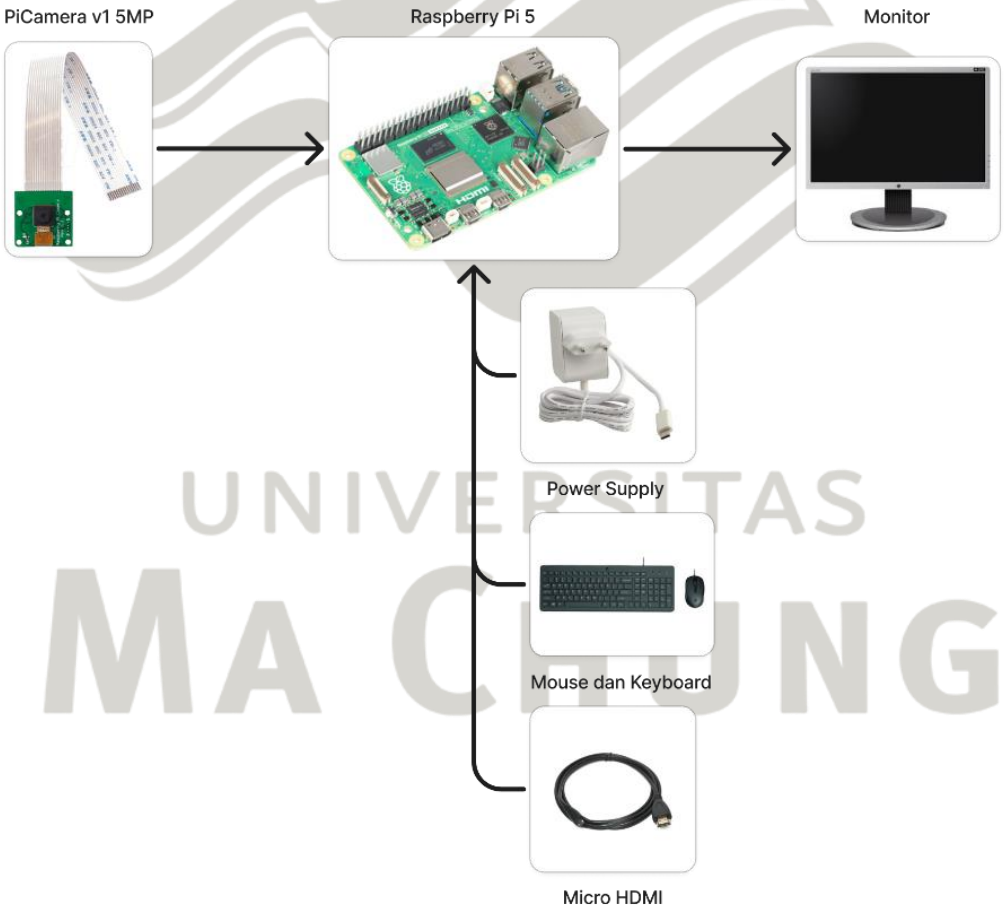
Berdasarkan Gambar 3.10, sistem *hybrid* dimulai dari proses akuisisi citra secara *real-time* menggunakan kamera yang terhubung ke *Raspberry Pi*. Setiap frame yang diterima kemudian dikonversi menjadi format RGB dan diproses oleh *MediaPipe Hands* untuk mendeteksi serta mengekstraksi 21 titik koordinat landmark dari satu atau dua tangan pengguna. Koordinat tersebut disimpan secara berurutan dalam bentuk urutan *frame* agar dapat dianalisis lebih lanjut oleh sistem.

Selanjutnya, sistem menganalisis perubahan posisi tangan antar *frame* untuk menentukan apakah gerakan yang dilakukan bersifat statis atau dinamis. Jika pergerakan tangan terdeteksi stabil dalam beberapa *frame* berturut-turut, maka sistem menganggapnya sebagai gestur statis dan mengirimkan data ke model *Random Forest* untuk klasifikasi. Sebaliknya, apabila sistem mendeteksi adanya perubahan posisi tangan yang signifikan dalam rentang waktu tertentu, maka urutan *frame* tersebut diteruskan ke model LSTM *TensorFlow Lite* / *Transformer TensorFlow Lite* untuk mengenali pola pergerakan dinamis.

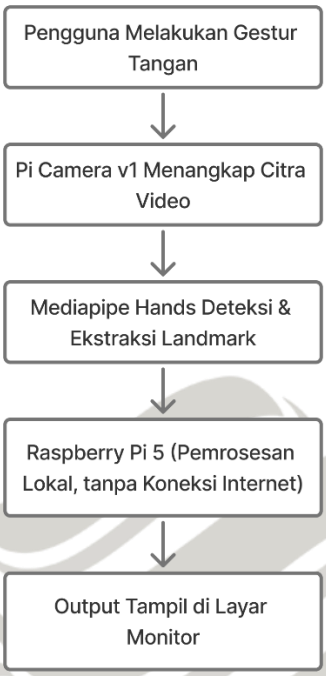
Hasil klasifikasi dari kedua model akan ditampilkan secara langsung di layar dengan keterangan jenis model yang digunakan. Sistem juga memberikan jeda waktu singkat setelah setiap prediksi untuk mencegah pengenalan ganda terhadap gestur yang sama. Dengan pendekatan ini, sistem mampu melakukan klasifikasi secara adaptif berdasarkan pola pergerakan tangan pengguna, sehingga menghasilkan proses deteksi gestur yang efisien dan akurat pada perangkat *Raspberry Pi 5* tanpa memerlukan koneksi internet.

Agar sistem tetap ringan dijalankan pada *Raspberry Pi 5*, penelitian ini menggunakan pendekatan hybrid bergantian. Skenario pertama adalah kombinasi RF + LSTM untuk menguji akurasi klasifikasi statis dan dinamis. Skenario kedua adalah kombinasi RF + *Transformer* dengan fungsi yang sama. Hasil dari kedua skenario tersebut kemudian dibandingkan untuk mengevaluasi performa model sekuensial (LSTM dan *Transformer*) dalam konteks pengenalan BISINDO secara *real-time*. Dengan strategi ini, pengujian dapat dilakukan secara objektif tanpa membebani perangkat keras karena hanya dua model dijalankan pada satu waktu.

Selain arsitektur algoritmik, penelitian ini juga memperhatikan topologi perangkat keras yang digunakan. Sistem terdiri atas *Raspberry Pi 5* sebagai pusat komputasi, kamera CSI *PiCamera v1* sebagai perangkat akuisisi data visual, serta kipas pendingin untuk menjaga kestabilan suhu operasional. *Raspberry Pi* terhubung ke monitor melalui *micro*-HDMI untuk menampilkan hasil klasifikasi secara *real-time*, dengan dukungan catu daya eksternal 5V 3A. Topologi perangkat ini divisualisasikan pada Gambar 3.11, yang menunjukkan susunan komponen serta keterhubungan antarperangkat dalam sistem secara keseluruhan.



Gambar 3.11 Topologi Perangkat



Gambar 3.12 Topologi Tanpa Internet

Topologi sistem yang digunakan dalam penelitian ini bersifat sepenuhnya *offline*, Visualisasi arsitektur ini disajikan pada Gambar 3.12, di mana seluruh proses deteksi dan klasifikasi dilakukan secara lokal pada *Raspberry Pi 5* tanpa memerlukan koneksi internet. Sistem diawali dengan pengguna yang memperagakan gestur tangan di depan kamera (menggunakan *Pi Camera v1*). Kamera berfungsi menangkap citra tangan secara *real-time*, yang kemudian dikirim ke *MediaPipe Hands* untuk mendeteksi dan mengekstraksi titik-titik koordinat tangan (*landmark*) dalam bentuk tiga dimensi (x, y, z).

Hasil ekstraksi landmark ini menjadi masukan bagi sistem klasifikasi yang terdiri dari tiga model, yaitu *Random Forest*, serta LSTM dan *Transformer* yang berbasis *TensorFlow Lite*. Model *Random Forest* digunakan untuk mengenali gestur statis, yaitu gestur yang tidak melibatkan pergerakan tangan secara berurutan antar *frame*.

Sementara itu, LSTM dan *Transformer* digunakan untuk mengenali gestur dinamis yang mengandung urutan gerakan dalam waktu tertentu. Kedua model ini

bekerja dengan cara menganalisis pola sekuensial dari 20 *frame* berurutan (*sequence*) untuk menentukan jenis gestur yang dilakukan.

Dalam implementasinya, baik model LSTM maupun *Transformer* dikonversi ke dalam format *TensorFlow Lite* (TFLite) agar dapat dijalankan secara efisien pada perangkat *Raspberry Pi* yang memiliki sumber daya terbatas. Penggunaan format TFLite pada kedua model ini bertujuan untuk memastikan perbandingan performa yang adil dalam hal kecepatan inferensi dan penggunaan memori, khususnya untuk melihat kemampuan *Transformer* sebagai pembanding terhadap LSTM dalam mengenali urutan gerakan dengan kompleksitas tinggi.

Seluruh proses mulai dari penangkapan citra, deteksi *landmark*, klasifikasi gestur, hingga penampilan hasil di layar dilakukan secara lokal dan *offline*, sehingga sistem dapat beroperasi tanpa ketergantungan terhadap server eksternal atau koneksi jaringan.

Sistem penerjemah BISINDO berbasis *Raspberry Pi* ini dibangun dengan memanfaatkan kombinasi sumber daya perangkat keras dan perangkat lunak yang mendukung pengolahan citra serta inferensi model secara *offline*. Seluruh komponen yang digunakan dirancang agar sistem dapat berjalan mandiri tanpa ketergantungan pada koneksi internet. Komponen utama yang digunakan meliputi *Raspberry Pi 5* sebagai unit pemrosesan utama, *Pi Camera v1* untuk menangkap citra tangan, serta perangkat pendukung seperti *monitor*, *keyboard*, dan *mouse* untuk interaksi pengguna.

Dari sisi perangkat lunak, sistem dikembangkan menggunakan bahasa pemrograman *Python* dengan dukungan pustaka *MediaPipe Hands* untuk ekstraksi landmark tangan, *OpenCV* untuk pemrosesan citra, dan *TensorFlow Lite* sebagai mesin inferensi ringan untuk model LSTM dan *Transformer* yang dijalankan pada *Raspberry Pi*. Selain itu, sistem juga mengintegrasikan model *Random Forest* untuk mendeteksi gestur statis, serta LSTM dan *Transformer* (TFLite) untuk mengenali gestur dinamis. Penggunaan *TensorFlow Lite* bertujuan agar model dapat berjalan lebih efisien dengan konsumsi memori rendah tanpa mengorbankan kecepatan prediksi.

Rincian lengkap sumber daya perangkat keras dan perangkat lunak yang digunakan dalam penelitian ini dapat dilihat pada Tabel 3.6.

Tabel 3.6 Sumber Daya Sistem

Jenis Sumber Daya	Perangkat/Komponen	Keterangan
Perangkat Keras (Hardware)	Raspberry Pi 5	Unit pemrosesan utama dengan 16 GB RAM, menjalankan seluruh proses inferensi dan klasifikasi secara offline
	Pi Camera Module v1	Kamera utama dengan sensor OV5647, digunakan untuk menangkap citra tangan secara *real-time*
	Monitor, Keyboard, Mouse	Antarmuka pengguna untuk menampilkan hasil dan melakukan interaksi selama pengujian
	MicroSD 32 GB	Media penyimpanan sistem operasi, program, dan model machine learning
Perangkat Lunak (Software)	Raspberry Pi OS	Sistem operasi utama berbasis Linux
	Python	Bahasa pemrograman utama untuk pengembangan sistem
	OpenCV	Digunakan untuk pengambilan citra dan pengolahan frame video
	MediaPipe Hands	Untuk deteksi dan ekstraksi 21 titik landmark tangan dalam format koordinat (x, y, z)
	TensorFlow & TensorFlow Lite	Framework machine learning; TFLite digunakan untuk menjalankan model LSTM secara efisien di Raspberry Pi

Jenis Sumber Daya	Perangkat/Komponen	Keterangan
	Scikit-learn	Digunakan untuk implementasi dan inferensi model Random Forest
	NumPy & Pandas	Digunakan untuk manipulasi dan analisis data landmark
	Matplotlib	Untuk visualisasi hasil pelatihan dan evaluasi model
	Joblib	Untuk memuat model Random Forest yang telah dilatih sebelumnya

Bab IV
Hasil dan Pembahasan

4.1. Profil Partisipan

Berikut adalah data subjek deteksi bahasa isyarat BISINDO menggunakan input kamera secara *real-time* pada Raspberry Pi yang terlampir pada Tabel 4.1..

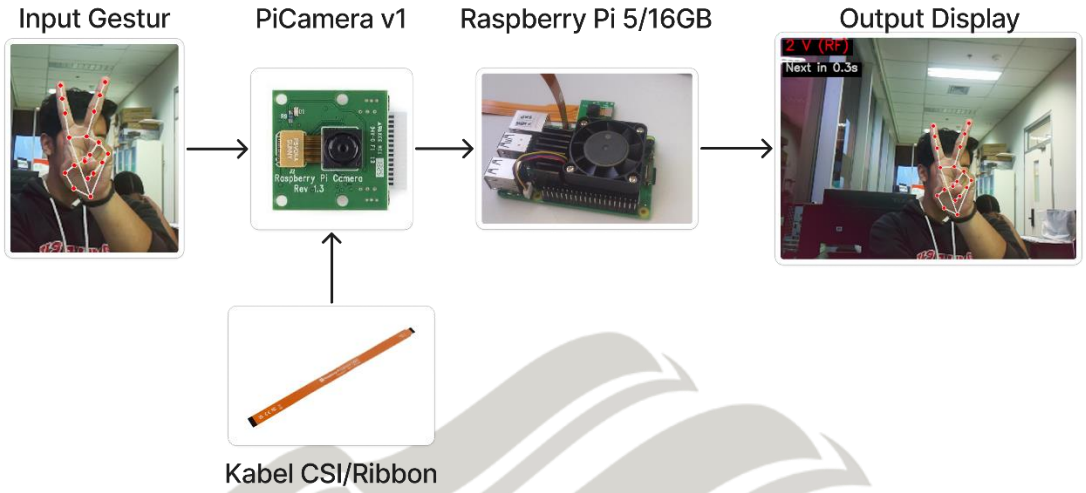
Tabel 4.1 Tabel Data Subjek

No	Umur	Jenis Kelamin	Profesi
1	21 Tahun	Laki - Laki	Mahasiswa
2	23 Tahun	Perempuan	Mahasiswa
3	21 Tahun	Laki - Laki	Mahasiswa
4	21 Tahun	Laki - Laki	Mahasiswa
5	21 Tahun	Perempuan	Mahasiswa
6	23 Tahun	Laki - Laki	Mahasiswa

4.2. Implementasi Sistem

4.2.1. Implementasi Perangkat Keras (*Hardware*)

(a)



(b)

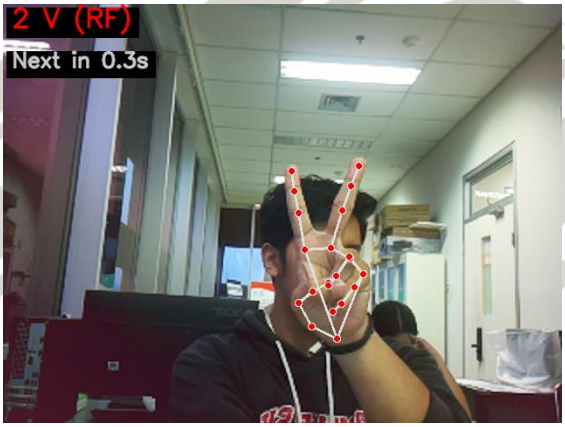
Gambar 4.1 Implementasi Perangkat Keras: (a) Setup Perangkat Keseluruhan dan (b) Diagram Blok Koneksi Perangkat.

Realisasi perangkat keras pada sistem ini berfokus pada integrasi unit *embedded system* yang berperan sebagai pusat pemrosesan algoritma kecerdasan buatan secara mandiri. Komponen utama yang digunakan adalah Raspberry Pi 5 yang bertugas menjalankan sistem operasi sekaligus mengeksekusi beban komputasi dari arsitektur model hibrida yang diterapkan. Untuk kebutuhan akuisisi visual, sistem ini memanfaatkan modul kamera khusus yaitu *Pi Camera v1* yang dihubungkan langsung ke papan utama melalui antarmuka serial berkecepatan tinggi, bukan melalui jalur USB standar. Penggunaan jalur antarmuka dedikasi ini dipilih untuk meminimalisir latensi pengiriman data citra (*frame*), sehingga proses deteksi gestur tangan baik yang bersifat diam maupun bergerak dapat direspons oleh sistem tanpa penundaan yang signifikan.

Implementasi fisik dari konfigurasi tersebut diperlihatkan secara rinci pada Gambar 4.1. Dalam ilustrasi tersebut, unit komputasi utama terlihat ditempatkan pada permukaan datar dan terhubung ke sensor visual menggunakan kabel fleksibel pipih, sementara modul kamera diposisikan secara strategis menempel pada bingkai layar monitor eksternal. Penempatan ini bertujuan untuk mendapatkan sudut pandang yang

optimal dan sejajar dengan pengguna, memastikan area tubuh bagian atas dan pergerakan tangan masuk sepenuhnya ke dalam bingkai tangkapan kamera. Hasil pemrosesan *visual* dan antarmuka pengguna kemudian ditransmisikan ke *monitor* tersebut, memungkinkan pengguna untuk melihat umpan balik sistem secara langsung saat melakukan gerakan isyarat. Sementara itu, perangkat periferal standar diintegrasikan di sekitar unit utama untuk keperluan inisialisasi program dan konfigurasi sistem selama tahap pengujian berlangsung.

4.2.2. Implementasi Antarmuka Pengguna (*User Interface*)



Gambar 4.2 Implementasi Antarmuka Program

Implementasi antarmuka pada sistem, sebagaimana divisualisasikan pada Gambar 4.2, dirancang untuk menyediakan umpan balik visual secara *real-time* dengan memanfaatkan pustaka *OpenCV* sebagai *backend* visualisasi utama. Alur pemrosesan visual dimulai dengan membalikkan *frame* kamera secara horizontal (*horizontal flip*) menggunakan fungsi *cv2.flip* untuk menciptakan efek cermin yang intuitif bagi pengguna. Di atas lapisan video tersebut, sistem mengintegrasikan modul visualisasi *MediaPipe* (*mp_draw*) yang memetakan kerangka tangan (*hand landmarks*) beserta garis koneksi antar sendi.

Mekanisme interaksi pengguna dibangun berbasis *state-driven display* yang memberikan panduan tekstual pada setiap tahapan klasifikasi. Untuk menjamin keterbacaan informasi di berbagai kondisi pencahayaan latar, seluruh elemen teks

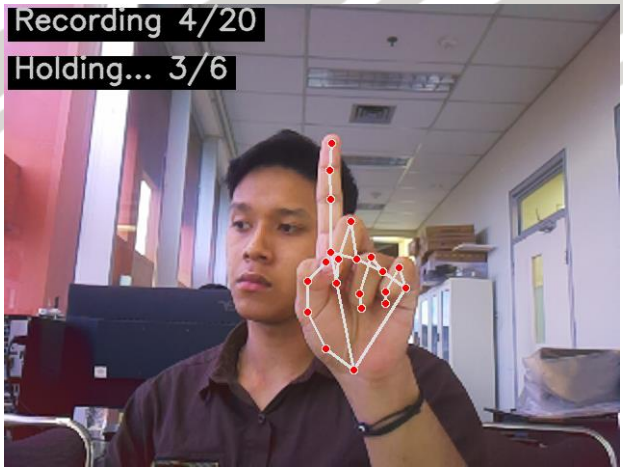
dirender menggunakan fungsi kustom yang menambahkan latar belakang persegi panjang berwarna hitam pada setiap label informasi. Sistem secara dinamis menampilkan status sistem berdasarkan stabilitas gerakan tangan:

- Fase Stabilisasi: Visualisasi fase stabilisasi sistem dapat dilihat pada Gambar 4.3. Saat tangan pertama kali dideteksi, sistem mengaktifkan timer jeda (*grace time*) dan menampilkan status "*Stabilizing...*" untuk mencegah pengambilan data yang prematur.

Gambar 4.3 Fase Stabilisasi

- Fase Akuisisi Data: Berdasarkan kalkulasi rata-rata pergerakan (*motion score*), antarmuka akan menampilkan status sistem sebagaimana divisualisasikan pada Gambar 4.4 dan Gambar 4.5. Sistem akan menampilkan status "*Recording*" (Gambar 4.4) yang menghitung jumlah buffer sekuensial untuk model dinamis, atau beralih ke status "*Holding*" (Gambar 4.5) jika gerakan berada di bawah ambang batas (*threshold*) 0.005 untuk memicu klasifikasi statis.

Gambar 4.4 Fase Recording



Gambar 4.5 Fase Holding

Sebagai bentuk validasi akhir kepada pengguna, hasil prediksi ditampilkan dengan pembedaan visual yang tegas berdasarkan model yang melakukan inferensi. Hasil klasifikasi dari model *Random Forest* (gestur statis) direpresentasikan dengan label teks berwarna merah, sedangkan luaran dari model *Deep Learning* (LSTM atau *Transformer*) ditandai dengan warna hijau. Setiap label prediksi juga menyertakan sufiks sumber model (misalnya "(RF)", "(LSTM)", atau "(*Transformer*)") untuk transparansi proses *hybrid* yang berjalan di latar belakang. Selain itu, fitur *cooldown*

visual diterapkan pasca-prediksi dengan menampilkan hitung mundur ("*Next in...*") untuk mencegah redundansi luaran pada satu gerakan yang sama.

4.3. Implementasi dan Analisis Kode Program

4.3.1. Implementasi Augmentasi Citra untuk Gestur Statis

Pada tahap awal eksperimen pelatihan model *Random Forest*, ditemukan indikasi *overfitting* di mana model memiliki performa sangat tinggi pada data latih namun kurang optimal dalam mengenali variasi gestur pada pengujian *real-time*. Hal ini disebabkan oleh terbatasnya variasi posisi dan skala tangan pada dataset murni hasil pengambilan awal. Untuk mengatasi permasalahan tersebut, penelitian ini mengimplementasikan teknik augmentasi citra (*image augmentation*) secara terprogram sebelum data diekstraksi fitur landmark-nya.

Implementasi augmentasi dilakukan menggunakan pustaka *OpenCV* dengan menerapkan transformasi geometris yang tetap mempertahankan makna semantik gestur. Kode program dirancang untuk menghasilkan variasi data baru secara otomatis melalui tiga teknik manipulasi utama:

- Rotasi (*Rotation*): Citra diputar dengan sudut acak antara -10 hingga 10 derajat untuk mensimulasikan orientasi tangan pengguna yang tidak selalu tegak lurus.
- Penskalaan (*Scaling/Safe Zoom*): Citra diperbesar atau diperkecil dengan rasio 0.85 hingga 1.05. Teknik *padding* (penambahan piksel hitam) diterapkan saat *zoom-out* untuk memastikan tidak ada bagian tangan yang terpotong.
- Translasi (*Translation*): Objek tangan digeser secara horizontal atau vertikal dalam rentang 5% dari dimensi citra untuk mengantisipasi posisi tangan yang tidak selalu tepat di tengah frame.

Realisasi teknis dari ketiga transformasi tersebut ditunjukkan pada Gambar 4.6 berikut, yang memperlihatkan fungsi inti *safe_augment_image* dalam memproses matriks citra input.

```
1    def safe_augment_image(image):
2        h, w = image.shape[:2]
3
```

```
        # 1. ROTASI (-10 s/d 10 derajat)
        angle = random.uniform(-10, 10)
        center = (w // 2, h // 2)
        M_rot = cv2.getRotationMatrix2D(center, angle, 1.0)
        image = cv2.warpAffine(image, M_rot, (w, h), borderMode=cv2.BORDER_CONSTANT)

        # 2. SAFE ZOOM (Fokus Zoom Out untuk menjaga fitur tangan)
        scale = random.uniform(0.85, 1.05)

        if scale < 1.0: # Logika Zoom Out dengan Padding
            new_h, new_w = int(h * scale), int(w * scale)
            resized = cv2.resize(image, (new_w, new_h))

            # Buat kanvas hitam seukuran asli agar dimensi tetap terjaga
            canvas = np.zeros((h, w, 3), dtype=np.uint8)
            y_off = (h - new_h) // 2
            x_off = (w - new_w) // 2
            canvas[y_off:y_off+new_h, x_off:x_off+new_w] = resized
            image = canvas

        # 3. TRANSLASI (Geser Posisi max 5%)
        tx = random.uniform(-0.05, 0.05) * w
        ty = random.uniform(-0.05, 0.05) * h
        M_trans = np.float32([[1, 0, tx], [0, 1, ty]])
        image = cv2.warpAffine(image, M_trans, (w, h), borderMode=cv2.BORDER_CONSTANT)

        return image
```

Gambar 4.6 Cuplikan Kode Augmentasi Citra

Melalui skrip augmentasi di atas, jumlah dataset gestur statis dilipatgandakan dengan faktor pengali (*AUGMENT_MULTIPLIER*) sebesar 10 kali lipat. Proses ini memungkinkan model *Random Forest* untuk mempelajari distribusi fitur *landmark* yang lebih luas dan menghasilkan model yang lebih *robust* terhadap variasi pengambilan gambar di lapangan.

Augmentasi data difokuskan secara intensif pada kategori gestur statis untuk mengatasi variabilitas posisi dan orientasi tangan yang sering memicu misklasifikasi pada pengujian awal.

Sebaliknya, pada kategori gestur dinamis, augmentasi data sintetis tidak diterapkan. Keputusan ini didasarkan pada hasil eksperimen pendahuluan yang menunjukkan bahwa fitur temporal pada gestur dinamis memiliki tingkat perbedaan yang sangat kuat dibandingkan gestur statis. Model terbukti mampu mencapai generalisasi yang optimal dan performa klasifikasi yang tinggi hanya dengan menggunakan variasi data natural. Oleh karena itu, integritas data urutan waktu pada gestur dinamis dipertahankan tanpa modifikasi buatan untuk mencegah distorsi pola gerakan yang justru dapat menurunkan akurasi deteksi pada skenario *real-time*.

4.3.2. Implementasi Model Random Forest (Gestur Statis)

Implementasi klasifikasi gestur statis dibangun menggunakan algoritma *Random Forest* dengan memanfaatkan pustaka *Scikit-learn*. Sebelum proses pelatihan, dataset yang memuat koordinat landmark tangan dipisahkan menjadi set fitur dan label. Untuk menjaga distribusi kelas yang seimbang antara data latih dan data uji, diterapkan teknik *stratified sampling* saat pembagian dataset dengan rasio 80:20. Hal ini diimplementasikan menggunakan fungsi *train_test_split* dengan parameter *stratify*, yang memastikan setiap kelas gestur terwakili secara proporsional di kedua subset data, sehingga mencegah bias pada model saat proses evaluasi.

```
1    # Split dataset (80% train, 20% test) dengan stratify
2    X_train, X_test, y_train, y_test = train_test_split(
3        X, y_encoded, test_size=0.2, stratify=y_encoded,
random_state=42
4    )
```

Gambar 4.7 Code Pembagian Dataset dengan Stratifikasi

Konfigurasi model diatur berdasarkan parameter yang telah ditentukan pada tahap perancangan. Model diinisialisasi dengan parameter *n_estimators=100*, yang berarti model akan membentuk seratus pohon keputusan (*decision trees*) untuk

melakukan prediksi secara *ensemble*. Kriteria pemisahan *node* menggunakan indeks *gini* untuk mengukur tingkat *impurity*. Selanjutnya, untuk menguji konsistensi performa model terhadap variasi data yang berbeda, diterapkan metode *Stratified K-Fold Cross Validation* dengan k=5. Metode ini membagi data latih menjadi lima lipatan (*folds*) berbeda, di mana model dilatih dan divalidasi secara iteratif pada setiap lipatan untuk mendapatkan metrik akurasi rata-rata yang lebih reliabel.

```
1   # Inisialisasi model Random Forest dengan 100 pohon
2   rf = RandomForestClassifier(
3       n_estimators=100,
4       criterion="gini",
5       max_depth=None,
6       random_state=42
7   )
8
9   # Penerapan 5-Fold Cross Validation
10  skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)
11  cv_scores = cross_val_score(rf, X, y_encoded,
cv=skf)
```

Gambar 4.8 Code Konfigurasi Model dan Cross-Validation

4.3.3. Implementasi Model LSTM (Gestur Dinamis)

Pada pengenalan gestur dinamis, arsitektur *Long Short-Term Memory* (LSTM) diimplementasikan menggunakan kerangka kerja *TensorFlow Keras*. Model disusun secara sekuensial (*Sequential Model*) untuk memproses data urutan (*sequence data*) dengan panjang tetap, yaitu 20 *frame* per sampel. Lapisan inti LSTM dikonfigurasi dengan 64 unit memori. Parameter krusial yang diterapkan di sini adalah *unroll=True*. Pengaturan ini memaksa jaringan untuk membuka gulungan (*unroll*) *loop* LSTM saat kompilasi, yang bertujuan untuk meningkatkan kecepatan eksekusi pada saat inferensi, meskipun dengan konsekuensi penggunaan memori yang sedikit lebih besar. Selanjutnya, hasil pemrosesan LSTM diteruskan ke lapisan *Dense* dengan 32 *neuron*

yang dilengkapi aktivasi *ReLU* dan regularisasi L2 (*kernel_regularizer*) sebesar 0.001 untuk membatasi besaran bobot dan mencegah overfitting.

```
1   model = models.Sequential([
2       layers.Input(shape=(SEQ_LENGTH, FEATURES)),
3       # Unroll=True untuk mempercepat eksekusi (speed
optimization)
4       layers.LSTM(64,          return_sequences=False,
unroll=True),
5       # Regularisasi L2 untuk mencegah overfitting pada
dense layer
6       layers.Dense(32,              activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
7       layers.Dropout(0.3),
8       layers.Dense(num_classes, activation='softmax')
9   ])
```

Gambar 4.9 Code Arsitektur Model LSTM

Untuk mengoptimalkan proses pembelajaran, model dikompilasi menggunakan pengoptimal (*optimizer*) *adam* dan fungsi kerugian *categorical_crossentropy* yang sesuai untuk klasifikasi multi-kelas. Strategi pelatihan diperkuat dengan penerapan mekanisme *callbacks*. *EarlyStopping* digunakan untuk memantau nilai *validation loss* dan akan menghentikan pelatihan secara otomatis jika tidak terjadi penurunan *loss* selama 10 *epoch* berturut-turut (parameter *patience*=10), serta mengembalikan bobot terbaik yang pernah dicapai (*restore_best_weights=True*). Selain itu, *ReduceLROnPlateau* diterapkan untuk menurunkan *learning rate* sebesar 50% (faktor 0.5) jika model mengalami stagnasi, memungkinkan model untuk mencari *local minima* yang lebih presisi.

```
1   model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
2
3   callbacks = [
4       # Hentikan training jika val_loss tidak membaik
dalam 10 epoch
5       EarlyStopping(monitor="val_loss",   patience=10,
```

```
restore_best_weights=True),
6       # Turunkan learning rate jika performa stagnan
7       ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=5, min_lr=1e-6)
8   ]
```

Gambar 4.10 Code Konfigurasi Pelatihan dan Callbacks

4.3.4. Implementasi Model Transformer (Gestur Dinamis)

Sebagai pembanding utama terhadap model LSTM, penelitian ini mengimplementasikan arsitektur *Transformer* yang menawarkan pendekatan berbeda dalam mempelajari pola waktu. Jika model rekuren seperti LSTM memproses data secara berurutan (sekuensial) yang seringkali membatasi kecepatan pelatihan *Transformer* dirancang untuk memproses seluruh urutan data sekaligus secara paralel. Namun, keunggulan pemrosesan paralel ini membawa tantangan tersendiri: model kehilangan pemahaman inheren mengenai urutan waktu. Tanpa penanda khusus, *Transformer* tidak dapat membedakan antara gerakan awal dan gerakan akhir dalam sebuah gestur. Untuk mengatasi hal ini, langkah pertama dalam implementasi kode adalah melakukan proyeksi fitur *input* ke dalam dimensi *internal model* (*Dense Projection*) sebesar 64 *unit*, yang kemudian dipadukan dengan informasi posisi (*Positional Embedding*). Teknik ini secara efektif mengintegrasikan"penanda waktu" ke dalam setiap *frame data*, sehingga model dapat memahami konteks urutan gerakan dengan benar misalnya, membedakan arah gerakan tangan dari kiri ke kanan dengan sebaliknya meskipun seluruh data diproses secara bersamaan.

```
1   # Proyeksi fitur input ke dimensi d_model
2   x = layers.Dense(64, name="dense_projection")(inputs)

3   # Membuat dan menambahkan Positional Embedding
4   positions = tf.range(start=0, limit=input_shape[0],
delta=1)
5   pos_embedding                                          =
keras.layers.Embedding(input_dim=input_shape[0],
output_dim=64)(positions)
6   x = x + pos_embedding
```

Gambar 4.11 Code Implementasi Positional Embedding pada Transformer

Komponen inti dari model ini terletak pada blok *Multi-Head Attention*. Dalam implementasinya, blok ini dikonfigurasi untuk memiliki 4 "kepala" (*heads*) perhatian. Secara sederhana, ini memungkinkan *model* untuk memfokuskan perhatiannya pada empat aspek berbeda dari gerakan tangan secara bersamaan misalnya, satu kepala fokus pada posisi ibu jari, sementara kepala lain fokus pada pergerakan kelingking. Hal ini membuat *model* sangat peka terhadap detail-detail kecil dalam gestur yang kompleks. Selain itu, untuk menjaga agar proses belajar model tetap stabil dan tidak stabil (*exploding gradient*), setiap hasil pemrosesan dilapisi dengan normalisasi (*LayerNormalization*) dan koneksi sisa (*residual connection*), yang memastikan informasi asli dari input tidak hilang tertimpa oleh proses komputasi yang dalam.

```
# Blok Attention: Memungkinkan model fokus pada bagian penting dari gestur
attn_output = layers.MultiHeadAttention(
    num_heads=4, key_dim=64, dropout=0.3
)(query=x, value=x, key=x)

# Normalisasi dan Koneksi Sisa (menjaga kestabilan informasi)
x = layers.LayerNormalization(epsilon=1e-6)(x + attn_output)

# Jaringan Feed Forward untuk pemrosesan lebih lanjut
ffn_output = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64)
])(x)
x = layers.LayerNormalization(epsilon=1e-6)(x + ffn_output)
```

Gambar 4.12 Code Mekanisme Perhatian (*Attention*) dan Stabilisasi Model

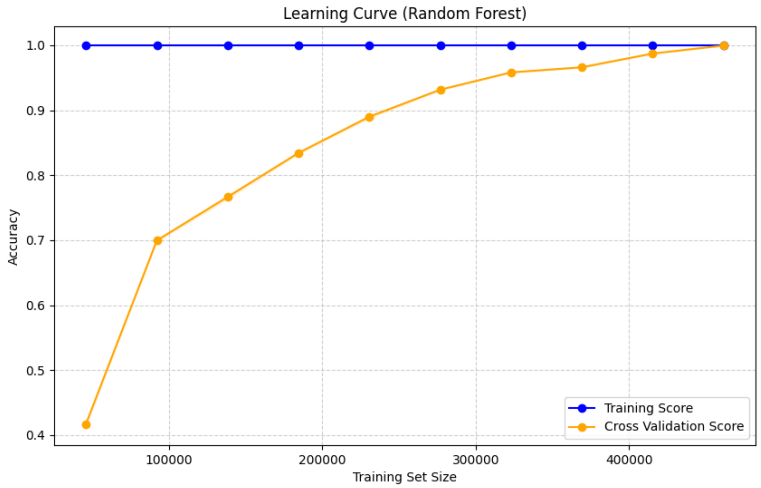
4.4. Hasil Evaluasi Model

Evaluasi model dilakukan secara *offline* menggunakan data uji *(test set*) yang telah dipisahkan sebesar 20% dari total *dataset* sebelum proses pelatihan dimulai. Pengujian ini bertujuan untuk mengukur kinerja *model* dalam mengenali gestur pada

data yang belum pernah dilihat sebelumnya, serta memastikan bahwa *model* tidak mengalami *overfitting*.

4.4.1. Evaluasi Model Gestur Statis (Random Forest)

Kinerja model *Random Forest* dievaluasi secara komprehensif menggunakan data uji (*test set*) yang mencakup gestur statis berupa huruf (A-Z), angka (0-10), dan kosakata statis lainnya. Evaluasi diawali dengan analisis kurva pembelajaran (*learning curve*) untuk memastikan model mempelajari pola data dengan benar tanpa mengalami *overfitting* atau *underfitting*.



Gambar 4.13 Kurva Pembelajaran (*Learning Curve*) Model *Random Forest*

Sebagaimana terlihat pada Gambar 4.13, kurva pembelajaran (*learning curve*) menunjukkan dampak signifikan dari penerapan augmentasi data terhadap performa model. Penting untuk dicatat bahwa kurva validasi (oranye) pada grafik ini merepresentasikan rata-rata akurasi dari proses *5-Fold Cross-Validation*, yang menguji model pada seluruh bagian dataset secara objektif. Garis skor validasi (oranye) memperlihatkan tren peningkatan yang tajam dan konsisten seiring dengan bertambahnya volume data latih yang kini mencapai lebih dari 450.000 sampel. Meskipun skor validasi berawal dari titik yang relatif rendah (sekitar 0.4) yang mengindikasikan kompleksitas variasi data akibat rotasi dan penskalaan model mampu

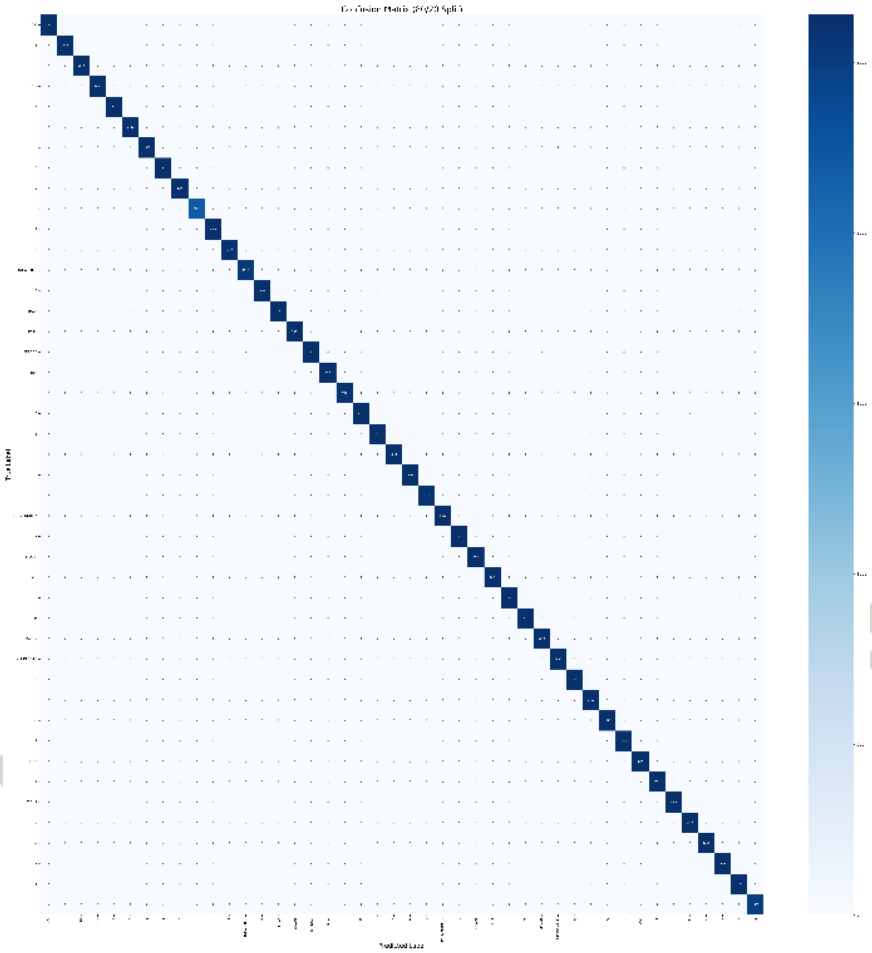
mempelajari pola tersebut secara efektif seiring penambahan data. Pada akhirnya, kurva validasi berhasil konvergen sempurna dengan skor pelatihan (garis biru) di titik akurasi 1.0 (100%). Pertemuan kedua garis pada titik maksimal ini membuktikan bahwa strategi augmentasi data berhasil mengeliminasi indikasi *overfitting* yang sebelumnya muncul, menghasilkan model yang memiliki kemampuan generalisasi sangat baik dan stabil (*robust*) terhadap variasi gestur.

Hasil evaluasi kuantitatif terhadap kinerja model *Random Forest* dirangkum secara rinci dalam Tabel 4.2. Berdasarkan hasil pengujian yang melibatkan 115.392 sampel data jumlah yang meningkat signifikan akibat proses augmentasi model *Random Forest* menunjukkan performa yang sempurna. Model berhasil mencapai tingkat akurasi global sebesar 100%. Konsistensi kinerja ini juga tercermin secara merata pada seluruh metrik evaluasi, di mana nilai rata-rata tertimbang (*weighted average*) untuk presisi (*precision*), *recall*, dan *f1-score* seluruhnya tercatat sempurna pada angka 1.00. Capaian akurasi absolut ini dapat diatribusikan pada dua faktor utama: karakteristik fitur *landmark* tangan statis yang memiliki distingsi spasial yang sangat tegas antar-kelas, serta efektivitas algoritma *Random Forest* dalam mempelajari pola data berdimensi tinggi yang telah diperkaya variabilitasnya melalui teknik augmentasi.

Tabel 4.2 Ringkasan Performa Model *Random Forest* pada Data Uji

Metrik Evaluasi	Precision	Recall	F1-Score
Macro Average	1.00	1.00	1.00
Weighted Average	1.00	1.00	1.00
Akurasi Global	1.00 (100%)		
Total Sampel Uji	115.392 Data		

Untuk memverifikasi detail prediksi per kelas, dilakukan analisis menggunakan Confusion Matrix.

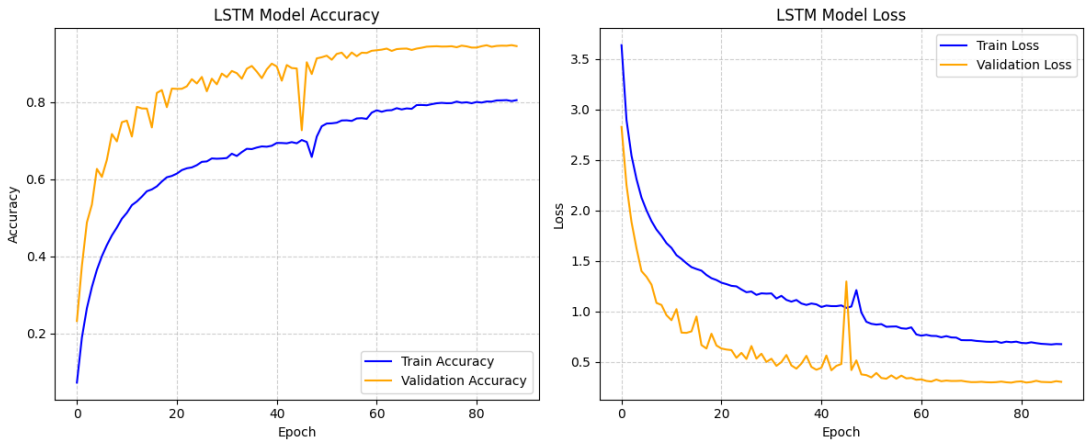


Gambar 4.14 Confusion Matrix Model Random Forest pada Data Uji

Visualisasi *Confusion Matrix* pada Gambar 4.14 memperlihatkan dominasi pola diagonal utama yang sangat tegas, merefleksikan akurasi absolut *model* terhadap 115.392 sampel uji. Seluruh prediksi terkonsentrasi sempurna pada garis diagonal (*True Positive*), sementara area *off-diagonal* tampak benar-benar bersih tanpa adanya indikasi kesalahan klasifikasi (*misclassification*). Absennya sebaran nilai di luar diagonal ini membuktikan bahwa model memiliki kemampuan diskriminatif yang sangat tinggi; ia mampu membedakan setiap gestur statis termasuk yang memiliki kemiripan konfigurasi jari dengan presisi mutlak. Hal ini juga mengonfirmasi bahwa variasi data yang dihasilkan melalui proses augmentasi berhasil dipelajari dengan baik oleh model tanpa menimbulkan ambiguitas atau kebingungan antar-kelas.

4.4.2. Evaluasi Model Gestur Dinamis: LSTM

Evaluasi terhadap model *Long Short-Term Memory* (LSTM) difokuskan pada kemampuannya mengenali pola gestur dinamis yang melibatkan urutan waktu. Analisis diawali dengan pemeriksaan kurva pelatihan untuk memastikan proses pembelajaran berjalan dengan baik.



Gambar 4.15 Kurva Akurasi dan Loss Model LSTM

Evaluasi model dilakukan dengan strategi pembagian data yang bertingkat untuk menjamin validitas hasil. Sebelum pelatihan final, stabilitas arsitektur diuji terlebih dahulu melalui metode *5-Fold Cross Validation*, di mana data validasi diambil secara bergantian dari 80% himpunan data latih (*training set*). Setelah validitas internal teruji, dilakukan pelatihan final yang hasilnya ditunjukkan pada Gambar 4.15. Perlu diperjelas bahwa kurva validasi (oranye) pada grafik ini merepresentasikan evaluasi terhadap 20% data uji (test set) yang sepenuhnya terpisah dan tidak pernah digunakan dalam proses K-Fold maupun pelatihan.

Pada grafik tersebut, terlihat pola konvergensi yang unik di mana garis akurasi validasi (oranye) cenderung bergerak lebih tinggi dibandingkan akurasi pelatihan (biru) pada fase awal hingga pertengahan. Fenomena ini terjadi akibat penerapan lapisan *Dropout* sebesar 0.3 selama proses pelatihan, yang secara acak mematikan 30% *neuron* untuk mencegah model menghafal data (*overfitting*). Namun, saat fase validasi pada

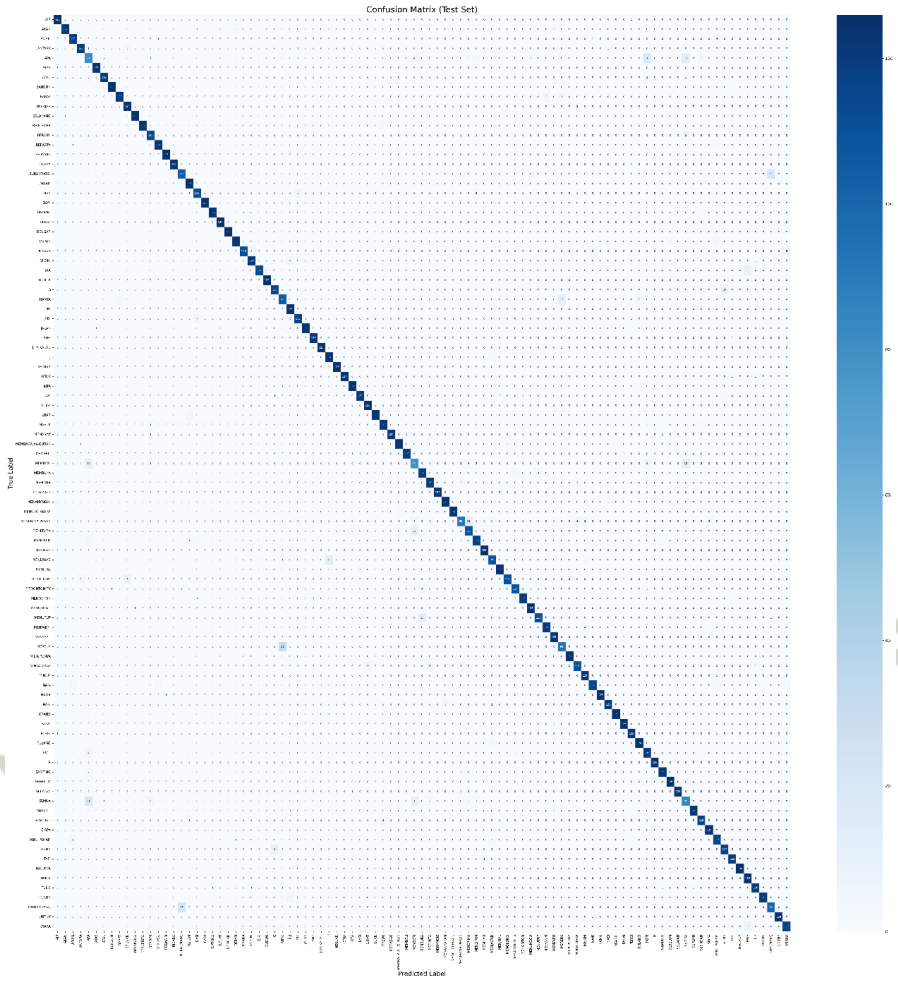
data uji, *Dropout* dinonaktifkan sehingga model dapat menggunakan kapasitas penuh jaringannya, menghasilkan performa yang lebih optimal. Secara umum, kurva *loss* menunjukkan penurunan yang stabil, menandakan model berhasil meminimalkan kesalahan seiring bertambahnya *epoch*.

Hasil evaluasi kuantitatif terhadap model LSTM pada tahap pengujian akhir dirangkum dalam Tabel 4.3. Berdasarkan data yang tersaji, model menunjukkan kinerja yang memuaskan dengan mencapai tingkat akurasi global sebesar 94,50% dari total 11.970 sampel uji. Konsistensi performa model juga tercermin dari keseimbangan nilai rata-rata tertimbang (*weighted average*) pada metrik evaluasi lainnya, di mana presisi tercatat sebesar 94,57%, *recall* sebesar 94,50%, dan *F1-score* sebesar 94,48%. Angka-angka ini mengindikasikan bahwa model LSTM cukup handal dalam mengenali pola gestur dinamis meskipun terdapat kompleksitas urutan gerakan.

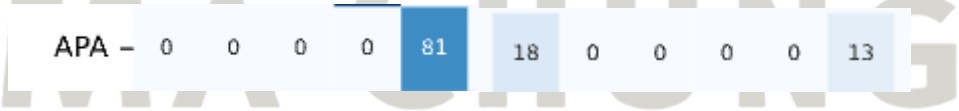
Tabel 4.3 Ringkasan Performa Model LSTM pada Data Uji

Metrik Evaluasi	**Precision**	**Recall**	**F1-Score**
Macro Average	0.95	0.95	0.94
Weighted Average	0.95	0.95	0.94
Akurasi Global	0.95 (94.50%)		
Total Sampel Uji	11.970 Data		

Meskipun akurasi keseluruhan tinggi, analisis lebih dalam menggunakan *Confusion Matrix* dan detail per kelas mengungkapkan adanya kesulitan model pada gestur-gestur tertentu yang memiliki ambiguitas gerakan.



Gambar 4.16 Confusion Matrix Model LSTM



Gambar 4.17 Misklasifikasi Gestur “APA” Confusion Matrix LSTM

Gambar 4.18 Misklasifikasi Gestur "HANYA" Confusion Matrix LSTM

Gambar 4.19 Misklasifikasi Gestur "MEMBERI" Confusion Matrix LSTM

Visualisasi *Confusion Matrix* secara keseluruhan pada Gambar 4.16 memperlihatkan distribusi prediksi model yang mengindikasikan adanya tantangan signifikan pada beberapa kelas gestur yang memiliki kemiripan visual tinggi. Untuk menganalisis kesalahan ini secara lebih mendalam, dilakukan proses perbesaran pada area yang terindikasi memiliki tingkat *error* tertinggi.

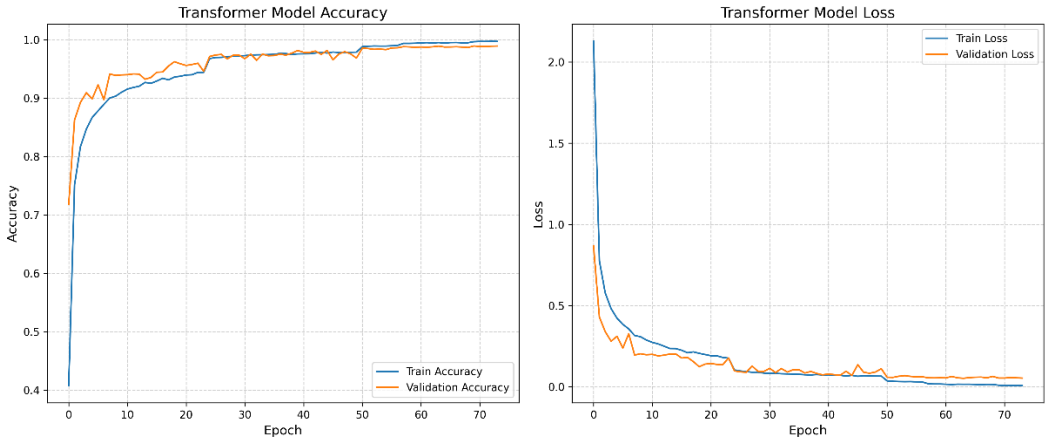
Berdasarkan Gambar 4.17, terlihat secara spesifik bahwa gestur 'APA' sering mengalami misklasifikasi sebagai gestur 'ROTI' dan 'SEMUA'. Hal ini disebabkan oleh pola lintasan gerak yang sangat mirip antar-ketiga gestur tersebut, sehingga arsitektur LSTM kesulitan membedakan nuansa transisinya. Pola kesalahan serupa juga terlihat pada Gambar 4.18, di mana gestur 'MEMBERI' sering salah dideteksi menjadi gestur 'APA', 'ROTI', dan 'SEMUA' akibat tumpang tindih fitur gerakan yang kompleks. Selain itu, Gambar 4.19 menunjukkan bahwa gestur 'HANYA' juga mengalami kebingungan prediksi, di mana model cenderung salah mengklasifikasikannya sebagai gestur 'MEREKA' atau `PULANG` dikarenakan kemiripan posisi tangan.

Sebaliknya, di luar kasus-kasus tersebut, konsentrasi warna yang pekat sempurna pada garis diagonal utama untuk gestur 'DARI', 'DENGAR', dan 'DUDUK' menunjukkan bahwa model mampu memprediksi gestur-gestur ini tanpa kesalahan. Keberhasilan ini dikarenakan ketiga gestur tersebut memiliki pola gerakan yang berbeda secara signifikan dibandingkan gestur lainnya, sehingga fitur-fiturnya dapat diekstraksi dengan baik tanpa ambiguitas.

4.4.3. Evaluasi Model Gestur Dinamis: Transformer

Berbeda dengan LSTM yang berbasis rekurensi, evaluasi pada *model Transformer* difokuskan untuk melihat efektivitas mekanisme *Self-Attention* dalam

menangkap pola global dari gestur tangan. Analisis diawali dengan pengamatan terhadap dinamika pembelajaran model melalui kurva pelatihan.



Gambar 4.20 Kurva Akurasi dan Loss Model Transformer

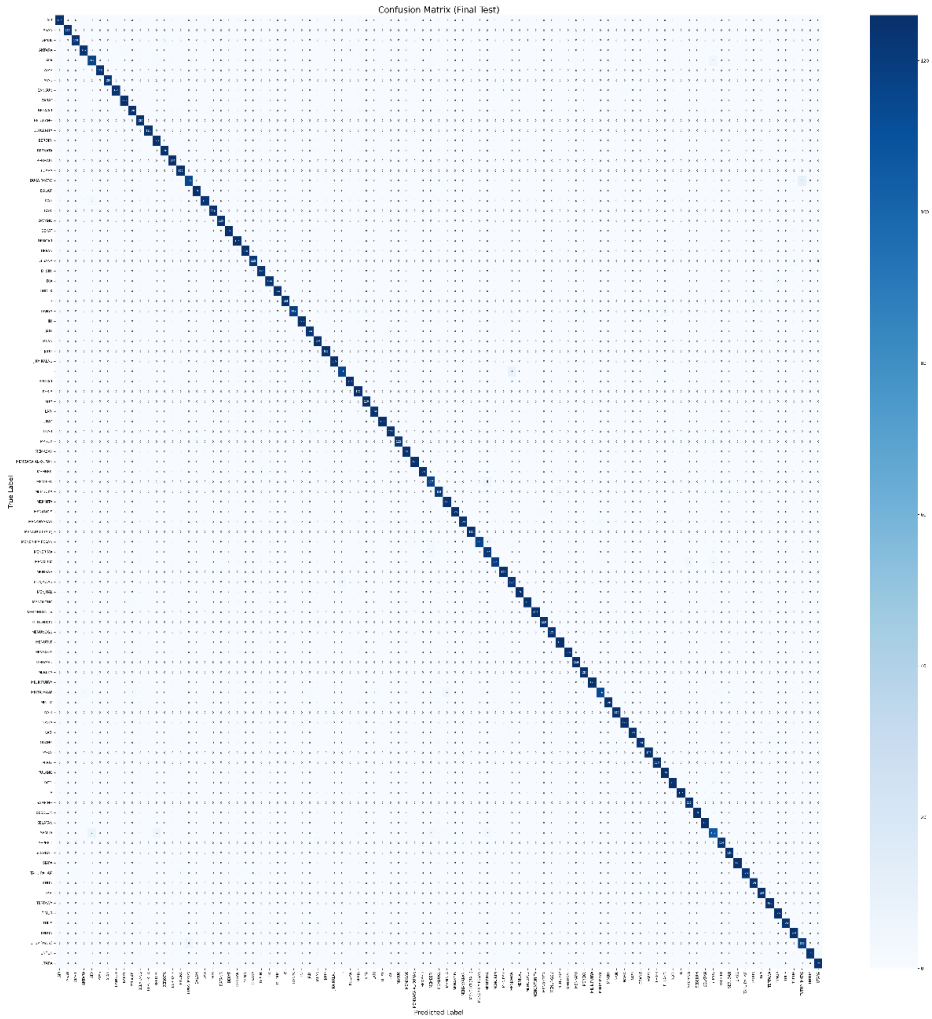
Sebagaimana diperlihatkan pada Gambar 4.20, kurva pelatihan model *Transformer* menunjukkan karakteristik konvergensi yang sangat cepat dan stabil. Pada grafik sebelah kiri (*Accuracy*), garis akurasi validasi (oranye) yang dievaluasi menggunakan 20% data uji (*test set*) yang terpisah sepenuhnya dari proses pelatihan melesat naik sejak *epoch* awal dan stabil di angka tinggi (>95%) hanya dalam kurang dari 20 *epoch*, seiring dengan garis akurasi pelatihan (biru). Hal yang sama terlihat pada grafik *Loss* (kanan), di mana *loss* validasi menurun tajam dan mendatar di angka yang sangat rendah (mendekati 0.0) tanpa adanya divergensi yang menandakan *overfitting* parah. Hal ini mengindikasikan bahwa arsitektur *Transformer* dengan konfigurasi *Dropout* 0.3 mampu mempelajari generalisasi pola gestur dengan sangat efisien. Dibandingkan dengan kurva LSTM yang membutuhkan waktu lebih lama untuk stabil, *Transformer* membuktikan efisiensi pemrosesan paralelnya dalam menangkap fitur *spasio-temporal* yang kompleks. Konsistensi hasil ini sekaligus memvalidasi tahap pengujian sebelumnya yang menggunakan metode *5-Fold Cross-Validation*, di mana validasi internal dilakukan dengan merotasi 80% data latih utama, memastikan bahwa model memiliki stabilitas yang kokoh sebelum dievaluasi pada data uji final.

Secara kuantitatif, hasil pengujian pada data uji menempatkan model *Transformer* sebagai model dengan performa paling superior, sebagaimana dirincikan pada Tabel 4.4. Berdasarkan laporan klasifikasi (*Classification Report*) yang dihasilkan pada data uji final, model ini berhasil mencatatkan tingkat akurasi global sebesar 98.57% pada 11.970 sampel uji. Kualitas prediksi juga sangat konsisten di seluruh metrik, dengan nilai rata-rata tertimbang (*weighted average*) untuk presisi, *recall*, dan *f1-score* semuanya mencapai 98.57%. Angka ini menunjukkan peningkatan performa yang signifikan dibandingkan model LSTM.

Tabel 4.4 Ringkasan Performa Model Transformer pada Data Uji

Metrik Evaluasi	**Precision**	**Recall**	**F1-Score**
Macro Average	0.99	0.99	0.99
Weighted Average	0.99	0.99	0.99
Akurasi Global	0,99 (98,57%)		
Total Sampel Uji	11.970 Data		

Untuk memverifikasi distribusi kesalahan prediksi secara mendetail, dilakukan analisis visual menggunakan *Confusion Matrix.*



Gambar 4.21 Confusion Matrix Model Transformer pada Data Uji

Visualisasi *Confusion Matrix* pada Gambar 4.21 memperlihatkan dominasi warna biru gelap yang sangat pekat pada garis diagonal utama, yang menandakan tingginya densitas prediksi yang benar untuk hampir seluruh kelas gestur. Area *off-diagonal* (kesalahan prediksi) terlihat sangat bersih, mengindikasikan minimnya kesalahan klasifikasi antar-kelas (*misclassification*).

Gambar 4.22 Misklasifikasi Gestur “APA” Confusion Matrix Transformer

Gambar 4.23Misklasifikasi Gestur “HANYA” Confusion Matrix Transformer

Gambar 4.24 Misklasifikasi Gestur “MEMBERI” Confusion Matrix Transformer

Keunggulan arsitektur *Transformer* terlihat lebih nyata ketika dilakukan analisis mendalam pada gestur-gestur yang sebelumnya bermasalah di model LSTM. Berdasarkan hasil perbesaran pada Gambar 4.22, gestur 'APA' kini menunjukkan tingkat kesalahan yang sangat minim, di mana misklasifikasi hanya terjadi dalam frekuensi rendah terhadap gestur 'SEMUA' akibat kemiripan gerakan. Hal serupa terlihat pada Gambar 4.23, di mana gestur 'MEMBERI' berhasil dikenali dengan sangat baik dengan hanya sedikit kesalahan klasifikasi terhadap gestur 'MENERIMA' yang memiliki bentuk tangan serupa namun arah berlawanan. Peningkatan stabilitas model juga terbukti pada Gambar 4.24, di mana gestur 'HANYA' yang sebelumnya sangat rentan tertukar, kini hanya memiliki tingkat kesalahan yang dapat diabaikan terhadap gestur 'MEREKA' dan 'PULANG'.

Secara komparatif, jika disandingkan dengan hasil model LSTM yang mengalami kebingungan signifikan pada ketiga gestur tersebut, hasil ini membuktikan bahwa mekanisme *self-attention* pada *Transformer* jauh lebih andal dalam membedakan fitur-fitur halus. Model mampu memisahkan gestur dengan kemiripan visual tinggi

secara presisi, menjadikannya solusi yang lebih superior untuk sistem penerjemah bahasa isyarat yang membutuhkan akurasi tinggi.

4.4.4. Analisis Komparatif Model Dinamis (LSTM vs *Transformer*)

Setelah dilakukan pengujian dan evaluasi terhadap masing-masing model secara terpisah, tahap selanjutnya adalah melakukan analisis komparatif antara algoritma LSTM dan *Transformer* untuk mengukur efektivitasnya dalam mengenali gestur dinamis BISINDO. Perbandingan ini difokuskan pada metrik akurasi global serta rata-rata performa (*precision*, *recall*, *f1-score*) pada data uji yang sama kemudian efisiensi komputasi selama proses pelatihan dan tingkat keyakinan (*confidence score*) model dalam melakukan prediksi benar. Ringkasan perbandingan kinerja kedua model disajikan pada Tabel 4.5.

Tabel 4.5 Perbandingan Performa Model LSTM dan Transformer

Model Arsitektur	Akurasi Global	Rata-rata Precision	Rata-rata Recall	Rata-rata F1-Score	Selisih (Akurasi)
LSTM	94,50%	94,57%	94,50%	94,48%	-
Transformer	98,57%	98,59%	98,57%	98,57%	+4,07%

Berdasarkan hasil *benchmark* kecepatan pelatihan pada lingkungan komputasi yang sama, ditemukan perbedaan signifikan dalam waktu eksekusi per *epoch*. Sebagaimana tercatat dalam laporan estimasi, model LSTM mencatatkan waktu pelatihan rata-rata 8,86 detik per *epoch*, sedangkan model Transformer membutuhkan waktu 18,80 detik per *epoch*. Data ini menunjukkan bahwa arsitektur Transformer memiliki beban komputasi sekitar 2,12 kali lebih tinggi dibandingkan LSTM. Peningkatan beban ini merupakan konsekuensi logis dari mekanisme *Multi-Head Attention* pada *Transformer* yang harus memproses matriks hubungan antar-seluruh *frame* secara paralel, berbeda dengan LSTM yang memproses data secara sekuensial namun dengan operasi matematis yang lebih ringkas dan efisien pada setiap langkah waktunya.

Meskipun menuntut sumber daya komputasi yang lebih besar, model *Transformer* membuktikan keunggulannya melalui superioritas nyata dalam hal ketegasan prediksi. Berdasarkan analisis probabilitas pada data uji, rata-rata skor keyakinan (*avg confidence*) pada prediksi yang benar untuk *Transformer* mencapai angka 99,62%, lebih tinggi 6,55% dibandingkan model LSTM yang mencatatkan rata-rata keyakinan sebesar 93,07%. Tingginya probabilitas ini mengindikasikan bahwa fitur *Self-Attention* pada *Transformer* sangat efektif dalam memisahkan batas keputusan (*decision boundary*) antar-kelas gestur secara tegas. Hal ini bermakna bahwa ketika *Transformer* memprediksi suatu gestur, model tersebut memiliki tingkat kepastian yang hampir absolut, meminimalkan keraguan ambiguitas yang terkadang masih terlihat pada prediksi model LSTM.

Analisis lebih mendalam dilakukan untuk melihat seberapa efektif model *Transformer* memperbaiki kesalahan yang sering dilakukan oleh model LSTM. Tabel 4.6 memperlihatkan daftar gestur yang mengalami peningkatan akurasi paling drastis.

Tabel 4.6 Perbandingan Gestur dengan Kemiripan Visual Tinggi

Label Gestur	F1-Score LSTM	F1-Score Transformer	Peningkatan
A. Gestur Ambigu			
APA	62,79%	92,49%	+29,70%
MEMBERI	66,67%	92,13%	+25,46%
HANYA	78,36%	97,14%	+18,78%
B. Gestur Invers			
BUKA (MATA)	81,57%	92,74%	+11,17%
TUTUP (MATA)	81,12%	93,33%	+12,21%
C. Gestur Variasi			
MENERIMA PESAN	79,49%	95,12%	+15,63%
MENERIMA	86,27%	93,85%	+7,58%

Peningkatan performa paling dramatis terlihat pada kategori gestur ambigu, yaitu kelompok gestur yang memiliki pola gerakan samar atau sangat mirip secara

visual dengan gestur lain sehingga sering memicu kebingungan model. Contoh signifikannya adalah gestur 'APA' dan 'MEMBERI', di mana *Transformer* mampu meningkatkan *f1-score* masing-masing sebesar 29,70% dan 25,46%. Hal ini mengindikasikan bahwa mekanisme *Self-Attention* sukses menangkap detail transisi gerakan halus yang sebelumnya gagal diproses oleh memori sekuensial LSTM.

Keunggulan *Transformer* juga teruji pada kategori gestur invers, yang melibatkan pasangan gerakan dengan arah lintasan yang saling berkebalikan. Pada model LSTM, pasangan gestur 'BUKA (MATA)' dan 'TUTUP (MATA)' sering mengalami kesalahan klasifikasi dengan skor di kisaran 81%, namun *Transformer* berhasil memperbaikinya hingga mencapai 93,33%.

Selain itu, model ini juga sangat sensitif terhadap kategori gestur variasi, yakni gestur gabungan yang terbentuk dari pengembangan atau penambahan gerakan pada gestur dasar. Kasus ini terlihat jelas pada gestur 'MENERIMA PESAN'. Jika LSTM kesulitan membedakan gestur ini dari gestur dasarnya ('MENERIMA') terlihat dari skor rendah 79,49% maka *Transformer* mampu mengidentifikasi nuansa tambahan tersebut dengan sangat baik, mencatatkan skor 95,12%. Secara keseluruhan, data ini mengonfirmasi bahwa *Transformer* memiliki kemampuan superior dalam membedakan gestur yang memiliki kemiripan struktural.

4.5. Pengujian Sistem Secara Real-Time pada Raspberry Pi

Setelah melalui tahap evaluasi model secara terpisah (*offline*), tahap selanjutnya adalah pengujian integrasi sistem secara *real-time*. Pengujian ini dilakukan dengan menjalankan aplikasi utama pada perangkat *Raspberry Pi 5* yang telah terhubung dengan *Pi Camera v1* dan layar monitor. Tujuan utama dari pengujian ini adalah untuk memvalidasi kemampuan sistem *hybrid* dalam mengenali gestur tangan pengguna secara langsung di lingkungan nyata, di mana terdapat variabel dinamis seperti kecepatan gerakan tangan dan variasi pencahayaan yang tidak ditemukan pada data latih statis.

4.5.1. Skenario Pengujian

Pengujian sistem secara *real-time* melibatkan tiga orang partisipan utama yang merupakan penutur asli BISINDO (tunarungu) dan dua penutur non-asli. Pelibatan

penutur asli bertujuan untuk memvalidasi kinerja sistem terhadap gestur yang alami, cepat, dan fluiditas tinggi. Sementara itu, partisipan non-asli disertakan untuk menguji ketahanan model dalam mengenali variasi gerakan yang mungkin kurang presisi atau memiliki tempo yang berbeda, sehingga merepresentasikan kemampuan generalisasi sistem terhadap berbagai tipe pengguna.

Pengujian performa sistem secara *real-time* dilaksanakan untuk memvalidasi kemampuan model saat dijalankan pada perangkat embedded. Skenario pengujian dilakukan di lingkungan Laboratorium HMI dengan kondisi pencahayaan terukur antara 200 hingga 300 *Lux* dan latar belakang kompleks. Kondisi lingkungan ini disetarakkan dengan kondisi pada fase pelatihan data untuk menjaga konsistensi variabel eksternal.

Perangkat akuisisi citra yang digunakan pada tahap ini adalah *PiCamera v1* yang terintegrasi dengan modul *Raspberry Pi*. Subjek ditempatkan pada jarak 50 hingga 80 cm di depan kamera, dengan resolusi input yang disesuaikan untuk pemrosesan model. Penggunaan latar belakang yang tidak polos dan pencahayaan ruang kerja yang natural dalam pengujian ini bertujuan untuk membuktikan ketahanan sistem dalam mengenali gestur pada kondisi operasional yang sesungguhnya, di mana noise visual dari lingkungan sekitar tidak dihilangkan.

Prosedur pengujian dilakukan dengan langkah-langkah sebagai berikut:

1. Lingkungan Pengujian: Pengujian dilakukan di dalam ruangan dengan pencahayaan lampu standar (sekitar 200-300 *lux*) untuk menyimulasikan kondisi penggunaan sehari-hari. Jarak antara pengguna dan kamera diatur pada kisaran 50–80 cm agar *frame* kamera dapat menangkap gestur tangan secara utuh.
2. Tugas Partisipan: Dalam setiap pengujian, partisipan diinstruksikan untuk memperagakan gestur secara tanpa adanya interupsi atau pergantian sesi. Alur tugas dimulai dengan memperagakan gestur statis (Huruf A-Z, Angka 0-10 dan Kata Statis), kemudian langsung dilanjutkan dengan memperagakan gestur dinamis untuk melihat transisi dan responsivitas sistem.

3. Metode Pengambilan Data: Untuk membandingkan performa arsitektur model, pengujian dibagi menjadi dua sesi eksperimen utama:
    - Sesi 1: *Hybrid* (*Random Forest* + LSTM *TFLite*).
    - Sesi 2: *Hybrid* (*Random Forest* + *Transformer TFLite*).

4.5.2. Hasil Pengujian Akurasi Real-Time

Pengujian sistem secara *real-time* dilakukan dengan melibatkan satu partisipan Tunarungu dan dua partisipan non-difabel untuk memvalidasi kinerja sistem dalam kondisi penggunaan yang sebenarnya. Pengujian ini bertujuan untuk mengukur tingkat akurasi sistem dalam mengenali gestur statis dan dinamis, tidak hanya pada gerakan yang natural dari penutur asli, tetapi juga pada pola gerakan yang diperagakan oleh penutur non-asli. Pengujian dibagi menjadi dua skenario utama, yaitu Skenario A yang menggunakan kombinasi *Random Forest* dan LSTM, serta Skenario B yang menggabungkan *Random Forest* dengan *Transformer*.

Pada Skenario A, sistem yang diuji dengan model *Hybrid* (*Random Forest* + LSTM) berhasil mencapai tingkat akurasi keseluruhan sebesar 88%, sebagaimana dirincikan pada Tabel 4.7. Secara spesifik, model *Random Forest* menunjukkan kinerja yang sangat andal dalam mendeteksi gestur statis (Huruf, Angka dan Kata Statis) dengan akurasi rata-rata mencapai 90%. Sementara itu, model LSTM yang bertugas menangani gestur dinamis mencatatkan akurasi sebesar 86%. Selisih akurasi ini mengindikasikan bahwa tantangan dalam pengenalan gestur dinamis relatif lebih tinggi dibandingkan gestur statis, terutama dalam hal menangkap variabilitas pola pergerakan tangan secara temporal.

Tabel 4.7 Hasil Pengujian Real-Time RF+LSTM

Rata – Rata	
Random Forest	LSTM
90%	86%

*Data lengkap berada di Lampiran A.1

Beralih ke Skenario B, di mana model LSTM digantikan oleh *Transformer*, hasil pengujian yang disajikan pada Tabel 4.8 menunjukkan adanya peningkatan performa secara umum dengan capaian akurasi total sebesar 93%. Pada sesi ini, deteksi gestur statis mencatatkan akurasi yang lebih tinggi yaitu 91%, yang kembali menegaskan konsistensi algoritma *Random Forest* dalam mengenali bentuk tangan statis. Untuk pengenalan gestur dinamis, model *Transformer* memperoleh akurasi rata-rata sebesar 93%, mencatatkan peningkatan performa yang signifikan dibandingkan model LSTM pada skenario sebelumnya yang hanya mencapai 86%.

Tabel 4.8 Hasil Pengujian Real-Time RF+Transformer

Rata – Rata	
Random Forest	Transformer
91%	93%

*Data lengkap berada di Lampiran A.2

Berdasarkan analisis komparatif antara kedua skenario, terlihat bahwa Skenario B (RF + *Transformer*) memberikan hasil keseluruhan yang lebih unggul dibandingkan Skenario A. Analisis mendalam pada tingkat gestur dinamis mengungkapkan disparitas performa yang cukup signifikan, di mana model *Transformer* mencatatkan rata-rata akurasi sebesar 93%, mengungguli model LSTM yang berada di angka 86%. Meskipun demikian, model LSTM menunjukkan dominasi performa yang spesifik pada gestur-gestur dengan pola gerakan linier atau repetitif yang tegas, seperti gestur "MINUM", "NAIK" dan "TULIS". Pada gestur-gestur ini, LSTM berhasil mempertahankan stabilitas tinggi dengan akurasi sempurna (100%), sementara *Transformer* mengalami penurunan performa.

Sebaliknya, keunggulan utama *Transformer* terletak pada kemampuannya menangani gestur yang memiliki kompleksitas tinggi atau ambiguitas bentuk tangan antar-*frame*. Hal ini terbukti dari lonjakan akurasi yang drastis pada gestur "SEMUA", "DUDUK", "UTARA", dan "MEMBUKA". Pada gestur-gestur ini, *Transformer* mampu mencapai akurasi 100%, jauh meninggalkan LSTM yang hanya berkisar di

angka 33-47%, menciptakan selisih performa hingga lebih dari 60%. Temuan ini mengonfirmasi bahwa mekanisme *Self-Attention* pada *Transformer* sangat efektif dalam membedakan konteks gerakan dengan transisi halus yang sering luput dari arsitektur rekuren LSTM. Di sisi lain, keandalan sistem dalam mengenali gestur statis terbukti sangat konsisten pada kedua skenario, di mana mayoritas gestur abjad (seperti B, C, D, hingga W) dapat dikenali dengan akurasi 100%, memvalidasi peran *Random Forest* sebagai komponen *hybrid* yang efisien untuk menangani isyarat non-temporal.

Berdasarkan hasil evaluasi komparatif, teridentifikasi empat kelas gestur yang secara konsisten memiliki performa rendah baik pada skenario model LSTM maupun *Transformer*, yaitu gestur 'Z', 'MENDENGAR', 'ANTARA', dan 'BERKATA'. Persistensi kesalahan pada kedua arsitektur model mengindikasikan bahwa tantangan utama bukan terletak pada kemampuan model mempelajari urutan, melainkan pada ambiguitas fitur intrinsik dan kualitas representasi data pada kelas-kelas tersebut. Berikut adalah analisis mendalam untuk masing-masing kasus.

Gestur 'Z' (Statis) Gestur ini mencatatkan akurasi terendah (hampir 0%) di kedua skenario. Analisis menunjukkan bahwa kesalahan fatal ini disebabkan oleh keterbatasan variasi pada dataset latih statis. Minimnya sampel yang merepresentasikan variasi sudut pandang dan orientasi jari saat membentuk huruf 'Z' menyebabkan model gagal mengenali pola tersebut saat diuji secara *real-time* dengan pose yang sedikit berbeda dari data latih (*overfitting* pada pose statis tertentu).

Gestur 'MENDENGAR' (Statis) vs. 'KANAN' Kesalahan klasifikasi pada gestur 'MENDENGAR' didominasi oleh prediksi yang tertukar dengan gestur 'KANAN'. Berdasarkan observasi karakteristik gestur, gestur 'MENDENGAR' dilakukan dengan tangan terbuka di sebelah telinga, sedangkan gestur 'KANAN' dilakukan dengan tangan menggenggam di posisi yang sama. Meskipun konfigurasi jari kedua gestur ini berbeda secara signifikan, tingginya tingkat kesalahan menunjukkan bahwa model lebih dominan mempelajari fitur spasial/posisi dibandingkan fitur bentuk tangan. Posisi tangan yang identik yakni berada di area lateral kepala dekat telinga menyebabkan koordinat *keypoints* pergelangan tangan dan telapak tangan berada pada vektor lokasi yang sangat berdekatan. Dalam kondisi

pengujian *real-time*, model tampaknya kesulitan memprioritaskan fitur "kondisi jari" di atas fitur "lokasi tangan", sehingga sering mengabaikan perbedaan bentuk genggaman dan salah mengklasifikasikan gestur hanya berdasarkan kedekatan posisinya di area telinga.

Gestur 'ANTARA' (Dinamis) vs. 'BELAJAR' Pada kategori gestur dinamis, gestur 'ANTARA' sering mengalami misklasifikasi sebagai gestur 'BELAJAR'. Analisis visual terhadap data uji menunjukkan adanya tumpang tindih pada pola lintasan gerak. Kedua gestur ini melibatkan pergerakan kedua tangan di area depan dada dengan tempo gerakan yang hampir serupa. Kemiripan pola kinematik ini membuat fitur temporal yang diekstraksi baik oleh *gate* LSTM maupun *attention Transformer* menjadi kurang terlihat perbedaannya, sehingga model kesulitan menentukan batas pembeda yang tegas di antara keduanya.

Gestur 'BERKATA' (Dinamis) vs. 'AMBIL' Serupa dengan kasus sebelumnya, gestur 'BERKATA' mengalami kebingungan prediksi yang signifikan terhadap gestur 'AMBIL'. Kesalahan ini diakibatkan oleh kemiripan transisi gerakan awal dan akhir. Kedua gestur melibatkan pergerakan tangan yang bermula dari area tubuh bagian atas menuju ke arah luar atau sebaliknya. Dalam kondisi pengambilan data *real-time* yang bervariasi, nuansa perbedaan pada orientasi telapak tangan seringkali tersamarkan, menyebabkan model menangkap fitur gerak global yang identik antara 'BERKATA' (gerakan seperti mulut keluar) dan 'AMBIL' (gerakan meraih), yang berujung pada kesalahan klasifikasi.

4.5.3. Analisis Statistik Signifikansi Performa (Uji Wilcoxon)

Meskipun data deskriptif pada subbab sebelumnya menunjukkan bahwa rata-rata akurasi model *Hybrid* RF + *Transformer* (93%) lebih tinggi dibandingkan RF + LSTM (86%), pembuktian secara statistik diperlukan untuk memastikan bahwa perbedaan tersebut bersifat signifikan dan bukan terjadi karena kebetulan semata. Oleh karena itu, dilakukan uji beda dua sampel berpasangan (*paired sample test*) terhadap hasil akurasi dari 95 gestur dinamis.

Mengingat data akurasi tidak terdistribusi secara normal, metode statistik non-parametrik *Wilcoxon Signed-Rank Test* dipilih sebagai instrumen pengujian. Pengujian

ini dilakukan dengan taraf signifikansi sebesar 0,05 atau 5%. Hipotesis yang diajukan dalam pengujian ini adalah sebagai berikut:

- Hipotesis Nol (H_0): Tidak terdapat perbedaan yang signifikan antara performa akurasi model RF + LSTM dan RF + *Transformer*.
- Hipotesis Alternatif (H_1): Terdapat perbedaan yang signifikan antara performa akurasi model RF + LSTM dan RF + *Transformer*.

Hasil perhitungan statistik menggunakan perangkat lunak SPSS disajikan pada Gambar 4.25 dan Gambar 4.26 berikut ini.

Ranks

		N	Mean Rank	Sum of Ranks
Akurasi_Transformer - Akurasi_LSTM	Negative Ranks	11[a]	22.45	247.00
	Positive Ranks	35[b]	23.83	834.00
	Ties	49[c]		
	Total	95		

a. Akurasi_Transformer < Akurasi_LSTM

b. Akurasi_Transformer > Akurasi_LSTM

c. Akurasi_Transformer = Akurasi_LSTM

Gambar 4.25 Peringkat Tanda (Ranks) Uji Wilcoxon

Test Statistics[a]

	Akurasi_Transformer - Akurasi_LSTM
Z	-3.219[b]
Asymp. Sig. (2-tailed)	.001

a. Wilcoxon Signed Ranks Test

b. Based on negative ranks.

Gambar 4.26 Hasil Statistik Uji Wilcoxon

Berdasarkan Gambar 4.26, hasil uji statistik menunjukkan nilai *Asymp. Sig. (2-tailed)* sebesar 0,001. Nilai ini jauh lebih kecil dari taraf signifikansi yang ditetapkan ($0.001 < 0.05$). Dengan demikian, H_0 ditolak dan H_1 diterima. Hal ini membuktikan secara statistik bahwa penggantian arsitektur dari LSTM ke *Transformer* memberikan dampak peningkatan akurasi yang nyata dan signifikan pada sistem klasifikasi BISINDO *real-time*.

Analisis lebih lanjut pada Gambar 4.20 (*Ranks*) memperlihatkan dominasi performa model *Transformer*. Data *Positive Ranks* (jumlah gestur di mana *Transformer* unggul atas LSTM) tercatat jauh lebih banyak dibandingkan *Negative Ranks* (jumlah gestur di mana LSTM unggul). Hal ini mengindikasikan bahwa perbaikan performa terjadi secara merata pada mayoritas gestur dinamis yang diujikan.

Secara statistik, model RF + *Transformer* terbukti lebih *robust* dan andal untuk diimplementasikan pada perangkat *Raspberry Pi*. Kesimpulannya, model Hybrid *Random Forest + Transformer* adalah arsitektur yang paling direkomendasikan untuk sistem penerjemah bahasa isyarat ini.

4.6. Evaluasi Kinerja Komputasi pada *Raspberry Pi 5*

Evaluasi kinerja komputasi dilakukan untuk memvalidasi efisiensi model *Deep Learning* (LSTM dan *Transformer*) saat dijalankan pada perangkat *edge Raspberry Pi 5*. Pengujian ini difokuskan pada pengukuran kecepatan inferensi murni (*inference benchmark*) untuk mengetahui batas maksimal kemampuan model dalam memproses data tanpa dipengaruhi oleh latensi kamera atau antarmuka grafis. Berdasarkan pengujian intensif yang dilakukan sebanyak 5 kali iterasi berturut-turut untuk setiap model, diperoleh data performa yang sangat impresif sebagaimana dirangkum dalam analisis berikut.

Sebelum membahas hasil performa waktu nyata, pengujian kecepatan inferensi dilakukan terlebih dahulu untuk mengukur efisiensi komputasi model saat diimplementasikan ke dalam format *TensorFlow Lite*. Metode pengukuran dilakukan dengan menjalankan proses prediksi model pada serangkaian sampel data gestur secara berulang pada perangkat uji, tanpa menyertakan waktu yang dibutuhkan untuk *pre-processing* data. Total waktu eksekusi yang tercatat kemudian dibagi dengan jumlah

sampel untuk mendapatkan rata-rata waktu latensi per prediksi dalam satuan milidetik (ms). Selain itu, metrik *Frames Per Second* (FPS) juga dihitung untuk mengetahui berapa banyak gestur yang dapat diproses sistem dalam satu detik, di mana nilai FPS berbanding terbalik dengan latensi.

Hasil pengujian kuantitatif menunjukkan bahwa model LSTM mencatatkan efisiensi komputasi tertinggi dengan kecepatan pemrosesan mencapai 2.123 FPS. Jika dikonversikan ke dalam satuan waktu respons, angka ini setara dengan latensi ultra-rendah sebesar 0,47 milidetik per prediksi, yang diperoleh melalui perhitungan matematis $Latensi = \frac{1000ms}{2.123\ FPS} = 0{,}47\ ms$. Sementara itu, Model Transformer, meskipun memiliki arsitektur yang lebih kompleks, tetap menunjukkan kinerja yang luar biasa dengan rata-rata kecepatan inferensi 1.757 FPS. Berdasarkan rumus konversi yang sama $Latensi = \frac{1000ms}{1.757\ FPS} = 0{,}57\ ms$, model ini menghasilkan latensi sebesar 0,57 milidetik. Selisih waktu inferensi sekitar 0,1 milidetik antara kedua model ini tergolong sangat tidak signifikan dalam konteks penggunaan nyata, mengingat standar waktu respons manusia berada di kisaran ratusan milidetik. Hal ini mengindikasikan bahwa kedua model telah teroptimasi dengan sangat baik melalui kuantisasi *TensorFlow Lite*, sehingga proses klasifikasi gestur dinamis hampir tidak memberikan beban latensi tambahan pada sistem utama..

Dari aspek penggunaan sumber daya, *Raspberry Pi 5* terbukti sangat mumpuni dalam menangani beban komputasi kedua model tersebut. Selama pengujian berlangsung, rata-rata penggunaan CPU untuk model LSTM tercatat stabil di angka 23,93%, sedangkan model *Transformer* sedikit lebih tinggi di angka 26,10%. Stabilitas termal perangkat juga terjaga dengan baik, di mana suhu rata-rata operasional terpantau berada di kisaran 48°C hingga 49°C, jauh di bawah batas *thermal throttling* (80°C). Rendahnya konsumsi sumber daya dan suhu operasional ini menjamin bahwa sistem dapat dijalankan dalam durasi panjang tanpa risiko penurunan performa (*overheating*), menjadikan solusi ini sangat layak untuk diimplementasikan sebagai perangkat penerjemah bahasa isyarat portabel yang andal.

Bab V
Simpulan dan Saran

5.1. Kesimpulan

Penelitian ini berhasil merancang dan mengimplementasikan sistem klasifikasi bahasa isyarat BISINDO hibrida secara *real-time* pada perangkat *edge Raspberry Pi 5* dengan efisiensi komputasi yang optimal. Melalui pemanfaatan format T*ensorFlow Lite*, sistem mampu memproses inferensi dengan latensi sangat rendah, berkisar antara 0,47 ms hingga 0,57 ms, sehingga memenuhi standar responsivitas yang dibutuhkan untuk komunikasi langsung. Hasil pengujian komparatif pada 95 gestur dinamis menunjukkan bahwa arsitektur *Hybrid Random Forest + Transformer* (Skenario B) memiliki performa yang lebih superior dengan akurasi rata-rata mencapai 93%, mengungguli arsitektur *Hybrid Random Forest* + LSTM (Skenario A) yang mencatatkan akurasi 86%. Keunggulan ini telah divalidasi secara statistik melalui uji *Wilcoxon Signed-Rank Test* dengan hasil yang signifikan ($p < 0.05$) membuktikan bahwa mekanisme *Self-Attention* pada *Transformer* jauh lebih efektif dalam menangani gestur kompleks dan ambigu seperti "BUKA (MATA)" dan "UTARA" dibandingkan arsitektur rekuren. Selain itu, sistem menunjukkan tingkat robustitas yang baik saat diuji oleh tiga partisipan yang terdiri dari satu penutur asli Tuli dan dua penutur dengar, serta mampu mempertahankan konsistensi akurasi sempurna (100%) pada klasifikasi gestur statis berkat keandalan komponen *Random Forest*.

Meskipun sistem menunjukkan performa komputasi dan akurasi yang optimal, evaluasi lapangan dengan partisipan tunarungu menyingkap adanya tantangan teknis pada sisi akuisisi data visual. Ditemukan adanya kesenjangan antara kecepatan alami gerak isyarat penutur asli dengan kapabilitas tangkapan kamera dan algoritma deteksi fitur. Kecepatan dan fluiditas gerakan tangan partisipan sering kali melampaui batas *frame rate* kamera serta kecepatan pemrosesan *landmark* oleh *MediaPipe* pada perangkat *edge*, yang mengakibatkan terjadinya efek motion blur dan hilangnya pelacakan titik tangan. Kondisi ini menyebabkan kegagalan segmentasi gestur yang mengharuskan pengguna untuk mengulang gerakan dari posisi awal, mengindikasikan

bahwa keandalan sistem dalam kondisi nyata tidak hanya bergantung pada model klasifikasi, tetapi juga sangat dipengaruhi oleh spesifikasi perangkat keras akuisisi citra dalam menangani dinamika gerakan cepat.

5.2. Saran

Berdasarkan hasil penelitian dan evaluasi kinerja sistem penerjemah bahasa isyarat yang telah dilakukan, terdapat sejumlah rekomendasi strategis untuk pengembangan penelitian di masa mendatang. Saran-saran ini difokuskan pada tiga aspek utama: optimalisasi implementasi perangkat keras, peningkatan metodologi pelatihan, dan perluasan fungsionalitas sistem.

Dari sisi implementasi perangkat keras, penelitian selanjutnya sangat disarankan untuk menerapkan teknik optimasi model tingkat lanjut berupa *Full Integer Quantization (Int8)* dan *Model Pruning.* Langkah ini bertujuan untuk mereduksi presisi bobot model dan memangkas redundansi arsitektur, sehingga memungkinkan sistem beroperasi secara efisien pada perangkat mikrokontroler hemat daya atau perangkat *wearable*. Sejalan dengan itu, penggunaan sensor kamera dengan *frame rate* tinggi (60 FPS atau lebih) direkomendasikan untuk meningkatkan resolusi temporal agar gerakan mikro penutur asli dapat tertangkap dengan presisi. Namun, guna menjaga efisiensi daya pada perangkat *portable*, penggunaan kamera ini sebaiknya disertai dengan penyesuaian resolusi input, mengingat ekstraksi fitur berbasis *MediaPipe* tidak menuntut resolusi citra yang tinggi.

Dalam aspek metodologi dan data, penelitian lanjutan sangat disarankan untuk memitigasi risiko bias demografis pada pembuatan *dataset*. Hal ini dapat dilakukan dengan memperluas akuisisi data yang melibatkan partisipan dari dua kelompok demografi utama, yaitu penutur asli dan non-penutur asli. Pelibatan penutur asli bertujuan untuk menangkap karakteristik gerakan yang natural, fluida, dan berkecepatan tinggi, sedangkan data dari non-penutur asli merepresentasikan variasi gerakan yang lebih terstruktur. Kombinasi kedua spektrum data ini krusial untuk melatih model agar memiliki kemampuan generalisasi yang tangguh dalam mengenali pola gestur pada berbagai tingkat kecepatan dan gaya bahasa. Selain itu, guna mendapatkan konfigurasi arsitektur model yang paling optimal tanpa bergantung pada

penyesuaian manual, disarankan untuk mengadopsi metode pencarian *hyperparameter* otomatis seperti *Bayesian Optimization.* Metode ini menawarkan eksplorasi ruang parameter yang lebih sistematis untuk menyeimbangkan akurasi dan kompleksitas komputasi.

Terakhir, pengembangan fungsionalitas sistem diharapkan dapat bergerak menuju pengenalan kalimat kontinu (*Continuous Sign Language Recognition*) yang mampu menerjemahkan percakapan utuh secara mengalir tanpa jeda antar-kata. Fungsionalitas ini idealnya diintegrasikan dengan fitur komunikasi dua arah (*bi-directional communication*) yang dilengkapi modul *Text-to-Speech* (TTS) dan *Speech-to-Text*. Dengan fitur ini, sistem tidak hanya memvisualisasikan isyarat ke dalam teks, tetapi juga dapat menyuarakan terjemahan tersebut secara verbal, serta mengonversi ucapan lawan bicara kembali menjadi teks atau visual isyarat, sehingga menciptakan ekosistem komunikasi yang inklusif dan menyeluruh bagi penyandang disabilitas rungu.

DAFTAR PUSTAKA

Abed, A.A. and Rahman, S.A., 2017. Python-based Raspberry Pi for hand gesture recognition. *International Journal of Computer Applications*, 173(4), pp.18-24.

Alatas, O.H and Widodo, R.B., 2024. Klasifikasi Bahasa Isyarat BISINDO dengan Input Kamera pada Raspberry Pi Menggunakan Algoritma Random Forest dan Long Short-Term Memory. *Laporan Praktik Kerja Lapangan. Malang: Universitas Ma Chung*.

Alexander, N., Widodo, R.B., & Swastika, W. Penggunaan Machine Learning Dalam Klasifikasi Bahasa Isyarat BISINDO Menggunakan Kamera. *Prosiding Seminar Nasional Universitas Ma Chung(SENAM)*, 2023, (pp. 11-26).

Ali, Z., 2024. A Comprehensive Overview and Comparative Analysis of CNN, RNN-LSTM and Transformer. RNN-LSTM and Transformer (December 31, 2024).

Aljabar, A., Suryani, D., and Prasetyo, E., 2020, 'BISINDO Sign Language Recognition Using CNN and LSTM', *Proceedings of the International Conference on Computer Engineering, Network and Intelligent Multimedia*, pp. 1–6.

Birajdar, G.S., Baz, M., Singh, R., Rashid, M., Gehlot, A., Akram, S.V., Alshamrani, S.S. and AlGhamdi, A.S., 2021. Realization of people density and smoke flow in buildings during fire accidents using raspberry and openCV. Sustainability, 13(19), p.11082.

Camgoz, N.C., Koller, O., Hadfield, S. and Bowden, R., 2020. Sign language transformers: Joint end-to-end sign language recognition and translation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10023-10033).

Chaudhary, L., Ananthanarayana, T., Hoq, E. and Nwogu, I., 2022. Signnet ii: A transformer-based two-way sign language translation model. IEEE

Transactions on Pattern Analysis and Machine Intelligence, 45(11), pp.12896-12907.

Chen, Z., Lei, X., Yuan, Q., Qi, Y., Ma, Z., Qian, S. and Lyu, X., 2024. Key technologies for autonomous fruit-and vegetable-picking robots: A review. Agronomy, 14(10), p.2233.

Coffen, B. and Mahmud, M.S., 2021, March. Tinydl: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor. In 2020 IEEE international conference on e-health networking, application & services (HEALTHCOM) (pp. 1-6). IEEE.

David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T. and Warden, P., 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of machine learning and systems, 3, pp.800-811.

Fadlilah, N., Suryani, D., and Prasetyo, E., 2022, 'Modelling of Basic Indonesian Sign Language Translator Based on Raspberry Pi Technology', *Journal of Physics: Conference Series*, 1(1), pp. 1–6.

Hoque, O.B., Jubair, M.I., Islam, M.S., Akash, A.F. and Paulson, A.S., 2018, December. Real time bangladeshi sign language detection using faster r-cnn. In 2018 international conference on innovation in engineering and technology (ICIET) (pp. 1-6). IEEE.

Karita, S., Chen, N., Hayashi, T., Hori, T., Inaguma, H., Jiang, Z., Someki, M., Soplin, N.E.Y., Yamamoto, R., Wang, X. and Watanabe, S., 2019, December. A comparative study on transformer vs rnn in speech applications. In 2019 IEEE automatic speech recognition and understanding workshop (ASRU) (pp. 449-456). IEEE.

Konaite, M., Owolawi, P.A., Mapayi, T., Malele, V., Odeyemi, K., Aiyetoro, G. and Ojo, J.S., 2021, December. Smart hat for the blind with real-time object

detection using raspberry pi and tensorflow lite. In Proceedings of the International Conference on Artificial Intelligence and its Applications (pp. 1-6).

Kothadiya, D., Bhatt, C., Sapariya, K., Patel, K., Gil-González, A.B. and Corchado, J.M., 2022. Deepsign: Sign language detection and recognition using deep learning. Electronics, 11(11), p.1780.

Pomaska, G., 2019. Stereo vision applying opencv and raspberry pi. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 42, pp.265-269.

Raspberry Pi Foundation (2013–2023) Camera Module Documentation. Available at: https://www.raspberrypi.com/documentation/accessories/camera.html (Accessed: 14 September 2025).

Raspberry Pi Ltd (2023) Raspberry Pi 5. Available at: https://www.raspberrypi.com/products/raspberry-pi-5/ (Accessed: 3 December 2025).

Shin, J., Musa Miah, A.S., Hasan, M.A.M., Hirooka, K., Suzuki, K., Lee, H.S. and Jang, S.W., 2023. Korean sign language recognition using transformer-based deep neural network. Applied Sciences, 13(5), p.3029.

Shiri, F.M., Perumal, T., Mustapha, N. and Mohamed, R., 2023. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU. arXiv preprint arXiv:2305.17473.

Symon, A.F., Hassan, N., Rashid, H., Ahmed, I.U. and Reza, S.T., 2017, September. Design and development of a smart baby monitoring system based on Raspberry Pi and Pi camera. In 2017 4th International Conference on Advances in Electrical Engineering (ICAEE) (pp. 117-122). IEEE.

Toyib, R., Affandi Mussa, A. P., Wijaya, A. and Sonita, A. (2025) "Indonesian Sign System Introduction Application with Tensorflow Lite and Firebase Authentication", *Jurnal Teknik Informatika dan Sistem Informasi. Jakarta, Indonesia*, 11(1), pp. 31–48.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.

Wungow, K.C., Widodo, R.B. and Subianto, M., 2022, September. Studi Klasifikasi dengan KNN dan ANN pada Sarung Tangan Penerjemah Angka dan Alfabet Bahasa Isyarat SIBI. In Prosiding Seminar Nasional Universitas Ma Chung (Informatika & Sistem Informasi; Bahasa dan Seni; Farmasi) (Vol. 2, pp. 60-74).

Xu, M., Yoon, S., Fuentes, A. and Park, D.S., 2023. A comprehensive survey of image augmentation techniques for deep learning. Pattern Recognition, 137, p.109347.

Zhu, J., 2023. Comparative study of sequence-to-sequence models: From RNNs to transformers. Appl Comput Eng, 42(67), pp.2755-2721.

Lampiran

Lampiran A. Hasil Evaluasi *Real-Time*

Lampiran A.1 Hasil Evaluasi *Real-Time* RF+LSTM

GESTURE	SUBJECT	AVERAGE (%)	ACCURACY (%)
0	1	100%	100%
	2	100%	
	3	100%	
1	1	100%	100%
	2	100%	
	3	100%	
2	1	100%	100%
	2	100%	
	3	100%	
3	1	100%	100%
	2	100%	
	3	100%	
4	1	100%	100%
	2	100%	
	3	100%	
5	1	60%	87%
	2	100%	
	3	100%	
6	1	100%	100%
	2	100%	
	3	100%	
7	1	100%	100%
	2	100%	
	3	100%	
8	1	40%	47%
	2	0%	
	3	100%	
9	1	0%	67%
	2	100%	
	3	100%	

10	1	100%	100%
	2	100%	
	3	100%	
A	1	40%	80%
	2	100%	
	3	100%	
B	1	100%	100%
	2	100%	
	3	100%	
C	1	100%	100%
	2	100%	
	3	100%	
D	1	100%	100%
	2	100%	
	3	100%	
E	1	100%	100%
	2	100%	
	3	100%	
F	1	100%	100%
	2	100%	
	3	100%	
G	1	100%	100%
	2	100%	
	3	100%	
H	1	100%	100%
	2	100%	
	3	100%	
I	1	100%	100%
	2	100%	
	3	100%	
J	1	100%	87%
	2	60%	
	3	100%	
K	1	100%	100%
	2	100%	
	3	100%	
L	1	100%	33%
	2	0%	

	3	0%	
M	1	100%	100%
	2	100%	
	3	100%	
N	1	100%	100%
	2	100%	
	3	100%	
O	1	100%	100%
	2	100%	
	3	100%	
P	1	100%	73%
	2	100%	
	3	20%	
Q	1	100%	100%
	2	100%	
	3	100%	
R	1	100%	100%
	2	100%	
	3	100%	
S	1	100%	100%
	2	100%	
	3	100%	
T	1	100%	100%
	2	100%	
	3	100%	
U	1	100%	100%
	2	100%	
	3	100%	
V	1	100%	100%
	2	100%	
	3	100%	
W	1	100%	100%
	2	100%	
	3	100%	
X	1	100%	100%
	2	100%	
	3	100%	
Y	1	100%	100%

	2	100%	
	3	100%	
Z	1	20%	7%
	2	0%	
	3	0%	
AIR	1	100%	100%
	2	100%	
	3	100%	
ANDA KAMU	1	100%	100%
	2	100%	
	3	100%	
ATAS	1	100%	100%
	2	100%	
	3	100%	
ATAU	1	100%	100%
	2	100%	
	3	100%	
BAWAH	1	80%	93%
	2	100%	
	3	100%	
BELAJAR	1	100%	100%
	2	100%	
	3	100%	
BISA	1	100%	100%
	2	100%	
	3	100%	
DARI	1	100%	100%
	2	100%	
	3	100%	
DIA	1	100%	33%
	2	0%	
	3	0%	
HANYA	1	80%	93%
	2	100%	
	3	100%	
INI	1	100%	100%
	2	100%	
	3	100%	

ITU MENUNJUK	1	100%	100%
	2	100%	
	3	100%	
JADI	1	100%	100%
	2	100%	
	3	100%	
JIKA KALAU	1	100%	100%
	2	100%	
	3	100%	
KERJA	1	100%	100%
	2	100%	
	3	100%	
LUAR	1	100%	100%
	2	100%	
	3	100%	
MAKAN	1	100%	100%
	2	100%	
	3	100%	
MINUM	1	100%	100%
	2	100%	
	3	100%	
NAIK	1	100%	100%
	2	100%	
	3	100%	
PERGI	1	100%	100%
	2	100%	
	3	100%	
ROTI	1	100%	100%
	2	100%	
	3	100%	
SAYA	1	100%	100%
	2	100%	
	3	100%	
SEPERTI	1	100%	100%
	2	100%	
	3	100%	
SIAPA	1	100%	100%
	2	100%	

	3	100%	
TAHUN	1	100%	100%
	2	100%	
	3	100%	
TAPI	1	100%	100%
	2	100%	
	3	100%	
UNTUK	1	100%	100%
	2	100%	
	3	100%	
AKAN	1	100%	100%
	2	100%	
	3	100%	
AMBIL	1	0%	40%
	2	60%	
	3	60%	
APA	1	60%	80%
	2	80%	
	3	100%	
BERKATA	1	100%	53%
	2	60%	
	3	0%	
DALAM	1	100%	100%
	2	100%	
	3	100%	
DAN	1	40%	80%
	2	100%	
	3	100%	
KITA	1	100%	100%
	2	100%	
	3	100%	
MEREKA	1	100%	87%
	2	60%	
	3	100%	
MILIK PUNYA	1	100%	100%
	2	100%	
	3	100%	
ORANG	1	100%	100%

	2	100%	
	3	100%	
SEMUA	1	20%	33%
	2	60%	
	3	20%	
TAHU PAHAM	1	100%	100%
	2	100%	
	3	100%	
TURUN	1	60%	73%
	2	100%	
	3	60%	
PULANG	1	100%	100%
	2	100%	
	3	100%	
NASI	1	100%	100%
	2	100%	
	3	100%	
TIDUR	1	100%	53%
	2	60%	
	3	0%	
BANGUN	1	100%	100%
	2	100%	
	3	100%	
LIHAT	1	100%	100%
	2	100%	
	3	100%	
DENGAR	1	60%	87%
	2	100%	
	3	100%	
BICARA	1	100%	100%
	2	100%	
	3	100%	
TULIS	1	100%	100%
	2	100%	
	3	100%	
BACA	1	100%	100%
	2	100%	
	3	100%	

JALAN	1	100%	87%
	2	100%	
	3	60%	
LARI	1	100%	100%
	2	100%	
	3	100%	
MEMBELI	1	100%	100%
	2	100%	
	3	100%	
MENJUAL	1	60%	87%
	2	100%	
	3	100%	
MEMBUKA	1	80%	33%
	2	0%	
	3	20%	
MENUTUP	1	100%	100%
	2	100%	
	3	100%	
MEMBERI	1	100%	47%
	2	40%	
	3	0%	
MENERIMA	1	60%	87%
	2	100%	
	3	100%	
MENOLONG	1	80%	67%
	2	100%	
	3	20%	
MENUNGGU	1	80%	93%
	2	100%	
	3	100%	
MEMINTA	1	80%	80%
	2	100%	
	3	60%	
MENONTON	1	60%	87%
	2	100%	
	3	100%	
DUDUK	1	40%	47%
	2	100%	

	3	0%	
BERDIRI	1	80%	87%
	2	100%	
	3	80%	
DATANG	1	100%	93%
	2	100%	
	3	80%	
MASUK	1	100%	93%
	2	100%	
	3	80%	
KELUAR	1	100%	93%
	2	100%	
	3	80%	
PAKAI	1	100%	87%
	2	100%	
	3	60%	
BUKA (MATA)	1	0%	13%
	2	40%	
	3	0%	
TUTUP (MATA)	1	40%	80%
	2	100%	
	3	100%	
MENANGIS	1	100%	80%
	2	100%	
	3	40%	
TERTAWA	1	100%	67%
	2	100%	
	3	0%	
MENJAWAB	1	60%	87%
	2	100%	
	3	100%	
MENANYAKAN	1	100%	100%
	2	100%	
	3	100%	
MENDENGAR	1	60%	20%
	2	0%	
	3	0%	
	1	100%	100%

MEMBACA AL-QURAN	2	100%	
	3	100%	
MENONTON TV	1	100%	100%
	2	100%	
	3	100%	
MENYAPU	1	60%	87%
	2	100%	
	3	100%	
MENCUCI (BAJU)	1	100%	100%
	2	100%	
	3	100%	
MEMASAK	1	100%	100%
	2	100%	
	3	100%	
MENGIRIM	1	60%	67%
	2	80%	
	3	60%	
NAMA	1	100%	100%
	2	100%	
	3	100%	
MENERIMA PESAN	1	40%	60%
	2	100%	
	3	40%	
MENIKAH	1	100%	60%
	2	0%	
	3	80%	
BERCERITA	1	100%	100%
	2	100%	
	3	100%	
BERDOA	1	100%	100%
	2	100%	
	3	100%	
MINTA MAAF	1	100%	60%
	2	60%	
	3	20%	
MENYANYI	1	60%	87%
	2	100%	
	3	100%	

BERMAIN	1	100%	100%
	2	100%	
	3	100%	
DEPAN	1	100%	100%
	2	100%	
	3	100%	
BELAKANG	1	100%	80%
	2	40%	
	3	100%	
SAMPING	1	100%	100%
	2	100%	
	3	100%	
ANTARA	1	0%	40%
	2	20%	
	3	100%	
DEKAT	1	100%	100%
	2	100%	
	3	100%	
JAUH	1	100%	100%
	2	100%	
	3	100%	
DI SINI	1	40%	80%
	2	100%	
	3	100%	
DI SANA	1	100%	87%
	2	100%	
	3	60%	
SEBELUM	1	100%	100%
	2	100%	
	3	100%	
SESUDAH	1	100%	100%
	2	100%	
	3	100%	
TIMUR	1	40%	60%
	2	100%	
	3	40%	
BARAT	1	100%	93%
	2	100%	

	3	80%	
SELATAN	1	80%	47%
	2	60%	
	3	0%	
UTARA	1	80%	33%
	2	20%	
	3	0%	
KANAN	1	100%	100%
	2	100%	
	3	100%	
KIRI	1	100%	100%
	2	100%	
	3	100%	
		AVERAGE	**88%**
		AVERAGE RF	**90%**
		AVERAGE LSTM	**86%**

*1= Ibu Sumiati, 2=Olfat, 3=Shelly

Lampiran A.2 Hasil Evaluasi Real-Time RF+Transformer

GESTURE	SUBJECT	AVERAGE (%)	ACCURACY (%)
0	1	100%	100%
	2	100%	
	3	100%	
1	1	100%	100%
	2	100%	
	3	100%	
2	1	100%	100%
	2	100%	
	3	100%	
3	1	100%	100%
	2	100%	
	3	100%	
4	1	100%	100%
	2	100%	
	3	100%	

5	1	100%	100%
	2	100%	
	3	100%	
6	1	100%	100%
	2	100%	
	3	100%	
7	1	100%	100%
	2	100%	
	3	100%	
8	1	100%	67%
	2	0%	
	3	100%	
9	1	100%	100%
	2	100%	
	3	100%	
10	1	100%	100%
	2	100%	
	3	100%	
A	1	40%	80%
	2	100%	
	3	100%	
B	1	100%	100%
	2	100%	
	3	100%	
C	1	100%	100%
	2	100%	
	3	100%	
D	1	100%	100%
	2	100%	
	3	100%	
E	1	100%	100%
	2	100%	
	3	100%	
F	1	100%	100%
	2	100%	
	3	100%	
G	1	100%	100%
	2	100%	

	3	100%	
H	1	100%	100%
	2	100%	
	3	100%	
I	1	100%	100%
	2	100%	
	3	100%	
J	1	100%	100%
	2	100%	
	3	100%	
K	1	100%	100%
	2	100%	
	3	100%	
L	1	100%	100%
	2	100%	
	3	100%	
M	1	100%	100%
	2	100%	
	3	100%	
N	1	100%	100%
	2	100%	
	3	100%	
O	1	100%	100%
	2	100%	
	3	100%	
P	1	100%	67%
	2	100%	
	3	0%	
Q	1	100%	100%
	2	100%	
	3	100%	
R	1	100%	100%
	2	100%	
	3	100%	
S	1	100%	100%
	2	100%	
	3	100%	
T	1	100%	67%

	2	100%	
	3	0%	
U	1	100%	100%
	2	100%	
	3	100%	
V	1	100%	100%
	2	100%	
	3	100%	
W	1	100%	100%
	2	100%	
	3	100%	
X	1	100%	33%
	2	0%	
	3	0%	
Y	1	100%	100%
	2	100%	
	3	100%	
Z	1	0%	0%
	2	0%	
	3	0%	
AIR	1	100%	100%
	2	100%	
	3	100%	
ANDA KAMU	1	100%	100%
	2	100%	
	3	100%	
ATAS	1	100%	100%
	2	100%	
	3	100%	
ATAU	1	100%	100%
	2	100%	
	3	100%	
BAWAH	1	100%	100%
	2	100%	
	3	100%	
BELAJAR	1	100%	100%
	2	100%	
	3	100%	

BISA	1	100%	100%
	2	100%	
	3	100%	
DARI	1	100%	100%
	2	100%	
	3	100%	
DIA	1	0%	67%
	2	100%	
	3	100%	
HANYA	1	60%	87%
	2	100%	
	3	100%	
INI	1	100%	100%
	2	100%	
	3	100%	
ITU MENUNJUK	1	100%	100%
	2	100%	
	3	100%	
JADI	1	100%	100%
	2	100%	
	3	100%	
JIKA KALAU	1	80%	93%
	2	100%	
	3	100%	
KERJA	1	100%	100%
	2	100%	
	3	100%	
LUAR	1	100%	100%
	2	100%	
	3	100%	
MAKAN	1	100%	100%
	2	100%	
	3	100%	
MINUM	1	0%	67%
	2	100%	
	3	100%	
NAIK	1	0%	67%
	2	100%	

	3	100%	
PERGI	1	100%	100%
	2	100%	
	3	100%	
ROTI	1	100%	100%
	2	100%	
	3	100%	
SAYA	1	100%	100%
	2	100%	
	3	100%	
SEPERTI	1	100%	100%
	2	100%	
	3	100%	
SIAPA	1	100%	100%
	2	100%	
	3	100%	
TAHUN	1	100%	100%
	2	100%	
	3	100%	
TAPI	1	100%	100%
	2	100%	
	3	100%	
UNTUK	1	100%	100%
	2	100%	
	3	100%	
AKAN	1	100%	100%
	2	100%	
	3	100%	
AMBIL	1	60%	87%
	2	100%	
	3	100%	
APA	1	80%	93%
	2	100%	
	3	100%	
BERKATA	1	100%	53%
	2	60%	
	3	0%	
DALAM	1	100%	100%

	2	100%	
	3	100%	
DAN	1	100%	100%
	2	100%	
	3	100%	
KITA	1	100%	100%
	2	100%	
	3	100%	
MEREKA	1	100%	100%
	2	100%	
	3	100%	
MILIK PUNYA	1	100%	100%
	2	100%	
	3	100%	
ORANG	1	100%	100%
	2	100%	
	3	100%	
SEMUA	1	100%	100%
	2	100%	
	3	100%	
TAHU PAHAM	1	100%	100%
	2	100%	
	3	100%	
TURUN	1	0%	67%
	2	100%	
	3	100%	
PULANG	1	100%	100%
	2	100%	
	3	100%	
NASI	1	100%	100%
	2	100%	
	3	100%	
TIDUR	1	100%	80%
	2	100%	
	3	40%	
BANGUN	1	100%	100%
	2	100%	
	3	100%	

LIHAT	1	100%	100%
	2	100%	
	3	100%	
DENGAR	1	100%	100%
	2	100%	
	3	100%	
BICARA	1	100%	100%
	2	100%	
	3	100%	
TULIS	1	0%	33%
	2	0%	
	3	100%	
BACA	1	100%	100%
	2	100%	
	3	100%	
JALAN	1	100%	100%
	2	100%	
	3	100%	
LARI	1	100%	100%
	2	100%	
	3	100%	
MEMBELI	1	100%	100%
	2	100%	
	3	100%	
MENJUAL	1	100%	100%
	2	100%	
	3	100%	
MEMBUKA	1	100%	100%
	2	100%	
	3	100%	
MENUTUP	1	100%	80%
	2	100%	
	3	40%	
MEMBERI	1	100%	100%
	2	100%	
	3	100%	
MENERIMA	1	100%	100%
	2	100%	

	3	100%	
MENOLONG	1	100%	100%
	2	100%	
	3	100%	
MENUNGGU	1	100%	100%
	2	100%	
	3	100%	
MEMINTA	1	40%	40%
	2	60%	
	3	20%	
MENONTON	1	60%	87%
	2	100%	
	3	100%	
DUDUK	1	100%	100%
	2	100%	
	3	100%	
BERDIRI	1	100%	93%
	2	100%	
	3	80%	
DATANG	1	0%	67%
	2	100%	
	3	100%	
MASUK	1	40%	80%
	2	100%	
	3	100%	
KELUAR	1	100%	100%
	2	100%	
	3	100%	
PAKAI	1	100%	100%
	2	100%	
	3	100%	
BUKA (MATA)	1	60%	87%
	2	100%	
	3	100%	
TUTUP (MATA)	1	40%	80%
	2	100%	
	3	100%	
MENANGIS	1	80%	80%

	2	100%	
	3	60%	
TERTAWA	1	60%	87%
	2	100%	
	3	100%	
MENJAWAB	1	0%	67%
	2	100%	
	3	100%	
MENANYAKAN	1	100%	100%
	2	100%	
	3	100%	
MENDENGAR	1	0%	7%
	2	20%	
	3	0%	
MEMBACA AL-QURAN	1	100%	100%
	2	100%	
	3	100%	
MENONTON TV	1	100%	100%
	2	100%	
	3	100%	
MENYAPU	1	100%	100%
	2	100%	
	3	100%	
MENCUCI (BAJU)	1	100%	100%
	2	100%	
	3	100%	
MEMASAK	1	100%	100%
	2	100%	
	3	100%	
MENGIRIM	1	100%	100%
	2	100%	
	3	100%	
NAMA	1	100%	100%
	2	100%	
	3	100%	
MENERIMA PESAN	1	80%	93%
	2	100%	
	3	100%	

MENIKAH	1	80%	93%
	2	100%	
	3	100%	
BERCERITA	1	100%	100%
	2	100%	
	3	100%	
BERDOA	1	100%	100%
	2	100%	
	3	100%	
MINTA MAAF	1	60%	87%
	2	100%	
	3	100%	
MENYANYI	1	100%	100%
	2	100%	
	3	100%	
BERMAIN	1	100%	100%
	2	100%	
	3	100%	
DEPAN	1	100%	100%
	2	100%	
	3	100%	
BELAKANG	1	100%	100%
	2	100%	
	3	100%	
SAMPING	1	100%	100%
	2	100%	
	3	100%	
ANTARA	1	20%	53%
	2	100%	
	3	40%	
DEKAT	1	100%	100%
	2	100%	
	3	100%	
JAUH	1	100%	100%
	2	100%	
	3	100%	
DI SINI	1	100%	100%
	2	100%	

	3	100%	
DI SANA	1	100%	100%
	2	100%	
	3	100%	
SEBELUM	1	100%	100%
	2	100%	
	3	100%	
SESUDAH	1	100%	100%
	2	100%	
	3	100%	
TIMUR	1	100%	80%
	2	100%	
	3	40%	
BARAT	1	100%	100%
	2	100%	
	3	100%	
SELATAN	1	100%	80%
	2	100%	
	3	40%	
UTARA	1	100%	100%
	2	100%	
	3	100%	
KANAN	1	100%	100%
	2	100%	
	3	100%	
KIRI	1	100%	100%
	2	100%	
	3	100%	
	AVERAGE		**93%**
	AVERAGE RF		**91%**
	AVERAGE Transformer		**93%**

*1= Ibu Sumiati, 2=Olfat, 3=Shelly

Lampiran B. ***Source Code***

Lampiran B.1 ***Source Code*** **Koleksi** ***Dataset Random Forest***

```
1    import cv2
```

```
2    import os
3    import time
4
5    def capture_dataset_auto(save_path, person_name,
max_images=200, delay_ms=100, start_delay=5000):
6        """
7        Capture dataset gestur BISINDO otomatis dari
webcam
8
9        Parameters:
10       -----------
11       save_path : str
12           Path folder tempat menyimpan dataset
(contoh: "E:/Dataset.../Nic/SIAPA")
13       person_name : str
14           Nama orang/responden (contoh: "NICO")
15       max_images : int
16           Target jumlah gambar
17       delay_ms : int
18           Jeda antar capture gambar (dalam milidetik)
19       start_delay : int
20           Jeda awal sebelum mulai capture (dalam
milidetik)
21       """
22
23       # Cek apakah folder sudah ada
24       if not os.path.exists(save_path):
25           os.makedirs(save_path)
26           print(f"📂 Folder baru dibuat: {save_path}")
27
28       # Inisialisasi webcam
29       cap = cv2.VideoCapture(0)
30       if not cap.isOpened():
31           print("❌ Tidak bisa membuka webcam")
32           return
33
34       print(f"▶ Persiapkan gestur di folder:
{save_path}")
35       print(f"Capture akan dimulai dalam
{start_delay/1000:.1f} detik...")
36
37       # Tampilkan countdown awal
38       start_time = time.time()
39       while (time.time() - start_time) * 1000 <
start_delay:
```

```
40            ret, frame = cap.read()
41            if not ret:
42                print("❌ Gagal membaca frame dari
webcam")
43                cap.release()
44                return
45            frame_resized = cv2.resize(frame, (640,
480))
46            remaining = int(start_delay/1000 -
(time.time() - start_time))
47            cv2.putText(frame_resized, f"Mulai dalam
{remaining} detik...",
48                        (150, 240),
cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,0,255), 3)
49            cv2.imshow("Capture Dataset", frame_resized)
50            cv2.waitKey(1)
51
52        # Cari nomor terakhir agar tidak overwrite file
lama
53        existing_files = [f for f in
os.listdir(save_path) if f.startswith(person_name)]
54        count_start = len(existing_files)
55
56        count = 0
57        print(f"▶ Mulai auto-capture: {max_images}
gambar | Delay: {delay_ms} ms per gambar")
58        print("Tekan 'q' jika ingin berhenti lebih
awal")
59
60        while count < max_images:
61            ret, frame = cap.read()
62            if not ret:
63                print("❌ Gagal membaca frame dari
webcam")
64                break
65
66            frame_resized = cv2.resize(frame, (640,
480))
67            cv2.putText(frame_resized, f"{person_name} |
{count+1}/{max_images}",
68                        (20, 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
69            cv2.imshow("Capture Dataset", frame_resized)
70
71            # Simpan gambar dengan nama berurutan
```

```
            filename = f"{person_name}_({count_start + count + 1}).jpg"
            cv2.imwrite(os.path.join(save_path, filename), frame)
            print(f"✅ Disimpan: {filename}")
            count += 1

            # Delay dalam ms
            key = cv2.waitKey(delay_ms) & 0xFF
            if key == ord("q"):
                break

        cap.release()
        cv2.destroyAllWindows()
        print(f"📂 Selesai: {count} gambar disimpan di {save_path}")


    # --------------------
    # 📌 Contoh penggunaan
    # --------------------

    # Path folder Anda (pastikan ganti sesuai path Anda)
    save_path = r"E:\Dataset Penelitian Bahasa Isyarat Olfat 2025\Tugas Akhir\Dataset RF\New Dataset RF\MIGOZ\TIDUR"

    # Capture 100 gambar, delay 500 ms, countdown awal 3000 ms
    capture_dataset_auto(save_path=save_path, person_name="MIGOZ", max_images=200, delay_ms=100, start_delay=5000)
```

Lampiran B.2 Source Code Koleksi Dataset LSTM dan Transformer

```
    import cv2
    import mediapipe as mp
    import numpy as np
    import os

    # ---------------------------
    # Konfigurasi
    # ---------------------------
    GESTURE_NAME = "UTARA"
```

```
10   SAVE_DIR = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Dataset LSTM\New Dataset LSTM
REV\MIGOZ"
11   SEQUENCE_LENGTH = 20    # jumlah frame per sequence
12   TOTAL_SEQUENCES = 105   # target dataset
13
14   # pastikan folder ada
15   os.makedirs(SAVE_DIR, exist_ok=True)
16
17   # ---------------------------
18   # Init MediaPipe
19   # ---------------------------
20   mp_hands = mp.solutions.hands
21   hands = mp_hands.Hands(
22       static_image_mode=False,
23       max_num_hands=2,
24       min_detection_confidence=0.6,
25       min_tracking_confidence=0.6
26   )
27   mp_draw = mp.solutions.drawing_utils
28
29   # ---------------------------
30   # Kamera
31   # ---------------------------
32   cap = cv2.VideoCapture(0)
33   if not cap.isOpened():
34       raise RuntimeError("Kamera tidak bisa dibuka")
35
36   print(f"[INFO] Mulai otomatis rekam
{TOTAL_SEQUENCES} sequence untuk gestur: {GESTURE_NAME}")
37   print("[INFO] Jumlah tangan akan divalidasi otomatis
berdasarkan deteksi MediaPipe")
38   print("Tekan 'q' untuk berhenti.")
39
40   all_sequences = []
41   seq_count = 0
42   frame_buffer = []
43   last_status = None  # simpan status valid/invalid
terakhir
44
45   def validate_sequence(seq, threshold=0.9):
46       """Validasi otomatis: pastikan sequence punya
cukup frame valid"""
47       seq = np.array(seq).reshape(SEQUENCE_LENGTH, 2,
21, 3)
```

```
48        valid_frames = 0
49        hand_counts = []
50
51        for frame in seq:
52            hands_detected = 0
53            for hand in frame:
54                if np.any(hand != 0):
55                    hands_detected += 1
56            hand_counts.append(hands_detected)
57            if hands_detected > 0:
58                valid_frames += 1
59
60        frame_valid_ratio = valid_frames /
SEQUENCE_LENGTH
61        avg_hands = round(np.mean(hand_counts))
62
63        # valid jika deteksi tangan stabil (>=
threshold)
64        return frame_valid_ratio >= threshold,
frame_valid_ratio, avg_hands
65
66    while seq_count < TOTAL_SEQUENCES:
67        ret, frame = cap.read()
68        if not ret:
69            continue
70
71        frame = cv2.flip(frame, 1)
72        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
73        results = hands.process(rgb)
74
75        landmarks_both = np.zeros((2, 21, 3))
76
77        if results.multi_hand_landmarks:
78            for h_idx, hand_landmarks in
enumerate(results.multi_hand_landmarks):
79                if h_idx > 1:
80                    break
81                for i, lm in
enumerate(hand_landmarks.landmark):
82                    landmarks_both[h_idx, i] = [lm.x,
lm.y, lm.z]
83                mp_draw.draw_landmarks(frame,
hand_landmarks, mp_hands.HAND_CONNECTIONS)
84
85        frame_buffer.append(landmarks_both.flatten())
```

```
86
87      # kalau buffer sudah SEQUENCE_LENGTH frame → cek
dan simpan
88      if len(frame_buffer) == SEQUENCE_LENGTH:
89          is_valid, ratio, avg_hands =
validate_sequence(frame_buffer)
90
91          if is_valid:
92
all_sequences.append(np.array(frame_buffer))
93              seq_count += 1
94              last_status = (f"VALID ({avg_hands}
hand)", (0, 255, 0))
95              print(f"[SAVED] Sequence
{seq_count}/{TOTAL_SEQUENCES} | Ratio={ratio:.2f},
Hands={avg_hands}")
96          else:
97              last_status = ("RETAKE", (0, 0, 255))
98              print(f"[RETAKE] Sequence tidak valid |
Ratio={ratio:.2f}")
99
100         frame_buffer = []  # reset buffer
101
102     # tampilkan info status
103     if last_status:
104         msg, color = last_status
105         cv2.putText(frame, f"{msg}", (10,70),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, color, 3)
106
107     cv2.putText(frame, f"{GESTURE_NAME} Seq
{seq_count}/{TOTAL_SEQUENCES}",
108                 (10,30), cv2.FONT_HERSHEY_SIMPLEX,
0.8, (0,0,255), 2)
109     cv2.imshow("Collect Dataset", frame)
110
111     if cv2.waitKey(1) & 0xFF == ord('q'):
112         break
113
114 cap.release()
115 cv2.destroyAllWindows()
116 hands.close()
117
118 # ---------------------------
119 # Simpan semua ke 1 file .npz
120 # ---------------------------
```

```
121  X = np.array(all_sequences)  # (N, SEQUENCE_LENGTH,
126)
122  y = np.array([0]*len(all_sequences))  # label gestur
ini = 0
123
124  save_path = os.path.join(SAVE_DIR,
f"{GESTURE_NAME}_dataset.npz")
125  np.savez_compressed(save_path, X=X, y=y,
gesture=GESTURE_NAME)
126
127  print("\n[INFO] Dataset selesai!")
128  print(f"Total sequence: {X.shape[0]} | Shape data:
{X.shape}")
129  print("Disimpan ke:", save_path)
```

Lampiran B.4 Source Code Augmentasi Citra Dataset Random Forest

```
1    import cv2
2    import os
3    import numpy as np
4    import random
5    from concurrent.futures import ThreadPoolExecutor
6
7    # ========================================
8    # ⚙️ KONFIGURASI
9    # ========================================
10   DATASET_ROOT = r"E:\Dataset Penelitian Bahasa
Isyarat Olfat 2025\Tugas Akhir\Dataset RF\New Dataset RF"
11   AUGMENT_MULTIPLIER = 10  # Total variasi per gambar
12
13   def safe_augment_image(image):
14       h, w = image.shape[:2]
15
16       # --- 1. ROTASI (-10 s/d 10 derajat) ---
17       # Sudut diperkecil sedikit agar aman
18       angle = random.uniform(-10, 10)
19       center = (w // 2, h // 2)
20       M_rot = cv2.getRotationMatrix2D(center, angle,
1.0)
21       image = cv2.warpAffine(image, M_rot, (w, h),
borderMode=cv2.BORDER_CONSTANT, borderValue=(0,0,0))
22
23       # --- 2. SAFE ZOOM (Fokus Zoom Out) ---
24       # Rentang scale: 0.85 (Jauh) sampai 1.05
(Sedikit Dekat)
```

```
25        # Kebanyakan akan menjauh (Zoom Out) agar tangan
tidak terpotong
26        scale = random.uniform(0.85, 1.05)
27
28        if scale < 1.0:
29            # === ZOOM OUT (Mengecil) ===
30            # Gambar dikecilkan, lalu ditempel di tengah
background hitam
31            # Ini 100% AMAN, tangan tidak akan hilang
32            new_h, new_w = int(h * scale), int(w *
scale)
33            resized = cv2.resize(image, (new_w, new_h))
34
35            # Buat kanvas hitam seukuran asli
36            canvas = np.zeros((h, w, 3), dtype=np.uint8)
37
38            # Hitung posisi tengah
39            y_off = (h - new_h) // 2
40            x_off = (w - new_w) // 2
41
42            # Tempel gambar kecil ke kanvas
43            canvas[y_off:y_off+new_h, x_off:x_off+new_w]
= resized
44            image = canvas
45
46        elif scale > 1.0:
47            # === ZOOM IN (Membesar) ===
48            # Dibatasi maksimal 5% agar tidak memotong
jari
49            new_h, new_w = int(h / scale), int(w /
scale)
50            top = (h - new_h) // 2
51            left = (w - new_w) // 2
52
53            cropped = image[top:top+new_h,
left:left+new_w]
54            image = cv2.resize(cropped, (w, h))
55
56        # --- 3. GESER SEDIKIT (Translation) ---
57        # Geser maksimal 5% dari lebar/tinggi gambar
58        tx = random.uniform(-0.05, 0.05) * w
59        ty = random.uniform(-0.05, 0.05) * h
60        M_trans = np.float32([[1, 0, tx], [0, 1, ty]])
61        image = cv2.warpAffine(image, M_trans, (w, h),
borderMode=cv2.BORDER_CONSTANT, borderValue=(0,0,0))
```

```
62
63          return image
64
65      def process_file(file_info):
66          root, filename = file_info
67
68          if "_aug_" in filename:
69              return 0
70
71          img_path = os.path.join(root, filename)
72          image = cv2.imread(img_path)
73
74          if image is None: return 0
75
76          count = 0
77          base_name = os.path.splitext(filename)[0]
78
79          for i in range(AUGMENT_MULTIPLIER):
80              try:
81                  aug_img = safe_augment_image(image)
82
83                  new_filename =
f"{base_name}_aug_{i}.jpg"
84                  save_path = os.path.join(root,
new_filename)
85                  cv2.imwrite(save_path, aug_img)
86                  count += 1
87              except Exception as e:
88                  print(f"Error processing {filename}:
{e}")
89
90          return count
91
92      def main():
93          print(f"🚀 Memulai SAFE AUGMENTATION pada:
{DATASET_ROOT}")
94
95          all_files = []
96          for root, dirs, files in os.walk(DATASET_ROOT):
97              for file in files:
98                  if file.lower().endswith(('.jpg',
'.jpeg', '.png')):
99                      all_files.append((root, file))
100
101         print(f"📄 Memproses {len(all_files)} file
```

```
gambar...")
102
103      with ThreadPoolExecutor(max_workers=8) as
executor:
104          results = executor.map(process_file,
all_files)
105          total = sum(results)
106
107      print(f"\n✅ SELESAI! {total} gambar variasi
aman dibuat.")
108
109  if __name__ == "__main__":
110      main()
```

Lampiran B.4 Source Code Ekstrak Landmark Dataset Random Forest

```
1    import os
2    import cv2
3    import mediapipe as mp
4    import pandas as pd
5
6    # Inisialisasi Mediapipe Hands
7    mp_hands = mp.solutions.hands
8    hands = mp_hands.Hands(static_image_mode=True,
max_num_hands=2, min_detection_confidence=0.6)
9
10   def extract_landmarks_from_image(image_path,
do_flip=True):
11       """
12       Ekstrak landmark dari 1 gambar.
13       Bisa di-flip horizontal tanpa mengubah file
aslinya.
14       """
15       image = cv2.imread(image_path)
16       if image is None:
17           return None
18
19       # FLIP gambar jika ingin konsisten dengan kamera
20       if do_flip:
21           image = cv2.flip(image, 1)
22
23       image_rgb = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
24       results = hands.process(image_rgb)
25
```

```
26        if not results.multi_hand_landmarks:
27            return None
28
29        data = []
30        for hand_landmarks in
results.multi_hand_landmarks:
31            for lm in hand_landmarks.landmark:
32                data.extend([lm.x, lm.y, lm.z])
33
34        # padding kalau hanya 1 tangan
35        if len(results.multi_hand_landmarks) == 1:
36            data.extend([0.0] * 63)
37
38        if len(data) != 126:
39            return None
40
41        return data
42
43    def process_dataset(root_dir, output_csv):
44        """Loop semua folder → ekstrak landmark → simpan
ke CSV"""
45        dataset = []
46
47        for person in os.listdir(root_dir):
48            person_path = os.path.join(root_dir, person)
49            if not os.path.isdir(person_path):
50                continue
51
52            for gesture in os.listdir(person_path):
53                gesture_path = os.path.join(person_path,
gesture)
54                if not os.path.isdir(gesture_path):
55                    continue
56
57                for filename in
os.listdir(gesture_path):
58                    if not
filename.lower().endswith((".jpg", ".jpeg", ".png")):
59                        continue
60
61                    file_path =
os.path.join(gesture_path, filename)
62                    landmarks =
extract_landmarks_from_image(file_path)
63
```

```
                    if landmarks is not None:
                        dataset.append(landmarks + [gesture])

        # Buat DataFrame
        num_landmarks = 42 * 3  # 42 titik (2 tangan), tiap titik ada x,y,z
        columns = [f"{axis}{i}" for i in range(42) for axis in ["x", "y", "z"]]
        columns.append("label")

        df = pd.DataFrame(dataset, columns=columns)

        # Simpan ke CSV
        df.to_csv(output_csv, index=False)
        print(f"✅ Ekstraksi selesai, hasil disimpan di: {output_csv}")


    # --------------------
    # 📌 Contoh penggunaan
    # --------------------
    root_dir = r"E:\Dataset Penelitian Bahasa Isyarat Olfat 2025\Tugas Akhir\Dataset RF Old"
    output_csv = r"E:\Dataset Penelitian Bahasa Isyarat Olfat 2025\Tugas Akhir\Dataset RF Old\dataset_landmarks_RF_OLD_DATASET.csv"

    process_dataset(root_dir, output_csv)
```

Lampiran B.5 Source Code Train Random Forest

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, cross_val_predict, learning_curve
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pickle
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
11
12   # ========================
13   # 1. Load dataset
14   # ========================
15   data_path = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Dataset RF
Old\dataset_landmarks_RF_OLD_DATASET.csv"
16   df = pd.read_csv(data_path)
17
18   # Pisahkan fitur dan label
19   X = df.drop("label", axis=1).values
20   y = df["label"].values
21
22   # Encode label jika masih teks
23   if y.dtype.kind in {'U', 'S', 'O'}:
24       le = LabelEncoder()
25       y_encoded = le.fit_transform(y)
26       class_names = le.classes_
27   else:
28       y_encoded = y
29       le = None
30       class_names = np.unique(y)
31
32   # ========================
33   # 2. Split dataset (80/20)
34   # ========================
35   X_train, X_test, y_train, y_test = train_test_split(
36       X, y_encoded, test_size=0.2, stratify=y_encoded,
random_state=42
37   )
38
39   # ========================
40   # 3. Train Random Forest (train-test split)
41   # ========================
42   print("[INFO] Melatih model Random Forest...")
43   rf = RandomForestClassifier(
44       n_estimators=100,
45       criterion="gini",
46       max_depth=None,
47       random_state=42
48   )
49   rf.fit(X_train, y_train)
50
51   # ========================
52   # 4. Evaluasi train-test split
```

```
53   # ========================
54   y_pred = rf.predict(X_test)
55   acc = accuracy_score(y_test, y_pred)
56   print(f"✅ Akurasi test (80/20 split): {acc:.2f}\n")
57
58   cm = confusion_matrix(y_test, y_pred)
59
60   # ========================
61   # 5. K-Fold Cross Validation & Learning Curve
62   # ========================
63   k = 5  # jumlah fold
64   skf = StratifiedKFold(n_splits=k, shuffle=True,
random_state=42)
65
66   print(f"[INFO] Menjalankan {k}-Fold Cross
Validation...")
67   cv_scores = cross_val_score(rf, X, y_encoded,
cv=skf)
68   print(f"✅ K-Fold CV mean accuracy:
{cv_scores.mean():.2f} ± {cv_scores.std():.2f}")
69
70   y_pred_cv = cross_val_predict(rf, X, y_encoded,
cv=skf)
71   cv_cm = confusion_matrix(y_encoded, y_pred_cv)
72
73   # --- PLOT LEARNING CURVE (Kurva Pembelajaran) ---
74   print("[INFO] Membuat Learning Curve...")
75   train_sizes_abs, train_scores, val_scores =
learning_curve(
76       rf, X, y_encoded, cv=skf, n_jobs=-1,
77       train_sizes=np.linspace(0.1, 1.0, 10),
78       scoring="accuracy"
79   )
80
81   train_scores_mean = np.mean(train_scores, axis=1)
82   train_scores_std = np.std(train_scores, axis=1)
83   val_scores_mean = np.mean(val_scores, axis=1)
84   val_scores_std = np.std(val_scores, axis=1)
85
86   # Direktori penyimpanan
87   results_dir = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Source Code\results train RF Old"
88   os.makedirs(results_dir, exist_ok=True)
89
90   plt.figure(figsize=(10, 6))
```

```
91   plt.title("Learning Curve (Random Forest)")
92   plt.xlabel("Jumlah Data Latih")
93   plt.ylabel("Akurasi")
94   plt.grid(True, linestyle="--", alpha=0.6)
95   plt.fill_between(train_sizes_abs, train_scores_mean
- train_scores_std,
96                    train_scores_mean +
train_scores_std, alpha=0.1, color="blue")
97   plt.fill_between(train_sizes_abs, val_scores_mean -
val_scores_std,
98                    val_scores_mean + val_scores_std,
alpha=0.1, color="orange")
99   plt.plot(train_sizes_abs, train_scores_mean, 'o-',
color="blue", label="Skor Training")
100  plt.plot(train_sizes_abs, val_scores_mean, 'o-',
color="orange", label="Skor Validasi (CV)")
101  plt.legend(loc="best")
102  plt.savefig(os.path.join(results_dir,
"learning_curve_rf.png"))
103  plt.close()
104
105  # ========================
106  # 6. Simpan Model
107  # ========================
108  model_path = os.path.join(results_dir,
"random_forest_bisindo_kcross.pkl")
109  with open(model_path, "wb") as f:
110      pickle.dump(rf, f)
111
112  if le is not None:
113      le_path = os.path.join(results_dir,
"label_encoder_kcross.pkl")
114      with open(le_path, "wb") as f:
115          pickle.dump(le, f)
116
117  print(f"\n✅ Model disimpan di: {model_path}")
118
119  # ========================
120  # 7. Simpan Laporan & Confusion Matrix (SUPER JELAS)
121  # ========================
122
123  # --- A. Simpan Classification Reports ---
124  report_path = os.path.join(results_dir,
"classification_report_split_rf.txt")
125  with open(report_path, "w") as f:
```

```
126      f.write("=== Classification Report (80/20 Split)
===\n")
127      f.write(classification_report(y_test, y_pred,
target_names=class_names))
128
129  cv_report_path = os.path.join(results_dir,
"classification_report_cv_rf.txt")
130  with open(cv_report_path, "w") as f:
131      f.write(f"=== Classification Report ({k}-Fold
Cross Validation) ===\n")
132      f.write(classification_report(y_encoded,
y_pred_cv, target_names=class_names))
133
134  # --- B. Simpan Confusion Matrix 80/20 Split (UKURAN
BESAR) ---
135  print("[INFO] Menyimpan Confusion Matrix 80/20 (High
Res)...")
136  fig_cm, ax_cm = plt.subplots(figsize=(40, 40))  # <-
-- Ukuran Raksasa
137
138  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
139              xticklabels=class_names,
yticklabels=class_names,
140              ax=ax_cm, annot_kws={"size": 6}) # Font
angka kecil agar muat
141
142  ax_cm.set_xticklabels(ax_cm.get_xticklabels(),
rotation=90, fontsize=8)
143  ax_cm.set_yticklabels(ax_cm.get_yticklabels(),
rotation=0, fontsize=8)
144  ax_cm.set_xlabel("Predicted Label", fontsize=15)
145  ax_cm.set_ylabel("True Label", fontsize=15)
146  ax_cm.set_title("Confusion Matrix (80/20 Split)",
fontsize=20)
147
148  plt.savefig(os.path.join(results_dir,
"confusion_matrix_split_rf.png"), dpi=300,
bbox_inches='tight')
149  plt.savefig(os.path.join(results_dir,
"confusion_matrix_split_rf.svg"), bbox_inches='tight')
150  plt.close(fig_cm)
151
152  # --- C. Simpan Confusion Matrix Cross-Validation
(UKURAN BESAR) ---
153  print("[INFO] Menyimpan Confusion Matrix CV (High
```

```
Res)...")
154  fig_cvcm, ax_cvcm = plt.subplots(figsize=(40, 40)) #
<--- Ukuran Raksasa
155
156  sns.heatmap(cv_cm, annot=True, fmt="d",
cmap="Greens",
157              xticklabels=class_names,
yticklabels=class_names,
158              ax=ax_cvcm, annot_kws={"size": 6}) #
Font angka kecil agar muat
159
160  ax_cvcm.set_xticklabels(ax_cvcm.get_xticklabels(),
rotation=90, fontsize=8)
161  ax_cvcm.set_yticklabels(ax_cvcm.get_yticklabels(),
rotation=0, fontsize=8)
162  ax_cvcm.set_xlabel("Predicted Label", fontsize=15)
163  ax_cvcm.set_ylabel("True Label", fontsize=15)
164  ax_cvcm.set_title(f"Confusion Matrix ({k}-Fold Cross
Validation)", fontsize=20)
165
166  plt.savefig(os.path.join(results_dir,
"confusion_matrix_cv_rf.png"), dpi=300,
bbox_inches='tight')
167  plt.savefig(os.path.join(results_dir,
"confusion_matrix_cv_rf.svg"), bbox_inches='tight')
168  plt.close(fig_cvcm)
169
170  print(f"\n✅ Semua hasil (Learning Curve, Conf
Matrix Besar, Laporan) tersimpan di: {results_dir}")
```

Lampiran B.6 Source Code Train LSTM

```
1    import os
2    import numpy as np
3    import tensorflow as tf
4    from tensorflow.keras import layers, models,
regularizers
5    from sklearn.model_selection import
train_test_split, KFold
6    from tensorflow.keras.utils import to_categorical
7    from tensorflow.keras.callbacks import
EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
8    import matplotlib.pyplot as plt
9    import seaborn as sns
10   from sklearn.metrics import confusion_matrix,
```

```
classification_report
11
12   # ------------------------------
13   # Konfigurasi
14   # ------------------------------
15   dataset_path = r"E:\Dataset Penelitian Bahasa
Isyarat Olfat 2025\Tugas Akhir\Dataset LSTM\New Dataset
LSTM REV"
16   SEQ_LENGTH = 20
17   FEATURES = 126
18   K_FOLDS = 5
19
20   RESULTS_DIR = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Source Code\results train
new\LSTM"
21   os.makedirs(RESULTS_DIR, exist_ok=True)
22
23   # ------------------------------
24   # Load dataset
25   # ------------------------------
26   X_all, y_all = [], []
27   gestures = {}
28   class_counter = 0
29
30   def normalize_sequence(seq, target_len=SEQ_LENGTH):
31       if seq.shape[0] < target_len:
32           pad_width = target_len - seq.shape[0]
33           padding = np.zeros((pad_width,
seq.shape[1]))
34           return np.vstack([seq, padding])
35       elif seq.shape[0] > target_len:
36           return seq[:target_len]
37       else:
38           return seq
39
40   print("[INFO] Memuat dataset...")
41   for root, dirs, files in os.walk(dataset_path):
42       for file in files:
43           if file.endswith(".npz"):
44               path = os.path.join(root, file)
45               data = np.load(path, allow_pickle=True)
46               X, y = data["X"], data["y"]
47               X_norm =
np.array([normalize_sequence(seq) for seq in X])
48               gesture_name =
```

```
file.replace("_dataset.npz", "")
49                if gesture_name not in gestures:
50                    gestures[gesture_name] =
class_counter
51                    class_counter += 1
52                y_label = gestures[gesture_name]
53                X_all.append(X_norm)
54                y_all.append(np.full(len(X_norm),
y_label))
55                print(f"[INFO] Loaded {file} from
{os.path.basename(root)}: {X_norm.shape}")
56
57    X_all = np.concatenate(X_all, axis=0)
58    y_all = np.concatenate(y_all, axis=0)
59    gesture_list = list(gestures.keys())
60    num_classes = len(gesture_list)
61    y_cat_all = to_categorical(y_all,
num_classes=num_classes)
62
63    print(f"[INFO] Total Data: {X_all.shape}, Total
Kelas: {num_classes}")
64
65    # ------------------------------
66    # Split dataset 80% train+val / 20% test
67    # ------------------------------
68    X_train_val, X_test, y_train_val, y_test =
train_test_split(
69        X_all, y_cat_all, test_size=0.2, stratify=y_all,
random_state=42
70    )
71
72    # ------------------------------
73    # K-Fold Cross Validation pada train+val
74    # ------------------------------
75    kf = KFold(n_splits=K_FOLDS, shuffle=True,
random_state=42)
76    fold_no = 1
77    histories = []
78
79    for train_idx, val_idx in kf.split(X_train_val):
80        print(f"\n[INFO] Training fold
{fold_no}/{K_FOLDS}")
81
82        X_train, X_val = X_train_val[train_idx],
X_train_val[val_idx]
```

```
83        y_train, y_val = y_train_val[train_idx],
y_train_val[val_idx]
84
85        # Dropout di sini Anda set 0.3 pada kode
sebelumnya,
86        # Anda bisa ubah ke 0.6 jika ingin kembali ke
konfigurasi awal
87        model = models.Sequential([
88            layers.Input(shape=(SEQ_LENGTH, FEATURES)),
89            layers.LSTM(64, return_sequences=False,
unroll=True),
90            layers.Dense(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
91            layers.Dropout(0.3),
92            layers.Dense(num_classes,
activation='softmax')
93        ])
94
95        model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
96
97        callbacks = [
98            EarlyStopping(monitor="val_loss",
patience=10, restore_best_weights=True),
99
ModelCheckpoint(f"{RESULTS_DIR}/best_model_fold{fold_no}.
h5", monitor="val_loss", save_best_only=True),
100           ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=5, min_lr=1e-6)
101       ]
102
103       history = model.fit(
104           X_train, y_train,
105           validation_data=(X_val, y_val),
106           epochs=300,
107           batch_size=16,
108           callbacks=callbacks,
109           verbose=1
110       )
111
112       histories.append(history)
113
114       # --- BAGIAN SIMPAN REPORT & MATRIX PER FOLD
DIHAPUS ---
115
```

```
116       print(f"[INFO] Fold {fold_no} selesai.")
117       fold_no += 1
118
119   # ------------------------------
120   # Train final model pada seluruh train+val
121   # ------------------------------
122   print("\n[INFO] Melatih model final pada seluruh
data train+val...")
123   # Pastikan nilai Dropout konsisten dengan yang Anda
inginkan (misal 0.3 atau 0.6)
124   model_final = models.Sequential([
125       layers.Input(shape=(SEQ_LENGTH, FEATURES)),
126       layers.LSTM(64, return_sequences=False,
unroll=True),
127       layers.Dense(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
128       layers.Dropout(0.6),
129       layers.Dense(num_classes, activation='softmax')
130   ])
131   model_final.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
132
133   callbacks_final = [
134       EarlyStopping(monitor="val_loss", patience=10,
restore_best_weights=True),
135
ModelCheckpoint(f"{RESULTS_DIR}/best_model_lstm.h5",
monitor="val_loss", save_best_only=True),
136       ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=5, min_lr=1e-6)
137   ]
138
139   history_final = model_final.fit(
140       X_train_val, y_train_val,
141       validation_data=(X_test, y_test),
142       epochs=300,
143       batch_size=16,
144       callbacks=callbacks_final
145   )
146
147   # ------------------------------
148   # Save final model & gestures
149   # ------------------------------
150   model_final.save(os.path.join(RESULTS_DIR,
"model_lstm.h5"))
```

```
151  np.save(os.path.join(RESULTS_DIR,
"gestures_labels.npy"), gesture_list)
152
153  # ------------------------------
154  # Plot Kurva Akurasi & Loss
155  # ------------------------------
156  plt.figure(figsize=(12, 5))
157
158  # Kurva Akurasi
159  plt.subplot(1, 2, 1)
160  plt.plot(history_final.history["accuracy"],
label="Train Accuracy", color='blue')
161  plt.plot(history_final.history["val_accuracy"],
label="Validation Accuracy", color='orange')
162  plt.title("LSTM Model Accuracy")
163  plt.xlabel("Epoch")
164  plt.ylabel("Accuracy")
165  plt.legend()
166  plt.grid(True, linestyle="--", alpha=0.6)
167
168  # Kurva Loss
169  plt.subplot(1, 2, 2)
170  plt.plot(history_final.history["loss"], label="Train
Loss", color='blue')
171  plt.plot(history_final.history["val_loss"],
label="Validation Loss", color='orange')
172  plt.title("LSTM Model Loss")
173  plt.xlabel("Epoch")
174  plt.ylabel("Loss")
175  plt.legend()
176  plt.grid(True, linestyle="--", alpha=0.6)
177
178  plt.tight_layout()
179  plt.savefig(os.path.join(RESULTS_DIR,
"training_curves_lstm.png"))
180  plt.show()
181
182  print(f"[INFO] Kurva akurasi dan loss disimpan di
{RESULTS_DIR}/training_curves_lstm.png")
183
184  # ------------------------------
185  # Evaluasi akhir pada test set (TETAP DISIMPAN)
186  # ------------------------------
187  print("[INFO] Evaluasi akhir pada Test Set...")
188  y_test_true = np.argmax(y_test, axis=1)
```

```
y_test_pred = np.argmax(model_final.predict(X_test,
verbose=0), axis=1)

# --- CONFUSION MATRIX FINAL (TEST SET) ---
cm_test = confusion_matrix(y_test_true, y_test_pred,
labels=range(num_classes))

# Buat figure besar
fig, ax = plt.subplots(figsize=(40, 40))
sns.heatmap(
    cm_test,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=gesture_list,
    yticklabels=gesture_list,
    ax=ax,
    annot_kws={"size": 6}
)

ax.set_xticklabels(ax.get_xticklabels(),
rotation=90, fontsize=8)
ax.set_yticklabels(ax.get_yticklabels(), rotation=0,
fontsize=8)
ax.set_title("Confusion Matrix (Test Set)",
fontsize=20)
ax.set_ylabel("True Label", fontsize=15)
ax.set_xlabel("Predicted Label", fontsize=15)

# Simpan PNG & SVG
plt.savefig(os.path.join(RESULTS_DIR,
"confusion_matrix_test_lstm_cross.png"), dpi=300,
bbox_inches='tight')
plt.savefig(os.path.join(RESULTS_DIR,
"confusion_matrix_test_lstm_cross.svg"),
bbox_inches='tight')
plt.close(fig)

report_test = classification_report(y_test_true,
y_test_pred, target_names=gesture_list, digits=4)
with open(os.path.join(RESULTS_DIR,
"classification_report_test_lstm_cross.txt"), "w") as f:
    f.write(report_test)

print(f"[INFO] Evaluasi test set selesai. Confusion
```

```
matrix & classification report disimpan di
{RESULTS_DIR}")
```

Lampiran B.7 Source Code Train Transformer

```
1    import numpy as np
2    import tensorflow as tf
3    from tensorflow import keras
4    from tensorflow.keras import layers
5    from sklearn.model_selection import StratifiedKFold,
train_test_split
6    from sklearn.metrics import classification_report,
confusion_matrix
7    import matplotlib.pyplot as plt
8    import seaborn as sns
9    import os
10
11   # ============================================
12   # 1. Konfigurasi & Memuat Dataset
13   # ============================================
14   DATASET_DIR = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Dataset LSTM\New Dataset LSTM REV"
15   SAVE_DIR = r"E:\Dataset Penelitian Bahasa Isyarat
Olfat 2025\Tugas Akhir\Source Code\results train
new\Transformer Dropout 0.3"
16   SEQ_LENGTH = 20
17   NUM_FOLDS = 5
18   BATCH_SIZE = 16   # DISAMAKAN dengan LSTM
19   EPOCHS = 300      # DISAMAKAN dengan LSTM
20   DROPOUT_RATE = 0.3 # DISAMAKAN dengan LSTM
21
22   # Buat folder simpan
23   os.makedirs(SAVE_DIR, exist_ok=True)
24
25   def normalize_sequence(seq, target_len=SEQ_LENGTH):
26       if seq.shape[0] > target_len: return
seq[:target_len]
27       if seq.shape[0] < target_len:
28           pad_width = target_len - seq.shape[0]
29           padding = np.zeros((pad_width,
seq.shape[1]))
30           return np.vstack([seq, padding])
31       return seq
32
33   X_all, y_all, gestures, class_counter = [], [], {},
```

```
0
34
35   print("[INFO] Memuat dataset secara rekursif...")
36   for root, dirs, files in os.walk(DATASET_DIR):
37       for file in files:
38           if file.endswith(".npz"):
39               path = os.path.join(root, file)
40               try:
41                   data = np.load(path,
allow_pickle=True)
42                   X_sequences = data["X"]
43
44                   # Cek jika file kosong
45                   if len(X_sequences) == 0:
46                       continue
47
48                   X_norm =
np.array([normalize_sequence(seq) for seq in
X_sequences])
49                   gesture_name =
file.replace("_dataset.npz", "")
50
51                   if gesture_name not in gestures:
52                       gestures[gesture_name] =
class_counter
53                       class_counter += 1
54
55                   y_label = gestures[gesture_name]
56                   X_all.append(X_norm)
57                   y_all.append(np.full(len(X_norm),
y_label))
58                   # print(f"[INFO] Loaded {file}")
59               except Exception as e:
60                   print(f"[ERROR] Gagal load {file}:
{e}")
61
62   if len(X_all) == 0:
63       print("[CRITICAL] Tidak ada data ditemukan!")
64       exit()
65
66   X = np.concatenate(X_all, axis=0)
67   y = np.concatenate(y_all, axis=0)
68   CLASS_NAMES = list(gestures.keys())
69   NUM_CLASSES = len(CLASS_NAMES)
70
```

```
71   print(f"[INFO] Total Data: {X.shape[0]} sequences,
{NUM_CLASSES} kelas.")
72
73   # --- SPLIT 80% (train+val) dan 20% (test) ---
74   X_train_val, X_test, y_train_val, y_test =
train_test_split(
75       X, y, test_size=0.2, stratify=y, random_state=42
76   )
77
78   # ================================================
79   # 2. Arsitektur Transformer (DIPERBAIKI)
80   # ================================================
81   def build_transformer_model(input_shape,
num_classes, d_model=64, num_heads=4, ff_dim=64,
num_transformer_blocks=2, dropout=DROPOUT_RATE):
82       inputs = keras.Input(shape=input_shape)
83       x = layers.Dense(d_model,
name="dense_projection")(inputs)
84
85       # Positional Encoding
86       positions = tf.range(start=0,
limit=input_shape[0], delta=1)
87       pos_embedding =
keras.layers.Embedding(input_dim=input_shape[0],
output_dim=d_model)(positions)
88       x = x + pos_embedding
89
90       # Transformer Blocks
91       for _ in range(num_transformer_blocks):
92           # Attention dengan Dropout variabel
93           attn_output = layers.MultiHeadAttention(
94               num_heads=num_heads, key_dim=d_model,
dropout=dropout
95           )(query=x, value=x, key=x)
96           x = layers.LayerNormalization(epsilon=1e-
6)(x + attn_output)
97
98           # Feed Forward
99           ffn_output = keras.Sequential([
100              layers.Dense(ff_dim, activation="relu"),
101              layers.Dense(d_model)
102          ])(x)
103          x = layers.LayerNormalization(epsilon=1e-
6)(x + ffn_output)
104
```

```
105       # Classification Head
106       x = layers.GlobalAveragePooling1D()(x)
107
108       # --- PERBAIKAN: Menggunakan variabel dropout
(0.3), bukan hardcode 0.2 ---
109       x = layers.Dropout(dropout)(x)
110       # ----------------------------------------------
-------------------------
111
112       x = layers.Dense(ff_dim, activation="relu")(x)
113       outputs = layers.Dense(num_classes,
activation="softmax")(x)
114
115       return keras.Model(inputs=inputs,
outputs=outputs)
116
117   # ================================================
118   # 3. K-Fold Cross-Validation (Setara LSTM)
119   # ================================================
120   skf = StratifiedKFold(n_splits=NUM_FOLDS,
shuffle=True, random_state=42)
121   cv_scores = []
122   fold_no = 1
123
124   print("\n[INFO] Memulai 5-Fold Cross Validation...")
125   for train_index, val_index in skf.split(X_train_val,
y_train_val):
126       print(f"--- FOLD {fold_no}/{NUM_FOLDS} ---")
127
128       X_train, X_val = X_train_val[train_index],
X_train_val[val_index]
129       y_train, y_val = y_train_val[train_index],
y_train_val[val_index]
130
131       INPUT_SHAPE = (X_train.shape[1],
X_train.shape[2])
132
133       # Build Model
134       model =
build_transformer_model(input_shape=INPUT_SHAPE,
num_classes=NUM_CLASSES, dropout=DROPOUT_RATE)
135
136
model.compile(optimizer=keras.optimizers.Adam(learning_ra
te=0.001),
```

```
137
loss="sparse_categorical_crossentropy",
138                     metrics=["accuracy"])
139
140       # Callbacks (Sama seperti LSTM: EarlyStopping +
ReduceLR)
141       callbacks_cv = [
142
keras.callbacks.EarlyStopping(monitor="val_loss",
patience=10, restore_best_weights=True),
143
keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=5, min_lr=1e-6)
144       ]
145
146       # Train dengan Batch Size 16 & Epoch 300
147       model.fit(X_train, y_train,
148                 batch_size=BATCH_SIZE,
149                 epochs=EPOCHS,
150                 validation_data=(X_val, y_val),
151                 callbacks=callbacks_cv,
152                 verbose=0) # verbose 0 agar rapi
153
154       scores = model.evaluate(X_val, y_val, verbose=0)
155       print(f"   Skor untuk fold {fold_no}: Akurasi
{scores[1]*100:.2f}%")
156       cv_scores.append(scores[1])
157       fold_no += 1
158
159   print("\n--- Hasil Cross-Validation ---")
160   print(f"Akurasi Rata-rata:
{np.mean(cv_scores)*100:.2f}% (+/-
{np.std(cv_scores)*100:.2f}%)")
161
162   # ================================================
163   # 4. Training Final & Evaluasi Akhir (Setara LSTM)
164   # ================================================
165   print("\n[INFO] Melatih model final pada seluruh 80%
data...")
166
167   INPUT_SHAPE = (X_train_val.shape[1],
X_train_val.shape[2])
168   final_model =
build_transformer_model(input_shape=INPUT_SHAPE,
num_classes=NUM_CLASSES, dropout=DROPOUT_RATE)
```

```
169
170  final_model.compile(optimizer=keras.optimizers.Adam(
learning_rate=0.001),
171
loss="sparse_categorical_crossentropy",
172                      metrics=["accuracy"])
173
174  # Callbacks Final (Sama seperti LSTM)
175  callbacks_final = [
176
keras.callbacks.EarlyStopping(monitor="val_loss",
patience=10, restore_best_weights=True),
177
keras.callbacks.ModelCheckpoint(os.path.join(SAVE_DIR,
"transformer_best.keras"), save_best_only=True,
monitor="val_accuracy"),
178
keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=5, min_lr=1e-6)
179  ]
180
181  # Training Final (Gunakan validation_data=(X_test,
y_test) seperti LSTM)
182  history = final_model.fit(
183      X_train_val, y_train_val,
184      batch_size=BATCH_SIZE,  # 16
185      epochs=EPOCHS,          # 300
186      validation_data=(X_test, y_test), # Konsisten
dengan LSTM
187      callbacks=callbacks_final,
188      verbose=1
189  )
190
191  print("\n[INFO] Mengevaluasi model final pada 20%
data test...")
192  test_loss, test_acc = final_model.evaluate(X_test,
y_test)
193  print(f"\nAkurasi pada data uji final:
{test_acc:.4f}")
194
195  # Simpan Model & Label
196  final_model.save(os.path.join(SAVE_DIR,
"transformer_final.keras"))
197  np.save(os.path.join(SAVE_DIR, "labels.npy"),
np.array(CLASS_NAMES))
```

```
198
199  # Prediksi
200  y_pred = np.argmax(final_model.predict(X_test),
axis=1)
201
202  # Simpan Classification Report
203  report = classification_report(y_test, y_pred,
target_names=CLASS_NAMES, digits=4)
204  with open(os.path.join(SAVE_DIR,
"classification_report.txt"), "w") as f:
205      f.write(report)
206
207  # ================================================
208  # === CONFUSION MATRIX RESOLUSI TINGGI ===
209  # ================================================
210  print("[INFO] Membuat Confusion Matrix Resolusi
Tinggi...")
211  cm = confusion_matrix(y_test, y_pred)
212
213  fig, ax = plt.subplots(figsize=(40, 40)) # Ukuran
Besar
214  sns.heatmap(
215      cm,
216      annot=True,
217      fmt="d",
218      cmap="Blues",
219      xticklabels=CLASS_NAMES,
220      yticklabels=CLASS_NAMES,
221      ax=ax,
222      annot_kws={"size": 6}
223  )
224
225  ax.set_xticklabels(ax.get_xticklabels(),
rotation=90, fontsize=8)
226  ax.set_yticklabels(ax.get_yticklabels(), rotation=0,
fontsize=8)
227  ax.set_title("Confusion Matrix (Final Test)",
fontsize=20)
228  ax.set_ylabel("True Label", fontsize=15)
229  ax.set_xlabel("Predicted Label", fontsize=15)
230
231  plt.savefig(os.path.join(SAVE_DIR,
"confusion_matrix.png"), dpi=300, bbox_inches='tight')
232  plt.savefig(os.path.join(SAVE_DIR,
"confusion_matrix.svg"), bbox_inches='tight')
```

```
233  plt.close(fig)
234
235  # Simpan Kurva
236  plt.figure(figsize=(12, 5))
237  plt.subplot(1, 2, 1)
238  plt.plot(history.history['accuracy'], label='Train
Acc')
239  plt.plot(history.history['val_accuracy'], label='Val
Acc')
240  plt.title('Accuracy')
241  plt.legend()
242  plt.grid(True, linestyle="--")
243
244  plt.subplot(1, 2, 2)
245  plt.plot(history.history['loss'], label='Train
Loss')
246  plt.plot(history.history['val_loss'], label='Val
Loss')
247  plt.title('Loss')
248  plt.legend()
249  plt.grid(True, linestyle="--")
250
251  plt.tight_layout()
252  plt.savefig(os.path.join(SAVE_DIR, "curves.png"))
253  plt.close()
254
255  print(f"\n[INFO] Selesai! Hasil disimpan di:
{SAVE_DIR}")
```

Lampiran B.8 Source Code Konversi Tensorflow Lite

```
1    import os
2    import numpy as np
3    import tensorflow as tf
4
5    # ------------------------------
6    # Path model input/output
7    # ------------------------------
8    keras_model_path = r"E:\Dataset Penelitian Bahasa
Isyarat Olfat 2025\Tugas Akhir\Source Code\results train
new\Transformer Dropout 0.3\transformer_best.keras"
9    tflite_out_dir = r"E:\Dataset Penelitian Bahasa
Isyarat Olfat 2025\Tugas Akhir\Source Code\results train
new\Transformer Dropout 0.3"
10   tflite_model_path = os.path.join(tflite_out_dir,
```

```
"model_transformer.tflite")
11
12   os.makedirs(tflite_out_dir, exist_ok=True)
13
14   # ------------------------------
15   # 1. Load model .keras
16   # ------------------------------
17   print("[INFO] Loading Keras model...")
18   model = tf.keras.models.load_model(keras_model_path,
compile=False)
19   model.summary()
20
21   # ------------------------------
22   # 2. Convert to TFLite (Float16 quantization)
23   # ------------------------------
24   print("\n[INFO] Converting to TensorFlow Lite
(float16 quantization)...")
25   converter =
tf.lite.TFLiteConverter.from_keras_model(model)
26
27   # Gunakan optimisasi default
28   converter.optimizations = [tf.lite.Optimize.DEFAULT]
29   # Targetkan tipe data float16 untuk bobot (lebih
ringan & cepat di ARM)
30   converter.target_spec.supported_types = [tf.float16]
31   # Pastikan inference tetap float32 agar kompatibel
32   converter.target_spec.supported_ops =
[tf.lite.OpsSet.TFLITE_BUILTINS]
33
34   # Konversi dan simpan
35   tflite_model = converter.convert()
36   with open(tflite_model_path, "wb") as f:
37       f.write(tflite_model)
38
39   size_kb = os.path.getsize(tflite_model_path) / 1024
40   print(f"[OK] Model berhasil disimpan:
{tflite_model_path} ({size_kb:.1f} KB)")
41
42   # ------------------------------
43   # 3. (Opsional) Cek input/output signature
44   # ------------------------------
45   interpreter =
tf.lite.Interpreter(model_path=tflite_model_path)
46   interpreter.allocate_tensors()
47   input_details = interpreter.get_input_details()
```

```
48   output_details = interpreter.get_output_details()
49
50   print("\n[INFO] Input Tensor:", input_details)
51   print("[INFO] Output Tensor:", output_details)
52   print("\n[INFO] Konversi selesai. Model siap
digunakan di Raspberry Pi.")
```

Lampiran B.9 Source Code Hybrid Model RF+LSTM

```
1    import sys
2    sys.path.append('/usr/lib/python3/dist-packages')  #
pastikan picamera2 bisa diimport
3
4    import cv2
5    import numpy as np
6    import mediapipe as mp
7    import joblib
8    import time
9    # [UBAH BAGIAN INI] Menggunakan tflite_runtime
menggantikan tensorflow full
10   from tflite_runtime.interpreter import Interpreter
11   from picamera2 import Picamera2
12   from collections import deque
13
14   # ------------------------------
15   # Load Models & Labels
16   # ------------------------------
17   rf_model = joblib.load(
18
"/home/olfat/Desktop/projects/projectsenv/random_forest_b
isindo_kcross.pkl"
19   )
20
21   # [UBAH BAGIAN INI] Load TFLite LSTM menggunakan
Interpreter dari tflite_runtime
22   interpreter =
Interpreter(model_path="/home/olfat/Desktop/projects/proj
ectsenv/model_lstm.tflite")
23   interpreter.allocate_tensors()
24   input_details = interpreter.get_input_details()
25   output_details = interpreter.get_output_details()
26
27   labels = np.load(
28
"/home/olfat/Desktop/projects/projectsenv/all_gestures_la
```

```
bels_lstm.npy", allow_pickle=True
29   ).item()  # dict -> {"static": [...], "dynamic":
[...]}
30
31   # ------------------------------
32   # CONFIG
33   # ------------------------------
34   SEQ_LENGTH = 20
35   MIN_SEQ_FOR_LSTM = 12
36   MOTION_THRESHOLD = 0.005
37   HOLD_FRAMES = 6
38   COOLDOWN_TIME = 1.5
39   HAND_GRACE_TIME = 1.0   # jeda detik saat tangan
baru muncul
40
41   # ------------------------------
42   # State & Buffers
43   # ------------------------------
44   sequence_buffer = deque(maxlen=SEQ_LENGTH)
45   motion_scores = []
46   hold_counter = 0
47   last_pred_time = 0.0
48   last_result_text = ""
49   last_result_color = (255,255,255)
50   last_result_source = ""
51   prev_landmarks = None
52
53   hand_present = False
54   grace_start_time = None
55
56   print(f"[INFO] Models loaded. Using TFLite LSTM.
SEQ_LENGTH={SEQ_LENGTH}")
57
58   # ------------------------------
59   # Mediapipe
60   # ------------------------------
61   mp_hands = mp.solutions.hands
62   hands = mp_hands.Hands(
63       static_image_mode=False,
64       max_num_hands=2,
65       min_detection_confidence=0.7,
66       min_tracking_confidence=0.7
67   )
68   mp_draw = mp.solutions.drawing_utils
69
```

```
70   # ------------------------------
71   # Helpers
72   # ------------------------------
73   def extract_landmarks(results):
74       if not results.multi_hand_landmarks:
75           return None
76       row = []
77       for hl in results.multi_hand_landmarks:
78           for lm in hl.landmark:
79               row.extend([lm.x, lm.y, lm.z])
80       if len(results.multi_hand_landmarks) == 1:
81           row.extend([0.0]*63)
82       if len(row) < 126:
83           row.extend([0.0]*(126-len(row)))
84       elif len(row) > 126:
85           row = row[:126]
86       return np.array(row, dtype=np.float32)
87
88   def predict_rf(feat):
89       probs = rf_model.predict_proba([feat])[0]
90       cid = int(np.argmax(probs))
91       return labels["static"][cid]
92
93   def predict_lstm_tflite(seq):
94       # ubah jadi numpy array
95       seq = np.array(seq, dtype=np.float32)
96
97       # kalau sequence belum penuh, lakukan padding
(ulang frame terakhir)
98       if len(seq) < SEQ_LENGTH:
99           last_frame = seq[-1] if len(seq) > 0 else
np.zeros((126,), dtype=np.float32)
100          pad_len = SEQ_LENGTH - len(seq)
101          pad_frames =
np.repeat(last_frame[np.newaxis, :], pad_len, axis=0)
102          seq = np.concatenate([seq, pad_frames],
axis=0)
103
104      # pastikan bentuknya (1, SEQ_LENGTH, 126)
105      X = np.expand_dims(seq, axis=0)
106
107
interpreter.set_tensor(input_details[0]['index'], X)
108      interpreter.invoke()
109      probs =
```

```
interpreter.get_tensor(output_details[0]['index'])[0]
110       cid = int(np.argmax(probs))
111       return labels["dynamic"][cid]
112
113   def draw_text_bg(img, text, pos=(10,40),
font_scale=1.0, color=(255,255,255)):
114       x,y = pos
115       font = cv2.FONT_HERSHEY_SIMPLEX
116       thickness = 2
117       (w,h), _ = cv2.getTextSize(text, font,
font_scale, thickness)
118       cv2.rectangle(img, (x-6,y-6), (x+w+6, y+h+6),
(0,0,0), -1)
119       cv2.putText(img, text, (x, y+h-6), font,
font_scale, color, thickness, cv2.LINE_AA)
120
121   # ------------------------------
122   # Camera Init
123   # ------------------------------
124   picam2 = Picamera2()
125   config =
picam2.create_preview_configuration(main={"format":
"XRGB8888", "size": (640, 480)})
126   picam2.configure(config)
127   picam2.start()
128
129   # ------------------------------
130   # Main loop
131   # ------------------------------
132   try:
133       while True:
134           frame = picam2.capture_array()
135           frame = cv2.flip(frame, 1)
136
137           image_rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
138           results = hands.process(image_rgb)
139           image = cv2.cvtColor(image_rgb,
cv2.COLOR_RGB2BGR)
140
141           if results.multi_hand_landmarks:
142               for hl in results.multi_hand_landmarks:
143                   mp_draw.draw_landmarks(image, hl,
mp_hands.HAND_CONNECTIONS)
144
```

```
145          # cooldown prediksi
146          if time.time() - last_pred_time <
COOLDOWN_TIME:
147              if last_result_text:
148                  draw_text_bg(image,
f"{last_result_text} ({last_result_source})",
pos=(10,10), color=last_result_color)
149                  remaining = COOLDOWN_TIME -
(time.time() - last_pred_time)
150                  draw_text_bg(image, f"Next in
{remaining:.1f}s", pos=(10,60), font_scale=0.8,
color=(200,200,200))
151              cv2.imshow("Gesture Recognition", image)
152              if cv2.waitKey(1) & 0xFF == ord('q'):
153                  break
154              continue
155
156          lm = extract_landmarks(results)
157          if lm is None:
158              sequence_buffer.clear()
159              motion_scores.clear()
160              hold_counter = 0
161              prev_landmarks = None
162              hand_present = False
163              grace_start_time = None
164
165              draw_text_bg(image, "Show your hand
(waiting)...", pos=(10,10), color=(200,200,200))
166              cv2.imshow("Gesture Recognition", image)
167              if cv2.waitKey(1) & 0xFF == ord('q'):
168                  break
169              continue
170          else:
171              if not hand_present:
172                  hand_present = True
173                  grace_start_time = time.time()
174
175              if grace_start_time is not None and
(time.time() - grace_start_time < HAND_GRACE_TIME):
176                  remaining = HAND_GRACE_TIME -
(time.time() - grace_start_time)
177                  draw_text_bg(image, f"Stabilizing...
{remaining:.1f}s", pos=(10,10), color=(0,200,200))
178                  cv2.imshow("Gesture Recognition",
image)
```

```
179                     if cv2.waitKey(1) & 0xFF ==
ord('q'):
180                         break
181                     continue
182
183             sequence_buffer.append(lm)
184
185             if prev_landmarks is not None:
186                 motion = np.mean(np.abs(lm -
prev_landmarks))
187             else:
188                 motion = 0.0
189             prev_landmarks = lm
190
191             motion_scores.append(motion)
192             if len(motion_scores) > SEQ_LENGTH:
193                 motion_scores.pop(0)
194
195             draw_text_bg(image, f"Recording
{len(sequence_buffer)}/{SEQ_LENGTH}", pos=(10,10),
color=(200,200,200))
196
197             if len(sequence_buffer) >= 2:
198                 avg_motion =
float(np.mean(motion_scores)) if motion_scores else 0.0
199
200                 if avg_motion > MOTION_THRESHOLD and
len(sequence_buffer) >= MIN_SEQ_FOR_LSTM:
201                     result =
predict_lstm_tflite(list(sequence_buffer))
202                     last_result_text = result
203                     last_result_source = "LSTM"
204                     last_result_color = (0,255,0)
205                     last_pred_time = time.time()
206                     sequence_buffer.clear()
207                     motion_scores.clear()
208                     prev_landmarks = None
209                     hold_counter = 0
210                     draw_text_bg(image,
f"{last_result_text} (LSTM)", pos=(10,10),
color=last_result_color)
211
212                 elif avg_motion <= MOTION_THRESHOLD:
213                     hold_counter += 1
214                     draw_text_bg(image, f"Holding...
```

```
{hold_counter}/{HOLD_FRAMES}", pos=(10,60),
color=(200,200,200))
215                   if hold_counter >= HOLD_FRAMES and
len(sequence_buffer) >= 1:
216                       result =
predict_rf(sequence_buffer[-1])
217                       last_result_text = result
218                       last_result_source = "RF"
219                       last_result_color = (0,0,255)
220                       last_pred_time = time.time()
221                       sequence_buffer.clear()
222                       motion_scores.clear()
223                       prev_landmarks = None
224                       hold_counter = 0
225                       draw_text_bg(image,
f"{last_result_text} (RF)", pos=(10,10),
color=last_result_color)
226
227           cv2.imshow("Gesture Recognition", image)
228           if cv2.waitKey(1) & 0xFF == ord('q'):
229               break
230
231   finally:
232       cv2.destroyAllWindows()
233       hands.close()
234       picam2.stop()
```

Lampiran B.10 Source Code Hybrid Model RF+Transformer

```
1    import sys
2    sys.path.append('/usr/lib/python3/dist-packages')  #
pastikan picamera2 bisa diimport
3
4    import cv2
5    import numpy as np
6    import mediapipe as mp
7    import joblib
8    import time
9    from tflite_runtime.interpreter import Interpreter
# ✅ lebih ringan dibanding tensorflow
10   from picamera2 import Picamera2
11   from collections import deque
12
13   #
==================================================
```

```
14   # 1. LOAD MODEL & LABELS
15   #
=====================================================
16   print("[INFO] Memuat model Random Forest dan
Transformer...")
17
18   # Random Forest untuk gestur statis
19   rf_model =
joblib.load("/home/olfat/Desktop/projects/projectsenv/ran
dom_forest_bisindo_kcross.pkl")
20
21   # Transformer (TensorFlow Lite)
22   interpreter =
Interpreter(model_path="/home/olfat/Desktop/projects/proj
ectsenv/model_transformer.tflite")
23   interpreter.allocate_tensors()
24   input_details = interpreter.get_input_details()
25   output_details = interpreter.get_output_details()
26
27   # Label
28   labels =
np.load("/home/olfat/Desktop/projects/projectsenv/all_ges
tures_labels_transformer.npy", allow_pickle=True).item()
# {"static": [...], "dynamic": [...]}
29
30   print(f"[INFO] Model loaded. Input:
{input_details[0]['shape']}, Output:
{output_details[0]['shape']}")
31
32   #
=====================================================
33   # 2. KONFIGURASI
34   #
=====================================================
35   SEQ_LENGTH = 20
36   MIN_SEQ_FOR_TRANSFORMER = 12
37   MOTION_THRESHOLD = 0.005
38   HOLD_FRAMES = 6
39   COOLDOWN_TIME = 1.5
40   HAND_GRACE_TIME = 1.0   # detik jeda stabilisasi
tangan
41
42   #
=====================================================
43   # 3. STATE & BUFFER
```

```
44   #
==================================================
45   sequence_buffer = deque(maxlen=SEQ_LENGTH)
46   motion_scores = []
47   hold_counter = 0
48   last_pred_time = 0.0
49   last_result_text = ""
50   last_result_color = (255, 255, 255)
51   last_result_source = ""
52   prev_landmarks = None
53
54   hand_present = False
55   grace_start_time = None
56
57   #
==================================================
58   # 4. MEDIAPIPE
59   #
==================================================
60   mp_hands = mp.solutions.hands
61   hands = mp_hands.Hands(
62       static_image_mode=False,
63       max_num_hands=2,
64       min_detection_confidence=0.7,
65       min_tracking_confidence=0.7
66   )
67   mp_draw = mp.solutions.drawing_utils
68
69   #
==================================================
70   # 5. HELPER FUNCTIONS
71   #
==================================================
72   def extract_landmarks(results):
73       if not results.multi_hand_landmarks:
74           return None
75       row = []
76       for hl in results.multi_hand_landmarks:
77           for lm in hl.landmark:
78               row.extend([lm.x, lm.y, lm.z])
79       if len(results.multi_hand_landmarks) == 1:
80           row.extend([0.0] * 63)
81       if len(row) < 126:
82           row.extend([0.0] * (126 - len(row)))
83       elif len(row) > 126:
```

```
            row = row[:126]
        return np.array(row, dtype=np.float32)

    def predict_rf(feat):
        probs = rf_model.predict_proba([feat])[0]
        cid = int(np.argmax(probs))
        return labels["static"][cid]

    def predict_transformer_tflite(seq):
        seq = np.array(seq, dtype=np.float32)

        # padding jika panjang sequence < 20
        if len(seq) < SEQ_LENGTH:
            last_frame = seq[-1] if len(seq) > 0 else
np.zeros((126,), dtype=np.float32)
            pad_len = SEQ_LENGTH - len(seq)
            pad_frames =
np.repeat(last_frame[np.newaxis, :], pad_len, axis=0)
            seq = np.concatenate([seq, pad_frames],
axis=0)

        X = np.expand_dims(seq, axis=0)  # (1, 20, 126)


interpreter.set_tensor(input_details[0]['index'], X)
        interpreter.invoke()
        probs =
interpreter.get_tensor(output_details[0]['index'])[0]
        cid = int(np.argmax(probs))
        return labels["dynamic"][cid]

    def draw_text_bg(img, text, pos=(10,40),
font_scale=1.0, color=(255,255,255)):
        x, y = pos
        font = cv2.FONT_HERSHEY_SIMPLEX
        thickness = 2
        (w, h), _ = cv2.getTextSize(text, font,
font_scale, thickness)
        cv2.rectangle(img, (x-6, y-6), (x+w+6, y+h+6),
(0,0,0), -1)
        cv2.putText(img, text, (x, y+h-6), font,
font_scale, color, thickness, cv2.LINE_AA)

    #
==================================================
```

```
# 6. INISIALISASI KAMERA
#
=====================================================
picam2 = Picamera2()
config =
picam2.create_preview_configuration(main={"format":
"XRGB8888", "size": (640, 480)})
picam2.configure(config)
picam2.start()

print("[INFO] Sistem siap. Tekan 'q' untuk keluar.")

#
=====================================================
# 7. LOOP UTAMA
#
=====================================================
try:
    while True:
        frame = picam2.capture_array()
        frame = cv2.flip(frame, 1)

        image_rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
        results = hands.process(image_rgb)
        image = cv2.cvtColor(image_rgb,
cv2.COLOR_RGB2BGR)

        if results.multi_hand_landmarks:
            for hl in results.multi_hand_landmarks:
                mp_draw.draw_landmarks(image, hl,
mp_hands.HAND_CONNECTIONS)

        # cooldown prediksi
        if time.time() - last_pred_time <
COOLDOWN_TIME:
            if last_result_text:
                draw_text_bg(image,
f"{last_result_text} ({last_result_source})",
pos=(10,10), color=last_result_color)
                remaining = COOLDOWN_TIME -
(time.time() - last_pred_time)
                draw_text_bg(image, f"Next in
{remaining:.1f}s", pos=(10,60), font_scale=0.8,
color=(200,200,200))
```

```
150                 cv2.imshow("Gesture Recognition", image)
151                 if cv2.waitKey(1) & 0xFF == ord('q'):
152                     break
153                 continue
154 
155             lm = extract_landmarks(results)
156             if lm is None:
157                 sequence_buffer.clear()
158                 motion_scores.clear()
159                 hold_counter = 0
160                 prev_landmarks = None
161                 hand_present = False
162                 grace_start_time = None
163 
164                 draw_text_bg(image, "Show your hand
(waiting)...", pos=(10,10), color=(200,200,200))
165                 cv2.imshow("Gesture Recognition", image)
166                 if cv2.waitKey(1) & 0xFF == ord('q'):
167                     break
168                 continue
169             else:
170                 if not hand_present:
171                     hand_present = True
172                     grace_start_time = time.time()
173 
174                 if grace_start_time is not None and
(time.time() - grace_start_time < HAND_GRACE_TIME):
175                     remaining = HAND_GRACE_TIME -
(time.time() - grace_start_time)
176                     draw_text_bg(image, f"Stabilizing...
{remaining:.1f}s", pos=(10,10), color=(0,200,200))
177                     cv2.imshow("Gesture Recognition",
image)
178                     if cv2.waitKey(1) & 0xFF ==
ord('q'):
179                         break
180                     continue
181 
182             # ------------------------------
183             # Proses sequence & motion
184             # ------------------------------
185             sequence_buffer.append(lm)
186             motion = np.mean(np.abs(lm -
prev_landmarks)) if prev_landmarks is not None else 0.0
187             prev_landmarks = lm
```

```
188          motion_scores.append(motion)
189          if len(motion_scores) > SEQ_LENGTH:
190              motion_scores.pop(0)
191
192          draw_text_bg(image, f"Recording
{len(sequence_buffer)}/{SEQ_LENGTH}", pos=(10,10),
color=(200,200,200))
193
194          if len(sequence_buffer) >= 2:
195              avg_motion =
float(np.mean(motion_scores)) if motion_scores else 0.0
196
197              # ------------------------------
198              # Dynamic Gesture → Transformer
199              # ------------------------------
200              if avg_motion > MOTION_THRESHOLD and
len(sequence_buffer) >= MIN_SEQ_FOR_TRANSFORMER:
201                  result =
predict_transformer_tflite(list(sequence_buffer))
202                  last_result_text = result
203                  last_result_source = "Transformer"
204                  last_result_color = (0,255,0)
205                  last_pred_time = time.time()
206                  sequence_buffer.clear()
207                  motion_scores.clear()
208                  prev_landmarks = None
209                  hold_counter = 0
210                  draw_text_bg(image,
f"{last_result_text} (Transformer)", pos=(10,10),
color=last_result_color)
211
212              # ------------------------------
213              # Static Gesture → Random Forest
214              # ------------------------------
215              elif avg_motion <= MOTION_THRESHOLD:
216                  hold_counter += 1
217                  draw_text_bg(image, f"Holding...
{hold_counter}/{HOLD_FRAMES}", pos=(10,60),
color=(200,200,200))
218                  if hold_counter >= HOLD_FRAMES and
len(sequence_buffer) >= 1:
219                      result =
predict_rf(sequence_buffer[-1])
220                      last_result_text = result
221                      last_result_source = "RF"
```

```
                        last_result_color = (0,0,255)
                        last_pred_time = time.time()
                        sequence_buffer.clear()
                        motion_scores.clear()
                        prev_landmarks = None
                        hold_counter = 0
                        draw_text_bg(image,
f"{last_result_text} (RF)", pos=(10,10),
color=last_result_color)

            cv2.imshow("Gesture Recognition", image)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    finally:
        cv2.destroyAllWindows()
        hands.close()
        picam2.stop()
```