

Analisis Perbandingan Performa Integrasi Model Deep Learning Berbasis TensorFlow Lite dan REST API untuk Klasifikasi Sampah pada Aplikasi Mobile Flutter

Sustri Elina Simamora^{*1}, Agung Setya Ariowirawan², Moch.Nurul Indra Al Fauzan³

^{1,2,3} Departemen Informatika, Fakultas Teknologi Industri dan Kreatif, Universitas Internasional Semen Indonesia
Email: ¹sustri.simamora23@student.ac.id, ²agung.ariowirawan23@student.ac.id, ³moch.fauzan@uisi.ac.id

Abstrak

Pengelolaan sampah secara efisien memerlukan sistem klasifikasi citra otomatis yang dapat diimplementasikan pada perangkat mobile. Penelitian ini membandingkan performa dua pendekatan integrasi model *deep learning* pada aplikasi Flutter bernama GreenShift: *on-device inference* menggunakan TensorFlow Lite (TFLite) dan *server-side inference* melalui REST API. Pengujian dilakukan terhadap 51 citra yang mencakup tiga kategori sampah (anorganik, organik, dan B3) menggunakan arsitektur model yang sama. Parameter yang diukur meliputi *inference time*, *preprocessing time*, *response time*, *total processing time*, *success rate*, dan konsistensi prediksi. Hasil pengujian menunjukkan bahwa TFLite mencapai rata-rata *total processing time* sebesar 217,26 ms, sedangkan REST API memerlukan rata-rata 1.002,53 ms. *Success rate* kedua pendekatan mencapai 100% dengan konsistensi prediksi 100% (51/51 citra), yang membuktikan bahwa proses kuantisasi TFLite tidak mengubah hasil klasifikasi. Rata-rata *inference time* TFLite sebesar 126,60 ms, lebih rendah 41,65% dibandingkan *inference* pada sisi server (216,98 ms). Secara keseluruhan, TFLite bekerja 4,6 kali lebih cepat dibandingkan REST API dalam *total processing time* karena eliminasi latensi jaringan yang mencapai rata-rata 774,94 ms pada jalur API. Penelitian menyimpulkan bahwa TFLite lebih unggul untuk skenario yang membutuhkan respons cepat dan kemampuan *offline*, sementara REST API lebih sesuai untuk fleksibilitas pembaruan model tanpa pembaruan aplikasi klien.

Kata kunci: Flutter, klasifikasi sampah, *machine learning*, REST API, TensorFlow Lite

Abstract

Efficient waste management requires an automated image classification system that can be implemented on mobile devices. This study compares the performance of two deep learning model integration approaches in a Flutter application named GreenShift: *on-device inference* using TensorFlow Lite (TFLite) and *server-side inference* via REST API. Testing was conducted on 51 images covering three waste categories (inorganic, organic, and hazardous/B3) using the same model architecture. Parameters measured include *inference time*, *preprocessing time*, *response time*, *total processing time*, *success rate*, and *prediction consistency*. The results showed that TFLite achieved an average *total processing time* of 217.26 ms, while the REST API required an average of 1,002.53 ms. The success rate for both approaches reached 100% with 100% prediction consistency (51/51 images), proving that the TFLite quantization process does not alter classification results. The average *inference time* for TFLite was 126.60 ms, which is 41.65% lower than *server-side inference* (216.98 ms). Overall, TFLite performs 4.6 times faster than the REST API in *total processing time* due to the elimination of network latency, which averaged 774.94 ms in the API path. The study concludes that TFLite is superior for scenarios requiring rapid response and *offline* capabilities, while REST API is more suitable for model update flexibility without client-side application updates.

Keywords: Flutter, machine learning, TensorFlow Lite, REST API, waste classification

I. PENDAHULUAN

Pengelolaan sampah merupakan salah satu tantangan lingkungan yang mendesak di era modern. Peningkatan volume sampah setiap harinya dipengaruhi oleh pertumbuhan populasi, urbanisasi, dan perubahan pola konsumsi masyarakat yang semakin bergantung pada produk sekali pakai. Permasalahan ini menimbulkan tantangan serius dalam proses pengelolaan dan pemilahan sampah, terutama pada tahap identifikasi jenis sampah yang masih banyak dilakukan secara manual. Metode manual dinilai kurang efisien, membutuhkan waktu yang lama, rentan terhadap kesalahan klasifikasi, serta berisiko terhadap kesehatan pekerja akibat kontak langsung dengan limbah. Oleh karena itu, diperlukan solusi otomatis yang mampu melakukan klasifikasi sampah secara cepat, akurat, dan dapat diimplementasikan secara luas pada perangkat yang mudah diakses masyarakat.

Perkembangan kecerdasan buatan, khususnya deep learning berbasis Convolutional Neural Network (CNN), telah menunjukkan performa yang sangat baik dalam tugas klasifikasi citra, termasuk pada domain pengelolaan sampah. Berbagai penelitian membuktikan bahwa model CNN mampu mengenali karakteristik visual sampah dengan tingkat akurasi yang tinggi. Sebuah studi menunjukkan bahwa optimasi model *deep learning* melalui teknik *quantization*, *pruning*, dan *knowledge distillation* mampu mengurangi ukuran model serta *operation delay* secara signifikan dengan penurunan akurasi yang minimal, sehingga cocok diterapkan pada perangkat *edge* dengan sumber daya terbatas[1]. Hal ini membuka peluang implementasi sistem klasifikasi sampah secara real-time pada perangkat mobile.

Salah satu arsitektur CNN yang banyak digunakan pada perangkat mobile adalah MobileNetV2. Arsitektur ini dirancang untuk menghasilkan performa klasifikasi yang baik dengan kebutuhan komputasi yang ringan. Penelitian oleh [2] membuktikan bahwa MobileNetV2 mampu mengklasifikasikan sampah rumah tangga ke dalam empat kategori dengan akurasi sebesar 82,92%, meningkat 15,42% dibandingkan CNN konvensional. Selain itu, model tersebut memiliki ukuran yang relatif kecil sehingga sesuai untuk implementasi pada smartphone. Penelitian lain [3] membandingkan beberapa arsitektur CNN untuk deteksi sampah berbasis smartphone dan menyimpulkan bahwa keluarga YOLO unggul dalam akurasi deteksi, processing time, serta memory *footprint* pada perangkat mobile. Temuan tersebut menunjukkan bahwa pemilihan metode deployment model menjadi faktor penting dalam menentukan performa aplikasi berbasis machine learning.

Dalam implementasi aplikasi mobile, integrasi model deep learning umumnya dilakukan melalui dua pendekatan utama, yaitu *on-device inference* dan *server-side inference*. Pendekatan *on-device inference* dilakukan dengan menjalankan model secara langsung pada perangkat menggunakan TensorFlow Lite (TFLite). Pendekatan ini memungkinkan proses prediksi dilakukan tanpa koneksi internet sehingga memiliki keunggulan dari sisi privasi data, latensi rendah, dan kemampuan offline. Penulis [4] telah mengimplementasikan pendekatan ini pada aplikasi Android klasifikasi sampah berbasis TFLite dan menunjukkan bahwa proses inference dapat dijalankan secara stabil pada perangkat mobile. Namun, performa pendekatan ini sangat dipengaruhi oleh spesifikasi perangkat dan teknik optimasi model yang digunakan.

Sebaliknya, pendekatan *server-side inference* menjalankan model pada server dan aplikasi mobile hanya bertindak sebagai client yang mengirimkan gambar melalui REST API untuk memperoleh hasil prediksi dalam format JSON. Pendekatan ini memiliki keunggulan dalam fleksibilitas pengelolaan model dan kemampuan menggunakan model berukuran besar tanpa membebani perangkat pengguna. Akan tetapi, performa sistem sangat dipengaruhi oleh kualitas jaringan dan keandalan layanan API. Parameter penting dalam pengujian *Representational State Transfer Application Programming Interface* (REST API) meliputi *response time*, *success rate*, *error rate*, dan reliabilitas format respons [5]. Evaluasi performa layanan REST API perlu dilakukan secara sistematis untuk memastikan kualitas layanan pada kondisi operasional nyata [6].

Perbandingan antara pendekatan *on-device* dan *server-side inference* menjadi isu krusial dalam pengembangan aplikasi *machine learning* berbasis *mobile*. Faktor kondisi jaringan, spesifikasi perangkat, dan metode kompresi model memiliki pengaruh signifikan terhadap *end-to-end latency* yang dirasakan pengguna [7]. Terdapat *trade-off* antara ukuran model, akurasi, dan *inference time* pada implementasi TensorFlow Lite menggunakan metode *post-training quantization* [8]. Oleh karena itu, pemilihan pendekatan *deployment* tidak hanya mempertimbangkan akurasi model, tetapi juga performa keseluruhan aplikasi dari sisi waktu proses, efisiensi sumber daya, dan pengalaman pengguna.

Dari sisi pengembangan aplikasi *mobile*, Flutter menjadi salah satu *framework* yang banyak digunakan karena mampu menghasilkan performa tinggi pada berbagai platform. Flutter terbukti memiliki *overhead* eksekusi dan konsumsi energi yang lebih rendah dibandingkan beberapa *framework cross-platform* lainnya pada skenario *CPU-intensive* [9]. Hal ini menjadikan Flutter relevan digunakan dalam pengembangan aplikasi klasifikasi sampah berbasis *machine learning* yang memerlukan efisiensi komputasi.

Berdasarkan penelitian-penelitian terdahulu, sebagian besar studi hanya berfokus pada pengembangan model klasifikasi sampah atau implementasi salah satu metode deployment secara terpisah. Penelitian yang secara khusus membandingkan performa integrasi TensorFlow Lite dan REST API pada aplikasi Flutter untuk klasifikasi sampah masih terbatas. Selain itu, belum banyak penelitian yang mengevaluasi kedua pendekatan menggunakan parameter teknis yang terukur secara komprehensif, seperti *inference time*, total processing time, response time, success rate, error rate, ukuran aplikasi, ketergantungan jaringan, dan kesesuaian hasil prediksi.

Oleh karena itu, penelitian ini bertujuan untuk: (1) mengimplementasikan model deep learning klasifikasi sampah menggunakan pendekatan TensorFlow Lite dan REST API pada aplikasi Flutter, (2) melakukan pengujian performa kedua pendekatan berdasarkan parameter teknis yang telah ditentukan, dan (3) memberikan rekomendasi deployment yang paling optimal untuk aplikasi klasifikasi sampah pada perangkat mobile berdasarkan hasil evaluasi empiris. Penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan sistem pengelolaan sampah berbasis kecerdasan buatan yang efisien, adaptif, dan sesuai untuk implementasi pada perangkat mobile.

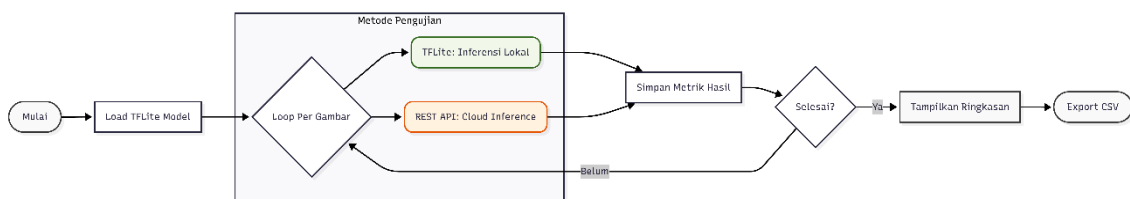
II. METODE PENELITIAN

2.1 Desain Penelitian

Penelitian ini menggunakan pendekatan *comparative study* antara dua arsitektur inferensi deep learning, yaitu (1) model Keras yang berjalan melalui REST API di server, dan (2) model TensorFlow Lite (TFLite) yang berjalan langsung di perangkat mobile. Pemilihan kedua pendekatan ini bertujuan membandingkan performa akurasi, latensi respons, dan kemandirian konektivitas jaringan [10], [11].

2.2 Arsitektur Sistem

Sistem yang dikembangkan penulis dalam penelitian ini bernama GreenShift, sebuah aplikasi mobile berbasis Flutter untuk klasifikasi sampah ke dalam tiga kategori yaitu organik, anorganik, dan B3 (Bahan Berbahaya dan Beracun).



Gambar 1. Arsitektur Alur Pengujian Performa Komparatif TFLite dan REST API

Arsitektur pengujian performa pada aplikasi GreenShift dirancang menggunakan pendekatan eksperimental sekuensial yang membandingkan dua paradigma komputasi berbeda, yaitu *edge computing* melalui TensorFlow Lite dan *cloud computing* melalui REST API, secara bersamaan untuk setiap sampel data. Alur kerja dimulai dengan inisialisasi model lokal, diikuti oleh mekanisme *looping* yang mengekstraksi aset gambar untuk diproses melalui dua jalur paralel: jalur pertama melakukan inferensi langsung di atas perangkat keras ponsel guna meminimalkan latensi jaringan, sementara jalur kedua melibatkan manajemen penyimpanan sementara serta transmisi data *multipart* melalui protokol HTTP ke server eksternal. Seluruh metrik performa yang dihasilkan, termasuk waktu inferensi dan respons jaringan, dikonsolidasikan ke dalam struktur data terpusat sebelum akhirnya diolah menjadi ringkasan statistik pada antarmuka pengguna dan diekspor ke dalam format CSV untuk kebutuhan analisis data penelitian yang lebih mendalam.

2.3 Dataset Pengujian

Dataset yang digunakan terdiri dari 51 gambar yang terdistribusi merata ke dalam tiga kelas:

Tabel 1 Pembagian Dataset

Kelas	Jumlah Gambar	Deskripsi
Organik	17	Sampah yang dapat terurai secara alami (sisa makanan, makanan busuk)
Anorganik	17	Sampah <i>non-biodegradable</i> (plastik, botol, kemasan)
B3	17	Bahan berbahaya dan beracun (baterai, elektronik, lampu)
Total	51	-

2.4 Parameter Pengujian

Parameter teknis yang diukur dalam penelitian ini didefinisikan sebagai berikut:

Tabel 2. Defenisi Operasional Parameter Pengujian Performa

Parameter	Defenisi	Satuan
<i>Inference Time</i>	Durasi model mengeksekusi komputasi neural network untuk menghasilkan klasifikasi.	ms
<i>Preprocessing Time</i>	Waktu untuk konversi citra, resizing, dan normalisasi byte sebelum masuk ke model.	ms
<i>Response Time</i>	Latensi jaringan yang diukur dari <i>MultipartRequest</i> dikirim hingga JSON diterima.	ms
<i>Total Processing Time</i>	Akumulasi waktu dari input gambar mentah hingga hasil ditampilkan pada <i>State</i> aplikasi	ms
<i>Success Rate</i>	Rasio antara jumlah gambar yang berhasil diprediksi dengan total 51 gambar uji.	%
<i>Confidence Score</i>	Nilai probabilitas (0 hingga 1) yang menunjukkan tingkat keyakinan model terhadap kelas tertentu	-

2.5 Alat dan Lingkungan Pengujian

Tabel 3. Lingkungan Pengujian Perangkat Keras dan Perangkat Lunak

Kategori	Komponen	Spesifikasi/Keterangan
Perangkat Lunak	<i>Framework Mobile</i>	Flutter SDK (Dart Language)
	<i>Machine Learning</i>	TensorFlow Lite
	<i>Backend API</i>	Flask Framework (Python)
	<i>Data Logging</i>	CSV <i>automated logging</i> via <i>path_provider</i>
Infrastruktur	Server	Localhost / Private Server (Flask)
	Koneksi Jaringan	WiFi IEEE 802.11 (Statis/Lokal)
Dataset	Sampel Uji	51 Citra Sampah (Organik, Anorganik, B3)
	Format Gambar	JPEG/JPG

Untuk menjaga validitas data pada parameter Response Time, seluruh pengujian dilakukan dalam satu sesi waktu yang sama dengan kondisi beban jaringan yang minim. Server backend dijalankan pada perangkat dengan spesifikasi tetap untuk memastikan bahwa variasi waktu pemrosesan pada REST API murni disebabkan oleh mekanisme *offloading* dan latensi jaringan, bukan karena fluktuasi beban prosesor server.

III. HASIL DAN PEMBAHASAN

3.1. Success Rate dan Keandalan Sistem

Pengujian dilakukan terhadap 51 gambar menggunakan kedua pendekatan secara terpisah. Hasil pengujian menunjukkan bahwa kedua pendekatan mencapai success rate 100% (51/51), artinya tidak terdapat satu pun request yang gagal dieksekusi, baik karena error model, error koneksi, maupun error preprocessing selama pengujian berlangsung.

Tabel 4. Metrik Keberhasilan Eksekusi Prediksi pada TFLite dan REST API

Parameter	TFLite	REST API
Total Data Uji	51	51
Prediksi Berhasil	51	51
Prediksi Gagal	0	0
Success Rate	100%	100%

Keberhasilan penuh pada REST API mengindikasikan stabilitas infrastruktur jaringan WiFi lokal yang digunakan. Hal ini sejalan dengan pernyataan [5] mengenai pengaruh stabilitas jaringan terhadap keandalan API. Namun, dalam kondisi lapangan dengan sinyal fluktuatif, *success rate* API berpotensi menurun, sementara TFLite akan tetap konsisten karena sifatnya yang *offline*.

3.2. Akurasi Prediksi Per Kelas

Model menghasilkan distribusi prediksi yang identik untuk kedua metode. Dengan membandingkan hasil terhadap *ground truth* (label asli), diperoleh tingkat akurasi sebesar 90,2%.

Tabel 5. Perbandingan Akurasi Klasifikasi Per Kategori Sampah

Kelas	Jumlah Sampel	Benar (TFLite/API)	Akurasi (%)
Organik	17	17	100,0%
Anorganik	17	14	82,4%
B3	17	15	88,2%

Total/Rata-rata	51	46	90,2%
-----------------	----	----	-------

Hasil yang identik membuktikan bahwa konversi model ke format .tflite tidak menurunkan kualitas ekstraksi fitur, konsisten dengan temuan [8].

3.3. Analisis Inference Time

Inference time mengukur waktu murni model dalam menghasilkan klasifikasi. Data menunjukkan keunggulan signifikan pada pemrosesan lokal.

Tabel 6. Statistik Perbandingan Inference Time (dalam milidetik)

Metode	Mean (ms)	Std.Dev	Min (ms)	Max (ms)
TFLite (Lokal)	126,60	16,83	101,18	192,81
REST API (Server)	216,98	24,06	181,67	269,79

TFLite 41,6% lebih cepat daripada REST API. Hal ini dikarenakan TFLite berjalan secara *native* dan memanfaatkan akselerasi perangkat keras ponsel, sementara REST API terbebani oleh *overhead* protokol HTTP dan eksekusi di lingkungan *framework* server.

3.4. Analisis Preprocessing Time

Preprocessing mencakup penyesuaian ukuran gambar dan normalisasi data sebelum masuk ke tahap inferensi.

Tabel 7. Perbandingan Waktu Preprocessing Citra

Metode	Mean (ms)	Std.Dev	Karakteristik Beban
TFLite	90,66	152,81	Beban penuh pada CPU ponsel
REST API	10,61	29,50	Sebagian besar dilakukan di server

REST API unggul 88,3% dalam kecepatan *preprocessing* di sisi klien karena aplikasi hanya melakukan *byte encoding*, sedangkan beban komputasi berat dipindahkan ke server. Pada TFLite, terjadi lonjakan waktu pada kategori B3 (std: 234,03 ms) yang mengindikasikan adanya beban memori saat menangani citra dengan resolusi tinggi.

3.5. Response Time Jaringan (REST API)

Response time mencatat waktu transmisi data dari ponsel ke server hingga menerima kembali hasil prediksi.

Tabel 8. Analisis Latensi Jaringan pada Metode REST API

Kategori	Mean (ms)	Std.Dev	Max (ms)
Organik	783,49	17,43	809,39
Anorganik	778,55	74,33	997,18
B3	762,79	223,39	1.554,88

Rata-rata waktu respons adalah 774,94 ms. Fluktuasi tertinggi (std 223,39 ms) pada kelas B3 dipengaruhi oleh ukuran *payload* gambar yang lebih besar, yang secara langsung memperlambat durasi *upload* sesuai dengan teori latensi *end-to-end* [7].

3.6. Total Processing Time dan Pengalaman Pengguna

Tabel 9. Perbandingan Total Processing Time

Kelas	API Mean (ms)	TFLite Mean (ms)	Rasio Performa
Organik	783,49	160,70	TFLite 4,88 x lebih cepat
Anorganik	778,55	171,26	TFLite 4,55 x lebih cepat
B3	762,79	320,10	TFLite 2,38 x lebih cepat
Rata-rata	774,94	217,35	TFLite 3,57 x lebih cepat

TFLite memberikan pengalaman pengguna yang jauh lebih responsif dengan total waktu rata-rata 217,35 ms, jauh di bawah ambang batas persepsi latensi manusia (400 ms), sementara REST API mendekati satu detik.

3.7. Rekapitulasi dan Pembahasan Komparatif

TFLite terbukti lebih unggul untuk penggunaan lapangan yang memerlukan respons cepat (real-time) dan kemandirian terhadap sinyal internet. Sebaliknya, REST API menawarkan kemudahan

pemeliharaan model (update di server tanpa rilis ulang aplikasi) dan efisiensi *preprocessing* bagi perangkat dengan spesifikasi rendah.

Tabel 10. Rekapitulasi Perbandingan Strategi Integrasi

Indikator Performa	TensorFlow Lite (Lokal)	REST API (Cloud)
Kecepatan Inferensi	Unggul (126,60 ms)	Lebih Lambat (216,98)
Total Waktu Tunggu	Sangat Cepat (217 ms)	Lambat (774 ms)
Ketertgantungan Data	Mandiri (Offline)	Kritis (Online)
Akurasi Hasil	Identik (90,2%)	Identik (90,2%)
Skalabilitas Model	Terbatas memori ponsel	Luas (Resource Server)

Pola kesalahan yang identik pada kedua metode mengonfirmasi bahwa limitasi akurasi terletak pada model AI itu sendiri, bukan pada cara integrasinya. TFLite sangat direkomendasikan untuk aplikasi klasifikasi sampah mobile karena memberikan kecepatan 3,57 kali lebih baik daripada pendekatan API.

IV. KESIMPULAN

Penelitian ini berhasil menyimpulkan bahwa integrasi model *machine learning* menggunakan TensorFlow Lite (TFLite) menunjukkan keunggulan performa yang sangat signifikan dibandingkan dengan pendekatan REST API dalam konteks klasifikasi sampah pada aplikasi mobile Flutter. Dari sisi efisiensi waktu, TFLite terbukti jauh lebih responsif dengan rata-rata total processing time sebesar 217,26 ms, sementara REST API memerlukan waktu rata-rata hingga 1002,53 ms. Hal ini menunjukkan bahwa secara keseluruhan, pendekatan *on-device inference* melalui TFLite mampu bekerja sekitar 4,6 kali lebih cepat dibandingkan dengan mengandalkan server eksternal melalui jalur REST API. Keunggulan ini juga terlihat pada *inference time* murni, di mana TFLite mencatatkan waktu 126,60 ms, atau sekitar 41,65% lebih rendah dibandingkan inferensi pada sisi server yang mencapai 216,98 ms.

Ditinjau dari aspek reliabilitas dan kualitas prediksi, kedua metode menunjukkan hasil yang setara dengan capaian *success rate* sebesar 100%. Pengujian terhadap 51 gambar uji dari kategori sampah anorganik, organik, dan B3 menunjukkan bahwa tidak ditemukan kegagalan sistem pada kedua jalur integrasi tersebut. Lebih lanjut, penelitian ini mengonfirmasi adanya konsistensi prediksi sebesar 100%, di mana label klasifikasi yang dihasilkan oleh TFLite dan REST API selalu identik untuk setiap gambar yang diuji. Hal ini membuktikan bahwa proses kuantisasi pada model TFLite tidak mengurangi akurasi hasil prediksi dibandingkan model aslinya yang berjalan di server, dengan tingkat akurasi sistem mencapai 90,2%.

Analisis mendalam mengenai struktur waktu pemrosesan mengungkap bahwa hambatan utama (*bottleneck*) pada pendekatan REST API bukan terletak pada proses komputasi model di server, melainkan pada latensi jaringan. Komponen *network response time* pada REST API mendominasi waktu pemrosesan dengan rata-rata 774,94 ms. Sebaliknya, pada TFLite, seluruh proses dilakukan secara lokal sehingga variabel ketergantungan terhadap stabilitas koneksi internet dapat dieliminasi sepenuhnya. Temuan ini menegaskan bahwa kualitas infrastruktur jaringan menjadi faktor penentu utama jika pengembang memilih menggunakan REST API.

Sebagai rekomendasi akhir, penelitian ini menyimpulkan bahwa TFLite merupakan pilihan terbaik untuk skenario aplikasi yang membutuhkan respons *real-time*, kemampuan beroperasi secara *offline*, serta efisiensi latensi yang tinggi. Namun, REST API tetap memiliki relevansi untuk kebutuhan di mana pembaruan model perlu dilakukan secara berkala tanpa harus memperbarui aplikasi di sisi pengguna. Untuk pengembangan di masa depan, disarankan untuk mengeksplorasi perbandingan pada berbagai kondisi jaringan seluler serta melakukan evaluasi terhadap dampak konsumsi daya baterai pada perangkat mobile.

REFERENSI

- [1] H. E. Kim, M. E. Maros, F. Siegel, and T. Ganslandt, "Rapid Convolutional Neural Networks for Gram-Stained Image Classification at Inference Time on Mobile Devices : Empirical Study from Transfer Learning to Optimization," no. D1, 2022.
- [2] L. Yong, L. Ma, D. Sun, and L. Du Id, "Application of MobileNetV2 to waste classification," pp. 1–16, 2023, doi: 10.1371/journal.pone.0282336.
- [3] M. Córdova *et al.*, "Litter Detection with Deep Learning : A Comparative Study," pp. 1–19, 2022.
- [4] T. Handhayani and J. Hendryli, "Leboh: An Android Mobile Application for Waste Classification Using TensorFlow Lite," in *Intelligent Systems and Applications*, K. Arai, Ed., Cham: Springer International Publishing, 2023, pp. 53–67.
- [5] A. Ehsan, M. A. M. E. Abuhaliqa, and C. Catal, "applied sciences RESTful API Testing Methodologies : Rationale , Challenges , and Solution Directions," 2022.
- [6] A. Golmohammadi, M. A. N. Zhang, and A. Arcuri, "Testing RESTful APIs : A Survey," vol. 33, no. 1, pp.

-
- 1–41, 2023.
- [7] S. S. Ogden and T. Guo, “Characterizing the Deep Neural Networks Inference Performance of Mobile Applications,” 2019.
- [8] I. L. Orășan, C. Seiculescu, and C. D. Căleanu, “Benchmarking TensorFlow Lite Quantization Algorithms for Deep Neural Networks,” in *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2022, pp. 221–226. doi: 10.1109/SACI55618.2022.9919465.
- [9] B. Moraes, F. Castor, and J. P. Fernandes, *Analyzing the Resource Usage Overhead of Mobile App Development Frameworks*, vol. 1, no. 1. Association for Computing Machinery. doi: 10.1145/3593434.3593487.
- [10] K. He and J. Sun, “Deep Residual Learning for Image Recognition,” pp. 1–9.
- [11] M. Sandler, A. Howard, M. Zhu, and A. Zhmoginov, “MobileNetV2 : Inverted Residuals and Linear Bottlenecks,” pp. 4510–4520.