Analisis Komputasi Paralel pada Image Encoding Framework untuk Konversi Citra Data Deret Waktu Sistem Kontrol Industri

Helmy Rahadian[1], Muhammad Rizalul Wahid[2], Zaenal Arifin[3]

[Submission: 27-02-2023, Accepted: 25-05-2023]

***Abstract*— Sensors in industrial control systems send a series of data each time, known as time series data, to the controller. The data contains important information for the controller to determine the control signal for the actuator. The appearance of anomalies in time series data can be detected using the Convolutional Neural Network (CNN) method utilizing image encoding techniques such as Gramian Angular Field (GAF) and Markov Transition Field (MTF). This technique converts time series data into images through data preparation, encoding, and conversion. Dividing extensive data into many smaller segments requires repeated encoding and conversion processes. Repeated processes that are done serially take a long time, which slows down the detection of anomalies and the responses that must be taken. This research applies parallel computation with Joblib and Mpire libraries on the GAF and MTF image encoding provided by the Python-based pyts library. The n_jobs configuration determines the number of CPU logical cores used to execute the program. According to the number of CPU logic cores of the computer, applying the value of n_jobs = 8 can save an average processing time of 63% (Joblib) and 49% (Mpire), which theoretically will be able to detect anomalies that occur at least every 62.73 ms (Joblib) and 86.20 ms (Mpire) compared to 167.51 ms in serial computing.**

***Keywords*— Time series data, Anomaly, Encoding, Conversion, Image, Parallel, Python.**

***Intisari*— Sensor pada sistem kontrol industri mengirimkan serangkaian data tiap waktu dikenal dengan data deret waktu ke kontroler. Data memiliki informasi penting bagi kontroler untuk menentukan sinyal kontrol bagi aktuator. Munculnya anomali pada data deret waktu dapat dideteksi dengan metode Convolutional Neural Network (CNN) memanfaatkan teknik image encoding seperti Gramian Angular Field (GAF) dan Markov Transition Field (MTF). Teknik ini mengubah data deret waktu menjadi citra melalui serangkaian tahap mulai persiapan data, encoding data, dan konversi citra. Pembagian data berukuran besar menjadi sejumlah segmen yang lebih kecil membutuhkan proses encoding dan konversi yang berulang. Proses berulang yang dikerjakan secara serial membutuhkan waktu yang lama sehingga memperlambat deteksi anomali dan tanggapan yang harus dilakukan. Penelitian ini menerapkan komputasi paralel dengan Joblib dan Mpire pada image encoding GAF dan MTF yang disediakan oleh pustaka pyts berbasis Python. Konfigurasi n_jobs menentukan jumlah inti logika CPU yang digunakan untuk mengeksekusi program. Penerapan nilai n_jobs = 8 yang disesuaikan dengan jumlah inti logika CPU komputer penelitian menghasilkan penghematan waktu proses rata-rata sebesar 63% (Joblib) dan 49% (Mpire) yang secara teoritis akan mampu mendeteksi anomali yang muncul minimal tiap 62.73 ms (Joblib) dan 86.20 ms (Mpire) dibandingkan dengan komputasi serial yakni setiap 167.51 ms.**

***Kata Kunci*— Data deret waktu, Anomali, Encoding, Konversi, Citra, Paralel, Python.**

I. PENDAHULUAN

Sistem kontrol industri (*Industrial Control System* atau ICS) terdiri atas berbagai perangkat yang membentuk sebuah sistem terintegrasi yang bertujuan untuk mengontrol proses secara otomatis dan mempertahankan kestabilan. Sensor pada ICS secara kontinyu mengirimkan data kepada kontroler (seperti *Programmable Logic Controller* atau PLC). Data ini merupakan hasil pengukuran sensor pada suatu proses tertentu dengan laju pencuplikan yang seragam dalam periode waktu tertentu dan dikenal sebagai *time series data* atau data deret waktu. Analisis terhadap data deret waktu dilakukan untuk menemukan pola atau karakteristik data serta memprediksi nilai yang akan datang berdasarkan hasil pengamatan pada data sebelumnya untuk menjadi dasar pengambilan keputusan [1].

Sebagai hasil dari pengamatan kondisi suatu proses, sering kali data deret waktu menunjukkan anomali yaitu munculnya data yang menyimpang dari pola atau perilaku yang dikehendaki. Istilah anomali juga dikenal sebagai *outliers*, didefinisikan sebagai hasil pengamatan yang menyimpang jauh dari hasil pengamatan yang lain sehingga seolah-olah data tersebut dihasilkan oleh mekanisme pengamatan yang berbeda [2]. Tanggapan atas munculnya anomali bisa berupa pembersihan data maupun analisis lanjut [3]. Pembersihan data dilakukan jika anomali berasal dari *noise* atau kesalahan perangkat maupun jalur komunikasi dan analisis lanjut dilakukan pada sebuah fenomena yang tidak biasa namun unik seperti pada kasus kecurangan finansial [4].

Metode atau teknik untuk mendeteksi anomali pada data deret waktu terdiri atas teknik berbasis pembelajaran mesin (*machine learning*/ML) dan berbasis statistik [5]. Beberapa tahun terakhir deteksi anomali berbasis ML telah banyak diadaptasi oleh banyak peneliti untuk diterapkan di berbagai

[1]*Dosen, Program Studi Teknik Elektro Fakultas Teknik Universitas Dian Nuswantoro, Jl. Nakula I No. 5-11 Semarang 50131 Indonesia (tlp: 024-3555628; fax: 024-3555628 Ext 1; e-mail: helmyrahadian@dsn.dinus.ac.id)*

[2]*Dosen, Program Studi Mekatronik dan Kecerdasan Buatan, Universitas Pendidikan Indonesia, Jl. Veteran No. 8 Purwakarta 41115 Indonesia (tlp:+622-64200395; rizalulwahid@upi.edu)*

[3]*Dosen, Program Studi Teknik Elektro Fakultas Teknik Universitas Dian Nuswantoro, Jl. Nakula I No. 5-11 Semarang 50131 Indonesia (tlp: 024-3555628; fax: 024-3555628 Ext 1; e-mail: xzaenal@dsn.dinus.ac.id)*

p-ISSN:1693 – 2951; e-ISSN: 2503-2372

bidang. Selanjutnya ML bertransformasi menjadi pembelajaran mendalam (*deep learning*/DL) yang memiliki kapabilitas tinggi dalam pemrosesan dan pembelajaran data dengan kompleksitas tinggi termasuk *high dimensional data*, data temporal dan spasial. Sejumlah algoritma DL telah terbukti memiliki performa lebih baik dibandingkan metode konvensional seperti metode statistik dalam mengatasi persoalan di berbagai aplikasi nyata [6].

Convolutional neural network (CNN) adalah salah satu metode DL yang banyak dipakai untuk mendeteksi anomali. Metode ini terinspirasi oleh cara kerja bagian otak yang disebut *visual cortex* dalam menanggapi rangsangan visual. CNN memproses masukan berupa citra dan melalui serangkaian proses yang terjadi pada beberapa lapisan (berisi neuron-neuron) hingga diperoleh karakteristik atau ciri utama pada citra tersebut. Data *time series* pada ICS terlebih dahulu diubah menjadi citra menggunakan teknik *image encoding* seperti Gramian Angular Field (GAF) dan Markov Transition Field (MTF). Citra hasil *image encoding* tersebut kemudian menjadi masukan bagi CNN yang akan menentukan ada tidaknya anomali pada citra.

Penelitian yang mengombinasikan *image encoding* dan CNN sudah banyak dilakukan diantaranya: mendeteksi anomali pada trafik komunikasi OPC [7]; mendeteksi anomali pada penggunaan daya listrik [8]; deteksi anomali pada belitan mesin pencelupan (*dyeing machine*) [9]; deteksi anomali pada *structural health monitoring* (SHM) [10]; dan deteksi kondisi jantung melalui data ECG pada aplikasi e-Health [11].

Pustaka pyts berbasis Python menyediakan *framework image encoding* baik GAF, MTF dan Recurrent plot (RP) untuk mengubah data deret waktu menjadi citra [12]. Pustaka ini memiliki dua tahap: (a) tahap mengubah data deret waktu menjadi matriks sesuai jenis *image encoding*; dan (b) tahap mengubah matriks menjadi citra. Beberapa pustaka lain digunakan untuk keperluan tahapan tersebut seperti contohnya penggunaan pustaka matplotlib untuk menghasilkan citra. Data deret waktu yang berukuran besar akan dibagi menjadi segmen-segmen berukuran sama. Proses konversi dari data deret waktu menjadi citra dilakukan secara satu-persatu atau serial. Hal ini membutuhkan waktu relatif lama dan semakin lama jika jumlah segmen semakin banyak. Lamanya proses konversi mempengaruhi jeda waktu antara sejak munculnya anomali hingga anomali tersebut berhasil dideteksi. Sehingga dimungkinkan tanggapan atas anomali terlambat dilakukan dan dapat mengakibatkan kerugian.

Penelitian ini bermaksud untuk menerapkan konsep komputasi paralel untuk mempercepat waktu *encoding* dan konversi citra. Kecepatan eksekusi program bisa ditingkatkan dengan memanfaatkan seluruh inti CPU yang dimiliki komputer. Hal ini diharapkan dapat memangkas waktu proses *encoding* dan konversi citra sehingga memperkecil selisih waktu antara munculnya anomali hingga berhasil dideteksi dan tanggapan atas anomali bisa segera diambil untuk menghindari potensi kerugian.

Makalah ini disusun mengikuti struktur berikut: Bagian II menjelaskan mengenai data deret waktu dan deteksi anomali, *image encoding* GAF dan MTF serta konsep pemrograman paralel. Selanjutnya pada Bagian III ditunjukkan mengenai metodologi penelitian yang dilakukan. Hasil penelitian dan pembahasannya dijabarkan pada Bagian IV dan ditutup dengan kesimpulan dan saran pada Bagian V.

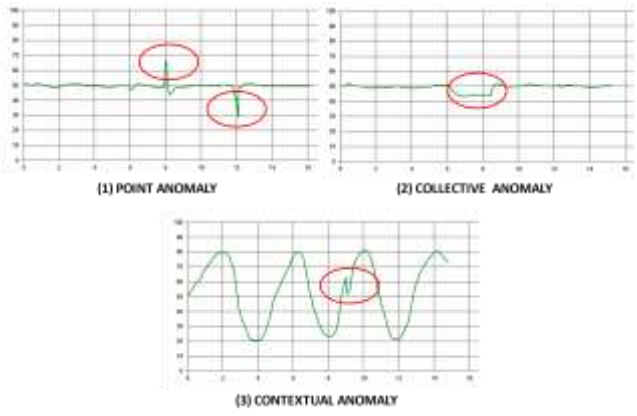
II. TINJAUAN PUSTAKA

A. *Data deret waktu dan deteksi anomali*

Data deret waktu diperoleh dari pengamatan atau pengukuran dalam kurun waktu atau periode tertentu. Data deret waktu banyak dijumpai di berbagai bidang seperti finansial, industri maupun kesehatan. Data deret waktu termasuk dalam data temporal yang memiliki karakteristik seperti jumlah data relatif besar, berdimensi tinggi (memiliki banyak variabel) dan diperbarui secara kontinu [13].

Analisis data deret waktu dilakukan untuk menemukan informasi yang terkandung di dalamnya. Berdasarkan informasi yang diperoleh dari data, maka dapat dilakukan: *forecasting* atau prediksi – yakni memperkirakan nilai yang akan muncul berdasarkan pola dari data-data sebelumnya; *clustering* – mengelompokkan data yang tanpa label menggunakan pendekatan *unsupervised learning* seperti jenis iklim yang berbeda dapat dikelompokkan atau ditentukan berdasarkan data temperatur; *classification* – bertujuan menghasilkan *classifier* yang secara otomatis mengelompokkan jenis data baru yang memiliki label dengan pendekatan *supervised learning*, seperti menentukan jantung dalam kondisi normal atau abnormal melalui pengamatan data sensor ECG [14].

Anomali dapat muncul pada data deret waktu dan menyimpan informasi tertentu yang berbeda dengan perilaku proses yang dikehendaki. Anomali memiliki beberapa jenis diantaranya: (1) *point anomaly* – anomali tunggal (individual) yang berbeda dari data yang lain; (2) *collective anomaly* – sekumpulan data yang tiap individu datanya masih termasuk data yang dapat ditoleransi (tidak berbeda) namun secara keseluruhan berbeda dari data yang lain; (3) *contextual anomaly* – data individu atau kelompok data yang masih termasuk dalam toleransi namun ketidakteraturan bila dibandingkan data di sekelilingnya [15]. Ilustrasi ketiga jenis anomali ditunjukkan oleh Gambar 1.



Gambar 1: Jenis-jenis anomali (a) *point anomaly*; (b) *collective anomaly*; (c) *contextual anomaly*

B. Open Platform Communication

Open Platform Communication (OPC) merupakan antarmuka komunikasi antar perangkat pada sektor industri dan dikenal juga dengan nama lain ysitu *open process control*. OPC menjadi jawaban atas semakin beragamnya jenis dan kompleksitas perangkat otomasi industri yang menyebabkan rendahnya interoperabilitas antar perangkat yang dapat menyebabkan kenaikan biaya perawatan, pengembangan, resiko keamanan dan dapat pula menurunkan resiliensi sistem [16]. Sejak tahun 1996, OPC digunakan sebagai protokol komunikasi standar yang memberikan jaminan keandalan dan keamanan dalam pertukaran data antar perangkat otomasi dalam industri. OPC generasi awal ini sering disebut dengan OPC *classic*. Pada tahun 2006, OPC UA (*unified architecture*) dirilis menggantikan OPC *classic* dan telah ditetapkan menjadi stadard internasional oleh *International Electrotechnical Commission* (IEC) yang dikenal sebagai IEC62541.

Komunikasi pada OPC UA mengadopsi model *client-server* dan *publish-subscribe*. Model *client-server* merupakan model komunikasi yang paling banyak dijumpai di industri. Model ini digunakan apabila diperlukan komunikasi asinkron dengan jumlah data yang besar. Komunikasi diinisiasi oleh OPC *client* untuk memperoleh data tertentu yang dimiliki oleh OPC *server*. OPC *client* umumnya berupa komputer yang menjalankan program seperti HMI dan SCADA sedangkan OPC *server* bisa berupa perangkat kecil seperti sensor dan PLC [17]. Selain itu sebuah komputer dapat diprogram untuk berperan baik sebagai OPC *client* maupun *server*.

Saat ini tersedia berbagai program atau perangkat lunak untuk mengimplementasikan protokol komunikasi OPC UA. Terdapat pilihan perangkat lunak *propietary* (berbayar atau bersifat komersial) dan *open source* (bisa digunakan atau dikembangkan tanpa biaya). Program OPC UA *propietary* banyak digunakan karena memiliki keunggulan dibandingkan dengan program *open source* dalam hal dukungan teknis dan kestabilan. Selain itu, kebanyakan pengguna tidak memiliki keahlian dalam melakukan konfigurasi yang dibutuhkan oleh program *open source*. Penggunaan program OPC UA *open source* bisa menjadi pilihan lebih baik apabila biaya menjadi pertimbangan utama. Beberapa program OPC UA *open source* diantaranya: open62541 (berbasis bahasa pemrograman C), UA-.NETStandard (C#), node-opcua (Java), python-opcua dan opcua-asyncio (python). Menurut [18], hanya open62541 dan UA-.NETStandard yang cocok untuk implementasi yang membutuhkan banyak *client* dan *server* (atau *node*).

C. Image encoding (GAF dan MTF)

Kombinasi *image encoding* dan CNN saat ini banyak digunakan dalam berbagai aplikasi dan riset serta menunjukkan performa yang baik. *Framework image encoding* Gramian Angular Field (GAF) dan Markov Transition Field (MTF) banyak digunakan untuk mengkonversi data deret waktu menjadi citra serta menunjukkan performa yang baik dalam klasifikasi dan perbaikan data pada citra [19].

GAF memiliki dua variasi yaitu *gramian angular difference field* (GADF) dan *gramian angular summation field* (GASF). GAF merepresentasikan data deret waktu yang biasa dalam koordinat kartesian menjadi koordinat polar. Perbedaan GASF dan GADF terletak pada operasi pada matriks *gramian*: pengurangan (GADF, dari kata *difference*) dan penambahan (GASF, dari kata *summation*). Tahapan proses mengubah data deret waktu X = (x_1, x_2, …, x_n) menjadi matriks gramian [19]:

- Normalisasi x_i pada rentang [-1, 1] atau [0, 1] menggunakan (1).
- Mengubah nilai deret waktu ke koordinat polar, melalui (2).
- Membentuk matriks *gramian* GADF (3) atau GASF (4).

$$\tilde{x}_{-1}^{i} = \frac{\left(x_i - max(\mathrm{X})\right) + \left(x_i - min(\mathrm{X})\right)}{max(\mathrm{X}) - min(\mathrm{X})}$$
$$or\ \tilde{x}_{0}^{i} = \frac{x_i - min(\mathrm{X})}{max(\mathrm{X}) - min(\mathrm{X})} \quad (1)$$

$$\begin{cases} \emptyset = arccos(\tilde{x}_i), -1 \le \tilde{x}_i \le 1, \tilde{x}_i \in \tilde{\mathrm{X}} \\ r = \frac{t_i}{\mathrm{N}}, t_i \in \mathbb{N} \end{cases} \quad (2)$$

$$GADF(i,j) = sin\left[\left(\emptyset_i - \emptyset_j\right)\right]$$
$$GADF = \begin{bmatrix} sin[(\emptyset_1 - \emptyset_1)] & \cdots & sin[(\emptyset_n - \emptyset)] \\ \vdots & \ddots & \vdots \\ sin[(\emptyset_1 - \emptyset_n)] & \cdots & sin[(\emptyset_n - \emptyset_n)] \end{bmatrix} \quad (3)$$

$$GASF(i,j) = cos\left[\left(\emptyset_i + \emptyset_j\right)\right]$$
$$GASF = \begin{bmatrix} cos[(\emptyset_1 + \emptyset_1)] & \cdots & cos[(\emptyset_n + \emptyset_1)] \\ \vdots & \ddots & \vdots \\ cos[(\emptyset_1 + \emptyset_n)] & \cdots & cos[(\emptyset_n + \emptyset_n)] \end{bmatrix} \quad (4)$$

MTF pada dasarnya melakukan *encoding* terhadap transisi dinamis secara statistik pada data deret waktu dengan penambahan probabilitas transisi Markov secara sekuensial untuk mempertahankan informasi di dalam domain waktu. Jika diketahui data deret waktu X = (x_1, x_2, …, x_n), maka tahapan *encoding* adalah [20]:

- Membagi data deret waktu dengan nilai quantile Q tertentu kemudian setiap x_i dipetakan pada quantile yang sesuai (q_j dengan $j \in [1, Q]$) sehingga diperoleh matriks probabilitas transisi W berdimensi Q x Q (5); dengan w_{ij} adalah probabilitas data x yang saat ini berada pada q_i akan berpindah ke q_j di waktu mendatang (6).
- Mengembangkan matriks W yang tiap elemennya saling independen menjadi matriks Markov transition field (M) melalui (7).

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1Q} \\ w_{21} & w_{22} & \cdots & w_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{Q1} & w_{Q2} & \cdots & w_{QQ} \end{bmatrix} \quad (5)$$

$$w_{ij} = P\{x_t \in q_i \mid x_{t-1} \in q_j\} \text{ dengan } \sum_{j=1}^{Q} w_{ij} = 1 \quad (6)$$

$$M = \begin{bmatrix} w_{ij} \mid x_1 \in q_i, x_1 \in q_j & \cdots & w_{ij} \mid x_1 \in q_i, x_n \in q_j \\ w_{ij} \mid x_2 \in q_i, x_1 \in q_j & \cdots & w_{ij} \mid x_2 \in q_i, x_n \in q_j \\ \vdots & \ddots & \vdots \\ w_{ij} \mid x_n \in q_i, x_1 \in q_j & \cdots & w_{ij} \mid x_n \in q_i, x_n \in q_j \end{bmatrix} \quad (7)$$

Encoding dengan MTF memiliki parameter "*strategy*" dan "n_bins" yang menentukan cara dan jumlah pembagian data deret waktu yang akan diproses. Terdapat tiga pilihan parameter "*strategy*" pada MTF yaitu:

- Uniform – membagi data deret waktu menjadi sejumlah n_bins bagian dengan tiap bagian memiliki rentang yang sama.
- Quantile – membagi data deret waktu menjadi sejumlah n_bins bagian dengan tiap bagian memiliki jumlah data yang sama
- Normal – data deret waktu dibagi dengan sejumlah n_bins yang mengikuti kuantil dari distribusi normal

Matriks yang dihasilkan oleh *image encoding* selanjutnya dikonversi menjadi citra. Pustaka pyts menggunakan pustaka lain yaitu matplotlib untuk mengkonversi matriks menjadi citra. Tahapan *encoding* dan konversi data deret waktu menjadi citra yang disediakan oleh pustaka pyts dijabarkan dalam algoritma 1 dan algoritma 2.

Algoritma 1 GAF

```
1  function GAF(mode)
2  dt = time series data
3  For i in range(num_of_segment)
4      X = dt[i]
5      Enc[i] = GAF(X)
6      Img[i] = fig.enc[i]
7  end function
```

Algoritma 2 MTF

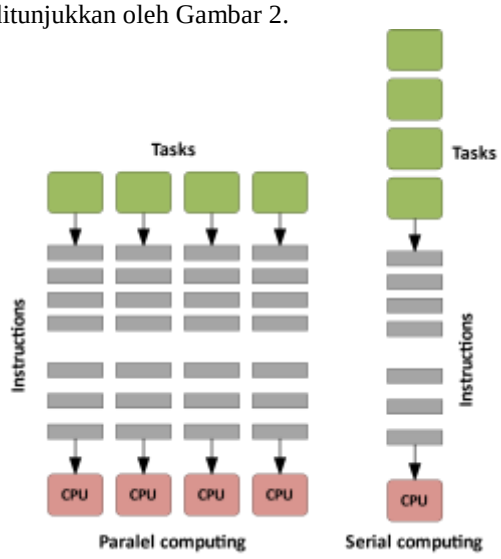
```
1  function MTF(strategi, bin)
2  dt = time series data
3  For i in range(num_of_segment)
4      X = dt[i]
5      Enc[i] = MTF(X)
6      Img[i] = fig.enc[i]
7  end function
```

D. Komputasi paralel

Algoritma komputasi berkembang menjadi semakin kompleks seiring dengan banyaknya permasalahan yang harus diselesaikan. Perangkat keras seperti komputer saat ini terus dikembangkan untuk dapat mendukung proses komputasi tersebut. Spesifikasi perangkat komputasi yang dapat mempengaruhi kecepatan eksekusi program diantaranya jumlah inti (*core*), frekuensi *clock central processing unit* (CPU), kecepatan memori dan *hard disk* serta dalam konteks aplikasi tertentu termasuk kapasitas atau *bandwith* jaringan komputer.

Python adalah salah satu bahasa pemrograman yang saat ini populer digunakan untuk komputasi saintifik seperti pengolahan citra serta beragam *problem solving* berbasis pembelajaran mesin dan pembelajaran mendalam seperti pada [21] serta untuk membangun sistem kontrol seperti kontrol robotika pada [22]. Hal ini didukung karena Python merupakan bahasa pemrograman tingkat tinggi yang mudah dipahami, *open source* (tidak membutuhkan biaya lisensi), memiliki dukungan pustaka yang sangat lengkap dan portabilitasnya yang baik karena bisa dijalankan di berbagai platform perangkat keras maupun perangkat lunak (sistem operasi komputer).

Semakin besar jumlah dan kompleksitas data maka peranan komputasi paralel menjadi semakin penting. Python adalah salah satu bahasa pemrograman yang mendukung komputasi secara paralel. Hal ini berarti, program Python bisa dijalankan secara bersamaan pada beberapa inti prosesor (CPU) atau beberapa CPU dengan memanfaatkan siklus CPU yang tidak terpakai ketika ada program yang menunggu sumber daya eksternal. Hal ini akan menjadikan *running time* program yang dikerjakan secara paralel lebih cepat dari pada dikerjakan secara serial [23]. Ilustrasi eksekusi program secara paralel dan serial ditunjukkan oleh Gambar 2.



Gambar 2: Komputasi paralel dan serial, berdasarkan [23]

Penerapan komputasi paralel pada Python dilakukan melalui dua pendekatan yaitu melalui multi-proses dan multi-threading. Sebuah proses merujuk pada sebuah program komputer. Setiap program Python adalah sebuah proses dan memiliki sebuah *thread* default disebut dengan main thread yang digunakan untuk mengeksekusi instruksi pada program [24]. Contoh sebuah proses adalah program *intepreter* Python yang mengeksekusi kode instruksi Python pada level *byte-code* seperti Jupyter Notebook. Sedangkan *thread* adalah urutan eksekusi pada sebuah program komputer. Setiap program adalah sebuah proses dan memiliki setidaknya satu thread yang mengeksekusi instruksi untuk proses tersebut. *Thread* menentukan urutan atau cara kode instruksi dieksekusi.

Joblib dan Mpire adalah contoh pustaka Python yang bisa digunakan untuk menjalankan komputasi paralel dan menambah kecepatan komputasi. Joblib mendukung multi-proses dan multi-*thread* serta memiliki fitur *transparent disk cache* untuk menyimpan obyek Python yang dihasilkan oleh proses atau *thread*. *Cache* ini tidak hanya membatu menghindari pekerjaan berulang namun bisa digunakan kembali untuk proses yang tertunda atau setelah mengalami crash [25]. Sedangkan Mpire adalah pustaka yang dikembangkan berdasarkan pustaka *multiprocessing* yang merupakan pustaka standar Python [26]. Joblib [27] dan Mpire [28] bersifat *open source* dan dapat diakses dan di-*install* Github.

III. METODOLOGI

Penelitian ini dilakukan melalui beberapa tahapan yaitu:

1. Akuisisi data
2. *Encoding* data menggunakan GAF dan MTF dengan pencatatan waktu:
   - Pemrosesan secara serial
   - Pemrosesan secara paralel
3. Analisis performa *encoding*

Perangkat lunak yang digunakan dalam penelitian ini adalah:

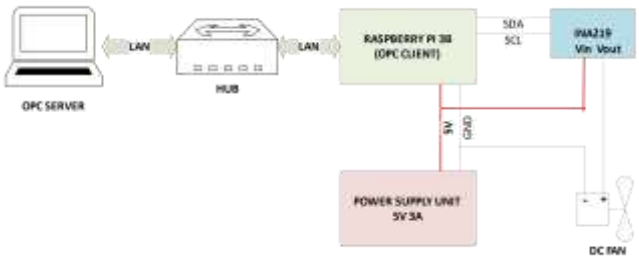
1. Platform Anaconda 2.3.3 dengan Python 3.9.15 dan Jupyter Lab 3.5.3
2. Pustakan pyts versi 0.12.0
3. Pustaka opcua-asyncio library v1.0.0
4. Pustaka Joblib versi 1.2.0
5. Pustaka Mpire versi 2.6.0
6. Windows 11 Pro 64-bit OS

Sedangkan perangkat keras yang digunakan dalam penelitian ini adalah komputer laptop dengan spesifikasi perangkat keras:

1. CPU Intel i7-8650u @1.90 GHz dengan 8 buah *logical processors*
2. RAM 16 GB 2400 MHz
3. SSD 1 TB nvme PCIe 1.3
4. Komputer dijalankan dengan sumber daya AC PLN (mode *performance*)

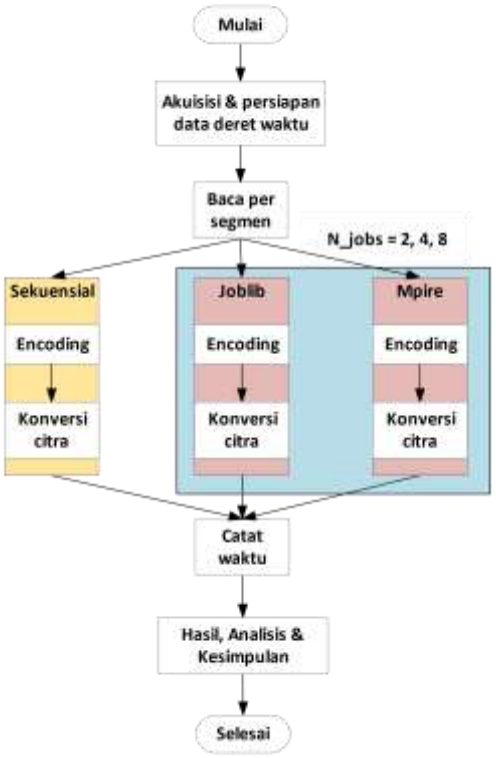
A. *Akuisisi data*

Data penelitian diperoleh dari pembacaan arus mini fan DC oleh Raspberry Pi 3B melalui sensor INA219 menggunakan protokol I^2C. Nilai arus dibaca secara kontinyu dengan periode 200ms hingga terkumpul 45.000 data. Raspberry bertindak sebagai sebuah OPC *client* yang mengirimkan data arus yang baru menuju ke laptop sebagai OCP *server*. OPC *client* dan *server* dihubungkan melalui jaringan komputer lokal menggunakan pustaka komunikasi opcua-asyncio v1.0.0. Sistem akuisisi data ini meniru arsitektur OPC *client-server* pada ICS. Diagram blok sistem akuisisi data ditunjukkan oleh Gambar 3.



Gambar 3: Diagram blok sistem akuisisi data

Proses selanjutnya yang dilakukan setelah memperoleh data adalah *encoding* dan konversi citra. Proses ini mengikuti diagram alir yang ditunjukkan oleh Gambar 4.



Gambar 4: Diagram alir proses *encoding* dan konversi citra

Sensor INA219 membaca nilai arus listrik yang mensuplai DC fan dalam kondisi normal. Periode pembacaan data sebesar 200ms dengan total data yang diambil sebanyak 45000 data. Data ini dibagi menjadi 2250 segmen dengan tiap segmennya terdiri atas 20 data. Selanjutnya tiap segmen di-*encoding* dengan GAF dan MTF menghasilkan 2250 matriks berdimensi 20x20. Selain itu, pada tiap segmen disisipkan data anomali berupa arus bernilai antara nol hingga 1mA menggantikan data normal. Data anomali disebar di tiap segmen pada posisi dan jumlah yang acak (antara 1 hingga 10 data anomali per segmen).

Setelah dihasilkan total 4500 matriks berisi *encoded value* (2250 segmen normal dan 2250 segmen anomali) maka

p-ISSN:1693 – 2951; e-ISSN: 2503-2372

selanjutnya dilakukan konversi matriks tersebut menjadi citra. Hasil akhir yang diperoleh adalah 2250 citra normal dan 2250 citra anomali. Sehingga jumlah iterasi atau perulangan eksekusi program yang dilakukan adalah masing-masing 2250 untuk proses *encoding* dan konversi citra normal dan anomali.

B. Encoding dan konversi citra

Penelitian ini melakukan *encoding* GAF dengan metode "*difference*" (GADF) dengan asumsi *encoding* GASF memiliki kemiripan dengan GADF hanya berbeda pada operasi pengurangan dan penambahan sehingga beban komputasi dianggap sama. Sedangkan *encoding* MTF menggunakan "*strategy*": uniform, quantile, dan normal dengan jumlah bin *default* yaitu 5. Parameter GADF dan MTF yang lain, seperti "image_size", "sample_range", "*overlapping*", dan "*flatten*" tetap bernilai *default*.

C. Pustaka komputasi paralel

Penelitian ini memanfaatkan dua buah pustaka untuk melakukan komputasi secara paralel pada proses *encoding* dan konversi citra yaitu Joblib dan Mpire. Konfigurasi pustaka joblib dan mpire difokuskan pada parameter n_jobs yang menentukan jumlah inti pada CPU yang akan digunakan untuk menjalankan proses. Pada penelitian ini, proses *encoding* dan konversi citra akan dijalankan menggunakan parameter n_jobs = 2, 4, dan 8 (menyesuaikan jumlah maksimal inti logika CPU yang dimiliki perangkat komputasi). Setiap proses dilakukan pencatatan waktu untuk kemudian dilakukan analisis performa komputasi paralel serta peningkatan kecepatan yang diperoleh bila dibandingkan dengan proses yang dikerjakan secara serial.

Encoding GAF dan MTF dengan pustaka pyts pada dasarnya memiliki struktur kode program yang sama. Selain melakukan *encoding* juga melakukan konversi citra. Algoritma 1 dan algoritma 2 menggambarkan proses *encoding* data deret waktu dan konversi citra yang masih dikerjakan secara serial sedangkan algoritma 3 menunjukkan proses tersebut namun dekerjakan secara paralel dengan Pustaka Joblib dan Mpire. Algoritma 3 menggambarkan secara spesifik proses *encoding* dan konversi citra menggunakan *framework* MTF namun algoritma ini juga berlaku untuk *framework* GADF dengan mengganti fungsi MTF dengan fungsi GADF.

Algoritma 3 MTF Paralel

```
1  dt = time series data
2  function MTF(mode, bin)
3      X = dt[i]
4      Enc[i] = MTF(X)
5      Img[i] = fig.enc[i]
6  end function
7
8  # Joblib
9  with Parallel(n_jobs = nCPU) as parallel:
10     parallel([delayed(MTF)(i) for i in range(n_seg)]))
11
12 # Mpire
13 with WorkerPool(n_jobs = nCPU, shared_objects =
   dt) as pool:
14     pool.map(MTF, range(2250))
```

IV. HASIL DAN PEMBAHASAN

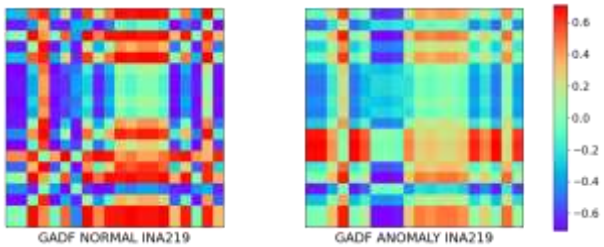
Bab ini berisi hasil penelitian berupa catatan waktu proses eksekusi program *encoding* dan konversi citra menggunakan *framework* GADF dan MTF yang dikerjakan secara seri dan paralel menggunakan pustaka Joblib dan Mpire dengan variasi nilai n_jobs. Selanjutnya dilakukan analisis terhadap data penelitian untuk mengetahui performa komputasi paralel bila diterapkan pada sistem deteksi anomali.

A. Citra hasil proses encoding dan konversi data deret waktu

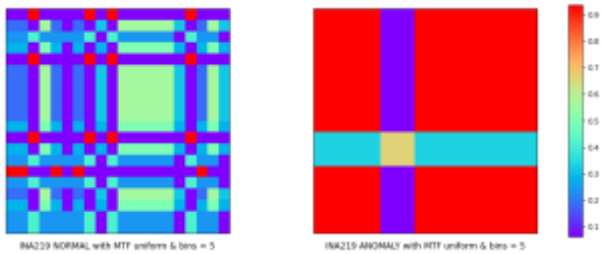
Contoh citra hasil proses *encoding* dan konversi diambil dari data deret waktu segmen pertama. Data arus listrik normal dan anomali pada segmen ini ditunjukkan oleh Gambar 5.

Gambar 5: Data deret waktu segmen pertama

Segmen pertama mengalami anomali pada data ke 7, 8, dan 9 dengan arus listrik yang tiba-tiba menjadi sangat kecil (hampir nol). Data normal dan anomali pada segmen pertama ini kemudian melewati proses *encoding* dan konversi menggunakan GADF, MTF uniform 5 (nilai bin = 5), MTF quantile 5, dan MTF normal 5 menghasilkan pasangan citra normal dan anomali. Sebagai contoh, citra GADF dan MTF uniform 5 ditunjukkan oleh Gambar 6 dan Gambar 7. Rata-rata setiap citra berukuran 4 KB.



Gambar 6: Citra hasil konversi data deret waktu dengan GADF pada segmen 1 – kondisi normal (kiri) dan anomali (kanan)



Gambar 7: Citra hasil konversi data deret waktu dengan MTF uniform 5 pada segmen 1 – kondisi normal (kiri) dan anomali (kanan)

Secara keseluruhan, dari pasangan 2250 segmen data deret waktu normal dan anomali akan menghasilkan 4500 pasang citra yang akan menjadi masukan pada proses selanjutnya yaitu klasifikasi citra CNN (tidak termasuk dalam fokus penelitian ini). Selanjutnya setelah terbukti menghasilkan citra normal dan anomali, dilakukan proses *encoding* dan konversi sebanyak 2250 kali secara serial dan paralel.

B. Serial

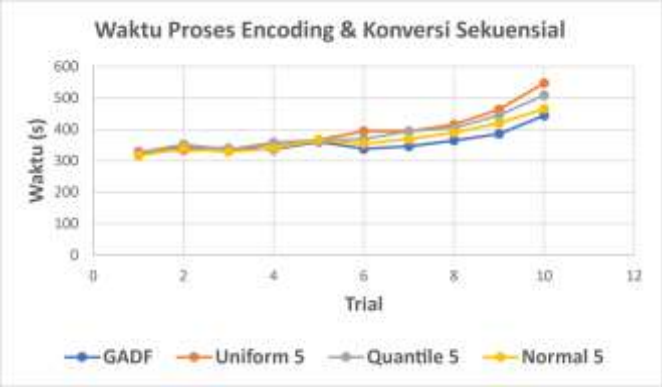
Proses *encoding* dan konversi citra dengan GADF dan MTF uniform 5, MTF quantile 5, dan MTF normal 5 secara serial menghasilkan catatan waktu yang ditunjukkan oleh Tabel 1.

TABEL 1
WAKTU *ENCODING* & KONVERSI CITRA GADF SECARA SERIAL

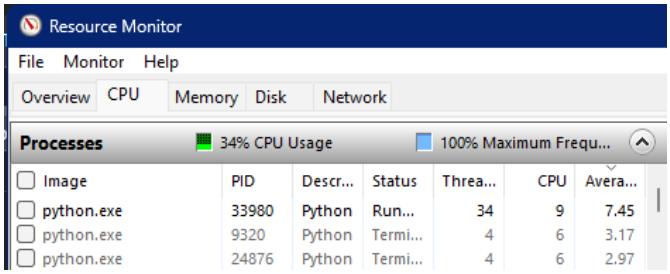
Percobaan	GADF (waktu, s)	MTF (waktu, s)		
		Uniform 5	Quantile 5	Normal 5
1	323.36	330.18	327.07	317
2	345.37	334.31	352.61	339.85
3	339.51	335.28	336.74	328.79
4	336.47	353.48	358.21	340.39
5	361.1	367.16	366.48	365.79
6	338.12	394.38	369.36	354.87
7	346.74	394.97	393.93	369.16
8	364.45	416.03	404.99	389.19
9	385.6	464	444.97	420.05
10	444.19	547.07	508.81	465.94
Rerata	**358.491**	**393.686**	**386.317**	**369.103**

Catatan waktu 10 kali percobaan *encoding* GADF dan MTF secara serial menunjukkan peningkatan khususnya setelah percobaan ke-7 seperti ditunjukkan pada Gambar 8. Peningkatan waktu proses *encoding* dan konversi disebabkan oleh memori (RAM) makin penuh. Penggunaan RAM di atas 90% terjadi pada percobaan ke-7 dan semakin bertambah pada percobaan terakhir. Hal ini kemungkinan terjadi karena citra yang dihasilkan pada percobaan sebelumnya tidak terhapus dan terakumulasi di memori. Komputer dapat mengalami kondisi *not responding* karena sistem operasi kekurangan *resource* memori untuk menjalankan perintah lainnya. Selama percobaan menghasilkan 450 ribu citra dengan ukuran 4 kB/citra sehingga membutuhkan memori sebesar 1.8 GB.

Oleh karena itu, eksekusi program secara serial tidak bisa sekedar dengan mengulang-ulang eksekusi. Dalam penelitian ini, eksekusi program sebanyak 2250 kali tidak langsung memenuhi memori namun jika dilakukan lebih banyak (2250 x 10 percobaan) akan meningkatkan resiko memori penuh sehingga diperlukan tambahan berupa fungsi atau metode untuk menghapus citra yang sudah tidak dipakai di memori. Tidak seperti efek yang terjadi pada memori, penggunaan CPU pada pemrosesan serial tidak terlalu tinggi. Terlihat pada *Windows Resource Monitor* (WRM) di Gambar 9, pada saat percobaan GADF ke-7, hanya ada satu proses Python yang aktif dengan utilisasi CPU sebesar 9% dengan rerata 7.45% dalam satu menit terakhir.



Gambar 8: Waktu proses *encoding* dan konversi serial



Gambar 9: *Windows resource monitor* (serial)

C. Paralel

Waktu yang diperlukan untuk melakukan *encoding* dan konversi citra GADF menggunakan pustaka Joblib dan Mpire dengan variasi parameter n_jobs sebesar 2, 4, dan 8 CPU ditunjukkan oleh Tabel 2 dan Tabel 3. Nilai pada tabel adalah rerata dari sepuluh kali percobaan tiap metode *encoding*.

TABEL 2
RERATA WAKTU PROSES *ENCODING* & KONVERSI SECARA PARALEL (JOBLIB)

Encoding & Konversi Citra	Joblib (waktu, s)		
	n_job = 2 CPU	n_job = 4 CPU	n_job = 8 CPU
GADF	252.29	190.71	150.49
Uniform 5	253.40	173.51	140.89
Quantile 5	253.47	175.95	137.79
Normal 5	249.73	171.98	135.35
Rerata	**252.22**	**178.04**	**141.13**

TABEL 3
RERATA WAKTU PROSES *ENCODING* & KONVERSI SECARA PARALEL (MPIRE)

Encoding & Konversi Citra	Mpire (waktu, s)		
	n_job = 2 CPU	n_job = 4 CPU	n_job = 8 CPU
GADF	314.35	234.67	209.23
Uniform 5	314.70	220.65	188.64
Quantile 5	306.60	221.91	191.31
Normal 5	312.83	219.40	186.64
Rerata	**312.12**	**224.16**	**193.95**

p-ISSN:1693 – 2951; e-ISSN: 2503-2372

Berdasarkan Tabel 2 dan Tabel 3, nilai n_jobs yang makin tinggi mempercepat waktu proses. Hal ini karena nilai n_jobs berpengaruh terhadap jumlah inti CPU yang bisa digunakan untuk mengeksekusi program. Nilai n_jobs = 1 berarti program hanya bisa dieksekusi secara bergantian (serial) pada satu inti CPU. Pada penelitian ini nilai maksimal n_jobs adalah 8 (sesuai dengan jumlah inti logika pada komputer) yang berarti eksekusi program bisa dilakukan secara paralel pada 8 inti CPU.

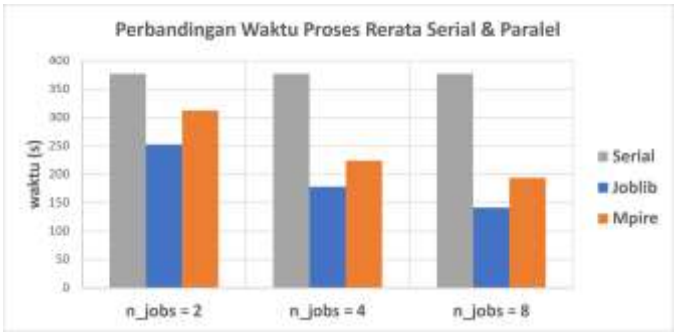
Sesuai pengamatan selama penelitian, saat n_jobs = 1. utilisasi CPU sekitar 36% kemudian saat n_jobs = 2 utilisasi CPU mencapai sekitar 90% dan meningkat menjadi 100% ketika n_jobs dinaikkan menjadi 4 dan 8. Pada saat n_jobs bernilai 1, setiap *task* dikerjakan secara bergantian (serial) hanya pada satu inti CPU sehingga membutuhkan waktu eksekusi yang lama. Utilisasi CPU pada inti yang mengeksekusi program relatif tinggi namun secara keseluruhan utilisasi cukup rendah karena inti CPU yang lain tidak digunakan. Sedangkan ketika nilai n_jobs dinaikkan (lebih dari 1) maka program akan dieksekusi secara paralel. Interpreter Python akan mengalokasikan jumlah *task* yang bisa dikerjakan secara independen ke inti CPU yang tersedia. Hal ini akan menyebabkan semua inti CPU bekerja mengeksekusi program (utilisasi CPU meningkat) sehingga waktu proses semakin cepat. Perbandingan utilisasi CPU antara eksekusi program secara serial (n_jobs = 1) dan paralel (n_jobs = 8) ditunjukkan melalui WRM pada Gambar 10.

Gambar 10: Perbandingan utilisasi CPU ketika mengeksekusi program (dengan pustaka Joblib) – secara serial (kiri) dan secara paralel (kanan)

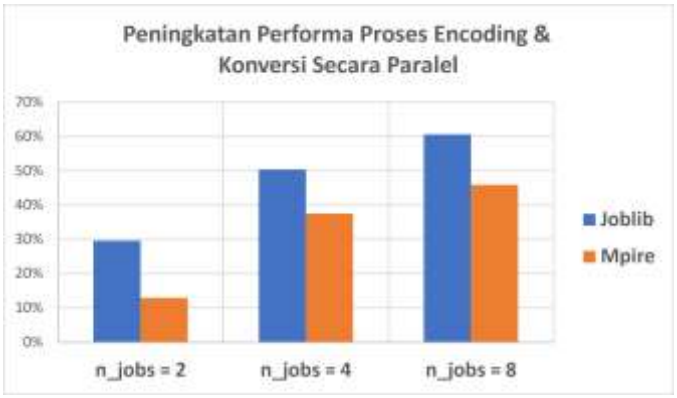
Sementara itu, masih berdasarkan pengamatan melalui WRM, jumlah proses Python yang aktif ketika menggunakan Joblib mengikuti nilai n_jobs. Hal yang agak berbeda terjadi pada Mpire, jumlah proses Python yang aktif lebih banyak satu buah daripada nilai n_jobs dan jumlah *thread* yang lebih banyak daripada Joblib. Meskipun demikian, berdasarkan Tabel 2 dan Tabel 3, secara umum rerata waktu Joblib lebih cepat daripada Mpire. Grafik durasi proses waktu eksekusi program dengan variasi n_jobs ditunjukkan oleh Gambar 11.

Gambar 12 menunjukkan persentase penghematan waktu proses yang diperoleh dengan eksekusi program secara paralel dengan n_jobs = 2, 4, dan 8 berturut-turut sebesar:

- 33%, 53%, dan 63% menggunakan Joblib
- 17%, 41%, dan 49% menggunakan Mpire



Gambar 11: Perbandingan waktu proses rata-rata serial & paralel (Joblib dan Mpire)



Gambar 12: Persentase peningkatan performa dengan komputasi secara paralel

Berdasarkan peningkatan performa yang diperoleh dengan mengunakan semua CPU (8 buah) maka proses *encoding* dan konversi citra dengan Joblib membutuhkan waktu lebih cepat yakni 62.73 ms untuk menghasilkan sepasang citra (normal dan anomali) sedangkan Mpire membutuhkan 86.2 ms/pasang citra. Waktu pemrosesan secara paralel dengan dua pustaka jauh lebih cepat jika dibandingkan dengan proses serial yang membutuhkan 167.51 ms/pasang citra. Secara teoritis, terkait dengan deteksi anomali, maka penerapan komputasi paralel dengan Joblib dan Mpire mampu menangkap munculnya anomali pada data deret waktu yang terjadi minimal setiap 62.73 ms dan 86.2 ms.

V. KESIMPULAN

Proses *encoding* dan konversi data deret waktu menggunakan beberapa *image encoding frameworks* seperti GAF dan MTF banyak digunakan untuk berbagai keperluan termasuk deteksi anomali pada sistem kontrol industri. Pembagian data deret waktu yang berukuran besar menjadi segmen-segmen yang lebih kecil memerlukan proses *encoding* dan konversi berulang. Penelitian ini menerapkan komputasi paralel pada proses *encoding* dan konversi data deret waktu yang diperoleh dari sistem kontrol industri dengan tujuan memperoleh waktu proses yang lebih cepat daripada komputasi serial. Waktu proses yang lebih cepat berarti semakin cepat anomali bisa terdeteksi dan tindakan penanggulangan bisa dilakukan.

Joblib dan Mpire merupakan pustaka Python yang menyediakan kapabilitas eksekusi program secara paralel melalui pengaturan nilai parameter n_jobs yang akan

menentukan jumlah inti logika CPU yang dapat digunakan untuk mengeksekusi program. Hasil yang diperoleh dari sepuluh kali percobaan eksekusi proses *encoding* data deret waktu dan konversi menjadi 4500 citra secara paralel dengan pustaka Joblib dan Mpire menunjukkan penghematan waktu proses yang diperoleh sebanding dengan kenaikan nilai parameter n_jobs.

Nilai n_jobs = 2, 4, dan 8 menghasilkan penghematan waktu proses rata-rata sebesar 33%, 53%, dan 63% (pada pustaka Joblib) dan sebesar 17%, 41%, dan 49% (pada pustaka Mpire) dibandingkan dengan eksekusi program secara serial. Semakin tinggi nilai n_jobs maka semakin banyak inti logika CPU yang bisa digunakan untuk memproses program secara paralel sehingga waktu proses yang diperlukan semakin cepat. Secara teoritis, penerapan komputasi paralel dengan Joblib dan Mpire dengan nilai n_jobs = 8 untuk deteksi anomaly akan mampu merekam munculnya anomali pada data deret waktu minimal setiap 62.73 ms dan 86.2 ms dibandingkan 167.51 ms pada komputasi serial.

REFERENSI

[1] Z. Liu, Z. Zhu, J. Gao, and C. Xu, "Forecast Methods for Time Series Data: A Survey," *IEEE Access*, vol. 9, pp. 91896–91912, 2021, doi: 10.1109/ACCESS.2021.3091162.

[2] D. M. Hawkins, *Identification of Outliers*. Dordrecht: Springer Netherlands, 1980. doi: 10.1007/978-94-015-3994-4.

[3] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A Review on Outlier/Anomaly Detection in Time Series Data," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–33, Apr. 2022, doi: 10.1145/3444690.

[4] C. C. Aggarwal, *Outlier Analysis*. Cham: Springer International Publishing, 2017. doi: 10.1007/978-3-319-47578-3.

[5] K. Shaukat *et al.*, "A Review of Time-Series Anomaly Detection Techniques: A Step to Future Perspectives," in *Advances in Information and Communication*, K. Arai, Ed., in Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2021, pp. 865–877. doi: 10.1007/978-3-030-73100-7_60.

[6] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep Learning for Anomaly Detection: A Review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–38, Apr. 2021, doi: 10.1145/3439950.

[7] H. Rahadian, S. Bandong, A. Widyotriatmo, and E. Joelianto, "Open Source OPC UA Data Traffic Characteristic and Anomaly Detection using Image-Encoding based Convolutional Neural Network," in *2022 7th International Conference on Electric Vehicular Technology (ICEVT)*, Bali, Indonesia: IEEE, Sep. 2022, pp. 52–59. doi: 10.1109/ICEVT55516.2022.9925002.

[8] S. Oh, S. Oh, T.-W. Um, J. Kim, and Y.-A. Jung, "Methods of Pre-Clustering and Generating Time Series Images for Detecting Anomalies in Electric Power Usage Data," *Electronics*, vol. 11, no. 20, p. 3315, Oct. 2022, doi: 10.3390/electronics11203315.

[9] C.-C. Wang and C.-H. Kuo, "Detecting dyeing machine entanglement anomalies by using time series image analysis and deep learning techniques for dyeing-finishing process," *Adv. Eng. Inform.*, vol. 55, p. 101852, Jan. 2023, doi: 10.1016/j.aei.2022.101852.

[10] G. Liu, Y. Niu, W. Zhao, Y. Duan, and J. Shu, "Data anomaly detection for structural health monitoring using a combination network of GANomaly and CNN," *Smart Struct. Syst.*, vol. 29, no. 1, pp. 53–62, Jan. 2022, doi: 10.12989/SSS.2022.29.1.053.

[11] H. Y. Yatbaz, E. Ever, and A. Yazici, "Activity Recognition and Anomaly Detection in E-Health Applications Using Color-Coded Representation and Lightweight CNN Architectures," *IEEE Sens. J.*, vol. 21, no. 13, pp. 14191–14202, Jul. 2021, doi: 10.1109/JSEN.2021.3061458.

[12] J. Faouzi and H. Janati, "pyts: A Python Package for Time Series Classification," *J. Mach. Learn. Res.*, vol. 21, no. 46, pp. 1–6, 2020.

[13] T. Fu, "A review on time series data mining," *Eng. Appl. Artif. Intell.*, vol. 24, no. 1, pp. 164–181, Feb. 2011, doi: 10.1016/j.engappai.2010.09.007.

[14] W. Jiang, "Time series classification: nearest neighbor versus deep learning models," *SN Appl. Sci.*, vol. 2, no. 4, p. 721, Apr. 2020, doi: 10.1007/s42452-020-2506-9.

[15] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using LSTM networks," *Comput. Ind.*, vol. 131, p. 103498, Oct. 2021, doi: 10.1016/j.compind.2021.103498.

[16] I. González, A. J. Calderón, J. Figueiredo, and J. M. C. Sousa, "A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments," *Electronics*, vol. 8, no. 5, p. 510, May 2019, doi: 10.3390/electronics8050510.

[17] A. Cartwright, "OPC-UA: the Flow of Data," *Ai Build TechBlog*, May 27, 2021. https://medium.com/ai-build-techblog/opc-ua-the-flow-of-data-7c3e5c870a4c (accessed May 21, 2023).

[18] N. Muhlbauer, E. Kirdan, M.-O. Pahl, and G. Carle, "Open-Source OPC UA Security and Scalability," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vienna, Austria: IEEE, Sep. 2020, pp. 262–269. doi: 10.1109/ETFA46521.2020.9212091.

[19] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[20] M. Wang, W. Wang, X. Zhang, and H. H.-C. Iu, "A New Fault Diagnosis of Rolling Bearing Based on Markov Transition Field and CNN," *Entropy*, vol. 24, no. 6, p. 751, May 2022, doi: 10.3390/e24060751.

[21] I. A. S. Dewi Paramitha, G. M. A. Sasmita, and I. M. S. Raharja, "Analisis Data Log IDS Snort dengan Algoritma Clustering Fuzzy C-Means," *Maj. Ilm. Teknol. Elektro*, vol. 19, no. 1, p. 95, Oct. 2020, doi: 10.24843/MITE.2020.v19i01.P14.

[22] A. Jayadi, T. Susanto, and F. D. Adhinata, "Sistem Kendali Proporsional pada Robot Penghindar Halangan (Avoider) Pioneer P3-DX," *Maj. Ilm. Teknol. Elektro*, vol. 20, no. 1, p. 47, Mar. 2021, doi: 10.24843/MITE.2021.v20i01.P05.

[23] Q. Kong, T. Siauw, and A. M. Bayen, *Python programming and numerical methods: a guide for engineers and scientists*. London: Elsevier, Academic Press, 2021.

[24] "Python Multiprocessing: The Complete Guide," *Super Fast Python*, Jun. 26, 2022. https://superfastpython.com/multiprocessing-in-python/ (accessed Feb. 26, 2023).

[25] T. Kim, Y. Cha, B. Shin, and B. Cha, "Survey and Performance Test of Python-based Libraries for Parallel Processing," in *The 9th International Conference on Smart Media and Applications*, Jeju Republic of Korea: ACM, Sep. 2020, pp. 154–157. doi: 10.1145/3426020.3426057.

[26] S. Jansen, "MPIRE for Python: MultiProcessing Is Really Easy!," *Medium*, Oct. 27, 2021. https://towardsdatascience.com/mpire-for-python-multiprocessing-is-really-easy-d2ae7999a3e9 (accessed Feb. 22, 2023).

[27] "joblib/joblib." joblib, Feb. 26, 2023. Accessed: Feb. 26, 2023. [Online]. Available: https://github.com/joblib/joblib

[28] "MPIRE (MultiProcessing Is Really Easy)." Slimmer AI, Feb. 20, 2023. Accessed: Feb. 26, 2023. [Online]. Available: https://github.com/Slimmer-AI/mpire

{Halaman ini sengaja dikosongkan}