Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 5 No. 2 (2026) pp: 21503-21517
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Implementasi Deep Learning untuk Prediksi Penggunaan Resource CPU dan RAM pada Server Debian Linux

Firamawati Hia[1], Ronita Olive Angelie[2], Lotar Mateus Sinaga[3]
[123]Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas Medan
[1]fmawatyhia@gmail.com, [2]ronitasitanggang589@gmail.com, [3]lotarmateus88@gmail.com

Abstrak

Penelitian ini mengimplementasikan metode deep learning berbasis Long Short-Term Memory (LSTM) untuk memprediksi penggunaan sumber daya CPU dan RAM pada server Debian Linux secara real-time. Data monitoring dikumpulkan otomatis setiap lima detik melalui modul collect.py dan menghasilkan 1.859 rekaman yang memuat timestamp, persentase CPU, persentase RAM, jumlah RAM terpakai, dan jumlah proses aktif. Data diproses melalui pengurutan waktu, ekstraksi fitur jam dan hari, normalisasi MinMaxScaler, serta pembentukan sequence menggunakan sliding window 30 timestep. Model dibangun dengan PyTorch menggunakan dua lapisan LSTM berukuran 64 dan 32 unit, diikuti lapisan fully connected, aktivasi ReLU, dan dropout. Pelatihan dilakukan selama 50 epoch dengan optimizer Adam dan fungsi kerugian Mean Squared Error. Hasil evaluasi akhir menunjukkan training loss 0,0140, test loss 0,0310, RMSE 0,1761, dan MAE 0,1426 pada skala ternormalisasi. Modul predict.py mampu menghasilkan prediksi CPU dan RAM setiap lima detik serta memberikan status NORMAL atau WASPADA berdasarkan ambang CPU 80% dan RAM 85%. Hasil ini menunjukkan bahwa LSTM dapat diterapkan sebagai dasar monitoring prediktif untuk mendukung tindakan proaktif administrator. Meskipun demikian, data dikumpulkan dalam durasi relatif singkat dan didominasi beban rendah, sehingga pengujian pada periode lebih panjang dan variasi beban yang lebih luas tetap diperlukan sebelum sistem digunakan pada lingkungan produksi. Sistem juga mempertahankan arsitektur modular agar mudah dipelihara dan dikembangkan.

Kata kunci: Deep Learning, Debian Linux, LSTM, Monitoring Server, Prediksi Resource, PyTorch

1. Latar Belakang

Kebutuhan terhadap pengelolaan infrastruktur server yang efisien dan berkelanjutan terus meningkat seiring berkembangnya skala sistem berbasis jaringan. Di antara berbagai platform sistem operasi yang tersedia, Debian Linux banyak digunakan sebagai fondasi server karena stabil, memiliki pengelolaan paket yang matang, dan mendukung otomasi layanan. Pada lingkungan yang melayani aplikasi secara terus-menerus, perubahan penggunaan CPU dan RAM perlu diamati karena peningkatan beban yang tidak terdeteksi dapat menyebabkan waktu respons melambat, antrean proses meningkat, dan layanan gagal memenuhi kebutuhan pengguna.

Pendekatan monitoring yang hanya menampilkan kondisi saat ini bersifat reaktif. Administrator biasanya mengambil tindakan setelah pemakaian sumber daya mendekati batas kapasitas atau gangguan telah terjadi. Pendekatan tersebut tetap penting untuk diagnosis, tetapi belum memberi waktu yang cukup untuk menyiapkan penambahan kapasitas, pemindahan proses, penjadwalan ulang pekerjaan, atau pengaturan autoscaling. Prediksi beban beberapa langkah ke depan dapat melengkapi monitoring konvensional sehingga pengelolaan sumber daya berubah dari reaktif menjadi proaktif. Penelitian terbaru menunjukkan bahwa prediksi workload memiliki hubungan langsung dengan efisiensi resource provisioning, pengurangan pemborosan kapasitas, dan pencegahan pelanggaran kualitas layanan [1]–[4].

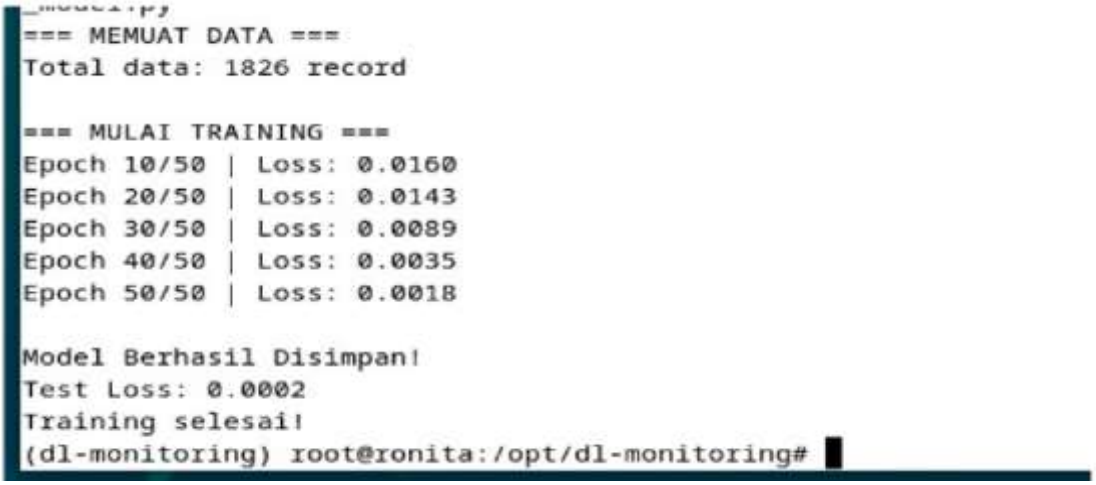
Penggunaan CPU dan RAM membentuk data deret waktu yang dipengaruhi urutan kejadian. Nilai pada satu waktu tidak berdiri sendiri, tetapi berkaitan dengan aktivitas sebelumnya, jumlah proses, pola penggunaan aplikasi, dan waktu pengukuran. Model statistik seperti ARIMA dapat digunakan pada pola yang relatif linier dan stasioner, sedangkan model recurrent neural network mampu mempelajari hubungan nonlinier dan ketergantungan temporal. LSTM mempertahankan informasi penting melalui mekanisme input gate, forget gate, dan output gate sehingga lebih tahan terhadap masalah hilangnya gradien pada sekuens panjang. Pendekatan LSTM, Bi-LSTM, GRU, dan model hibrida telah banyak digunakan untuk prediksi workload cloud, utilisasi host, serta kebutuhan resource virtual machine [1]–[4], [8], [10].

Perkembangan metode peramalan deret waktu juga melahirkan arsitektur Transformer, model dekomposisi, dan model berbasis patch. Temporal Fusion Transformer, Informer, Autoformer, FEDformer, DLinear, PatchTST, dan TimesNet memperlihatkan bahwa hubungan jangka panjang, pola musiman, serta interaksi multivariat dapat dipelajari melalui pendekatan yang berbeda [5]–[7], [9], [11]–[13]. Namun, kompleksitas model yang lebih baru belum selalu dibutuhkan untuk implementasi awal pada server berukuran terbatas. LSTM tetap relevan karena arsitekturnya mudah dikendalikan, kebutuhan komputasinya relatif terukur, dan inferensinya dapat dijalankan langsung pada server tanpa GPU untuk sequence pendek.

Penelitian ini merancang dan mengimplementasikan sistem monitoring serta prediksi resource berbasis LSTM pada Debian 12. Sistem dibangun dengan framework PyTorch dan terdiri atas tiga komponen terintegrasi: modul pengumpulan data collect.py, modul pelatihan train_model.py, dan modul prediksi aktif predict.py. Seluruh komponen dijalankan pada virtual environment Python di direktori /opt/dl-monitoring untuk menjaga isolasi dependensi. Fokus penelitian adalah membuktikan kelayakan teknis pengumpulan data real-time, pembentukan sequence, pelatihan model, dan penyajian prediksi CPU-RAM secara berkelanjutan. Evaluasi juga membahas keterbatasan periode data, dominasi kondisi beban rendah, serta konsekuensi penggunaan metrik ternormalisasi agar hasil penelitian tidak ditafsirkan melebihi cakupan eksperimen.

2. Metode Penelitian

Sistem yang dibangun mengadopsi arsitektur modular dengan tiga komponen utama yang memiliki fungsi spesifik namun dapat beroperasi secara independen. Ketiganya diimplementasikan sebagai skrip Python yang berjalan dalam virtual environment terdedikasi pada direktori /opt/dl-monitoring. Lingkungan eksperimen menggunakan Debian 12 yang dijalankan melalui VirtualBox. Alur implementasi dimulai dari pengumpulan metrik, penyimpanan data CSV, preprocessing dan pembentukan sequence, pelatihan model, penyimpanan model dan scaler, kemudian inferensi berulang untuk menghasilkan prediksi serta status kondisi server.



Gambar 1. Contoh keluaran awal proses pelatihan model pada lingkungan Debian Linux

Gambar 1 mempertahankan dokumentasi keluaran awal yang terdapat pada naskah sumber. Nilai loss pada dokumentasi tersebut berasal dari salah satu proses uji implementasi, sedangkan evaluasi akhir penelitian menggunakan rangkaian nilai training loss dan test loss yang disajikan pada Bagian 3. Pembedaan ini penting karena beberapa percobaan dilakukan selama pengembangan skrip sebelum konfigurasi evaluasi akhir digunakan.

2.1. Persiapan Lingkungan dan Instalasi

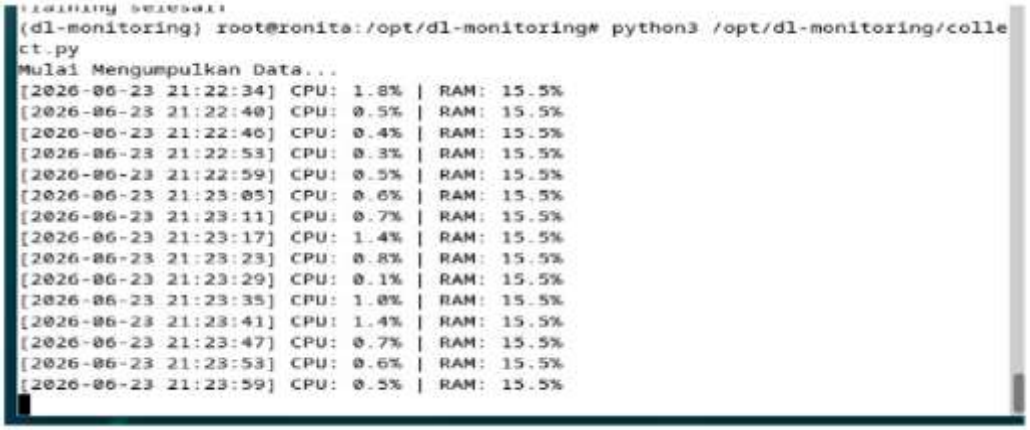
Sebelum sistem diimplementasikan, paket Debian diperbarui dan Python 3, pip, serta venv dipastikan tersedia. Virtual environment dibentuk agar library seperti torch, pandas, numpy, scikit-learn, joblib, dan psutil tidak bercampur dengan paket sistem. Isolasi ini juga mempermudah reproduksi karena versi dependensi dapat disimpan dalam requirements.txt. Direktori kerja dipakai untuk menempatkan skrip, file server_metrics.csv, scaler.pkl, dan berkas bobot model. Prinsip pemisahan lingkungan ini mempertahankan konfigurasi server utama dan mengurangi risiko konflik saat pembaruan paket dilakukan.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

```
sudo apt update && sudo apt upgrade -y
sudo apt install python3 python3-pip python3-venv -y
python3 -m venv /opt/dl-monitoring
source /opt/dl-monitoring/bin/activate
pip install torch pandas numpy scikit-learn joblib psutil
```

2.2. Modul Pengumpulan Data (collect.py)

Modul collect.py berjalan terus-menerus dan merekam metrik setiap lima detik. Pada setiap iterasi, psutil digunakan untuk memperoleh cpu_percent, ram_percent, ram_used_mb, dan process_count. Timestamp ditambahkan agar setiap sampel memiliki urutan waktu yang jelas. Data disimpan dengan mode append sehingga pengumpulan dapat dilanjutkan tanpa menimpa rekaman sebelumnya. Header CSV hanya ditulis ketika berkas belum tersedia. Interval lima detik dipilih sebagai kompromi antara resolusi monitoring dan overhead proses pencatatan.



Gambar 2. Output terminal modul collect.py saat pengumpulan data real-time

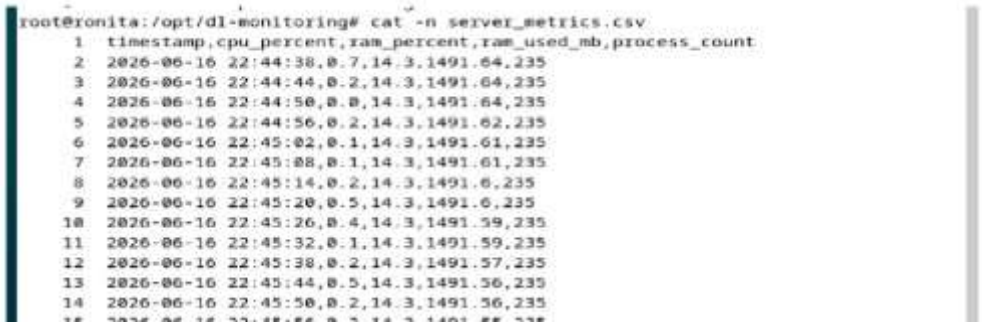
Selama cuplikan pengamatan, penggunaan CPU bergerak sekitar 0,1% sampai 1,8%, sementara RAM berada di sekitar 15,5%. Kondisi tersebut menggambarkan beban ringan dan relatif stabil. Setelah proses berjalan, pemeriksaan jumlah baris menunjukkan 1.860 baris, yaitu satu header dan 1.859 record data. Jumlah ini cukup untuk membuktikan alur kerja model, tetapi belum mewakili variasi harian, perubahan jumlah pengguna, kegiatan pencadangan, kompilasi, atau lonjakan permintaan yang biasanya terjadi pada server produksi.

2.3. Struktur Dataset

Tabel 1. Struktur dan deskripsi atribut dataset server_metrics.csv

Atribut	Tipe Data	Satuan	Deskripsi
timestamp	datetime	YYYY-MM-DD HH:MM:SS	Waktu pengambilan sampel data monitoring.
cpu_percent	float	%	Persentase utilisasi CPU pada saat pengambilan data.
ram_percent	float	%	Persentase penggunaan memori RAM.
ram_used_mb	float	MB	Jumlah kapasitas RAM yang sedang digunakan dalam megabyte.
process_count	integer	proses	Jumlah proses yang sedang aktif pada sistem.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)



```
root@ronita:/opt/dl-monitoring# cat -n server_metrics.csv
     1  timestamp,cpu_percent,ram_percent,ram_used_mb,process_count
     2  2026-06-16 22:44:38,0.7,14.3,1491.64,235
     3  2026-06-16 22:44:44,0.2,14.3,1491.64,235
     4  2026-06-16 22:44:50,0.0,14.3,1491.64,235
     5  2026-06-16 22:44:56,0.2,14.3,1491.62,235
     6  2026-06-16 22:45:02,0.1,14.3,1491.61,235
     7  2026-06-16 22:45:08,0.1,14.3,1491.61,235
     8  2026-06-16 22:45:14,0.2,14.3,1491.6,235
     9  2026-06-16 22:45:20,0.5,14.3,1491.6,235
    10  2026-06-16 22:45:26,0.4,14.3,1491.59,235
    11  2026-06-16 22:45:32,0.1,14.3,1491.59,235
    12  2026-06-16 22:45:38,0.2,14.3,1491.57,235
    13  2026-06-16 22:45:44,0.5,14.3,1491.56,235
    14  2026-06-16 22:45:50,0.2,14.3,1491.56,235
```

Gambar 3. Tampilan sebagian dataset server_metrics.csv

Cuplikan dataset memperlihatkan cpu_percent yang berubah pada rentang rendah, sedangkan ram_percent dan ram_used_mb cenderung stabil di sekitar 14,3% dan 1.491 MB. process_count konsisten pada sekitar 235 proses. Stabilitas ini menguntungkan untuk memeriksa apakah pipeline dapat berjalan tanpa error, tetapi menyebabkan distribusi target menjadi sempit. Model yang dilatih pada rentang sempit cenderung menghasilkan prediksi yang dekat dengan rata-rata data dan belum tentu mempertahankan akurasi ketika beban berubah secara ekstrem.

2.4. Pemrosesan Data dan Pembentukan Sequence

Data CSV diurutkan berdasarkan timestamp, kemudian dari waktu tersebut diekstrak fitur hour dan day_of_week. Empat fitur yang digunakan sebagai masukan model adalah cpu_percent, ram_percent, hour, dan day_of_week, sedangkan targetnya adalah CPU dan RAM pada timestep berikutnya. Fitur dinormalisasi ke rentang 0–1 menggunakan MinMaxScaler. Scaler hasil fitting disimpan dengan joblib dan harus digunakan kembali pada proses inferensi agar transformasi data konsisten dengan tahap pelatihan.

Sequence dibentuk menggunakan sliding window berukuran 30 timestep. Karena interval pencatatan adalah lima detik, setiap input model merepresentasikan sekitar 150 detik kondisi terakhir. Nilai pada timestep ke-31 menjadi target. Data kemudian dibagi secara kronologis dengan rasio 80:20 untuk pelatihan dan pengujian. Urutan tidak diacak agar informasi masa depan tidak bocor ke data latih. Pendekatan ini sejalan dengan prinsip evaluasi time series yang menempatkan observasi lebih awal sebagai data pelatihan dan observasi lebih akhir sebagai data uji.

```
features = ['cpu_percent', 'ram_percent', 'hour', 'day_of_week']
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df[features])
WINDOW = 30
# X: 30 timestep sebelumnya, y: CPU dan RAM timestep berikutnya
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

2.5. Arsitektur dan Pelatihan Model LSTM (train_model.py)

Model menggunakan dua lapisan LSTM berurutan. Lapisan pertama menerima empat fitur per timestep dan menghasilkan representasi 64 unit, sedangkan lapisan kedua mereduksi representasi menjadi 32 unit. Hanya output pada timestep terakhir yang diteruskan ke lapisan fully connected. Lapisan 32–16 menggunakan ReLU dan dropout 0,2 untuk mengurangi ketergantungan berlebihan pada neuron tertentu. Lapisan keluaran memiliki dua neuron yang masing-masing mewakili prediksi CPU dan RAM. DataLoader menggunakan batch size 16 dan shuffle=False untuk mempertahankan urutan temporal.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

Tabel 2. Arsitektur model LSTM

Lapisan	Tipe	Konfigurasi	Output
Input	Input layer	4 fitur × window 30	(batch, 30, 4)
LSTM-1	Recurrent	64 unit; batch_first=True	(batch, 30, 64)
LSTM-2	Recurrent	32 unit; batch_first=True	(batch, 30, 32)
Slice	Sequence selection	Output timestep terakhir	(batch, 32)
FC-1 + ReLU + Dropout	Fully connected	Linear 32→16; dropout 0,2	(batch, 16)
FC-2	Fully connected	Linear 16→2 (CPU dan RAM)	(batch, 2)

Pelatihan dilakukan selama 50 epoch menggunakan optimizer Adam dan fungsi kerugian Mean Squared Error. Pada setiap epoch, bobot diperbarui berdasarkan rata-rata error batch. Setelah pelatihan selesai, model disimpan dan dievaluasi pada data uji menggunakan model.eval() serta torch.no_grad(). RMSE dan MAE dihitung pada skala ternormalisasi untuk melengkapi nilai MSE. Karena skala target berasal dari MinMaxScaler, interpretasi error ke satuan persentase aktual harus selalu mempertimbangkan rentang minimum dan maksimum data yang digunakan saat fitting.

2.6. Modul Prediksi Real-Time (predict.py)

Modul predict.py memuat scaler dan bobot model yang telah disimpan, kemudian mengisi buffer hingga mencapai 30 timestep. Setelah buffer penuh, data terbaru dinormalisasi, dikonversi menjadi tensor, dan dimasukkan ke model. Output inverse-transform dikembalikan ke skala CPU dan RAM. Sistem menampilkan nilai aktual, nilai prediksi, waktu inferensi, dan klasifikasi kondisi server. Ambang batas ditetapkan pada CPU 80% dan RAM 85%. Ketika salah satu prediksi melewati ambang, status berubah menjadi WASPADA sebagai sinyal agar administrator melakukan pemeriksaan atau penyesuaian kapasitas.

Klasifikasi tersebut merupakan aturan operasional dan bukan keluaran klasifikasi yang dipelajari model. Dengan demikian, kualitas status WASPADA bergantung pada dua komponen: ketepatan prediksi regresi dan relevansi nilai threshold terhadap karakteristik server. Pada implementasi produksi, threshold sebaiknya ditentukan berdasarkan kapasitas, jenis layanan, pola antrean, dan toleransi risiko, serta dapat dibedakan antara peringatan awal, kritis, dan pemulihan agar tidak menghasilkan alarm berulang.

3. Hasil dan Pembahasan

Proses pelatihan akhir membaca 1.859 record dan menyelesaikan 50 epoch. Training loss menurun dari 0,0210 pada epoch ke-10 menjadi 0,0140 pada epoch ke-50. Pola penurunan yang bertahap menunjukkan optimizer mampu memperbaiki bobot model secara konsisten. Test loss sebesar 0,0310 lebih tinggi daripada training loss, tetapi selisih tersebut masih dapat diterima pada tahap proof of concept karena data uji berasal dari bagian waktu yang lebih akhir dan tidak digunakan untuk pembaruan bobot. Nilai test loss yang tidak melonjak tajam menunjukkan model tetap mengenali pola umum pada rangkaian data uji.

Dokumentasi terminal pada Gambar 4 memperlihatkan dua proses utama, yaitu pelatihan model dan prediksi aktif. Nilai loss yang tersimpan pada hasil evaluasi akhir digunakan sebagai dasar pembahasan. Sementara itu, cuplikan percobaan awal pada Gambar 1 memperlihatkan nilai yang berbeda karena konfigurasi data dan proses pengembangan belum identik dengan evaluasi akhir. Penyajian kedua dokumentasi dipertahankan untuk menunjukkan tahapan implementasi, tetapi klaim kinerja penelitian mengacu pada Tabel 3 dan Tabel 4.

Penurunan loss tidak boleh langsung diartikan sebagai kemampuan generalisasi yang kuat pada semua kondisi server. Data latih didominasi CPU kurang dari 2% dan RAM sekitar 14–18%. Dalam kondisi seperti ini, model dapat memperoleh error kecil dengan mempelajari level dasar yang relatif konstan. Agar kemampuan prediksi lonjakan beban dapat dinilai, data perlu mencakup proses komputasi berat, peningkatan jumlah koneksi, aktivitas baca-tulis, pencadangan, dan variasi jadwal. Studi prediksi cloud juga menekankan pentingnya variasi workload

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

dan pemodelan multivariat untuk menghindari keputusan provisioning yang terlalu bergantung pada pola historis sempit [2]–[4], [8].

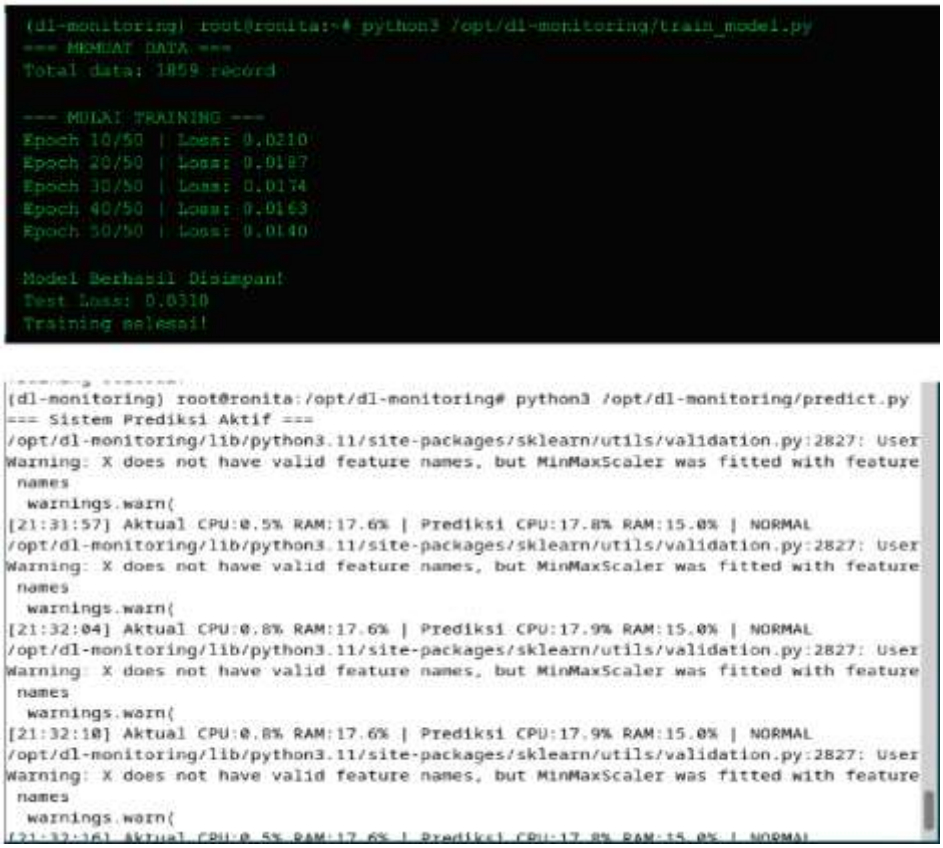
Pembagian data secara kronologis menjadi bagian pelatihan dan pengujian merupakan langkah penting dalam penelitian deret waktu. Data yang lebih awal digunakan untuk membentuk bobot model, sedangkan data yang lebih akhir dipertahankan sebagai simulasi kondisi yang belum pernah dilihat. Pendekatan ini lebih realistis daripada pengacakan acak karena pada penggunaan nyata administrator selalu memprediksi periode setelah data historis tersedia. Walaupun demikian, satu kali pembagian 80:20 belum cukup untuk memastikan kestabilan. Perubahan singkat pada bagian akhir, termasuk kenaikan proses atau pergeseran penggunaan memori, dapat membuat hasil test loss berbeda. Oleh sebab itu, nilai evaluasi akhir dalam penelitian ini diperlakukan sebagai indikasi awal kelayakan pipeline, bukan sebagai ukuran akurasi universal.

Prediksi dilakukan secara multivariat dengan empat fitur masukan, tetapi keluaran difokuskan pada CPU dan RAM. Hubungan kedua target perlu diperhatikan karena CPU dapat melonjak dalam waktu sangat singkat, sedangkan RAM biasanya berubah lebih lambat dan dapat bertahan setelah proses selesai. Satu model bersama memungkinkan representasi temporal dipakai untuk kedua target, namun skala dan volatilitasnya tidak identik. Evaluasi lanjutan sebaiknya melaporkan loss CPU dan loss RAM secara terpisah, melakukan inverse transform sebelum interpretasi, dan memeriksa apakah peningkatan akurasi salah satu target justru diikuti penurunan pada target lain. Pendekatan multi-task seperti ini tetap berguna selama keseimbangan kontribusi masing-masing target dijaga.

Tingginya autokorelasi pada interval lima detik juga menuntut pembanding yang sederhana. Ketika kondisi server stabil, nilai berikutnya sering sangat dekat dengan nilai terakhir, sehingga persistence baseline dapat menghasilkan error rendah tanpa mempelajari pola kompleks. Model LSTM baru dapat dinilai memberi manfaat tambahan apabila mampu mengungguli baseline pada periode perubahan beban, bukan hanya pada bagian data yang datar. Karena eksperimen awal lebih banyak merekam kondisi normal, pembahasan hasil menekankan keberhasilan integrasi sistem dan kemampuan inferensi real-time. Klaim keunggulan algoritmik harus menunggu pengujian terkontrol dengan pola beban naik, puncak, turun, dan pemulihan yang berulang.

Ukuran window 30 timestep merepresentasikan sekitar dua setengah menit riwayat. Pilihan ini cukup untuk menangkap fluktuasi sangat pendek dan tetap menjaga beban komputasi rendah. Namun, kapasitas prediksi akan berbeda ketika pola layanan memiliki siklus lima belas menit, satu jam, atau harian. Window yang terlalu pendek dapat kehilangan konteks, sedangkan window yang terlalu panjang menambah ukuran tensor, waktu pelatihan, dan risiko mempelajari informasi yang tidak relevan. Eksperimen lanjutan perlu membandingkan beberapa ukuran window dengan konfigurasi lain yang sama, lalu menilai bukan hanya RMSE dan MAE, tetapi juga latensi, penggunaan memori, serta kemampuan mendeteksi transisi menuju kondisi beban tinggi.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)



Gambar 4. Output terminal pelatihan model dan prediksi aktif

Tabel 3. Perkembangan loss selama pelatihan model LSTM

Epoch	Training Loss	Status Konvergensi
10	0,0210	Mulai konvergen
20	0,0187	Penurunan loss stabil
30	0,0174	Penurunan loss berlanjut
40	0,0163	Mendekati kondisi optimal
50	0,0140	Konvergen pada konfigurasi pengujian
Test Loss	0,0310	Kemampuan generalisasi awal pada data uji

3.1. Output Sistem Prediksi Aktif

Setelah model disimpan, predict.py dapat dijalankan tanpa mengulang pelatihan. Sistem menunggu buffer berisi 30 pengamatan, kemudian menghasilkan prediksi setiap lima detik. Pada contoh pukul 21:31:57, CPU aktual 0,5% diprediksi menjadi 17,8%, sedangkan RAM aktual 17,6% diprediksi menjadi 15,0%. Karena kedua nilai prediksi masih di bawah threshold, sistem menampilkan status NORMAL. Proses berulang pada waktu berikutnya dan menunjukkan bahwa integrasi pengambilan metrik, transformasi fitur, inferensi, inverse transform, dan pencetakan status berjalan secara berkelanjutan.

Prediksi CPU yang jauh lebih tinggi daripada nilai aktual perlu dibaca secara hati-hati. Pada skala data yang sangat rendah, sedikit perubahan hasil normalisasi dapat diterjemahkan menjadi perbedaan yang terlihat besar ketika dikembalikan ke skala persentase. Kondisi ini dapat pula menunjukkan bahwa model tertarik pada pola

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

sementara dalam window atau belum memiliki cukup contoh untuk memisahkan fluktuasi acak dari kenaikan beban yang nyata. Oleh karena itu, prediksi sebaiknya tidak langsung digunakan untuk mematikan proses atau memindahkan layanan. Pada tahap awal, keluaran lebih aman digunakan sebagai informasi tambahan yang dikonfirmasi dengan metrik aktual dan tren beberapa interval.

Peringatan dari scikit-learn mengenai feature names muncul karena scaler difit menggunakan DataFrame berkolom, sedangkan data inferensi diberikan dalam bentuk array. Peringatan tersebut tidak menghentikan proses, tetapi sebaiknya diperbaiki dengan membentuk DataFrame dengan nama kolom yang sama sebelum transformasi. Perbaikan ini meningkatkan konsistensi pipeline dan mencegah risiko urutan fitur berubah tanpa terdeteksi. Dalam sistem produksi, validasi tipe data, urutan kolom, nilai kosong, dan rentang input perlu dijalankan sebelum model melakukan inferensi.

Dari sisi operasional, latensi prediksi kurang dari satu detik lebih pendek daripada interval monitoring lima detik. Artinya, model dapat menyelesaikan inferensi sebelum sampel berikutnya tersedia. Keunggulan ini penting karena monitoring prediktif tidak boleh menambah beban berlebihan pada server yang sedang diamati. Meski demikian, overhead pengumpulan metrik dan inferensi perlu diukur kembali ketika sistem berjalan bersama layanan produksi, terutama pada perangkat dengan jumlah inti terbatas atau ketika beberapa model dijalankan secara paralel.

3.2. Analisis Performa Sistem

Tabel 4. Ringkasan evaluasi performa sistem

Dimensi Evaluasi	Hasil	Keterangan
Total data	1.859 record	Pengambilan setiap 5 detik selama sekitar 2,5–3 jam.
Training loss akhir	0,0140	Model mencapai penurunan loss yang stabil sampai epoch 50.
Test loss	0,0310	Generaliasi awal pada bagian data yang lebih akhir.
Latensi prediksi	< 1 detik	Lebih pendek daripada interval monitoring.
Stabilitas sistem	Stabil selama pengujian	Tidak ditemukan error kritis yang menghentikan pipeline.
Threshold CPU/RAM	80% / 85%	Dapat disesuaikan dengan kebijakan operasional.
RMSE ternormalisasi	0,1761	Menggambarkan akar rata-rata kuadrat error pada skala 0–1.
MAE ternormalisasi	0,1426	Rata-rata selisih absolut prediksi pada skala 0–1.

RMSE memberikan penalti lebih besar terhadap kesalahan yang jauh dari nilai aktual, sedangkan MAE menunjukkan rata-rata deviasi absolut. Nilai RMSE 0,1761 lebih tinggi daripada MAE 0,1426, sehingga terdapat beberapa prediksi dengan error relatif lebih besar dibanding mayoritas pengamatan. Perbedaan kedua metrik ini wajar pada data time series yang memiliki fluktuasi pendek. Namun, pernyataan bahwa error setara dengan persentase CPU tertentu tidak dapat dibuat hanya dengan mengalikan nilai ternormalisasi secara langsung; konversi harus menggunakan selisih nilai minimum dan maksimum yang dipakai oleh scaler untuk masing-masing target.

Test loss 0,0310 menunjukkan model masih memiliki ruang perbaikan. Nilai tersebut juga perlu dibandingkan dengan baseline sederhana, misalnya prediksi persistence yang menetapkan nilai berikutnya sama dengan nilai terakhir, moving average, dan model linier. Tanpa baseline pada dataset yang sama, belum dapat dipastikan seberapa besar manfaat LSTM dibanding metode yang jauh lebih sederhana. Prinsip evaluasi ini sejalan dengan temuan DLinear yang menunjukkan bahwa model linier dapat menjadi baseline kuat pada beberapa dataset peramalan jangka panjang [11].

Keberhasilan teknis utama penelitian bukan hanya terletak pada angka loss, tetapi pada terhubungnya seluruh pipeline secara otomatis. collect.py menghasilkan data, train_model.py memproses serta menyimpan model, dan predict.py memakai artefak yang sama untuk inferensi. Arsitektur modular membuat setiap komponen dapat diuji

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

secara terpisah. Jika terjadi gangguan pada pengumpulan data, model yang sudah tersimpan tetap dapat dipakai; jika model diperbarui, pengumpul data tidak perlu diubah. Pemisahan ini meningkatkan kemudahan pemeliharaan dan cocok dengan praktik deployment layanan kecil.

Stabilitas selama 50 menit atau lebih pada pengujian awal menunjukkan loop inferensi, pengelolaan buffer, dan pembacaan psutil berfungsi. Namun, kestabilan jangka pendek belum menggambarkan penggunaan selama berminggu-minggu. Sistem perlu menangani rotasi file CSV, perubahan waktu sistem, restart service, data duplikat, file model rusak, serta pembaruan library. Pada implementasi produksi, ketiga modul sebaiknya dijalankan sebagai service systemd dengan kebijakan restart, log terstruktur, dan batas penggunaan disk.

3.3. Karakteristik Data dan Dampaknya terhadap Prediksi

Distribusi data merupakan faktor paling menentukan dalam penelitian ini. CPU sebagian besar berada pada beban sangat rendah dan RAM cenderung stabil. Model mendapatkan banyak contoh kondisi normal tetapi hampir tidak memperoleh contoh kondisi WASPADA. Akibatnya, model lebih siap memperkirakan nilai di sekitar kondisi normal dibanding mengenali transisi menuju beban tinggi. Fenomena ini dapat disebut keterbatasan cakupan distribusi, bukan sekadar kekurangan jumlah baris. Menambah ribuan baris dengan pola yang sama tidak otomatis meningkatkan kemampuan model menghadapi kondisi yang belum pernah diamati.

Fitur hour dan day_of_week dimaksudkan untuk memberi konteks waktu. Akan tetapi, data yang dikumpulkan hanya beberapa jam membuat nilai day_of_week hampir konstan dan rentang hour terbatas. Dalam kondisi tersebut, fitur waktu belum mempunyai kesempatan untuk menunjukkan pola harian atau mingguan. Agar fitur tersebut bermakna, pengumpulan perlu berlangsung minimal beberapa minggu dan mencakup jam sibuk, jam sepi, hari kerja, serta akhir pekan. Alternatif lain adalah menggunakan representasi siklis sin dan cos untuk jam dan hari sehingga kedekatan antara pukul 23 dan 00 dapat dipelajari dengan lebih baik.

process_count dicatat tetapi tidak digunakan sebagai fitur masukan pada konfigurasi model yang dijelaskan. Informasi ini dapat ditambahkan pada penelitian berikut karena peningkatan jumlah proses dapat mendahului kenaikan CPU atau memori. Demikian pula, ram_used_mb dapat digunakan bersama ram_percent untuk membedakan perubahan relatif dan absolut. Meskipun keduanya berkorelasi, nilai absolut berguna ketika kapasitas RAM berubah atau model diterapkan pada server lain. Penambahan fitur perlu disertai analisis korelasi dan evaluasi ablation agar diketahui fitur mana yang benar-benar memperbaiki prediksi.

Interval lima detik menghasilkan data dengan autokorelasi tinggi. Dua pengamatan berurutan biasanya sangat mirip, terutama ketika server tidak menerima beban baru. Kondisi ini dapat membuat model terlihat akurat karena target berikutnya dekat dengan nilai terakhir. Untuk menguji manfaat LSTM, hasilnya perlu dibandingkan dengan persistence baseline dan dilakukan evaluasi pada periode yang berisi perubahan beban. Selain itu, horizon prediksi dapat diperluas menjadi 30 detik, satu menit, atau lima menit. Horizon yang lebih panjang memberi waktu tindakan lebih banyak kepada administrator, meskipun tingkat kesalahan biasanya meningkat.

Window 30 timestep setara sekitar 2,5 menit. Window tersebut cukup untuk menangkap fluktuasi sangat pendek, tetapi belum tentu mencakup pola proses yang berlangsung lebih lama. Pemilihan window perlu diuji melalui beberapa nilai, misalnya 12, 30, 60, dan 120 timestep. BHyPreC menunjukkan bahwa ukuran historical window memengaruhi akurasi prediksi workload [1], sedangkan model workload multivariat lain juga memanfaatkan struktur sekuens dan hubungan antarf fitur untuk meningkatkan estimasi [2], [8]. Oleh sebab itu, konfigurasi window sebaiknya diperlakukan sebagai hyperparameter, bukan nilai tetap yang berlaku untuk semua server.

3.4. Perbandingan LSTM dengan Metode ARIMA

Naskah awal membandingkan hasil LSTM dengan estimasi ARIMA yang diadaptasi dari literatur. Perbandingan ini dipertahankan sebagai gambaran indikatif, tetapi bukan eksperimen head-to-head. ARIMA tidak dilatih pada server_metrics.csv dengan pembagian data dan horizon yang sama. Oleh karena itu, selisih metrik tidak boleh dianggap sebagai bukti final bahwa LSTM unggul sebesar persentase tertentu pada dataset penelitian. Pengujian yang adil mensyaratkan data, preprocessing, horizon, dan metrik identik.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

Tabel 5. Perbandingan indikatif LSTM dan estimasi ARIMA dari literatur

Metrik	ARIMA (estimasi literatur)	LSTM (penelitian)	Interpretasi
MSE	0,0397	0,0310	LSTM lebih rendah secara indikatif.
RMSE	0,1992	0,1761	Selisih bersifat sementara karena dataset berbeda.
MAE	0,1633	0,1426	Perlu validasi langsung pada data yang sama.
Pola nonlinier	Terbatas	Didukung	LSTM memodelkan ketergantungan temporal nonlinier.
Prediksi real-time	Memerlukan pemutakhiran model	Didukung	Inferensi LSTM dapat dijalankan berulang setelah model dimuat.

ARIMA memiliki keunggulan dalam interpretasi, kebutuhan data yang relatif kecil, dan proses pelatihan yang ringan. Model ini dapat menjadi baseline penting ketika deret waktu stasioner atau dapat dibuat stasioner melalui differencing. LSTM lebih fleksibel untuk pola nonlinier dan masukan multivariat, tetapi memerlukan pemilihan arsitektur, normalisasi, epoch, serta pengendalian overfitting. Dalam konteks server yang bebannya sangat stabil, ARIMA atau persistence baseline mungkin sudah memadai. LSTM menjadi lebih bernilai ketika pola beban kompleks, memiliki interaksi beberapa metrik, dan berubah berdasarkan konteks waktu.

Literatur mutakhir menunjukkan bahwa tidak ada satu arsitektur yang selalu unggul. Informer, Autoformer, FEDformer, PatchTST, TimesNet, dan DLinear menggunakan mekanisme berbeda untuk menangkap pola temporal [6], [7], [9], [11]–[13]. Temuan tersebut mendukung perlunya evaluasi beberapa baseline. Penelitian ini memilih LSTM karena sasaran utamanya adalah implementasi end-to-end pada Debian Linux, bukan kompetisi komprehensif antararsitektur. Dengan keterbatasan data 1.859 record, model sederhana dan mudah dijalankan merupakan pilihan yang rasional untuk proof of concept.

Pengujian lanjutan sebaiknya melatih ARIMA, persistence, moving average, random forest berbasis lag, GRU, dan LSTM pada pembagian data yang sama. Metrik dilaporkan per target CPU dan RAM, bukan hanya agregat. Selain MSE, RMSE, dan MAE, evaluasi dapat menambahkan MAPE dengan perlakuan khusus untuk nilai CPU mendekati nol, serta waktu pelatihan, waktu inferensi, dan konsumsi memori model. Dengan desain tersebut, keputusan penggunaan model dapat mempertimbangkan keseimbangan antara akurasi dan biaya komputasi.

3.5. Perbandingan dengan Penelitian Terkini

BHyPreC menggabungkan Bi-LSTM, LSTM, dan GRU untuk memprediksi workload CPU virtual machine serta melaporkan keunggulan terhadap model statistik dan recurrent tunggal [1]. Ouhame, Hadi, dan Ullah menggabungkan CNN dan LSTM untuk memprediksi CPU, memori, disk, dan jaringan pada cloud data center [2]. Kedua penelitian tersebut menegaskan bahwa karakteristik spasial atau interaksi multivariat dapat dipelajari sebelum hubungan temporal diproses. Sistem pada penelitian ini lebih sederhana karena hanya menggunakan dua lapisan LSTM, tetapi kesederhanaan tersebut memudahkan deployment langsung pada server Debian dan mengurangi kebutuhan komputasi.

Bi et al. mengintegrasikan pemrosesan data, filtering, normalisasi, serta arsitektur LSTM untuk workload dan resource cloud [3]. Kesamaan dengan penelitian ini terletak pada penggunaan Min-Max scaling dan pemodelan sekuens. Perbedaannya adalah penelitian ini menitikberatkan implementasi operasional dalam tiga modul yang dapat dijalankan langsung pada server. Kumar, Singh, dan Buyya menambahkan mekanisme belajar dari error peramalan sebelumnya untuk meningkatkan prediksi workload [4]. Gagasan tersebut dapat diadaptasi melalui retraining berkala atau online learning ketika pola server berubah.

esDNN memproses workload multivariat dan menunjukkan manfaat GRU yang dimodifikasi untuk autoscaling [8]. START menggunakan encoder LSTM untuk memprediksi straggler dan mendukung mitigasi

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

proaktif pada cloud [10]. Setayesh, Hadian, dan Prodan menekankan kebutuhan prediksi host yang cepat, adaptif, dan multistep, serta menggunakan pruning untuk mengurangi biaya inferensi [15]. Dibanding penelitian tersebut, sistem ini belum memiliki mekanisme autoscaling, mitigasi otomatis, online learning, atau pruning. Kontribusinya berada pada bukti implementasi sederhana yang menghubungkan monitoring Debian dengan prediksi aktif.

Daraghmeh, Agarwal, dan Jararweh menggunakan pendekatan multilevel yang menggabungkan deteksi anomali, clustering, dan ensemble regression untuk prediksi CPU [14]. COSCO2 mengoptimalkan model prediksi workload untuk mendukung keberlanjutan cloud data center [16]. Simaiya et al. menggabungkan CNN-LSTM dan optimasi untuk load balancing serta prediksi utilisasi host [17]. Penelitian-penelitian tersebut menunjukkan bahwa prediksi resource dapat menjadi komponen dari sistem keputusan yang lebih besar. Pada penelitian ini, status NORMAL/WASPADA merupakan langkah awal menuju integrasi dengan penjadwalan, load balancing, atau notifikasi otomatis.

Model Transformer dan variasinya menawarkan kemampuan untuk memodelkan dependensi panjang dan hubungan multivariat [5]–[7], [9], [12], [13]. Namun, untuk dataset pendek dengan sampling lima detik dan window 30, penggunaan Transformer belum tentu memberikan manfaat yang sebanding dengan kompleksitasnya. DLinear bahkan menunjukkan bahwa baseline linier dapat mengungguli Transformer pada beberapa benchmark [11]. Oleh karena itu, pengembangan berikutnya sebaiknya tidak langsung mengganti LSTM dengan model lebih besar, tetapi melakukan perbandingan terkontrol berdasarkan akurasi, latensi, memori, dan kemudahan deployment.

3.6. Implikasi Operasional pada Server Debian Linux

Implementasi langsung pada Debian Linux memberi nilai praktis karena model tidak hanya diuji dalam notebook terpisah. File model, scaler, dan skrip ditempatkan pada direktori layanan yang jelas. Administrator dapat menjadwalkan collect.py dan predict.py melalui systemd, menyimpan log ke journal, dan menambahkan restart otomatis ketika proses berhenti. Struktur ini dapat diterapkan pada server lokal, virtual machine, atau perangkat edge selama Python dan dependensi tersedia.

Status WASPADA sebaiknya menjadi pemicu pemeriksaan berjenjang. Prediksi di atas threshold selama satu interval dapat dianggap peringatan awal, sedangkan kondisi yang bertahan beberapa interval dapat dinaikkan menjadi peringatan kritis. Pendekatan hysteresis dapat diterapkan agar status tidak berubah cepat ketika nilai berada dekat threshold. Sistem juga dapat menggabungkan nilai aktual dan prediksi, misalnya peringatan hanya aktif jika prediksi tinggi dan tren aktual meningkat. Kebijakan semacam ini mengurangi false alarm akibat satu prediksi ekstrem.

Untuk mendukung kapasitas proaktif, prediksi dapat dikirim ke dashboard atau sistem notifikasi. Grafana, Prometheus, atau aplikasi web sederhana dapat menampilkan nilai aktual, prediksi, interval kepercayaan, dan status. Namun, integrasi visualisasi tidak boleh menutupi kebutuhan penyimpanan data yang konsisten. CSV cocok untuk prototipe, tetapi basis data time-series lebih sesuai untuk volume jangka panjang, query rentang waktu, retensi, dan agregasi. Rotasi serta kompresi data juga perlu diterapkan agar monitoring tidak memenuhi disk.

Keamanan deployment perlu diperhatikan. Skrip tidak seharusnya selalu dijalankan sebagai root jika tidak diperlukan. Direktori model dan data diberi izin terbatas, dependensi dikunci versinya, dan input eksternal divalidasi. Jika model diunduh atau diperbarui dari lokasi lain, integritas file harus diperiksa. Logging juga tidak boleh menyimpan informasi sensitif dari proses atau pengguna. Prinsip least privilege mempertahankan manfaat monitoring tanpa memperbesar permukaan serangan server.

Dari sisi performa, pengambilan metrik setiap lima detik relatif ringan, tetapi interval dapat disesuaikan. Server dengan workload cepat mungkin memerlukan interval satu detik, sedangkan server yang stabil dapat menggunakan 15–60 detik. Interval yang lebih pendek meningkatkan resolusi sekaligus memperbesar ukuran data dan autokorelasi. Keputusan harus mempertimbangkan tujuan prediksi, horizon tindakan, serta overhead. Prediksi untuk autoscaling biasanya memerlukan horizon lebih panjang daripada prediksi untuk menampilkan peringatan kepada administrator.

3.7. Validitas, Keterbatasan, dan Risiko Interpretasi

Keterbatasan pertama adalah durasi pengumpulan sekitar 2,5–3 jam. Data ini belum merepresentasikan siklus harian dan mingguan. Keterbatasan kedua adalah dominasi beban rendah, sehingga model belum diuji pada kondisi

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

CPU dan RAM mendekati threshold. Keterbatasan ketiga adalah evaluasi hanya menggunakan satu urutan pembagian 80:20. Hasil dapat berubah jika periode uji mengandung pola berbeda. Time-series cross-validation atau rolling-origin evaluation diperlukan agar stabilitas model dinilai pada beberapa titik waktu.

Keterbatasan keempat adalah penggunaan metrik agregat. CPU total dan RAM total tidak menjelaskan proses mana yang menyebabkan perubahan. Penambahan load average, disk I/O, network throughput, jumlah koneksi, context switches, dan metrik per proses dapat meningkatkan konteks, tetapi juga memperbesar dimensi data. Pemilihan fitur perlu dilakukan secara hati-hati agar model tidak menjadi kompleks tanpa peningkatan akurasi.

Keterbatasan kelima berkaitan dengan ketidakpastian prediksi. Model menghasilkan satu nilai titik tanpa interval kepercayaan. Administrator tidak mengetahui seberapa yakin model terhadap estimasi tersebut. Pendekatan dropout saat inferensi, ensemble beberapa model, atau probabilistic forecasting dapat dipertimbangkan untuk menghasilkan rentang prediksi. Interval yang lebar dapat menandakan pola baru atau data di luar distribusi sehingga tindakan otomatis perlu dibatasi.

Risiko utama interpretasi adalah menganggap loss rendah sebagai jaminan kesiapan produksi. Pada data yang stabil, persistence baseline mungkin memperoleh hasil serupa. Selain itu, nilai error pada skala ternormalisasi tidak dapat disamakan langsung dengan persentase CPU tanpa inverse transform yang benar. Klaim keunggulan terhadap ARIMA juga masih indikatif. Penelitian ini karena itu diposisikan sebagai bukti kelayakan awal bahwa pipeline prediksi dapat dibangun dan dijalankan real-time pada Debian Linux.

Validitas dapat diperkuat melalui eksperimen beban terkontrol. Misalnya, stress-ng digunakan untuk menghasilkan level CPU 20%, 40%, 60%, dan 80%, sedangkan konsumsi memori dinaikkan bertahap. Setiap skenario dijalankan pada durasi tertentu dan disisipkan periode pemulihan. Data dari beban terkontrol digabungkan dengan beban alami sehingga model melihat transisi, puncak, dan penurunan. Pengujian pada hari berbeda dan server berbeda akan menunjukkan apakah model hanya menghafal karakteristik satu lingkungan atau dapat dipindahkan melalui retraining.

3.8. Arah Pengembangan Sistem

Pengembangan pertama adalah memperpanjang pengumpulan data minimal beberapa minggu. Data dipisahkan berdasarkan periode dan tidak diacak. Retraining dijadwalkan ketika jumlah data baru mencukupi atau ketika error monitoring meningkat. Model lama tetap disimpan agar dapat dilakukan rollback. Versi model, scaler, fitur, dan waktu pelatihan dicatat sebagai metadata sehingga hasil inferensi dapat ditelusuri.

Pengembangan kedua adalah evaluasi multi-horizon. Model dapat memprediksi 5 detik, 30 detik, 1 menit, dan 5 menit ke depan. Horizon pendek berguna untuk monitoring cepat, sedangkan horizon lebih panjang memberi waktu untuk provisioning. Arsitektur sequence-to-sequence, direct multi-output, atau recursive forecasting dapat dibandingkan. START dan model host workload adaptif menunjukkan pentingnya prediksi beberapa langkah untuk pengambilan keputusan lebih awal [10], [15].

Pengembangan ketiga adalah penambahan baseline dan model pembanding. Persistence, moving average, ARIMA, linear regression berbasis lag, GRU, CNN-LSTM, dan DLinear dapat diuji pada dataset yang sama. Model terbaik tidak hanya dipilih berdasarkan RMSE, tetapi juga latensi, ukuran file, penggunaan RAM, kestabilan, dan kemudahan pemeliharaan. Untuk server kecil, model sedikit kurang akurat tetapi jauh lebih ringan dapat menjadi pilihan lebih rasional.

Pengembangan keempat adalah integrasi notifikasi dan dashboard. Status WASPADA dapat dikirim melalui email atau aplikasi pesan setelah memenuhi aturan durasi tertentu. Dashboard menampilkan tren aktual-prediksi dan error historis. Ketika error meningkat atau input berada di luar rentang training, dashboard memberikan indikator model drift. Dengan demikian, sistem tidak hanya memprediksi resource, tetapi juga memantau kesehatan modelnya sendiri.

Pengembangan kelima adalah eksplorasi arsitektur modern secara bertahap. Transformer seperti Informer, Autoformer, FEDformer, PatchTST, dan TimesNet dapat diuji ketika data sudah panjang dan bervariasi [6], [7], [9], [12], [13]. Temporal Fusion Transformer dapat digunakan jika interpretabilitas fitur dan multi-horizon menjadi kebutuhan [5]. Namun, semua model harus dibandingkan dengan baseline sederhana sebagaimana disarankan oleh temuan DLinear [11]. Pendekatan tersebut menjaga agar peningkatan kompleksitas selalu disertai manfaat yang terukur.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

3.9. Rancangan Evaluasi Lanjutan dan Kesiapan Produksi

Rancangan evaluasi lanjutan perlu memisahkan akurasi CPU dan RAM karena karakteristik keduanya tidak sama. CPU dapat berubah cepat dalam hitungan detik, sedangkan RAM sering meningkat secara bertahap dan bertahan lebih lama. Pelaporan satu nilai agregat dapat menyembunyikan kelemahan pada salah satu target. MSE, RMSE, dan MAE dihitung untuk masing-masing variabel setelah inverse transform. Selain itu, directional accuracy dapat digunakan untuk menilai apakah model mampu memperkirakan arah kenaikan atau penurunan, sedangkan precision dan recall peringatan dapat dihitung setelah kondisi tinggi didefinisikan berdasarkan threshold.

Protokol rolling-origin evaluation dapat dimulai dengan sebagian data awal sebagai train set, kemudian model diuji pada blok waktu berikutnya. Jendela pelatihan digeser atau diperluas dan proses diulang beberapa kali. Cara ini lebih sesuai untuk time series dibanding satu pembagian 80:20 karena menunjukkan apakah performa stabil pada periode yang berbeda. Setiap fold harus mempertahankan scaler yang hanya difit pada data pelatihan. Jika scaler menggunakan seluruh data sebelum pembagian, informasi minimum dan maksimum dari masa depan dapat bocor ke proses pelatihan.

Kesiapan produksi juga memerlukan pemantauan model drift. Distribusi workload dapat berubah setelah aplikasi baru dipasang, jumlah pengguna meningkat, atau konfigurasi server dimodifikasi. Sistem dapat menyimpan error aktual setiap kali target berikutnya tersedia. Jika MAE bergerak melewati batas tertentu atau proporsi input di luar rentang training meningkat, retraining dipicu. Dengan mekanisme ini, model tidak dianggap sebagai artefak statis, tetapi sebagai komponen layanan yang harus dipelihara dan dievaluasi secara berkala.

Reproduksibilitas perlu diperkuat melalui penyimpanan seed, versi Python, versi library, konfigurasi hyperparameter, hash dataset, serta tanggal pelatihan. Seluruh eksperimen dicatat dalam berkas konfigurasi atau alat pelacakan eksperimen. Dokumentasi ini penting ketika hasil loss berbeda antara percobaan awal dan evaluasi akhir, sebagaimana terlihat pada dokumentasi terminal. Pencatatan konfigurasi memungkinkan perbedaan tersebut dijelaskan secara objektif dan mencegah pemilihan hasil terbaik tanpa konteks.

Dengan langkah-langkah tersebut, kontribusi sistem dapat berkembang dari demonstrasi teknis menjadi alat bantu pengelolaan kapasitas yang dapat dipertanggungjawabkan. Model yang akurat tetapi tidak dapat dipantau, direproduksi, atau dipulihkan akan sulit digunakan dalam operasi. Sebaliknya, pipeline sederhana dengan baseline jelas, logging lengkap, evaluasi berkala, dan prosedur rollback dapat memberikan manfaat nyata meskipun arsitektur modelnya tidak paling kompleks.

Kesiapan produksi selanjutnya perlu dinilai melalui matriks skenario yang merepresentasikan kondisi operasional. Skenario beban rendah digunakan untuk mengukur kestabilan dan konsumsi sumber daya monitoring, beban sedang untuk menilai respons terhadap perubahan normal, sedangkan beban tinggi dipakai untuk menguji ketepatan peringatan sebelum CPU atau RAM mencapai batas. Gangguan terencana seperti penghentian proses, restart layanan, perubahan jumlah koneksi, dan pekerjaan pencadangan juga perlu dimasukkan. Setiap skenario dicatat bersama waktu mulai, durasi, konfigurasi perangkat, nilai aktual, nilai prediksi, dan tindakan yang diharapkan agar evaluasi dapat diulang secara konsisten.

Kalibrasi threshold tidak dapat dipisahkan dari tujuan layanan. Ambang CPU 80% dan RAM 85% pada prototipe berfungsi sebagai nilai operasional awal, tetapi server basis data, web, dan komputasi batch dapat memiliki karakteristik yang berbeda. Threshold sebaiknya ditetapkan berdasarkan data historis, kapasitas headroom, waktu yang dibutuhkan administrator untuk merespons, dan konsekuensi kegagalan layanan. Penggunaan hysteresis dapat mencegah status berulang kali berubah ketika nilai berada tepat di sekitar ambang. Selain itu, peringatan dapat mensyaratkan beberapa prediksi tinggi secara berturut-turut agar fluktuasi sesaat tidak menghasilkan alarm palsu.

Beban tambahan dari sistem monitoring harus diukur sebagai bagian dari hasil, bukan dianggap selalu kecil. collect.py menjalankan pembacaan psutil dan penulisan CSV setiap lima detik, sedangkan predict.py melakukan normalisasi dan inferensi. Pada lingkungan eksperimen, proses tersebut tidak menghambat sistem, tetapi dampaknya dapat berbeda pada perangkat dengan RAM terbatas atau CPU satu inti. Pengukuran penggunaan CPU, RAM, ukuran log, pertumbuhan file CSV, dan waktu inferensi selama beberapa hari akan menunjukkan biaya operasional sebenarnya. Hasil pengukuran dapat digunakan untuk menentukan interval sampling, strategi rotasi data, dan apakah inferensi perlu dipindahkan ke host terpisah.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

Aspek keandalan juga mencakup penanganan kegagalan komponen. Sistem perlu mendeteksi file model yang hilang, scaler yang tidak sesuai, data CSV rusak, timestamp tidak berurutan, serta nilai metrik yang tidak wajar. Kesalahan tersebut harus menghasilkan pesan yang jelas dan tidak langsung menghentikan layanan monitoring. Penyimpanan model dengan versi, checksum, dan konfigurasi fitur membantu mencegah artefak yang salah digunakan. Untuk pemulihan, service manager dapat menjalankan ulang proses secara otomatis, sementara health check memastikan bahwa data baru benar-benar tercatat dan prediksi diperbarui sesuai interval.

Dari sisi manfaat praktis, keluaran prediksi perlu disajikan dalam bentuk yang mudah digunakan administrator. Angka aktual dan prediksi sebaiknya dilengkapi tren, selisih error, waktu pembaruan terakhir, serta alasan status peringatan. Informasi ini membantu pengguna membedakan antara lonjakan yang nyata, prediksi yang belum pasti, dan masalah pada pipeline. Uji pengguna sederhana dengan administrator server dapat menilai apakah tampilan mendukung pengambilan keputusan, berapa waktu yang dibutuhkan untuk memahami alarm, dan tindakan apa yang paling sering dipilih. Dengan demikian, keberhasilan sistem tidak hanya diukur dari loss model, tetapi juga dari kemampuannya memberi informasi yang tepat waktu, dapat dipercaya, dan dapat ditindaklanjuti.

4. Kesimpulan

1). Sistem berhasil mengumpulkan 1.859 record data monitoring CPU, RAM, kapasitas memori terpakai, dan jumlah proses dengan interval lima detik melalui modul collect.py. 2). Arsitektur dua lapisan LSTM berbasis PyTorch dapat dilatih selama 50 epoch dan menghasilkan training loss 0,0140, test loss 0,0310, RMSE 0,1761, serta MAE 0,1426 pada skala ternormalisasi. 3). Modul predict.py mampu menjalankan inferensi real-time setiap lima detik dan menampilkan status NORMAL atau WASPADA berdasarkan threshold CPU 80% dan RAM 85%. 4). Arsitektur modular yang memisahkan pengumpulan data, pelatihan, dan inferensi memudahkan pemeliharaan serta menjadi dasar integrasi dashboard, notifikasi, dan tindakan proaktif. 5). Hasil penelitian merupakan proof of concept karena data dikumpulkan dalam waktu relatif singkat dan didominasi beban rendah. Validasi produksi memerlukan data lebih panjang, variasi workload tinggi, baseline pada dataset yang sama, evaluasi rolling, dan pengujian pada server berbeda. Pengembangan berikutnya diarahkan pada penambahan metrik disk dan jaringan, evaluasi multi-horizon, retraining berkala, pengujian ARIMA dan baseline sederhana pada data yang sama, serta integrasi dashboard dan notifikasi. Eksplorasi GRU, CNN-LSTM, DLinear, dan Transformer dapat dilakukan setelah jumlah serta keragaman data mencukupi sehingga peningkatan kompleksitas model dapat dinilai secara objektif.

Referensi

[1] M. E. Karim, M. M. S. Maswood, S. Das, and A. G. Alharbi, “BHyPreC: A Novel Bi-LSTM Based Hybrid Recurrent Neural Network Model to Predict the CPU Workload of Cloud Virtual Machine,” IEEE Access, vol. 9, pp. 131476–131495, 2021, doi: 10.1109/ACCESS.2021.3113714.

[2] S. Ouhame, Y. Hadi, and A. Ullah, “An Efficient Forecasting Approach for Resource Utilization in Cloud Data Center Using CNN-LSTM Model,” Neural Computing and Applications, vol. 33, no. 16, pp. 10043–10055, 2021, doi: 10.1007/s00521-021-05770-9.

[3] J. Bi, S. Li, H. Yuan, and M. Zhou, “Integrated Deep Learning Method for Workload and Resource Prediction in Cloud Systems,” Neurocomputing, vol. 424, pp. 35–48, 2021, doi: 10.1016/j.neucom.2020.11.011.

[4] J. Kumar, A. K. Singh, and R. Buyya, “Self Directed Learning Based Workload Forecasting Model for Cloud Resource Management,” Information Sciences, vol. 543, pp. 345–366, 2021, doi: 10.1016/j.ins.2020.07.012.

[5] B. Lim, S. Ö. Arik, N. Loeff, and T. Pfister, “Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting,” International Journal of Forecasting, vol. 37, no. 4, pp. 1748–1764, 2021, doi: 10.1016/j.ijforecast.2021.03.012.

[6] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting,” in Advances in Neural Information Processing Systems, vol. 34, pp. 22419–22430, 2021.

[7] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 12, pp. 11106–11115, 2021, doi: 10.1609/aaai.v35i12.17325.

[8] M. Xu, C. Song, H. Wu, S. S. Gill, K. Ye, and C. Xu, “esDNN: Deep Neural Network Based Multivariate Workload Prediction Approach in Cloud Computing Environments,” ACM Transactions on Internet Technology, vol. 22, no. 3, 2022, doi: 10.1145/3524114.

[9] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “FEDformer: Frequency Enhanced Decomposed Transformer for Long-Term Series Forecasting,” in Proceedings of the 39th International Conference on Machine Learning, PMLR, vol. 162, pp. 27268–27286, 2022.

[10] S. Tuli, S. S. Gill, P. Garraghan, R. Buyya, G. Casale, and N. R. Jennings, “START: Straggler Prediction and Mitigation for Cloud Computing Environments Using Encoder LSTM Networks,” IEEE Transactions on Services Computing, vol. 16, no. 1, pp. 615–627, 2023, doi: 10.1109/TSC.2021.3129897.

[11] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are Transformers Effective for Time Series Forecasting?” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 9, pp. 11121–11128, 2023, doi: 10.1609/aaai.v37i9.26317.

[12] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A Time Series Is Worth 64 Words: Long-Term Forecasting with Transformers,” in International Conference on Learning Representations, 2023.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)

[13] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis," in International Conference on Learning Representations, 2023.

[14] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "A Multilevel Learning Model for Predicting CPU Utilization in Cloud Data Centers," in 2023 IEEE International Conference on Dependable, Autonomic and Secure Computing and Related Conferences, 2023, doi: 10.1109/DASC/PiCom/CBDCom/Cy59711.2023.10361305.

[15] A. Setayesh, H. Hadian, and R. Prodan, "An Efficient Online Prediction of Host Workloads Using Pruned GRU Neural Nets," arXiv preprint arXiv:2303.16601, 2023.

[16] R. Karthikeyan, V. Balamurugan, R. Cyriac, and B. Sundaravadivazhagan, "COSCO2: AI-Augmented Evolutionary Algorithm Based Workload Prediction Framework for Sustainable Cloud Data Centers," Transactions on Emerging Telecommunications Technologies, vol. 34, no. 1, e4652, 2023, doi: 10.1002/ett.4652.

[17] S. Simaiya, U. K. Lilhore, Y. K. Sharma, K. B. V. Brahma Rao, V. V. R. Maheswara Rao, A. Baliyan, A. Bijalwan, and R. Alroobaea, "A Hybrid Cloud Load Balancing and Host Utilization Prediction Method Using Deep Learning and Optimization Techniques," Scientific Reports, vol. 14, art. 1337, 2024, doi: 10.1038/s41598-024-51466-0.

DOI: https://doi.org/10.31004/riggs.v5i2.11453
Lisensi: Creative Commons Attribution 4.0 International (CC BY 4.0)