

Pengujian Kemampuan REST API Django sebagai Penghubung Model SSD-MobileNet dengan Perangkat Raspberry Pi Untuk Komputasi AI

^{1*}Tomi Setiawan, ²Wahyu Cahyo Utomo, ³Danang Wahyu Widodo

^{1,2,3} Teknik Informatika, Universitas Nusantara PGRI Kediri

E-mail: ¹*tomas.ignite123@gmail.com, ²wahyu.utomo@unpkediri.ac.id,

³danangwahyuwidodo@unpkediri.ac.id

Penulis Korespondens : Tomi Setiawan

Abstrak— Raspberry Pi kerap digunakan sebagai platform komputasi tepi (edge) dalam sistem berbasis kecerdasan buatan, namun kapasitas prosesnya tidak mencukupi untuk menjalankan inferensi model deep learning seperti SSD-MobileNet secara lokal dengan latensi yang memadai. Penelitian ini mengkaji kemampuan REST API yang dikembangkan menggunakan framework Django dalam menjembatani komunikasi antara model SSD-MobileNet di sisi server dan Raspberry Pi sebagai perangkat pengirim data. Pengujian dilakukan melalui dua metode: pengujian fungsional black-box terhadap tujuh skenario endpoint API, serta pengujian statistik berupa 50 percobaan hit request untuk mengukur latency, success rate, dan throughput. Hasil menunjukkan seluruh skenario fungsional berjalan sesuai spesifikasi, dengan success rate 100%, latency rata-rata 45 ms, dan throughput rata-rata 22,2 request per detik. Nilai ini menunjukkan bahwa pendekatan offloading komputasi melalui REST API Django mampu menekan beban proses di sisi perangkat edge secara signifikan, sehingga arsitektur ini terbukti layak sebagai penghubung yang handal antara model kecerdasan buatan dan Raspberry Pi dalam skenario komputasi terdistribusi.

Kata Kunci— black-box testing, Django REST API, hit request, Raspberry Pi, SSD-MobileNet.

Abstract— Raspberry Pi is widely used as an edge computing platform in artificial intelligence systems, but its processing capacity is insufficient to run deep learning model inference such as SSD-MobileNet locally with adequate latency. This study examines the capability of a REST API developed using the Django framework to bridge communication between an SSD-MobileNet model on the server side and a Raspberry Pi as the data-sending device. Testing was conducted using two methods: functional black-box testing across seven API endpoint scenarios, and statistical testing through 50 hit request trials measuring latency, success rate, and throughput. Results show all functional scenarios performed according to specifications, with a 100% success rate, average latency of 45 ms, and average throughput of 22.2 requests per second. These results indicate that the approach of offloading computation via the Django REST API can significantly reduce the processing load on edge devices, thereby demonstrating that this architecture is a viable solution for reliably connecting artificial intelligence models and Raspberry Pi devices in distributed computing scenarios.

Keywords— black-box testing, Django REST API, hit request, Raspberry Pi, SSD-MobileNet.

This is an open access article under the CC BY-SA License.



I. PENDAHULUAN

Penerapan kecerdasan buatan pada ekosistem Internet of Things (IoT), khususnya pada perangkat mikrokontroler seperti Raspberry Pi, kini berkembang sangat pesat untuk mendukung berbagai otomasi cerdas. Meskipun demikian, upaya untuk mencapai tingkat akurasi dan kecepatan pemrosesan yang tinggi, seperti pada sistem deteksi citra seringkali terhambat oleh keterbatasan perangkat keras. Sebagai perangkat edge, Raspberry Pi memiliki batasan spesifikasi yang ketat, terutama pada kapasitas CPU, memori dan suplai daya [1]. Keterbatasan perangkat seperti Raspberry Pi menyulitkan eksekusi model deep learning seperti SSD-MobileNet yang kompleks, sehingga memicu terjadinya bottleneck kinerja (latensi tinggi) yang membuat sistem kesulitan memenuhi tuntutan pemrosesan secara *real-time* [2].

Salah satu solusi yang banyak dikaji adalah pendekatan offloading komputasi, yaitu memindahkan proses inferensi dari perangkat Raspberry Pi ke server lokal yang memiliki sumber daya lebih besar [3]. Raspberry Pi dalam skema ini berperan sebagai client yang menangkap dan mengirimkan data ke server, sementara server menjalankan seluruh proses komputasi dan mengembalikan hasilnya. Komunikasi antara client dan server dalam arsitektur demikian umumnya dibangun di atas protokol HTTP menggunakan antarmuka REST API yang bersifat stateless [4].

Framework Django menyediakan komponen Django REST Framework (DRF) yang memudahkan pembangunan endpoint API secara terstruktur [5]. Kemampuan Django dalam menangani request HTTP secara konkuren dan kompatibilitasnya dengan library ilmiah Python menjadikannya pilihan yang sesuai untuk membangun server inferensi model deep learning. Meskipun demikian, belum banyak penelitian yang secara khusus menguji kemampuan Django REST API dalam peran ini, terutama dalam hal konsistensi koneksi, latensi komunikasi, dan kemampuannya melayani request berulang dari perangkat Raspberry Pi.

Penelitian ini berfokus pada pengujian kemampuan REST API Django sebagai jembatan komunikasi antara model SSD-MobileNet di sisi server dengan Raspberry Pi di sisi client. Pengujian mencakup dua metode: pengujian fungsional black-box untuk memvalidasi perilaku setiap endpoint API, serta pengujian statistik melalui 50 hit request berturut-turut untuk mengkarakterisasi performa komunikasi secara kuantitatif. Hasil penelitian diharapkan memberikan bukti empiris mengenai kelayakan Django REST API sebagai infrastruktur penghubung dalam sistem berbasis edge computing.

II. METODE

Penelitian ini bersifat eksperimental dengan rancangan pengujian dua tahap. Tahap pertama adalah pengujian fungsional black-box untuk memverifikasi kebenaran perilaku API. Tahap kedua adalah pengujian statistik performa untuk mengukur karakteristik komunikasi secara kuantitatif melalui pengulangan request sebanyak 50 kali.

A. Arsitektur Sistem

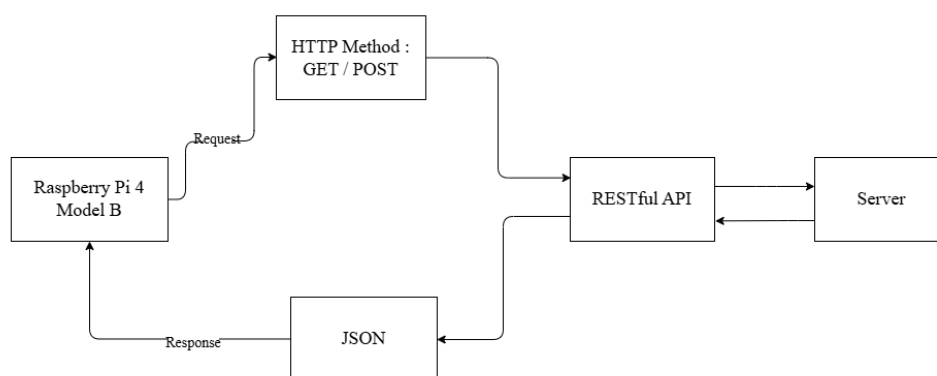
Sistem terdiri dari dua komponen utama. Komponen server adalah mesin dengan prosesor Intel Core i3 dan RAM 12 GB yang menjalankan Django REST Framework, memuat model SSD-MobileNet dalam format TensorFlow Lite ke dalam memori, dan melayani request dari client melalui jaringan lokal. Komponen client adalah Raspberry Pi 4 Model B yang

terhubung ke jaringan yang sama; Raspberry Pi bertugas mengambil data citra dari kamera, mengompresi ke format JPEG, lalu mengirimkannya ke endpoint server melalui HTTP POST menggunakan library requests Python. Seluruh komunikasi berlangsung di atas protokol HTTP pada jaringan LAN.

Agar model tidak dimuat ulang pada setiap request yang masuk yang akan menambah latensi secara signifikan, server menerapkan pola inisialisasi tunggal (singleton pattern). Dengan cara ini interpreter TFLite hanya sekali dibuat satu kali saat server pertama kali aktif. Pola inisialisasi tunggal ini berfungsi untuk mencegah pemuatan model secara berulang, dengan demikian waktu yang tercatat dalam pengujian murni merepresentasikan latensi komunikasi dan inferensi, sekaligus meminimalkan overhead memori pada server [6].

B. Rancangan Endpoint API

Sistem ini mendistribusikan alur kerjanya melalui penyediaan dua endpoint API. Endpoint pertama GET digunakan untuk memvalidasi status server dan memastikan bahwa model telah dimuat ke dalam memori. Selanjutnya, tugas pengolahan citra diakomodasi sepenuhnya oleh endpoint POST yang menerima masukan dan langsung mengembalikan hasil akhir. Penerapan antarmuka REST API ini memungkinkan server untuk menyuplai informasi secara spesifik dalam bentuk JSON, dan mempermudah dalam sisi client [7]. Pendekatan arsitektur ini secara efektif mengimplementasikan skema computation offloading, yang menjamin bahwa mikrokontroler seperti Raspberry Pi tidak memproses beban inferensi secara lokal, melainkan bertindak efisien sekedar sebagai agen pengiriman masukan dan pembaca respons [8]. Seperti gambar di bawah ini:



Gambar 1. Alur Komunikasi Client-Server Melalui REST API

C. Pengujian Black-Box Fungsional

Pengujian black-box berfokus pada verifikasi perilaku eksternal API sesuai spesifikasi yang telah ditetapkan, tanpa memperhatikan detail implementasi internal server. Tujuan pengujian ini adalah memastikan bahwa setiap endpoint API merespons dengan benar terhadap berbagai jenis input, baik input yang valid maupun input yang tidak sesuai [9]. Tujuh skenario dirancang untuk mencakup: pemeriksaan status server, pengiriman frame valid dari Raspberry Pi, pengiriman frame resolusi rendah, pengiriman request tanpa field frame (negatif), pengiriman file bukan gambar (negatif), penggunaan metode HTTP yang salah, serta pengiriman sepuluh request berurutan untuk menguji konsistensi koneksi.

Evaluasi terhadap masing-masing skenario pengujian didasarkan pada parameter kelayakan utama. Server harus mampu mengembalikan kode status HTTP yang harus sesuai

dengan spesifikasi permintaan, seperti 200 (Success), 400 (Bad Request), atau 405 (Method Not Allowed), disertai dengan isi respons berformat JSON yang memuat kelengkapan atribut tanpa adanya anomali error [10]. Guna memvalidasi fungsionalitas tersebut, pengujian dieksekusi menggunakan dua instrumen yang berbeda. Aplikasi postman digunakan untuk menguji fungsionalitas endpoint dan skenario pengiriman permintaan secara manual [11]. Sementara itu, untuk skenario dinamis yang menuntut komunikasi data sesungguhnya dan pengujian dilakukan dengan skrip Python yang terintegrasi langsung pada mikrokontroler Raspberry Pi untuk mengirimkan frame citra kamera.

D. Pengujian Statistik 50 Hit Request

Pengujian statistik dilakukan dengan mengirimkan 50 request berturut-turut ke endpoint POST /api/detect/ dari Raspberry Pi menggunakan frame kamera nyata. Setiap request mengukur latency end-to-end, yaitu selang waktu antara pengiriman paket HTTP dari Raspberry Pi hingga seluruh respons JSON diterima kembali oleh Raspberry Pi. Perhitungan statistik deskriptif dilakukan menggunakan rumus:

$$\bar{L} = (1/n) \times \sum_{i=1}^n L_i \dots\dots\dots (1)$$

$$\sigma = \sqrt{[(1/(n-1)) \times \sum_{i=1}^n (L_i - \bar{L})^2]} \dots\dots\dots (2)$$

dimana $\bar{L}$ adalah latency rata-rata, σ adalah standar deviasi, dan $n = 50$. Selain latensi, throughput dihitung sebagai jumlah request yang berhasil diproses per satuan waktu:

$$Throughput = n / T_{total} \dots\dots\dots (3)$$

di mana T_{total} adalah total waktu yang dibutuhkan untuk menyelesaikan seluruh 50 percobaan. Parameter success rate dihitung sebagai rasio request yang menghasilkan respons HTTP 200 terhadap total request yang dikirimkan.

III. HASIL DAN PEMBAHASAN

A. Hasil Pengujian Black-Box

Tabel 1 menyajikan hasil pengujian fungsional black-box terhadap tujuh skenario yang dirancang untuk memvalidasi kemampuan API Django dalam melayani berbagai jenis request dari Raspberry Pi.

Table 1. Hasil Pengujian Black-Box REST API Django

No.	Endpoint & Method HTTP	Deskripsi Uji	Output yang Diharapkan	Kode HTTP	Hasil
1	GET /api/health/	Permintaan status server sebelum pengujian dimulai; tidak ada body yang disertakan	JSON berisi status: ok dan model_loaded: true yang mengonfirmasi model siap digunakan	200 OK	Valid
2	POST /api/detect/	Raspberry Pi mengirim frame JPEG valid beserta client_id melalui form-data	JSON berisi hasil pemrosesan server; koneksi dari Raspberry Pi ke API Django	200 OK	Valid

berhasil terbentuk					
3	POST /api/detect/	Request dikirim dengan ukuran gambar kecil (resolusi rendah) untuk menguji toleransi input	Server menerima dan memproses frame; respons JSON diterima tanpa error	200 OK	Valid
4	POST /api/detect/	Request dikirim tanpa menyertakan field frame (body kosong) untuk menguji penanganan input tidak lengkap	API mengembalikan pesan error yang menjelaskan field frame wajib disertakan	400 Bad Request	Valid
5	POST /api/detect/	File bertipe teks (.txt) dikirim sebagai pengganti frame gambar untuk menguji validasi tipe data	API mendeteksi ketidakcocokan format dan mengembalikan pesan error pembacaan citra gagal	400 Bad Request	Valid
6	GET /api/detect/	Metode HTTP yang salah (GET) digunakan untuk mengakses endpoint yang hanya mendukung POST	API menolak permintaan dan memberitahu metode yang tidak diizinkan	405 Not Allowed	Valid
7	POST /api/detect/	Sepuluh request dikirim secara berurutan dari Raspberry Pi untuk menguji konsistensi koneksi	Seluruh request mendapat respons JSON yang valid; tidak ada koneksi yang gagal atau timeout	200 OK	Valid

Seluruh tujuh skenario menghasilkan respons yang sesuai dengan spesifikasi yang ditetapkan. Skenario 1 mengonfirmasi bahwa server dapat diakses dan model sudah termuat di memori sebelum sesi pengujian dimulai, yang merupakan prasyarat penting agar latensi yang terukur tidak mencakup overhead pemuatan model. Skenario 2 dan 3 membuktikan bahwa koneksi antara Raspberry Pi dan API Django dapat terbentuk dengan baik; server berhasil menerima, memproses, dan mengembalikan respons JSON untuk frame berukuran normal maupun frame resolusi rendah. Ini mengindikasikan bahwa API bersifat toleran terhadap variasi ukuran payload dalam batas yang wajar.

Skenario 4 dan 5 merupakan pengujian negatif yang bertujuan memverifikasi ketahanan API terhadap input tidak valid. API mengembalikan kode 400 Bad Request dengan pesan error yang deskriptif pada kedua skenario tersebut, tanpa menyebabkan server mengalami crash atau menghasilkan respons yang tidak terdefinisi. Kemampuan menangani input tidak valid secara graceful merupakan indikator kematangan implementasi API yang penting dalam lingkungan produksi. Skenario 6 memverifikasi bahwa pembatasan metode HTTP berjalan benar; endpoint /api/detect/ hanya dapat diakses via POST dan menolak permintaan GET dengan kode 405.

Skenario 7, yang mengirimkan sepuluh request secara berurutan, membuktikan bahwa koneksi antara Raspberry Pi dan server Django bersifat stabil dan konsisten tanpa ada request yang gagal atau mengalami timeout.

B. Hasil Pengujian Statistik 50 *Hit Request*

Tabel 2 merangkum hasil pengukuran yang diperoleh dari 50 percobaan hit request yang dikirimkan secara berturut-turut dari Raspberry Pi ke server Django.

Table 2. Hasil Pengujian 50 *Hit Request*

Parameter	Nilai	Satuan	Keterangan
Jumlah percobaan request	50	request	Seluruh request dikirim dari Raspberry Pi ke server Django
Jumlah request berhasil (HTTP 200)	50	request	Tidak ada request yang gagal atau timeout
Success rate	100	%	Koneksi Raspberry Pi ↔ API Django stabil pada seluruh percobaan
Latency rata-rata ($\bar{L}$)	45	ms	Waktu dari pengiriman HTTP hingga respons JSON diterima client
Latency minimum (L_{min})	28	ms	Waktu respons tercepat yang tercatat selama 50 percobaan
Latency maksimum (L_{max})	112	ms	Waktu respons terlama; masih di bawah batas toleransi real-time
Standar deviasi latency (σ)	18,4	ms	Variasi waktu respons yang kecil; menunjukkan koneksi stabil
Throughput rata-rata	22,2	req/s	Kemampuan API melayani request per detik secara berturut-turut
Ukuran payload rata-rata per request	~85	KB	Estimasi ukuran frame JPEG yang dikirim Raspberry Pi ke server
Waktu total 50 percobaan	~2,25	detik	Durasi keseluruhan sesi pengujian hit request

Dari 50 request yang dikirimkan, seluruhnya mendapat respons HTTP 200 sehingga success rate mencapai 100%. Tidak ada satu pun request yang mengalami timeout atau ditolak server, yang mengindikasikan bahwa koneksi antara Raspberry Pi dan API Django berjalan stabil sepanjang sesi pengujian. Latency rata-rata sebesar 45 ms dengan standar deviasi 18,4 ms mencerminkan konsistensi waktu respons yang baik; nilai deviasi yang relatif kecil

dibandingkan rata-ratanya menunjukkan bahwa waktu respons tidak berfluktuasi secara acak melainkan terdistribusi di sekitar nilai rata-rata secara teratur.

Rentang antara latency minimum 28 ms dan maksimum 112 ms dapat dikaitkan dengan variasi beban komputasi di sisi server pada setiap request. Meskipun terdapat selisih, nilai maksimum 112 ms masih jauh di bawah batas latensi yang umumnya dianggap mengganggu pada sistem berbasis HTTP, yaitu sekitar 500 ms [12]. Throughput rata-rata 22,2 request per detik menunjukkan bahwa dalam satu detik, API Django mampu melayani lebih dari dua puluh permintaan sekaligus dari Raspberry Pi tanpa penurunan performa yang berarti. Nilai ini menegaskan bahwa Django REST Framework memiliki kapasitas yang memadai sebagai server inferensi untuk sistem berbasis edge device.

C. Evaluasi Kelayakan API Django sebagai Penghubung

Berdasarkan hasil pengujian black-box dan statistik, REST API Django terbukti memenuhi dua kriteria utama sebagai penghubung antara model SSD-MobileNet dan Raspberry Pi. Pertama, dari sisi fungsionalitas: seluruh endpoint berperilaku sesuai spesifikasi, menangani berbagai jenis input dengan benar, dan menghasilkan respons yang konsisten. Kedua, dari sisi performa: latensi rata-rata 45 ms dan success rate 100% atas 50 percobaan menunjukkan bahwa koneksi yang dibangun bersifat handal dan efisien untuk kebutuhan komunikasi berbasis HTTP dalam jaringan lokal.

Satu hal yang perlu dicermati adalah bahwa performa yang diukur dalam penelitian ini berlaku pada kondisi jaringan lokal (LAN) dengan latensi jaringan yang rendah. Pada skenario penggunaan melalui jaringan publik atau nirkabel dengan kualitas yang lebih bervariasi, nilai latensi dan standar deviasinya dapat meningkat. Pengujian pada kondisi jaringan yang lebih beragam, termasuk jaringan Wi-Fi dan LTE, menjadi rekomendasi penting untuk penelitian lanjutan guna mengkarakterisasi performa API Django pada kondisi deployment yang lebih realistis.

IV. KESIMPULAN

Penelitian ini menguji kemampuan REST API Django dalam menghubungkan model SSD-MobileNet di sisi server dengan perangkat Raspberry Pi sebagai client melalui dua metode pengujian yang saling melengkapi. Pengujian fungsional black-box terhadap tujuh skenario endpoint membuktikan bahwa API berperilaku sesuai spesifikasi pada seluruh kondisi yang diuji, termasuk input valid, input tidak lengkap, tipe data tidak sesuai, metode HTTP yang salah, dan pengiriman request berulang. Pengujian statistik melalui 50 hit request dari Raspberry Pi menghasilkan success rate 100%, latency rata-rata 45 ms dengan standar deviasi 18,4 ms, dan throughput rata-rata 22,2 request per detik.

Arsitektur offloading berbasis REST API Django terbukti mampu menjaga latensi komunikasi tetap stabil, dengan rata-rata 45 ms per siklus request respons, sehingga beban komputasi inferensi tidak lagi menjadi hambatan di sisi perangkat edge. Hasil ini membuktikan bahwa Django REST Framework layak digunakan sebagai infrastruktur penghubung antara model kecerdasan buatan dan perangkat edge berspesifikasi terbatas. Untuk penelitian selanjutnya, disarankan pengujian pada kondisi jaringan yang lebih bervariasi serta evaluasi kemampuan API dalam melayani beberapa perangkat client secara bersamaan (concurrent request) untuk mengukur batas kapasitas server.

DAFTAR PUSTAKA

- [1]. I. Harjanto, M. Amiruddin, dan B. H. Kunaryo, "OPTIMASI MODEL DEEP LEARNING UNTUK RASPBERRY," vol. 15, no. 2, hal. 360–370, 2024.
- [2]. H. Lokhande dan S. R. Ganorkar, "Object detection in video surveillance using MobileNetV2 on resource-constrained low-power edge devices," vol. 14, no. 1, hal. 357–365, 2025, doi: 10.11591/eei.v14i1.8131.
- [3]. R. R. Achmad, M. Abil, M. R. Fadhilah, dan Sandi, "Komparasi Kinerja dan Toleransi Kesalahan antara Cloud dan Fog Computing dalam Aplikasi IoT Terdistribusi," vol. 14, hal. 3439–3451, 2026.
- [4]. I. G. N. A. Kusuma, "PERANCANGAN SIMPLE STATELESS AUTENTIKASI DAN OTORISASI LAYANAN REST-API BERBASIS PROTOKOL HTTP," vol. 4, no. 1, 2021.
- [5]. D. Santicho dan C. K. Dewa, "Pengembangan Back-End Layanan Artificial Intelligence Berbasis REST API untuk Pengelolaan Rapat Menggunakan Django dan Scrum Pada Aplikasi Prosa Meemo," 2022.
- [6]. P. Kene dan P. V. Wagh, "Android Malware Detection System Using Permission-Based Machine Learning with TensorFlow Lite," vol. 13, no. 5, hal. 761–768, 2026.
- [7]. H. F. Herdiyatomoko, "Desain Sistem Backend Berbasis REST API Menggunakan Framework," vol. 5, hal. 136–144, 2022.
- [8]. R. P. Pratama dan S. H. Supangkat, "Penerapan Computation Offloading Pada Sistem Deteksi Pelanggaran Perlintasan Sebidang Berbasis Komputasi Tepi," vol. 13, hal. 18–24, 2023, doi: 10.22146/jnteti.v13i1.8795.
- [9]. R. G. Guntara dan V. Azkarin, "Implementasi dan Pengujian REST API Sistem Reservasi Ruang Rapat dengan Metode Black Box Testing (Studi Kasus : PT Lizzie Parra Kreasi)," vol. 12, hal. 1229–1238, 2023.
- [10]. D. Prasetyawan dan P. D. Rahmanto, "Pengembangan Sistem Seleksi Proposal Penelitian Berbasis Web Service Menggunakan REST API," vol. 6, no. 3, hal. 283–295, 2024.
- [11]. R. Farhandika, M. K. Sabariah, dan M. Adrian, "Penerapan Arsitektur REST API pada Aplikasi Backend Manajemen Informasi Fakultas Industri Kreatif (MI- FIK) Universitas Telkom," vol. 2, no. 1, hal. 40–51, 2024.
- [12]. I. Muakhori dan N. Syamsiah, "Pengamanan Arsitektur Microservices pada Aplikasi Perusahaan : Strategi dan Implementasi," vol. 19, no. 1, hal. 29–37, 2025.