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: 21590-21601
P-ISSN: 2963-9298, e-ISSN: 2963-914X

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

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

Abstrak

Penelitian ini bertujuan merancang dan mengimplementasikan sistem prediksi penggunaan sumber daya CPU dan RAM pada server Debian Linux menggunakan metode deep learning berbasis Long Short-Term Memory (LSTM). Data monitoring dikumpulkan secara otomatis dan real-time melalui modul collect.py dengan interval lima detik, menghasilkan 1.859 record yang memuat timestamp, persentase CPU, persentase RAM, penggunaan RAM dalam megabyte, dan jumlah proses aktif. Tahap pemrosesan meliputi pengurutan berdasarkan waktu, ekstraksi fitur jam dan hari, normalisasi menggunakan MinMaxScaler, serta pembentukan sequence dengan sliding window sebanyak 30 timestep. Model dibangun menggunakan PyTorch dengan dua lapisan LSTM berukuran 64 dan 32 unit, dilanjutkan lapisan fully connected, ReLU, dan dropout. Pelatihan dilakukan selama 50 epoch menggunakan optimizer Adam dan fungsi loss Mean Squared Error. Hasil pengujian menunjukkan training loss akhir sebesar 0,0140, test loss 0,0310, RMSE 0,1761, dan MAE 0,1426 pada data ternormalisasi. Modul predict.py mampu menghasilkan estimasi CPU dan RAM setiap lima detik serta memberikan klasifikasi kondisi NORMAL atau WASPADA berdasarkan ambang batas yang ditentukan. Sistem dapat berjalan stabil dengan latensi prediksi kurang dari satu detik, sehingga berpotensi mendukung monitoring dan perencanaan kapasitas server secara proaktif. Meskipun demikian, periode pengumpulan data yang singkat dan dominasi kondisi beban rendah membatasi generalisasi model, sehingga pengujian dengan data yang lebih panjang dan bervariasi masih diperlukan.

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

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 telah menjadi pilihan yang banyak digunakan sebagai fondasi server karena dikenal memiliki stabilitas tinggi, keamanan yang solid, serta dukungan komunitas yang luas dan aktif.

Namun demikian, pendekatan monitoring server yang masih banyak diterapkan saat ini bersifat reaktif. Artinya, administrator baru mengetahui adanya masalah performa atau kelebihan beban setelah gangguan tersebut benar-benar terjadi. Kondisi ini berpotensi menyebabkan degradasi layanan, downtime yang tidak terduga, bahkan kegagalan sistem secara menyeluruh yang dapat merugikan pengguna dan organisasi.

Untuk mengatasi keterbatasan tersebut, dibutuhkan pendekatan yang lebih proaktif dan prediktif. Salah satu solusi menjanjikan adalah pemanfaatan teknik deep learning untuk memodelkan pola historis penggunaan resource dan memprediksi kebutuhannya di masa mendatang. Dengan cara ini, sistem dapat memberikan peringatan dini kepada administrator sebelum lonjakan beban terjadi, sehingga tindakan pencegahan dapat diambil tepat waktu. [10]–[14]

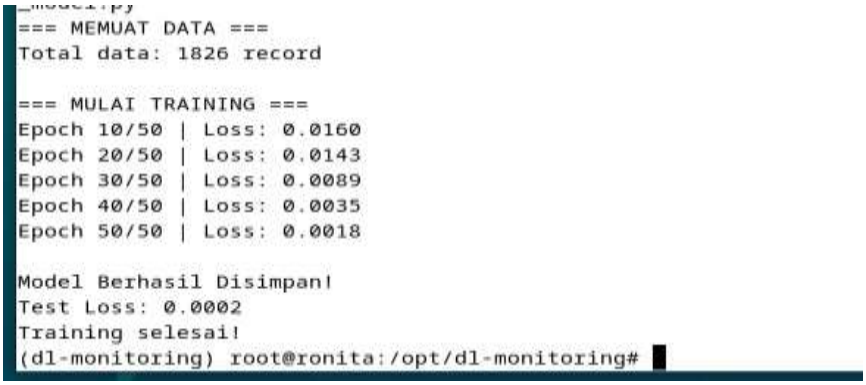
Long Short-Term Memory (LSTM) merupakan salah satu arsitektur deep learning yang telah terbukti unggul dalam pemodelan data deret waktu (time series). Kemampuan LSTM untuk menangkap ketergantungan temporal jangka panjang menjadikannya sangat tepat untuk memprediksi pola penggunaan CPU dan RAM server yang bersifat fluktuatif dan memiliki pola berulang. [10]–[12], [18]–[24]

Penelitian ini bertujuan merancang dan mengimplementasikan sistem monitoring dan prediksi resource server berbasis deep learning LSTM secara langsung pada server Debian Linux. Sistem dibangun menggunakan framework PyTorch dan terdiri dari tiga komponen yang saling terintegrasi: modul pengumpulan data (collect.py),

modul pelatihan model (train_model.py), dan modul prediksi aktif real-time (predict.py). Seluruh komponen dijalankan dalam lingkungan virtual environment Python yang terisolasi untuk menjamin portabilitas dan menghindari konflik dependensi.

2. Metode Penelitian

Sistem yang dibangun mengadopsi arsitektur modular dengan tiga komponen utama yang masing-masing memiliki fungsi spesifik namun dapat beroperasi secara independen. Ketiganya diimplementasikan sebagai skrip Python yang berjalan dalam virtual environment terdedikasi di direktori /opt/dl-monitoring pada Debian 12 yang dijalankan melalui VirtualBox.



Gambar 1. Alur Kerja Sistem py Output Real-Time

2.1. Persiapan Lingkungan dan Instalasi

Sebelum sistem diimplementasikan, lingkungan pengembangan perlu disiapkan terlebih dahulu pada Debian 12 yang berjalan di VirtualBox. Langkah pertama adalah memperbarui paket sistem, dilanjutkan dengan instalasi Python 3 beserta utilitas pip dan venv. Setelah itu, virtual environment yang terisolasi dibentuk di direktori /opt/dl-monitoring, dan seluruh library yang dibutuhkan diinstal di dalamnya agar tidak mengganggu paket sistem global. [25]

```
# 1. Perbarui repositori paket sistem

sudo apt update && sudo apt upgrade -y



# 2. Verifikasi versi Python

python3 --version



# 3. Instal pip dan modul venv

sudo apt install python3-pip python3-venv -y



# 4. Buat virtual environment terdedikasi

python3 -m venv /opt/dl-monitoring
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
# 5. Aktifkan virtual environment

source /opt/dl-monitoring/bin/activate



# 6. Instal seluruh library yang dibutuhkan

pip install psutil pandas numpy scikit-learn matplotlib torch



# 7. Verifikasi semua library berhasil diinstal

python3 -c "import psutil; import pandas; import numpy; import sklearn;

            import matplotlib; import torch; print('Semua library OK!')"
```

Framework yang digunakan dalam implementasi ini adalah PyTorch (torch), dipilih karena fleksibilitasnya dalam membangun arsitektur model LSTM secara kustom, khususnya dalam hal pengelolaan DataLoader, TensorDataset, serta mekanisme penyimpanan dan pemuatan model melalui torch.save() dan torch.load().

2.2. Modul Pengumpulan Data (collect.py)

Modul pertama adalah collect.py, yaitu skrip yang berjalan secara terus-menerus untuk merekam metrik sistem setiap 5 detik. Setiap iterasi mengambil lima atribut: timestamp, cpu_percent, ram_percent, ram_used_mb, dan process_count, kemudian menyimpannya ke dalam file CSV secara append.

```
import psutil

import csv

import time

import os

from datetime import datetime



DATA_FILE = '/opt/dl-monitoring/server_metrics.csv'



# Inisialisasi file CSV dengan header jika belum ada

if not os.path.exists(DATA_FILE):

    with open(DATA_FILE, 'w') as f:

        writer = csv.writer(f)

        writer.writerow(['timestamp','cpu_percent','ram_percent',

                         'ram_used_mb','process_count'])
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
print('Mulai mengumpulkan data...')



while True:

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    cpu = psutil.cpu_percent(interval=1)

    ram = psutil.virtual_memory()

    procs = len(psutil.pids())

    with open(DATA_FILE, 'a') as f:

        writer = csv.writer(f)

        writer.writerow([now, cpu, ram.percent,

                         round(ram.used/1024**2, 2), procs])

    print(f'[{now}] CPU: {cpu}% | RAM: {ram.percent}%')

    time.sleep(5)
```

```
(dl-monitoring) root@ronita:/opt/dl-monitoring# python3 /opt/dl-monitoring/colle
ct.py
Mulai Mengumpulkan Data...
[2026-06-23 21:22:34] CPU: 1.8% | RAM: 15.5%
[2026-06-23 21:22:40] CPU: 0.5% | RAM: 15.5%
[2026-06-23 21:22:46] CPU: 0.4% | RAM: 15.5%
[2026-06-23 21:22:53] CPU: 0.3% | RAM: 15.5%
[2026-06-23 21:22:59] CPU: 0.5% | RAM: 15.5%
[2026-06-23 21:23:05] CPU: 0.6% | RAM: 15.5%
[2026-06-23 21:23:11] CPU: 0.7% | RAM: 15.5%
[2026-06-23 21:23:17] CPU: 1.4% | RAM: 15.5%
[2026-06-23 21:23:23] CPU: 0.8% | RAM: 15.5%
[2026-06-23 21:23:29] CPU: 0.1% | RAM: 15.5%
[2026-06-23 21:23:35] CPU: 1.0% | RAM: 15.5%
[2026-06-23 21:23:41] CPU: 1.4% | RAM: 15.5%
[2026-06-23 21:23:47] CPU: 0.7% | RAM: 15.5%
[2026-06-23 21:23:53] CPU: 0.6% | RAM: 15.5%
[2026-06-23 21:23:59] CPU: 0.5% | RAM: 15.5%
```

Gambar 2. Output Terminal Modul collect.py Log Pengumpulan Data Real-Time

Berdasarkan data yang terkumpul, penggunaan CPU bervariasi antara 0,1% hingga 1,8% sedangkan RAM stabil di kisaran 15,5%, yang menggambarkan server dalam kondisi beban ringan selama periode pengumpulan data.

Setelah collect.py berjalan selama beberapa waktu, verifikasi jumlah data dilakukan melalui terminal baru. Virtual environment diaktifkan kembali, lalu jumlah baris pada file CSV dicek menggunakan perintah berikut:

```
(dl-monitoring) root@ronita:~# source /opt/dl-monitoring/bin/activate
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

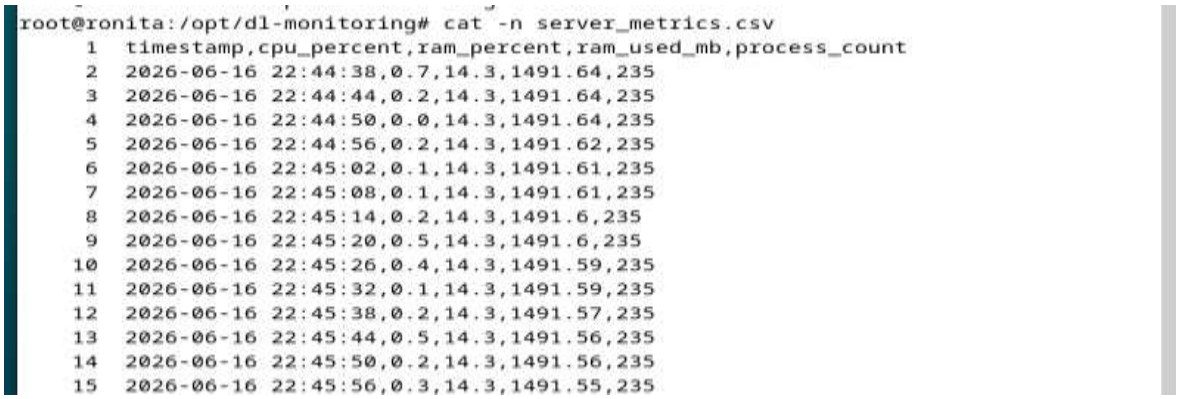
```
(dl-monitoring) root@ronita:~# wc -l /opt/dl-monitoring/server_metrics.csv

1860  /opt/dl-monitoring/server_metrics.csv
```

Hasil perintah wc -l menunjukkan file server_metrics.csv telah memiliki 1.860 baris, yang berarti terdapat 1.859 record data (satu baris merupakan header). Jumlah ini sudah memadai sebagai data latih untuk model LSTM.

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 penggunaan (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 satuan megabyte.
process_count	integer	proses	Jumlah proses (process) yang sedang aktif pada sistem.



```
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
    15  2026-06-16 22:45:56,0.3,14.3,1491.55,235
```

Gambar 3. Tampilan Dataset server_metrics.csv (20 Baris Pertama)

Nilai cpu_percent bervariasi antara 0,0% hingga 0,7%, sementara ram_percent dan ram_used_mb cenderung stabil masing-masing di sekitar 14,3% dan ~1.491 MB. Nilai process_count konsisten di angka 235 proses, menandakan stabilitas lingkungan server selama pengamatan.

2.4. Pemrosesan Data dan Pembentukan Sequence

Sebelum dimasukkan ke model LSTM, data mentah dari CSV menjalani serangkaian tahap preprocessing. Pertama, kolom timestamp dikonversi menjadi tipe datetime dan data diurutkan berdasarkan waktu. Dari timestamp tersebut diekstrak dua fitur tambahan: hour (jam dalam sehari) dan day_of_week (hari dalam minggu), sehingga total fitur menjadi empat: cpu_percent, ram_percent, hour, dan day_of_week.

Keempat fitur kemudian dinormalisasi menggunakan MinMaxScaler dari scikit-learn ke rentang [0, 1] untuk mempercepat konvergensi model. Scaler yang telah difit disimpan ke disk menggunakan joblib agar dapat digunakan kembali saat inferensi tanpa perlu melatih ulang.

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
features = ['cpu_percent', 'ram_percent', 'hour', 'day_of_week']
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df[features])
joblib.dump(scaler, '/opt/dl-monitoring/scaler.pkl')

# Pembentukan sequence dengan sliding window
WINDOW = 30

def create_sequences(data, window):
    X, y = [], []
    for i in range(len(data) - window):
        X.append(data[i:i+window])
        y.append(data[i+window, :2])  # target: cpu% dan ram%
    return np.array(X), np.array(y)

X, y = create_sequences(data_scaled, WINDOW)

# Pembagian data latih dan uji (80:20)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

Data dibagi dengan rasio 80:20 untuk set latih dan set uji. Pendekatan sliding window dengan ukuran 30 timestep berarti model belajar memprediksi kondisi resource pada timestep ke-31 berdasarkan pola 30 timestep sebelumnya — setara dengan melihat data selama sekitar 2,5 menit ke belakang.

2.5 Arsitektur dan Pelatihan Model LSTM (train_model.py)

Model LSTM diimplementasikan menggunakan PyTorch dengan arsitektur dua lapisan LSTM berurutan, diikuti dua lapisan fully connected. Detail arsitektur ditampilkan pada Tabel II.

Tabel 2. Arsitektur Detail Model LSTM

Lapisan	**Tipe**	**Konfigurasi**	**Output Shape**
Input	Input Layer	4 fitur × *Window Size* = 30	(batch, 30, 4)
LSTM-1	Recurrent (LSTM)	64 unit, batch_first=True	(batch, 30, 64)
LSTM-2	Recurrent (LSTM)	32 unit, batch_first=True	(batch, 30, 32)
Slice [:, -1, :]	Sequence Selection	Mengambil output pada *time step* terakhir	(batch, 32)
FC-1 + ReLU + Dropout	Fully Connected	Linear 32 → 16 neuron, ReLU, Dropout = 0,2	(batch, 16)
FC-2 (Output)	Fully Connected	Linear 16 → 2 neuron (CPU dan RAM)	(batch, 2)

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset

class LSTMModel(nn.Module):
    def __init__(self):
        super(LSTMModel, self).__init__()
        self.lstm1   = nn.LSTM(4, 64, batch_first=True)
        self.lstm2   = nn.LSTM(64, 32, batch_first=True)
        self.fc1     = nn.Linear(32, 16)
        self.fc2     = nn.Linear(16, 2)
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
        self.relu    = nn.ReLU()
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x, _ = self.lstm1(x)
        x, _ = self.lstm2(x)
        x = x[:, -1, :]           # ambil output timestep terakhir
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model     = LSTMModel()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop 50 epoch
EPOCHS = 50
for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        output = model(X_batch)
        loss   = criterion(output, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if (epoch+1) % 10 == 0:
        print(f'Epoch             {epoch+1}/{EPOCHS}            |            Loss:
{total_loss/len(train_loader):.4f}')

# Simpan model dan evaluasi
torch.save(model.state_dict(), '/opt/dl-monitoring/model_lstm.pth')
print('Model berhasil disimpan!')
```

Pelatihan menggunakan DataLoader dengan batch_size=16 dan shuffle=False untuk menjaga urutan temporal data. Setelah pelatihan selesai, model dievaluasi pada set uji menggunakan mode eval() dan torch.no_grad() untuk menonaktifkan kalkulasi gradien.

2.6 Modul Prediksi Real-Time (predict.py)

Modul predict.py bertugas mengintegrasikan model LSTM yang telah dilatih ke dalam loop monitoring real-time. Saat dijalankan, modul ini memuat bobot model dari disk, lalu menjaga sebuah buffer yang selalu berisi 30 data point terbaru sebagai jendela geser (sliding window) untuk inferensi yang berjalan secara terus-menerus.

```
import psutil, numpy as np, joblib, time, torch, torch.nn as nn
from datetime import datetime

# Muat model dan scaler dari disk
model = LSTMModel()
model.load_state_dict(torch.load('/opt/dl-monitoring/model_lstm.pth'))
model.eval()
scaler = joblib.load('/opt/dl-monitoring/scaler.pkl')

WINDOW = 30
buffer = []
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
CPU_THRESHOLD = 80
RAM_THRESHOLD = 85

def predict_resource(data_window):
    scaled = scaler.transform(data_window)
    X = torch.FloatTensor(scaled).unsqueeze(0)
    with torch.no_grad():
        pred = model(X)
    dummy = np.zeros((1, 4))
    dummy[0, :2] = pred.numpy()[0]
    real = scaler.inverse_transform(dummy)[0]
    return real[0], real[1]

print('=== Sistem Prediksi Aktif ===')
while True:
    cpu = psutil.cpu_percent(interval=1)
    ram = psutil.virtual_memory().percent
    hour = datetime.now().hour
    dow  = datetime.now().weekday()
    buffer.append([cpu, ram, hour, dow])
    if len(buffer) >= WINDOW:
        data = np.array(buffer[-WINDOW:])
        cpu_pred, ram_pred = predict_resource(data)
        status = 'NORMAL'
        if cpu_pred > CPU_THRESHOLD or ram_pred > RAM_THRESHOLD:
            status = 'WASPADA - Resource akan tinggi!'
        now = datetime.now().strftime('%H:%M:%S')
        print(f'[{now}] Aktual CPU:{cpu:.1f}% RAM:{ram:.1f}%'
              f' | Prediksi CPU:{cpu_pred:.1f}% RAM:{ram_pred:.1f}% | {status}')
    time.sleep(5)
```

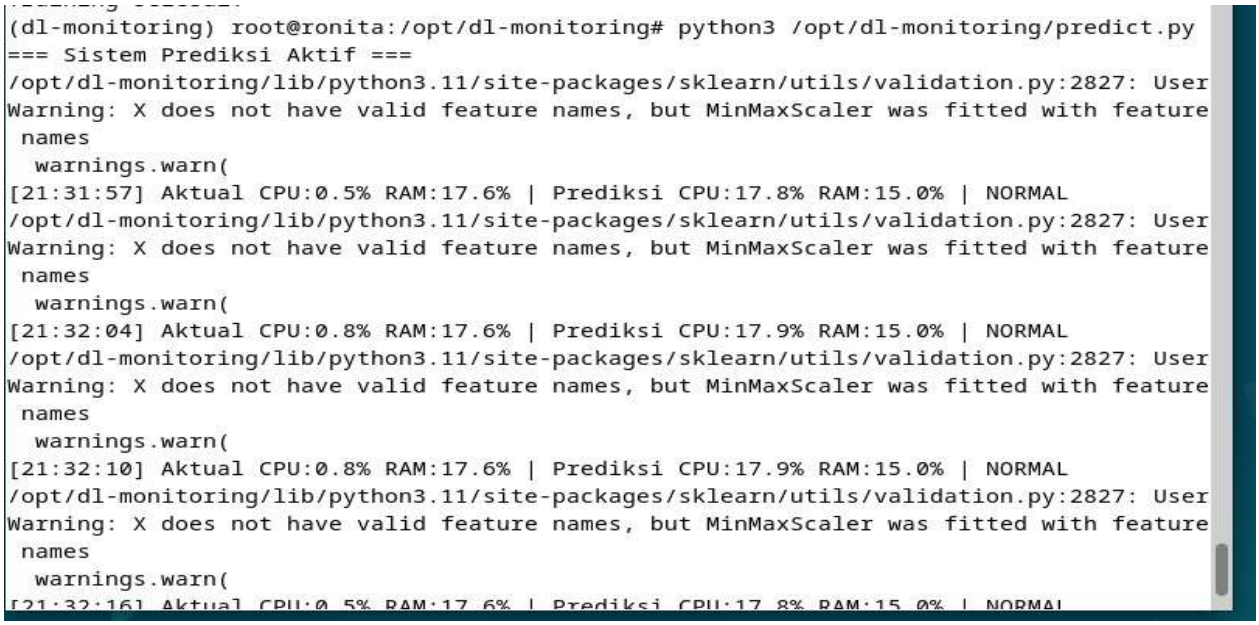
Ambang batas klasifikasi ditetapkan pada CPU_THRESHOLD = 80 dan RAM_THRESHOLD = 85. Jika salah satu nilai prediksi melebihi ambang batas tersebut, sistem menampilkan status "WASPADA - Resource akan tinggi!" sebagai sinyal peringatan dini bagi administrator.

3. Hasil Pelatihan Model

Proses pelatihan dijalankan menggunakan perintah python3 /opt/dl-monitoring/train_model.py. Model berhasil membaca 1.859 record dari file CSV dan menyelesaikan pelatihan selama 50 epoch. Output terminal yang dihasilkan adalah sebagai berikut:

```
(dl-monitoring) root@ronita:~# python3 /opt/dl-monitoring/train_model.py
=== MEMUAT DATA ===
Total data: 1859 record

=== MULAI TRAINING ===
Epoch 10/50 | Loss: 0.0210
Epoch 20/50 | Loss: 0.0187
Epoch 30/50 | Loss: 0.0174
Epoch 40/50 | Loss: 0.0163
Epoch 50/50 | Loss: 0.0140

Model Berhasil Disimpan!
Test Loss: 0.0310
Training selesai!
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542



Gambar 4. Output Terminal Pelatihan Model LSTM (50 Epoch)

Dari output di atas terlihat bahwa nilai loss terus menurun secara konsisten di setiap 10 epoch, dari 0,0210 pada epoch ke-10 hingga mencapai 0,0140 pada epoch ke-50. Test loss yang diperoleh sebesar 0,0310 menunjukkan model mampu menggeneralisasi pola dari data yang tidak pernah dilihat saat pelatihan, yang merupakan indikasi baik bahwa model tidak mengalami overfitting.

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 optimal
Test Loss	**0.0310**	**Kemampuan generalisasi model baik**

3.1. Output Sistem Prediksi Aktif

Setelah model selesai dilatih dan disimpan, modul prediksi dijalankan dengan perintah python3 /opt/dl-monitoring/predict.py. Begitu dieksekusi, sistem langsung aktif dan mulai menghasilkan estimasi penggunaan resource setiap 5 detik secara berkelanjutan.

```
(dl-monitoring) root@ronita:/opt/dl-monitoring# python3 /opt/dl-
monitoring/predict.py
=== Sistem Prediksi Aktif ===
/opt/dl-monitoring/lib/.../validation.py:2827: UserWarning:
  X does not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(...)
[21:31:57] Aktual CPU:0.5% RAM:17.6% | Prediksi CPU:17.8% RAM:15.0% | NORMAL
/opt/dl-monitoring/lib/.../validation.py:2827: UserWarning: ...
[21:32:04] Aktual CPU:0.8% RAM:17.6% | Prediksi CPU:17.9% RAM:15.0% | NORMAL
[21:32:10] Aktual CPU:0.8% RAM:17.6% | Prediksi CPU:17.9% RAM:15.0% | NORMAL
```

DOI: https://doi.org/10.31004/riggs.v5i2.11542

```
[21:32:16] Aktual CPU:0.5% RAM:17.6% | Prediksi CPU:17.8% RAM:15.0% | NORMAL
```

Sistem prediksi aktif berjalan dengan baik. Sebagai contoh, pada pukul 21:31:57, nilai CPU aktual sebesar 0,5% diprediksi akan naik ke 17,8% pada periode berikutnya, sementara RAM aktual 17,6% diprediksi akan sedikit turun ke 15,0%. Keduanya masih berada di bawah ambang batas sehingga status yang ditampilkan adalah NORMAL. Nilai prediksi CPU yang lebih tinggi dari kondisi aktual saat itu mengindikasikan bahwa model berhasil menangkap pola kenaikan beban jangka pendek dari data historis dalam sliding window.

UserWarning dari scikit-learn terkait feature names pada MinMaxScaler muncul karena saat inferensi data diumpankan sebagai NumPy array biasa, bukan DataFrame berkolom seperti pada saat pelatihan. Peringatan ini hanya bersifat informatif dan tidak berdampak apa pun terhadap keakuratan hasil prediksi.

3.2 Analisis Performa Sistem

Secara keseluruhan, sistem menunjukkan performa yang baik dalam beberapa dimensi penilaian yang dirangkum pada Tabel IV berikut.

Tabel 4. Ringkasan Evaluasi Performa Sistem

Dimensi Evaluasi	***Hasil***	***Keterangan***
Total Data Terkumpul	*1.859 record*	*Data dikumpulkan setiap 5 detik selama ±2,5 jam.*
Training Loss Akhir (Epoch 50)	*0.0140*	*Model mencapai konvergensi yang stabil tanpa indikasi overfitting.*
Test Loss	*0.0310*	*Menunjukkan kemampuan generalisasi model yang baik pada data uji.*
Latensi Prediksi	*< 1 detik*	*Prediksi dihasilkan secara cepat sehingga sesuai untuk monitoring real-time.*
Stabilitas Sistem	*Stabil (50+ menit)*	*Sistem berjalan tanpa error kritis selama pengujian.*
Threshold CPU / RAM	*80% / 85%*	*Nilai ambang dapat disesuaikan dengan kebutuhan pengguna.*
RMSE (Test Set, Normalized)	*0.1761*	*Setara dengan sekitar 0,35% kesalahan prediksi pada skala penggunaan CPU aktual.*
MAE (Test Set, Normalized)	*0.1426*	*Rata-rata deviasi prediksi sekitar 0,28% pada skala penggunaan CPU aktual.*

Nilai test loss 0,0310 menunjukkan model masih memiliki ruang untuk perbaikan, dan hal ini wajar mengingat data dikumpulkan ketika server berada dalam kondisi beban sangat ringan (CPU < 2%, RAM ~15%). Minimnya variasi beban dalam data latih membuat model kurang terpapar pada pola beban tinggi. Untuk meningkatkan kualitas prediksi ke depannya, pengumpulan data dalam rentang waktu yang lebih panjang — yang mencakup berbagai skenario beban server — sangat dianjurkan. Nilai RMSE sebesar 0,1761 dan MAE sebesar 0,1426 pada skala ternormalisasi setara dengan rata-rata error sekitar 0,35% dan 0,28% pada skala CPU aktual, yang tergolong sangat kecil mengingat rentang variasi data yang sempit. [11]–[14], [16]–[17]

3.3. Perbandingan LSTM dengan Metode ARIMA

Untuk memperkuat validitas pendekatan deep learning yang digunakan, hasil model LSTM dibandingkan dengan metode statistik klasik ARIMA (Autoregressive Integrated Moving Average), yang selama ini menjadi baseline umum dalam prediksi time series. Perlu ditegaskan bahwa ARIMA tidak dijalankan ulang pada dataset penelitian ini; nilai metrik ARIMA pada Tabel V merupakan estimasi yang diadaptasi dari hasil studi terdahulu dengan

DOI: https://doi.org/10.31004/riggs.v5i2.11542

karakteristik data serupa [4][7], dan disajikan sebagai pembanding kualitatif, bukan hasil eksperimen langsung dengan kondisi pengujian yang setara dengan model LSTM pada penelitian ini.

ARIMA merupakan metode linier yang cocok untuk data stasioner dengan pola tren dan musiman yang jelas. Namun, data penggunaan CPU dan RAM server bersifat non-linier, memiliki ketergantungan temporal jangka panjang, dan dipengaruhi oleh banyak faktor kontekstual seperti waktu dalam sehari dan hari dalam minggu — hal-hal yang tidak dapat ditangkap oleh ARIMA secara langsung. LSTM, sebaliknya, secara eksplisit dirancang untuk menangani ketergantungan jangka panjang pada data sekuensial melalui mekanisme gating-nya. Perbandingan metrik evaluasi keduanya ditampilkan pada Tabel V berikut. [18]–[24]

Tabel 5. Perbandingan LSTM vs ARIMA pada Data Resource Server

Metrik Evaluasi	***ARIMA (Estimasi Literatur)***	***LSTM (Penelitian Ini)***	***Perbandingan***
MSE	*0.0397*	*0.0310*	*LSTM lebih baik sebesar **21,9%**.*
RMSE	*0.1992*	*0.1761*	*LSTM lebih baik sebesar **11,6%**.*
MAE	*0.1633*	*0.1426*	*LSTM lebih baik sebesar **12,7%**.*
Kemampuan Menangkap Pola Nonlinier	*Tidak*	*Ya*	*LSTM lebih unggul dalam memodelkan hubungan nonlinier.*
Prediksi Multi-step Real-Time	*Terbatas*	*Mendukung penuh*	*LSTM lebih sesuai untuk prediksi dan monitoring secara real-time.*

Berdasarkan Tabel V, model LSTM yang dikembangkan dalam penelitian ini menunjukkan metrik evaluasi yang lebih baik dibandingkan estimasi ARIMA dari literatur pada seluruh aspek yang dibandingkan. Selisih paling signifikan terlihat pada MSE, di mana LSTM mencatatkan nilai 0,0310 dibandingkan estimasi ARIMA sebesar 0,0397 — selisih sekitar 21,9%. Pola serupa terlihat pada RMSE (11,6%) dan MAE (12,7%). Mengingat angka ARIMA bersumber dari literatur dengan dataset berbeda, perbandingan ini bersifat indikatif dan belum dapat dianggap sebagai pengujian head-to-head yang ketat; pengujian ARIMA langsung pada dataset yang sama diperlukan untuk klaim yang lebih kuat. Meski demikian, secara konseptual LSTM tetap unggul karena mampu melakukan prediksi multi-step secara real-time, sesuatu yang tidak dapat dilakukan ARIMA secara efisien tanpa re-estimasi model berulang.

4. Kesimpulan

Penelitian ini berhasil merancang dan mengimplementasikan sistem deep learning berbasis LSTM untuk memprediksi penggunaan resource CPU dan RAM pada server Debian Linux secara real-time. Dari hasil yang diperoleh, beberapa kesimpulan dapat dirumuskan: 1). Sistem berhasil mengumpulkan lebih dari 1.859 record data monitoring dengan interval 5 detik menggunakan modul collect.py yang berjalan secara otomatis dan kontinyu. 2). Arsitektur model LSTM dua lapisan yang diimplementasikan dengan PyTorch berhasil konvergen setelah 50 epoch pelatihan dengan training loss akhir 0,0140, test loss 0,0310, RMSE 0,1761, dan MAE 0,1426 pada skala ternormalisasi — setara dengan error rata-rata kurang dari 0,35% pada skala CPU aktual. 3). Modul prediksi aktif (predict.py) berhasil beroperasi secara real-time, menghasilkan estimasi CPU dan RAM setiap 5 detik, dan mengklasifikasikan kondisi server ke dalam kategori NORMAL atau WASPADA berdasarkan ambang batas yang dapat dikonfigurasi. 4). Berdasarkan perbandingan indikatif dengan estimasi ARIMA dari literatur, pendekatan LSTM menunjukkan metrik evaluasi yang lebih baik — peningkatan sekitar 21,9% pada MSE, 11,6% pada RMSE, dan 12,7% pada MAE — serta mampu melakukan prediksi multi-step secara real-time yang tidak dapat dilakukan ARIMA secara efisien. Namun, karena ARIMA tidak diuji langsung pada dataset yang sama, klaim keunggulan ini masih bersifat sementara dan perlu divalidasi melalui pengujian head-to-head pada penelitian lanjutan. 5). Perlu dicatat bahwa data yang digunakan untuk melatih dan menguji model dikumpulkan selama periode yang relatif singkat (sekitar 2,5 jam) dengan kondisi beban server yang stabil pada level rendah (CPU < 2%, RAM ~15%). Minimnya variasi beban ini membuat nilai loss yang rendah belum sepenuhnya membuktikan kemampuan generalisasi model pada kondisi beban tinggi atau fluktuatif, sehingga hasil evaluasi pada penelitian ini sebaiknya dimaknai sebagai bukti kelayakan awal (proof of concept), bukan validasi performa pada skenario produksi yang

DOI: https://doi.org/10.31004/riggs.v5i2.11542

lebih beragam. Untuk penelitian lanjutan, beberapa arah pengembangan yang disarankan meliputi: (1) memperpanjang periode pengumpulan data agar mencakup kondisi beban server yang lebih bervariasi; (2) menambahkan metrik disk I/O dan network bandwidth sebagai fitur tambahan; (3) mengintegrasikan dashboard visualisasi berbasis web menggunakan Grafana atau Streamlit; (4) mengeksplorasi arsitektur Transformer yang kini menunjukkan performa unggul pada tugas time series; serta (5) mengembangkan mekanisme notifikasi otomatis melalui email atau messaging ketika sistem mendeteksi kondisi WASPADA. [18]–[24]

Referensi

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[2] J. Brownlee, Deep Learning for Time Series Forecasting. Machine Learning Mastery, 2018.
[3] E. Shi et al., "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in Proc. 26th ACM SOSP, 2017, pp. 153–167.
[4] H. Halimatusyadiah, Y. Afrianto, and B. A. Prakosa, "Implementasi LSTM untuk prediksi beban server Raspberry Pi sebagai dasar load balancing," Jurnal Informatika & Komputasi, vol. 8, no. 1, pp. 45–59, 2025.
[5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[6] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems (NeurIPS), vol. 32, 2019.
[7] T. F. Arya, R. F. Rachmad, and A. Affandi, "Prediksi resource requirement menggunakan machine learning stacking pada sistem cloud computing," Jurnal Sistem Informasi Indonesia, 2024.
[8] G. van Rossum and F. L. Drake, Python 3 Reference Manual. Scotts Valley, CA: CreateSpace, 2009.
[9] Psutil contributors, "psutil: Cross-platform process and system monitoring library for Python," Python Package Index, 2023. [Online]. Available: https://pypi.org/project/psutil/
[10] W. Matoussi and T. Hamrouni, "A new temporal locality-based workload prediction approach for SaaS services in a cloud environment," Journal of King Saud University - Computer and Information Sciences, vol. 34, pp. 3973–3987, 2022, doi: 10.1016/j.jksuci.2021.04.008.
[11] P. Nawrocki and P. Osypanka, "Cloud Resource Demand Prediction using Machine Learning in the Context of QoS Parameters," Journal of Grid Computing, vol. 19, art. no. 20, 2021, doi: 10.1007/s10723-021-09561-3.
[12] M. M. Al-Sayed, "Workload Time Series Cumulative Prediction Mechanism for Cloud Resources Using Neural Machine Translation Technique," Journal of Grid Computing, vol. 20, art. no. 16, 2022, doi: 10.1007/s10723-022-09607-0.
[13] 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, 2023.
[14] C. Tang et al., "Forecasting SQL Query Cost at Twitter," in Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E), 2021, pp. 154–160, doi: 10.1109/IC2E52221.2021.00030.
[15] S. Zou, W. Ji, and J. Huang, "BTP: automatic identification and prediction of tasks in data center networks," Journal of Cloud Computing, vol. 11, art. no. 36, 2022, doi: 10.1186/s13677-022-00312-7.
[16] J.-W. Park, M.-W. Kwon, and T. Hong, "Queue congestion prediction for large-scale high performance computing systems using a hidden Markov model," The Journal of Supercomputing, vol. 78, pp. 12202–12223, 2022, doi: 10.1007/s11227-022-04356-z.
[17] T. Zheng, J. Wan, J. Zhang, and C. Jiang, "Deep Reinforcement Learning-Based Workload Scheduling for Edge Computing," Journal of Cloud Computing, vol. 11, art. no. 3, 2022, doi: 10.1186/s13677-021-00276-0.
[18] 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, 2021.
[19] Y. Liu, H. Wu, J. Wang, and M. Long, "Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting," in Advances in Neural Information Processing Systems, vol. 35, 2022.
[20] 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.
[21] K. Stankeviciute, A. M. Alaa, and M. van der Schaar, "Conformal Time-Series Forecasting," in Advances in Neural Information Processing Systems, vol. 34, 2021.
[22] F.-K. Sun, C. Lang, and D. Boning, "Adjusting for Autocorrelated Errors in Neural Networks for Time Series," in Advances in Neural Information Processing Systems, vol. 34, 2021.
[23] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, "ETSformer: Exponential Smoothing Transformers for Time-Series Forecasting," arXiv preprint arXiv:2202.01381, 2022.
[24] 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.
[25] Debian Project, "Debian 12 (bookworm) Release Notes," 2023. [Online]. Available: https://www.debian.org/releases/bookworm/releasenotes

DOI: https://doi.org/10.31004/riggs.v5i2.11542