Terbit *online* pada laman : http://teknosi.fti.unand.ac.id/

Jurnal Nasional Teknologi dan Sistem Informasi

| ISSN (Print) 2460-3465 | ISSN (Online) 2476-8812 |

Artikel Penelitian

Kajian Performa Efisiensi Infrastruktur Big Data Hemat Energi Menggunakan *Single Board Computer* dan *Framework* Apache Spark

Syahel Rusfi Razaba [a], *Muhaza Liebenlito* [b,*], *Taufik Edy Sutanto* [c]

[a,b,c]*Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah Jakarta*

INFORMASI ARTIKEL

Sejarah Artikel:
Diterima Redaksi: 07 Juli 2025
Revisi Akhir: 27 Agustus 2025
Diterbitkan *Online*: 07 September 2025

KATA KUNCI

Apache Spark,
Big Data,
Green AI,
Klaster SBC,
Machine Learning

KORESPONDENSI

E-mail: muhazaliebenlito@uinjkt.ac.id*

A B S T R A C T

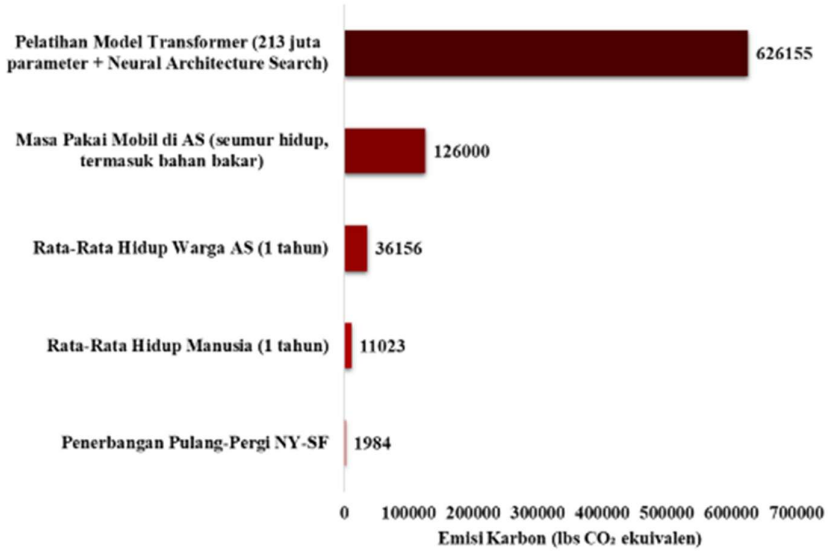
Meningkatnya kebutuhan komputasi untuk pemrosesan *big data* dan pelatihan model AI modern berdampak signifikan terhadap konsumsi energi komputasi global. Penelitian ini mengkaji efisiensi energi dan performa klaster *Single Board Computer* (SBC) dalam menjalankan beberapa algoritma *machine learning* menggunakan Apache Spark, sebagai alternatif ramah lingkungan terhadap infrastruktur komputasi konvensional. Tiga algoritma digunakan dalam eksperimen ini, yaitu Multi-Layer Perceptron (MLP), Regresi Logistik, dan Random Forest, yang dijalankan secara terdistribusi pada klaster SBC. Evaluasi dilakukan terhadap dua metrik utama, yaitu waktu eksekusi dan konsumsi energi, dengan tiga skenario ukuran dataset dan lima variasi jumlah inti (*core*). Hasil menunjukkan bahwa klaster SBC mampu mencapai percepatan waktu pelatihan hingga 59.7% pada algoritma Multi-Layer Perceptron dan hingga 49.3% pada Random Forest saat menangani data berukuran besar. Konsumsi daya listrik juga tetap rendah dan stabil, yakni sekitar 11.4 *watt* untuk konfigurasi satu core dan 12.6 *watt* untuk konfigurasi *multi-core*. Temuan ini menegaskan bahwa penggunaan klaster SBC berdaya rendah merupakan pendekatan potensial untuk mendukung komputasi hemat energi dan inisiatif *Green AI*.

1. PENDAHULUAN

Pelatihan model AI (*Artificial Intelligence*) yang populer saat ini, seperti GPT-3, Gopher, dan Open Pre-trained Transformer (OPT), masing-masing tercatat mengonsumsi daya sebesar 1.287, 1.066, dan 324 MWh (*Megawatt-hour*), dengan setiap model dilatih menggunakan data berukuran hingga *terabyte* dan sekitar 175 miliar atau lebih parameter [1]. Hal ini menunjukkan bahwa model AI modern, terutama yang menggunakan *deep learning*, memerlukan sumber daya komputasi yang sangat besar tidak hanya untuk proses pelatihan, tetapi juga dalam penggunaannya [2]. Penggunaan kerangka kerja (*framework*) *big data* seperti Apache Spark dapat dioptimalkan menggunakan perangkat komputasi berdaya rendah, seperti klaster SBC, sebagai pendekatan alternatif untuk menekan konsumsi energi dalam pelatihan model *machine learning*.

Konsumsi energi yang besar dalam komputasi AI, baik saat melatih model maupun dalam penggunaannya, turut menyumbang pada peningkatan emisi gas rumah kaca [3], [4], [5]. Penelitian terkini menunjukkan bahwa energi yang digunakan untuk melatih satu model AI canggih dapat menghasilkan emisi gas rumah kaca dengan dampak setara lebih dari 500 ton karbon dioksida, yang menyoroti tingginya biaya lingkungan dalam pengembangan teknologi AI [6]. Sebagai contoh, emisi yang dihasilkan oleh GPT-2 tercatat lima kali lipat lebih besar dibandingkan total emisi yang dihasilkan oleh sebuah mobil sepanjang masa penggunaannya [7].

https://doi.org/10.25077/TEKNOSI.v11i2.2025.152-160



Gambar 1. Perbandingan Emisi Karbon dari Berbagai Aktivitas [7].

Kekhawatiran tentang kebutuhan energi dan dampak lingkungan yang mungkin ditimbulkan oleh komputasi AI, telah meningkatkan minat pada konsep ”Green AI”, yaitu penggunaan teknologi AI dengan fokus pada efisiensi energi, pengurangan emisi karbon dioksida, dan keberlanjutan lingkungan [8]. Konsep ”Green AI” memiliki relevansi yang signifikan karena selaras dengan upaya keberlanjutan global, sebagaimana yang diutarakan oleh Perserikatan Bangsa-Bangsa (PBB) dalam Tujuan Pembangunan Berkelanjutan atau yang dikenal dengan *Sustainable Development Goals* (SDGs) [9], [10]. Oleh karena itu, berbagai upaya sedang dilakukan untuk memenuhi permintaan daya pemrosesan yang meningkat tanpa mengonsumsi banyak energi, dan akhirnya para peneliti mencoba menerapkan klaster komputer berbasis perangkat keras berbiaya rendah dan hemat energi [11].

Penggunaan klaster komputer berbiaya rendah dan hemat energi telah dilakukan oleh beberapa peneliti terdahulu yang mengadopsi SBC sebagai solusi perangkat keras yang efisien dari segi biaya dan energi. Penelitian yang dilakukan oleh Ioannis Stamelos, dkk [12] pada tahun 2016, mengevaluasi kinerja dan efisiensi energi aplikasi Apache Spark (Spark MLlib dan GraphX) pada prosesor *System-on-Chip* (SoC) seperti Raspberry Pi 3 dan Snapdragon 410, dibandingkan dengan prosesor berkinerja tinggi seperti Intel Xeon dan Intel i5. Hasil penelitian menunjukkan bahwa meskipun SoC memiliki waktu eksekusi lebih lambat, efisiensi energi yang dihasilkan 2 hingga 3.5 kali lebih baik dibandingkan prosesor berkinerja tinggi [12]. Temuan ini membuka peluang untuk menggunakan SoC dalam aplikasi *big data* yang hemat energi. Selain itu, João C. Saffran, dkk [13] pada tahun 2016 membangun klaster Raspberry Pi untuk menjalankan algoritma *data mining* (Apriori dan K-Means). Hasil eksperimen menunjukkan bahwa klaster Raspberry Pi dapat mengonsumsi hingga 88,35% dan 85,17% lebih sedikit daya dibandingkan Intel Xeon Phi saat menjalankan Apriori dan K-Means, serta menghemat hingga 45,51% energi saat menjalankan Apriori [13]. Berbeda dengan penelitian sebelumnya yang menggunakan Raspberry Pi untuk menjalankan Spark MLlib, GraphX, atau algoritma *data mining*, penelitian ini memanfaatkan tiga unit Orange Pi Zero 3[1] untuk mengeksekusi algoritma klasifikasi *machine learning* (Random Forest, Regresi Logistik, dan Multi-Layer Perceptron) secara terdistribusi menggunakan Apache Spark MLlib. Fokus utama penelitian ini adalah evaluasi performa dan efisiensi energi klaster SBC tanpa membandingkannya dengan klaster konvensional.

Dalam konteks edukasi, Alonso Rodríguez-Iglesias, dkk [14] pada penelitiannya tahun 2024, memperkenalkan Clupiter, sebuah mini-superkomputer berbasis Raspberry Pi yang dirancang untuk mendemonstrasikan konsep superkomputasi dan pemrograman paralel kepada audiens non-teknis. Clupiter menggunakan *NAS Parallel Benchmarks* (NPB) untuk mengevaluasi kinerja aplikasi paralel. Meskipun fokus utama penelitian tersebut adalah edukasi, hasilnya menunjukkan potensi SBC dalam mensimulasikan lingkungan *High Performance Computing* (HPC) yang kompleks dengan biaya rendah [14].

Berdasarkan studi-studi tersebut, penelitian ini mengadopsi pendekatan serupa, yaitu mengkaji performa klaster SBC, namun dengan eksplorasi jenis prosesor yang berbeda, yaitu 3 unit prosesor Allwinner H618 quad-core Cortex-A53 1.5 GHz. Dengan menggunakan *framework* Apache Spark, penelitian ini mengevaluasi penggunaan energi dan kemampuan sistem dalam menjalankan algoritma klasifikasi *machine learning* seperti Random Forest, Regresi Logistik, dan percobaan dengan *neural network* (Multi-Layer Perceptron). Melalui pendekatan ini, diharapkan dapat ditemukan solusi komputasi yang tidak hanya efisien dalam memproses data tetapi juga hemat energi, mendukung inisiatif *Green AI* dan *Big Data Processing* yang ramah lingkungan.

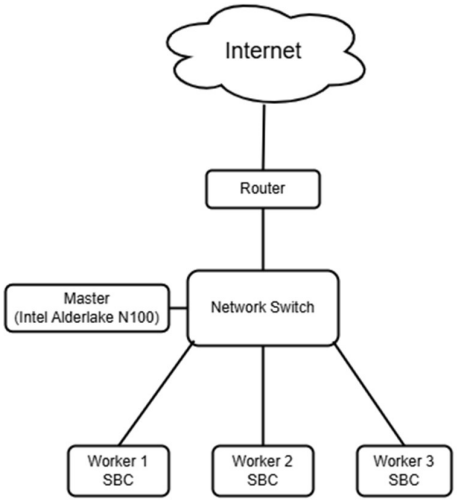
[1] Orange Pi - Orangepi

2. METODE

Untuk mengevaluasi efisiensi energi dan performa sistem dalam menjalankan tugas komputasi *machine learning* secara terdistribusi, dilakukan serangkaian eksperimen berbasis klaster yang memanfaatkan *framework* Apache Spark. Metode yang digunakan mencakup desain eksperimen, pemilihan algoritma dan data, serta teknik pengukuran performa dan konsumsi energi.

2.1. Desain Eksperimen

Penelitian ini menggunakan konsep komputasi terdistribusi yang diterapkan pada klaster SBC. Komputasi terdistribusi sendiri merupakan konsep di mana dua atau lebih komputer yang terhubung dalam jaringan yang sama dan saling berbagi tugas komputasi yang sama [15]. Dalam penelitian ini, eksperimen dilakukan dengan arsitektur sebagai berikut:



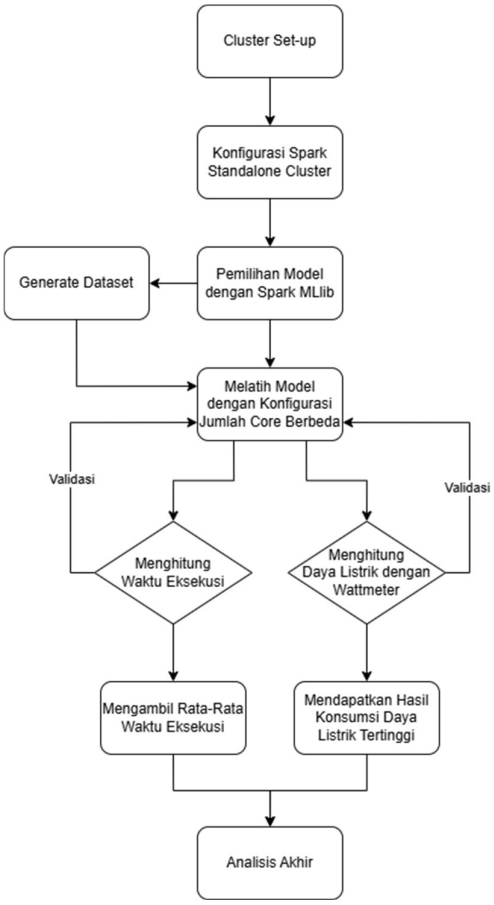
Gambar 2. Arsitektur Klaster SBC.

Gambar 2 memperlihatkan arsitektur klaster SBC yang terdiri dari tiga unit Orange Pi Zero 3 sebagai *worker node* yang bertugas menjalankan proses komputasi secara paralel. Sementara itu, Intel Alder Lake-N100 berperan sebagai *master node* yang bertanggung jawab dalam mendistribusikan tugas, mengelola sumber daya, serta mengoordinasikan komunikasi antar node. Seluruh node terhubung melalui *network switch* agar proses komunikasi antar perangkat berjalan secara optimal. Spesifikasi perangkat keras yang digunakan ditampilkan pada Tabel 1.

Tabel 1. Spesifikasi Perangkat Keras.

Komponen	Klaster SBC
Prosesor	Allwinner H618 quad-core Cortex-A53 1.5 GHz
Jumlah Node	3
Memori (RAM)	4 GB per node (LPDDR4)
Penyimpanan	MicroSD (64 GB)
Sistem Operasi	Ubuntu 22.04.4 LTS

Eksperimen dilakukan dengan membangun klaster SBC, melakukan konfigurasi Spark dalam mode Standalone, dan menjalankan proses pelatihan model *machine learning* menggunakan pustaka Spark MLlib. Selama proses tersebut, dilakukan pengukuran performa berdasarkan waktu eksekusi dan konsumsi energi. Alur lengkap tahapan eksperimen ditampilkan pada Gambar 3.



Gambar 3. Alur Eksperimen.

2.2. Apache Spark

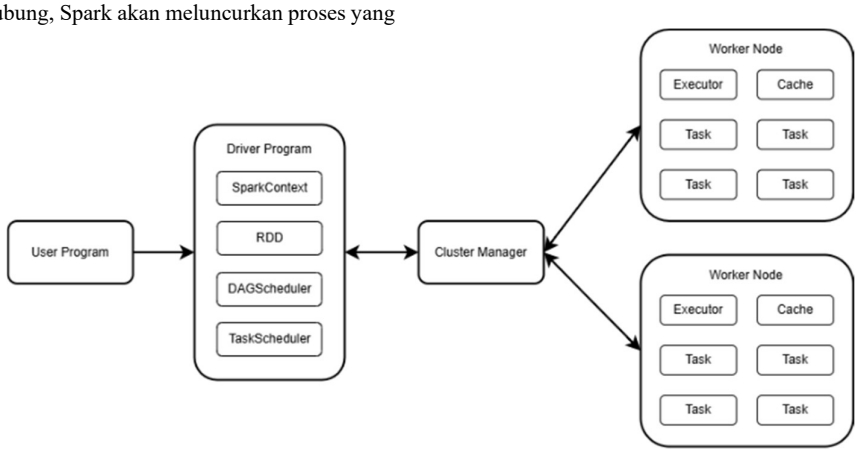
Apache Spark merupakan platform komputasi terdistribusi berbasis *open-source* yang dirancang untuk pemrosesan data skala besar. Dengan desain yang fleksibel, performa cepat, dan kemudahan penggunaan, teknologi ini sangat cocok untuk analisis *big data* serta penerapan *machine learning* [16]. Spark pertama kali dikembangkan sebagai proyek riset di AMPLab, UC, Berkeley pada tahun 2009 [17]. Kemunculan Apache Spark berawal dari upaya menjawab keterbatasan kerangka kerja Hadoop MapReduce, khususnya dalam menangani pemrosesan data yang bersifat interaktif dan iteratif. Spark memperkenalkan pendekatan komputasi dalam memori (*in-memory computation*) sebagai solusi untuk mengurangi latensi yang disebabkan oleh mekanisme baca-tulis ke disk yang digunakan oleh MapReduce.

Untuk mendukung komputasi berbasis memori, Apache Spark menggunakan abstraksi data yang dikenal sebagai *Resilient Distributed Dataset* (RDD). RDD merupakan struktur data hanya-baca (*read-only*) yang disimpan di memori, sehingga memungkinkan pemrosesan data dilakukan secara efisien tanpa harus menulis hasil ke disk setelah setiap operasi. Selain itu, RDD dirancang untuk tetap andal dalam menghadapi kegagalan sistem (*fault-tolerance*) [18]. Spark mendukung beberapa mode klaster

https://doi.org/10.25077/TEKNOSI.v11i2.2025.152-160

seperti Standalone, Hadoop YARN (*Yet Another Resource Negotiator*), Apache Mesos, dan Kubernetes. Dalam penelitian ini, Spark dijalankan menggunakan mode Standalone, yaitu manajer klaster (*cluster manager*) sederhana yang sudah terintegrasi dalam Spark dan dirancang untuk mempermudah proses penyusunan klaster [19].

Setiap aplikasi Spark dijalankan sebagai kumpulan proses yang dikoordinasikan oleh SparkContext, objek utama yang berada dalam *driver program* (program pengendali pusat). Ketika aplikasi dijalankan, SparkContext akan membangun koneksi ke *cluster manager* (dalam hal ini mode Standalone) untuk mengalokasikan sumber daya komputasi yang dibutuhkan. Setelah berhasil terhubung, Spark akan meluncurkan proses yang disebut *executor* pada setiap *worker node*, yang bertugas menjalankan tugas komputasi (*task*) dan menyimpan data di memori atau disk lokal. Selanjutnya, kode aplikasi (biasanya dalam bentuk file JAR atau skrip Python) dikirim ke masing-masing *executor* untuk dieksekusi. SparkContext kemudian menyusun urutan operasi pengguna ke dalam bentuk *Directed Acyclic Graph* (DAG) atau graf berarah tanpa siklus, yang menggambarkan alur logika pemrosesan data. DAG ini kemudian dipecah menjadi beberapa tahap (*stage*) oleh DAGScheduler, lalu dijadwalkan sebagai sekumpulan task oleh TaskScheduler untuk dijalankan secara paralel pada *executor* yang tersedia. Visualisasi dari arsitektur dan alur kerja internal Apache Spark dapat dilihat pada Gambar 4.



Gambar 4. Arsitektur dan Alur Kerja Apache Spark [19].

Apache Spark menyediakan beberapa pustaka bawaan, antara lain Spark SQL, Spark Streaming, MLlib, dan GraphX. Pada penelitian ini, Spark menggunakan pustaka MLlib untuk pemodelan *machine learning*, yang mendukung berbagai algoritma seperti regresi, klasifikasi, dan pengelompokan (*clustering*). Eksperimen dilakukan menggunakan Apache Spark versi 3.5.3, yang dijalankan dalam mode terdistribusi antar node. Sebagai contoh, pada konfigurasi 3 core, sistem menggunakan masing-masing satu core pada tiga node berbeda, bukan tiga core dalam satu node yang sama.

Oleh karena itu, pada kasus ini digunakan konsep *data parallelism*, yaitu pendekatan di mana data pelatihan dibagi ke dalam beberapa partisi dan setiap node dalam klaster Spark memproses bagian data tersebut secara paralel menggunakan model yang sama. Proses ini memungkinkan pemanfaatan sumber daya komputasi secara merata di seluruh node dan mempercepat pelatihan model dengan cara membagi beban kerja tanpa perlu membagi struktur model itu sendiri.

Selain *data parallelism*, Apache Spark juga dapat diadaptasi untuk mendukung *model parallelism*, khususnya pada pelatihan model jaringan saraf berskala besar yang memiliki jumlah parameter sangat banyak dan tidak dapat dimuat secara utuh dalam satu mesin [20]. Dalam pendekatan ini, struktur model dibagi ke beberapa bagian, dan setiap node menangani sebagian dari model tersebut secara terkoordinasi. *Model parallelism* menjadi relevan dalam konteks *deep learning* karena mampu mengatasi keterbatasan memori sekaligus memanfaatkan sumber daya komputasi secara terdistribusi.

Namun, dalam konteks penelitian ini yang menggunakan pustaka Spark MLlib, pendekatan yang digunakan tetap berada pada ranah *data parallelism*. Hal ini sejalan dengan implementasi resmi MLlib yang hingga saat ini mendukung pelatihan model seperti Multi-Layer Perceptron menggunakan pembagian data, tanpa membagi struktur model secara eksplisit antar node [20], [21].

2.3. Dataset

Dataset yang digunakan merupakan kumpulan bilangan acak kontinu dengan distribusi seragam (*uniform*) pada rentang nilai antara 0 hingga 1, yang dibangkitkan menggunakan pustaka *numpy*. Pemilihan dataset ini bertujuan untuk menekankan evaluasi performa sistem terhadap beban komputasi, tanpa mempertimbangkan aspek akurasi model *machine learning*. Fokus evaluasi meliputi waktu eksekusi dan konsumsi energi. Untuk menguji skalabilitas sistem, dataset dibagi ke dalam tiga skenario ukuran, yaitu data kecil (16384 × 20 sampel), data sedang (262144 × 20 sampel), dan data besar (524288 × 20 sampel).

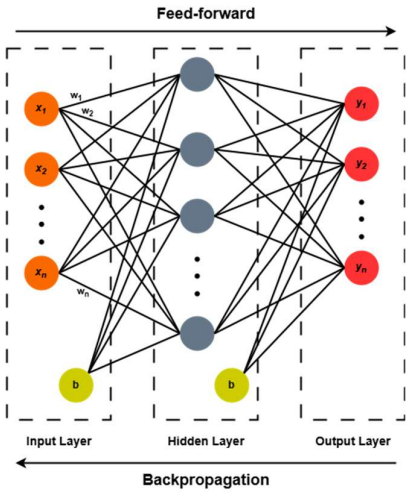
2.4. Algoritma

Algoritma yang digunakan pada penelitian ini adalah Multi-Layer Perceptron, Regresi Logistik, dan Random Forest.

2.4.1. Multi-layer Perceptron

Multi-Layer Perceptron adalah jaringan saraf tiruan yang terdiri dari lapisan input (*input layer*), satu atau lebih lapisan

tersembunyi (*hidden layer*), dan lapisan output (*output layer*). Setiap neuron terhubung penuh dengan neuron di lapisan berikutnya (*fully connected*) dan memproses input menggunakan kombinasi bobot dan bias, yang disebut *weighted sum*. Hasilnya kemudian dipetakan melalui fungsi aktivasi untuk menghasilkan output. MLP bekerja dalam dua tahap utama: *feed-forward*, di mana informasi mengalir maju dari input ke output, dan backpropagation, yaitu proses menghitung gradien kesalahan untuk memperbarui bobot. Pembaruan bobot dilakukan menggunakan algoritma optimasi seperti *Stochastic Gradient Descent* (SGD), Adagrad dan algoritma sejenis lainnya, dengan tujuan untuk meminimalkan kesalahan secara bertahap. Visualisasi arsitektur jaringan MLP ditunjukkan pada Gambar 5.



Gambar 5. Arsitektur Multi-layer Perceptron.

Pada implementasi konvensional, seperti menggunakan TensorFlow atau PyTorch, proses pelatihan MLP dijalankan secara lokal dalam satu mesin. Seluruh data dimuat ke memori, dan proses pembaruan bobot dilakukan secara iteratif dalam satu ruang eksekusi. Sebaliknya, pada lingkungan komputasi terdistribusi seperti Apache Spark, pelatihan model MLP dilakukan secara paralel melalui pustaka Spark MLlib. Data dibagi ke beberapa partisi dan didistribusikan ke node-node dalam klaster. Setiap node memproses subset data tersebut dengan menjalankan tahapan *feed-forward* dan *backpropagation* secara independen, lalu menghitung gradien lokal berdasarkan partisi datanya. Gradien dari seluruh node kemudian dikumpulkan dan disatukan oleh *driver*, yang selanjutnya digunakan untuk memperbarui bobot model secara global.

Meskipun konfigurasi model dalam Spark MLlib lebih terbatas, pendekatan ini memungkinkan pelatihan dilakukan secara efisien pada skenario *big data*. Dalam eksperimen ini, arsitektur jaringan MLP terdiri dari satu lapisan input dengan 20 neuron, tiga lapisan tersembunyi (*hidden layer*) dengan 128, 64, dan 32 neuron secara berturut-turut, serta dua neuron pada lapisan output untuk klasifikasi biner. Model dilatih menggunakan pustaka **MultilayerPerceptronClassifier** pada Spark MLlib dengan parameter default lainnya dan *seed* ditetapkan sebesar 42 untuk menjaga reprodusibilitas hasil.

2.4.2. *Regresi Logistik*

Regresi Logistik merupakan algoritma klasifikasi biner yang digunakan untuk memodelkan probabilitas suatu kejadian berdasarkan satu atau lebih variabel input. Berbeda dengan regresi linear yang menghasilkan output kontinu, regresi logistik menggunakan fungsi logistik (*sigmoid*) untuk memetakan output menjadi nilai probabilitas antara 0 dan 1. Secara matematis, hubungan antara variabel input dan probabilitas kejadian dinyatakan dalam bentuk fungsi logit sebagai berikut:

$$logit(p) = log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k \quad (1)$$

Di mana p adalah probabilitas kejadian, $x_1, \ldots, x_k$ adalah fitur input, dan $\beta_0, \ldots, \beta_k$ adalah parameter model. Proses pelatihan model dilakukan menggunakan metode optimisasi seperti *Stochastic Gradient Descent* (SGD) untuk meminimalkan fungsi *log-loss*. Pada implementasinya di Apache Spark, Regresi Logistik dijalankan secara terdistribusi menggunakan pustaka Spark MLlib. Dataset dibagi ke dalam beberapa partisi dan didistribusikan ke sejumlah node dalam klaster, di mana setiap node melakukan sebagian komputasi. Model regresi logistik dalam eksperimen ini dioptimasi menggunakan metode SGD. Beberapa parameter penting yang digunakan antara lain: jumlah iterasi sebanyak 1000, nilai *learning rate* (atau *step size*) sebesar 0.0005, dan *mini-batch fraction* sebesar 1.0. Model dilatih menggunakan pustaka **LogisticRegressionWithSGD** dari Spark MLlib.

2.4.3. *Random Forest*

Random Forest adalah algoritma *ensemble learning* yang membangun sejumlah pohon keputusan (*decision tree*) dan menggabungkan hasil prediksi dari masing-masing pohon untuk menghasilkan keputusan akhir. Algoritma ini bekerja dengan membentuk banyak pohon dari subset acak data dan fitur, yang membantu mengurangi *overfitting* dan meningkatkan generalisasi model. Prediksi akhir ditentukan melalui voting mayoritas untuk klasifikasi atau rata-rata untuk regresi.

Pada implementasinya di Apache Spark, algoritma Random Forest dapat dijalankan secara terdistribusi menggunakan pustaka Spark MLlib. Dataset dibagi ke dalam beberapa partisi dan didistribusikan ke *worker node* dalam klaster, di mana setiap node membangun satu atau lebih pohon keputusan secara paralel berdasarkan subset data dan fitur yang dipilih secara acak. Setelah seluruh pohon selesai dibangun, hasilnya dikumpulkan oleh *driver* dan disatukan menjadi satu model *ensemble* Random Forest yang utuh.

Dalam eksperimen ini, Random Forest dilatih dengan menggunakan 10 pohon keputusan (*decision tree*) dengan kedalaman maksimum 20 dan maksimal 80 *bins* untuk pemrosesan fitur numerik. Strategi pemilihan subset fitur yang digunakan adalah "sqrt", dan kriteria pemisahan berdasarkan *entropy*. Model ini dikembangkan menggunakan **RandomForest.trainClassifier** dari pustaka Spark MLlib dengan *seed* sebesar 42.

https://doi.org/10.25077/TEKNOSI.v11i2.2025.152-160

2.5. Pengukuran Kinerja dan Konsumsi Energi

Untuk mengevaluasi performa kedua klaster, penelitian ini mengukur dua aspek utama, yaitu waktu eksekusi (*execution time*) dan konsumsi energi (*energy consumption*) selama proses pelatihan model *machine learning*. Pelatihan dilakukan sebanyak lima kali percobaan, dan nilai rata-rata waktu eksekusi dari lima percobaan tersebut digunakan sebagai acuan evaluasi. Sementara itu, pengukuran konsumsi daya dilakukan dengan menggunakan *wattmeter* digital yang dipasang pada *power supply* masing-masing klaster. Nilai daya tertinggi yang tercatat selama proses pelatihan diambil sebagai representasi beban maksimum perangkat. Untuk menghitung energi listrik total yang dikonsumsi selama proses pelatihan, digunakan rumus dasar perhitungan energi listrik sebagai berikut:

$$E_{(J)} = P \times t \tag{2}$$

di mana:

- E = Energi listrik yang dikonsumsi (*Joule*),
- P = Daya listrik yang dikonsumsi oleh perangkat (*Watt*), dan
- t = Waktu penggunaan (detik).

Jika diperlukan untuk dikonversi ke dalam satuan *watt*-jam (Wh), maka digunakan rumus:

$$E_{(Wh)} = \frac{P \times t}{3600} \tag{3}$$

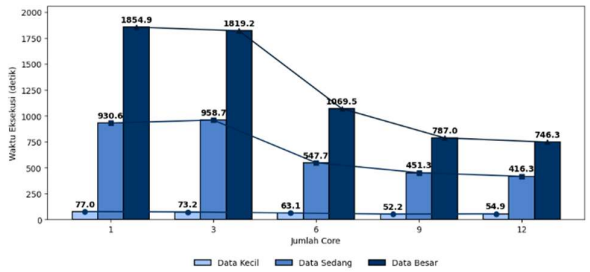
dengan:

- E = Energi listrik yang dikonsumsi (Wh),
- P = Daya listrik yang dikonsumsi oleh perangkat (*Watt*), dan
- t = Waktu penggunaan (detik).

3. HASIL DAN PEMBAHASAN

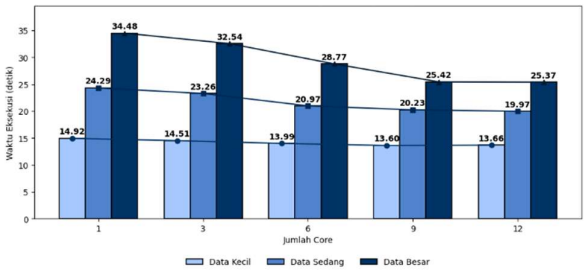
Bagian ini menyajikan hasil evaluasi performa klaster SBC dalam menjalankan algoritma *machine learning* secara terdistribusi menggunakan Apache Spark. Evaluasi dilakukan berdasarkan dua metrik utama, yaitu waktu eksekusi dan konsumsi energi, dengan tiga skenario ukuran data: kecil, sedang, dan besar. Seluruh eksperimen dilakukan dengan lima variasi jumlah core untuk mengamati pengaruh skalabilitas terhadap performa sistem.

3.1. Waktu Eksekusi



Gambar 6. Waktu Eksekusi Pelatihan Algoritma Multi-layer Perceptron.

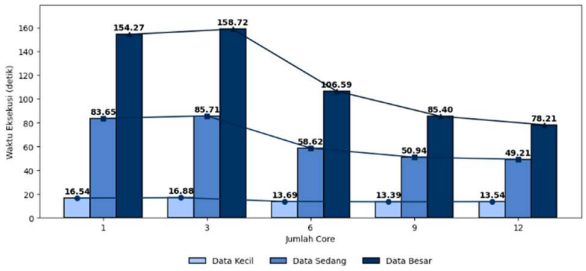
Gambar 6 menunjukkan hasil waktu eksekusi algoritma Multi-Layer Perceptron pada klaster SBC dengan konfigurasi jumlah core 1, 3, 6, 9, dan 12. Terlihat bahwa peningkatan jumlah core secara umum menurunkan waktu eksekusi, terutama pada data berukuran besar. Hal ini mengindikasikan bahwa sistem mampu memanfaatkan paralelisasi yang ditawarkan oleh Apache Spark untuk meningkatkan efisiensi proses pelatihan model. Kemampuan MLP dalam menjalankan komputasi secara paralel memang telah banyak dimanfaatkan dalam berbagai studi, karena sifatnya yang mendukung eksekusi cepat, paralel, dan mudah diimplementasikan [22]. Pada skenario MLP dengan dataset besar, penggunaan 12 core mampu memangkas waktu pelatihan dari 1854.918 detik menjadi 746.29 detik, yang setara dengan percepatan hingga 59.77% dibandingkan eksekusi satu core.



Gambar 7. Waktu Eksekusi Pelatihan Algoritma Regresi Logistik.

Gambar 7 memperlihatkan hasil waktu eksekusi algoritma Regresi Logistik pada klaster SBC dengan konfigurasi jumlah core 1, 3, 6, 9, dan 12. Secara umum, peningkatan jumlah core menyebabkan penurunan waktu eksekusi, terutama pada dataset berukuran besar. Namun, dibandingkan dengan algoritma MLP, penurunan waktu eksekusi pada Regresi Logistik tidak terlihat signifikan. Fenomena ini dapat disebabkan oleh karakteristik algoritma Regresi Logistik itu sendiri, yang relatif lebih sulit diparalelkan dibandingkan algoritma seperti MLP atau Random Forest [23], [24].

Dalam eksperimen ini, fenomena tersebut terlihat jelas pada algoritma Regresi Logistik, di mana penurunan waktu eksekusi tetap terbatas meskipun jumlah core ditingkatkan. Meskipun demikian, performa tetap mengalami peningkatan. Pada data berukuran besar, penggunaan 12 core menurunkan waktu pelatihan dari 34.48 detik menjadi 25.37 detik, atau sekitar 26.4% lebih cepat dibandingkan eksekusi dengan satu core.



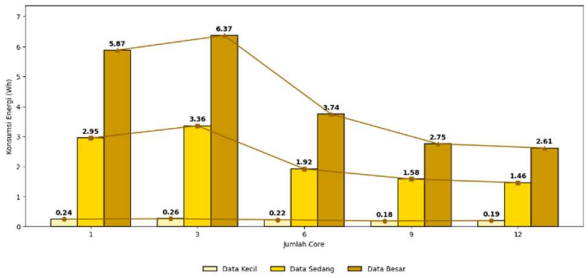
Gambar 8. Waktu Eksekusi Pelatihan Algoritma Random Forest.

Gambar 8 menunjukkan hasil waktu eksekusi algoritma Random Forest pada klaster SBC dengan konfigurasi jumlah core 1, 3, 6,

9, dan 12. Terlihat bahwa peningkatan jumlah core secara umum menurunkan waktu eksekusi, terutama pada data berukuran besar. Sama seperti MLP, paralelisasi pada algoritma Random Forest cukup mudah dilakukan. Ini dibuktikan dengan waktu eksekusi pelatihan model menurun seiring bertambahnya jumlah core. Hal ini sejalan dengan sifat dasar algoritma Random Forest yang memungkinkan setiap pohon keputusan dilatih secara independen, sehingga cocok untuk dijalankan secara paralel [24].

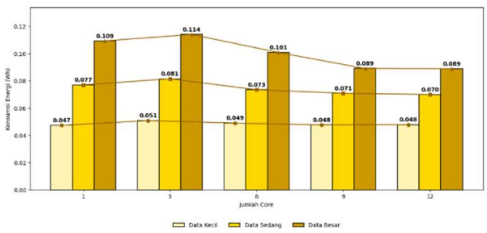
Hanya saja, pada mode serial (satu core) terlihat sedikit lebih cepat dibandingkan tiga core (satu core tiap node). Hal ini kemungkinan disebabkan oleh adanya tambahan proses koordinasi antar node yang terjadi pada mode terdistribusi, seperti penjadwalan *task* dan pembagian data antar mesin, sehingga menyebabkan waktu eksekusi sedikit lebih lama dibandingkan dengan mode eksekusi lokal yang berjalan secara langsung di satu komputer. Pada skenario data besar, konfigurasi 12 core mampu mempercepat pelatihan dari 154.27 detik menjadi 78.21 detik, atau setara dengan peningkatan performa hingga 49.3% dibandingkan eksekusi satu core.

3.2. Konsumsi Energi



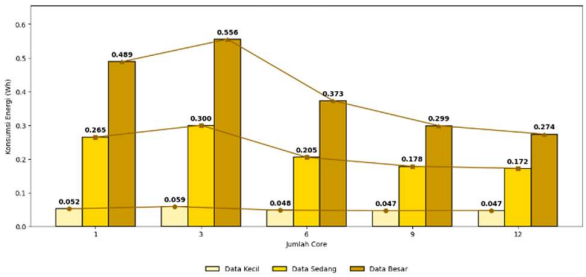
Gambar 9. Konsumsi Energi Pelatihan Algoritma Multi-layer Perceptron.

Gambar 9 memperlihatkan konsumsi energi oleh klaster SBC saat menjalankan algoritma MLP dengan berbagai konfigurasi jumlah core. Secara umum, semakin banyak core yang digunakan, konsumsi energi cenderung menurun. Hal ini disebabkan oleh fakta bahwa konsumsi energi dihitung berdasarkan hasil perkalian antara daya listrik (*watt*) dan waktu eksekusi (detik). Pada eksperimen sebelumnya telah ditunjukkan bahwa penambahan jumlah core mampu mempercepat waktu eksekusi pelatihan model. Sementara itu, daya listrik yang terukur selama eksekusi relatif konstan, yaitu sebesar 11.4 *watt* untuk mode serial (satu core), dan 12.6 *watt* untuk semua konfigurasi core paralel. Dengan demikian, penurunan waktu eksekusi yang signifikan turut berkontribusi terhadap pengurangan konsumsi energi total (dalam satuan Wh), terutama pada konfigurasi core yang lebih tinggi.



Gambar 10. Konsumsi Energi Pelatihan Algoritma Regresi Logistik.

Gambar 10 menunjukkan konsumsi energi oleh klaster SBC dalam menjalankan algoritma Regresi Logistik dengan variasi jumlah core. Secara umum, peningkatan jumlah core cenderung menurunkan konsumsi energi (dalam satuan Wh), meskipun penurunannya tidak se-signifikan pada algoritma MLP. Penurunan ini utamanya disebabkan oleh berkurangnya waktu eksekusi saat jumlah core bertambah, sementara daya listrik yang digunakan selama eksekusi tetap relatif stabil sebagaimana telah dijelaskan sebelumnya. Karena Regresi Logistik bukan algoritma yang mudah diparalelkan [23], waktu eksekusinya tidak turun secara drastis, sehingga konsumsi energi juga tidak menurun secara tajam.



Gambar 11. Konsumsi Energi Pelatihan Algoritma Random Forest.

Gambar 11 menunjukkan konsumsi energi oleh klaster SBC dalam menjalankan algoritma Random Forest pada berbagai konfigurasi jumlah core. Secara umum, tren penurunan konsumsi energi terlihat cukup signifikan seiring bertambahnya jumlah core, terutama pada data berukuran sedang dan besar. Penurunan konsumsi energi ini berkaitan langsung dengan berkurangnya waktu eksekusi, sementara daya listrik yang digunakan selama pelatihan tetap relatif konstan, sebagaimana telah dijelaskan sebelumnya.

Algoritma Random Forest sangat mendukung paralelisasi karena setiap pohon keputusan dalam *ensemble* dapat dilatih secara independen [24]. Oleh karena itu, peningkatan jumlah core dapat mempercepat proses pelatihan secara efektif, yang pada akhirnya berkontribusi langsung terhadap efisiensi energi. Secara keseluruhan, peningkatan jumlah core pada klaster SBC memberikan dampak positif terhadap efisiensi energi, terutama untuk algoritma seperti MLP dan Random Forest yang dirancang untuk pemrosesan paralel. Semakin cepat waktu eksekusi pelatihan, maka semakin rendah pula energi yang dikonsumsi.

4. KESIMPULAN

Penelitian ini menunjukkan bahwa klaster SBC mampu menjalankan algoritma *machine learning* secara terdistribusi menggunakan Apache Spark dengan efisien, baik dari sisi waktu eksekusi maupun konsumsi energi. Eksperimen membuktikan bahwa peningkatan jumlah core memberikan dampak positif terhadap percepatan pelatihan, khususnya pada algoritma yang mudah diparalelkan seperti Multi-Layer Perceptron dan Random Forest, dengan percepatan hingga hampir 59.7% dan 49.3% pada data berukuran besar. Konsumsi daya listrik juga tercatat rendah dan stabil, yaitu sekitar 11.4 *watt* untuk 1 core dan 12.6 *watt* untuk

https://doi.org/10.25077/TEKNOSI.v11i2.2025.152-160

konfigurasi *multi-core*. Hasil ini menegaskan potensi klaster SBC sebagai solusi *Green AI* yang ringan dan ramah lingkungan.

Dengan keterbatasan pada penggunaan dataset sintetis dan fokus pengukuran hanya pada proses pelatihan model, penelitian lanjutan dapat diarahkan pada penggunaan dataset nyata untuk menguji performa klaster SBC dalam skenario riil. Selain itu, evaluasi terhadap strategi distribusi dan partisi data serta konsumsi daya saat inferensi menjadi hal penting yang dapat diteliti guna memperoleh pemahaman yang lebih menyeluruh terhadap efisiensi energi sistem secara *end-to-end*.

DAFTAR PUSTAKA

[1] A. de Vries, "The Growing Energy Footprint of Artificial Intelligence," *Joule*, vol. 7, no. 10, pp. 2191–2194, 2023, doi: 10.1016/j.joule.2023.09.004.

[2] D. Xiao, "Neuroscience-Inspired Continuous Learning: A Sustainable Approach to AI's Energy Challenge," *OSF Preprints*, 2023, doi: 10.31219/osf.io/twn9q.

[3] J. Cowls, A. Tsamados, M. Taddeo, and L. Floridi, "The AI Gambit: Leveraging Artificial Intelligence to Combat Climate Change—Opportunities, Challenges, and Recommendations," *AI Soc*, vol. 38, no. 1, pp. 283–307, 2023, doi: 10.1007/s00146-021-01294-x.

[4] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of Energy Consumption in Machine Learning," *J Parallel Distrib Comput*, vol. 134, pp. 75–88, 2019, doi: 10.1016/j.jpdc.2019.07.007.

[5] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-All: Train One Network and Specialize It for Efficient Deployment," *8th International Conference on Learning Representations, ICLR 2020*, pp. 1–15, 2020, doi: 10.48550/arXiv.1908.09791.

[6] R. Gitzel, M. C. Platenius-Mohr, and A. Burger, "Estimating the Sustainability of AI Models Based on Theoretical Models and Experimental Data," *atp magazin*, vol. 66, no. 3, 2024, doi: 10.17560/atp.v66i3.2691.

[7] E. Strubell, A. Ganesh, and A. Mccallum, "Energy and Policy Considerations for Deep Learning in NLP," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019, doi: 10.18653/v1/P19-1355.

[8] Y. I. Alzoubi and A. Mishra, "Green Artificial Intelligence Initiatives: Potentials and Challenges," *J Clean Prod*, vol. 468, p. 143090, 2024, doi: 10.1016/j.jclepro.2024.143090.

[9] A. Tabbakh, L. Al Amin, M. Islam, G. M. I. Mahmud, I. K. Chowdhury, and M. S. H. Mukta, "Towards Sustainable AI: A Comprehensive Framework for Green AI," *Discover Sustainability*, vol. 5, no. 1, 2024, doi: 10.1007/s43621-024-00641-4.

[10] S. Fund, "Sustainable Development Goals." Accessed: Jan. 27, 2025. [Online]. Available: https://www.un.org/sustainabledevelopment/inequality/

[11] S. Bourhnane, M. R. Abid, K. Zine-Dine, N. Elkamoun, and D. Benhaddou, "High-Performance Computing: A Cost Effective and Energy Efficient Approach," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 6, pp. 1598–1608, 2020, doi: 10.25046/aj0506191.

[12] I. Stamelos, D. Soudris, and C. Kachris, "Performance and Energy Evaluation of Spark Applications on Low-Power SoCs," *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pp. 300–305, 2016, doi: 10.1109/SAMOS.2016.7818362.

[13] J. C. Saffran *et al.*, "A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms," *European Conference on Parallel and Distributed Computing Workshops (Euro-Par Workshops)*, vol. 10104, pp. 788–799, 2016, doi: 10.1007/978-3-319-58943-5_63.

[14] A. Rodriguez-Iglesias, M. J. Martin, and J. Tourino, "Clupiter: A Raspberry Pi Mini-Supercomputer for Educational Purposes," *IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 2400–2405, 2023, doi: 10.1109/TrustCom60117.2023.00338.

[15] Md. F. Ali and R. Z. Khan, "Distributed Computing: An Overview," *International Journal of Advanced Networking and Applications (IJANA)*, vol. 7, no. 01, pp. 2630–2635, 2015.

[16] A. M. Rajpurohit, R. R. Kumar, R. Kumar, and P. Kumar, "A Review on Apache Spark," *Proceedings of the KILBY 100 7th International Conference on Computing Sciences 2023 (ICCS 2023)*, 2023, doi: 10.2139/ssrn.4492445.

[17] H. Luu, "Introduction to Apache Spark," in *Beginning Apache Spark 2: With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning Library*, Berkeley, CA: Apress, 2018, ch. 1, pp. 1–13. doi: 10.1007/978-1-4842-3579-9_1.

[18] E. Shaikh, I. Mohiuddin, Y. Alufaisan, and I. Nahvi, "Apache Spark: A Big Data Processing Engine," in *2nd IEEE Middle East and North Africa Communications Conference, MENACOMM*, Institute of Electrical and Electronics Engineers Inc., 2019, pp. 1–6. doi: 10.1109/MENACOMM46666.2019.8988541.

[19] Apache Spark, "Cluster Mode Overview," Apache.org. Accessed: May 20, 2025. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview

[20] D. Shrivastava, S. Chaudhury, and Dr. Jayadeva, "A Data and Model-Parallel, Distributed and Scalable Framework for Training of Deep Networks in Apache Spark," 2017, [Online]. Available: https://arxiv.org/abs/1708.05840

[21] Apache Spark, "Multilayer Perceptron Classifier," Apache.org. Accessed: Jul. 07, 2025. [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html

[22] K. Assi, "Traffic Crash Severity Prediction—A Synergy by Hybrid Principal Component Analysis and Machine Learning Models," *Int J Environ Res Public Health*, vol. 17, no. 20, pp. 1–16, 2020, doi: 10.3390/ijerph17207598.

[23] D. Lei, M. Du, H. Chen, Z. Li, and Y. Wu, "Distributed Parallel Sparse Multinomial Logistic Regression," *IEEE Access*, vol. 7, pp. 55496–55508, 2019, doi: 10.1109/ACCESS.2019.2913280.

[24] J. Chen *et al.*, "A Parallel Random Forest Algorithm for Big Data in A Spark Cloud Computing Environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2017, doi: 10.1109/TPDS.2016.2603511.

NOMENKLATUR

$logit(p)$	Fungsi logit regresi logistik
p	Peluang kejadian sukses
$1-p$	Peluang kejadian gagal
$\beta_0, \dots, \beta_k$	Parameter model
$x_1, \dots, x_k$	Fitur input
E	Energi listrik yang dihasilkan
P	Daya listrik yang dikonsumsi
t	Waktu eksekusi program
MLP	Multi-Layer Perceptron
DAG	*Directed Acyclic Graph*
RDD	*Resilient Distributed Dataset*
SBC	*Single Board Computer*
SGD	*Stochastic Gradient Descent*

BIODATA PENULIS

Syahel Rusfi Razaba adalah mahasiswa di Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah Jakarta. Minat penelitiannya meliputi Klasifikasi Gambar, *Computer Vision* dan Komputasi Terdistribusi.

Muhaza Liebenlito adalah dosen di Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah Jakarta. Bidang Keahliannya meliputi *Medical Imaging* dan CFD-DL.

Taufik Edy Sutanto adalah dosen di Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah Jakarta. Bidang Keahliannya meliputi Data Sains dan Analisis Media Sosial.