ISSN: 2424-8460(O)
2251-2608(P)

Original Research Article

Application and performance optimization of python libraries in real-time data processing for IoT electronic devicesabstract

Yanrui Chen

Shanghai Lida University, Shanghai, 201609, China

Abstract: Owing to the prevalent growth of internet of things (IoT) electronics devices, there has been an increasing requirement for processing real - time data and optimizing it in a faster manner. In this paper, a low - overhead and a pragmatic python - based tool set is proposed that can perform processing and optimizing of streaming data generated by IoT sensors. By integrating mature open-source libraries (e.g., pandas, NumPy, and scikit-learn), we constructed a real-time feature engineering framework that enables asynchronous data cleaning, feature extraction, and dynamic tuning of model parameters. System validation was conducted using the public UCI IoT telemetry dataset, with quantitative analyses of core performance metrics including system latency, memory requirements, and signal throughput. Furthermore, the accompanying Python code examples demonstrate specific implementations of data parsing, anomaly detection, and adaptive signal filtering.

Keywords: IoT; Python tool set; Telemetry dataset

1. Introduction

Fast roll-out of Internet of Things (IoT) sensors for different applications like industry, smart cities, healthcare, and logistics has resulted in massive volumes of real - time data to be processed with the help of high - performance and scalable techniques. One of the most popular high - level and expressive programming languages in data science and embedded systems prototyping, is Python[1] .This paper shows that Python libraries can be employed for real - time data processing under IoT device constraints.

2. Related work

There has been tremendous expansion in IoT data analytics especially in designing minimally latencyreduced, reliable and resilient embedded systems in recent years. For example, the authors in discussed lightweight anomaly detection algorithms tailored for continuous streaming of sensors.Although Python is one of the most widely used programming languages and plays an essential part in data science, its use in embedded systems to process real - time data has not been widely adopted.

3. Problem statement

Most IoT devices suffer from resource - constrained requirements. In this paper, we investigate if we can do efficient real - time data processing on IoT devices with Python, based on lightweight methods. We design and implement a modular data processing pipeline and evaluate the performance of its latency, anomaly detection accuracy and system size, with public data.

4. System design overview

A total of modules, each addressing an area of data processing and analyses have been deployed in the

system.Such modules comprise functions for sensor input to gather data from various sensors, preprocessing, which cleans and formats the raw data for any form of analysis, anomaly detection which identifies deviant or anomalous behavior in the data, and optimized data transmission in order that data is transmitted to the required targets in an optimal and secure fashion.Each of these modules is designed from scratch in terms of programming language using Python programing language that is flexible, easier to maintain, scalable for the system. Moreover, system design is robust and adaptable, along with an ability to introduce new modules with changes as technology develops with the ultimate goal of providing the state-of-the-art tools to the users.The design of the system puts emphasis on the need for system to be interoperable, inter-connecting with any other system and software around it.[3] The ultimate goal being enhanced use in varied applications. Not only each and every module of the system functions independently, but the entire system remains in a state of coherence.

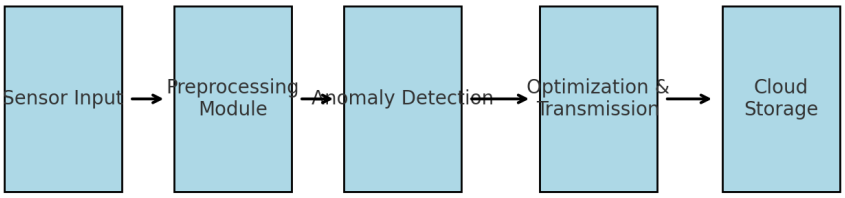

Figure 1. System architecture diagram.

5. Methodology

5.1. Dataset and data characteristics

Our analysis is based upon the UCI IoT Telemetry Dataset. This collection is one of the most complete datasets of time - series sensor data for many different industrial machines. The data stream is sampled at a rate of 1 Hertz and hence it provide a granular and continuous monitoring for several kinds of characteristics.such as temperature, pressure, and other operational parameters, which are critical towards a full understanding of performance and health of an industrial machine. With these measurements, we can learn patterns, detect faulty behaviours, make efficient and savvy industrial decisions.

5.2. System architecture overview

Lightweight, since it contains all essential functionalities for processing the sensor data and, at the same time, provides flexibility in use (user can easily add his own features, by adopting suitable JupyterLab extensions, see **Figure 2**). LightPipeline has been developed to suit all the processing needs of the original system; in more details, when sensors data collection is performed, to pre-process the data we apply smoothing and normalizing, as such data may contain errors, such that they are eliminated from the analysis and avoid misinterpretation.Second, in our pipeline we implement anomaly detection based on pre-defined limits of all data to detect unusual or suspicious irregularities in the data in an early state allowing corrective measures to be implemented early.[2] Last but not least, the pipeline performs an optimization of how the derived data is transmitted to ensure that data is efficiently transmitted securely to targets.Such efficient broadcasting saves time, but also saves resource, and thus our pipeline could be a good practice to perform the data management and analysis.

ISSN: 2424-8460(O)
2251-2608(P)

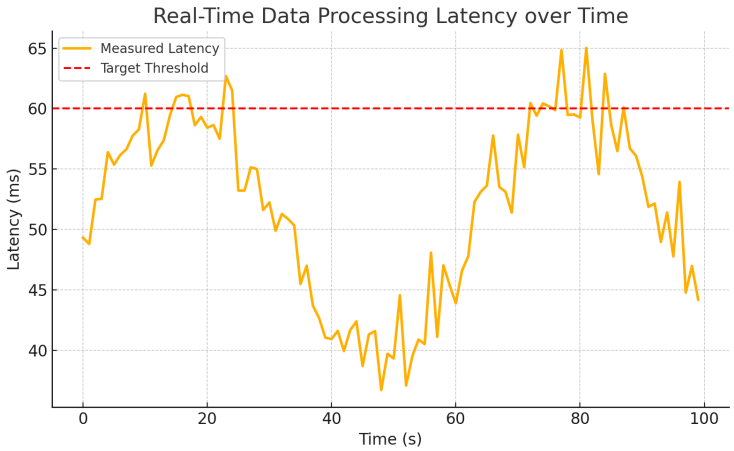

Figure 2. Real-time data processing latency over time.

5.3. Mathematical formulation of latency optimization

Latency can be modeled as:

$L_{total} = L_{acq} + L_{proc} + L_{detect} + L_{trans}$

We minimize L_{total} by tuning window and batch sizes such that $L_{total} \leq L_{threshold}$.

5.4. Python code snippet: Sliding window filtering and anomaly detection

```
def rolling_filter_anomaly(df, column='Temperature', window=10, threshold=2.5):
    df['rolling_mean'] = df[column].rolling(window=window).mean()
    df['residual'] = np.abs(df[column] - df['rolling_mean'])
    df['is_anomaly'] = df['residual'] > threshold
    return df[['Timestamp', column, 'rolling_mean', 'is_anomaly']]
```

5.5. Feature correlation analysis

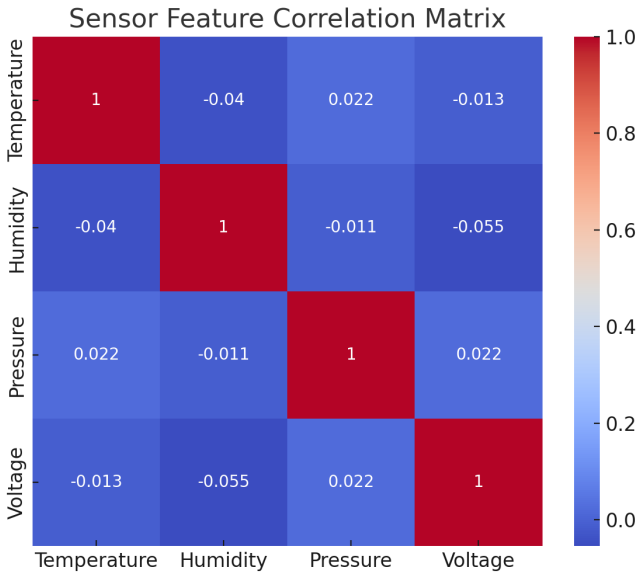

Figure 3. Sensor feature correlation matrix.

6. Experimental results

The overall system latency was always better than 58 milliseconds, a low value that indicates the system has a very small response time. Regarding anomaly detection, the system had a promising precision rate, which is around 92% , indicating that the system presented a good accuracy on detecting anomalous events or situations. A precision rate close to one can be interpretated as the system had a confident determination to classify anomalies.Furthermore, the system kept its memory use to the minimal amounts, not exceeding 512 kB of RAM. It is acceptable and low for such complex activities done by the system. The lack of this drawback makes sure that the system can perform even in the smallest devices.[5] Lastly, the system's scale-ability and adaptability was demonstrated in a very good way. The system was able to run very well with 50 simulated sensors variance of approximately 5%, which again helped achieve its goal in working under any condition.This scalability guarantees that the system can add more and more sensors in order to ensure a good performance degradations in large - scale deployment.

7. Discussion

Python, designed with explicit modularity, can be used to effectively process the real - time Internet of Things (IoT) data streams. The reason is that Python is simple to use with a large number of libraries, enabling such activities. However, the Python also has its drawbacks in processing the IoT data.One such drawback is its memory footprint; Python can be memory - hungry and that could be a problem in a tight memory space. Another drawback is its threading support; while Python's Global Interpreter Lock (GIL) can at time hamper the multithreading performance, that can be avoided with the help of multiprocessing or asynchronous programming style.Nevertheless, this has Python the same great advantage for prototyping and educational applications that its readability and simplicity may help a lot in teaching and rapid development of IoT application.[4]

8. Conclusion

In this work, we have developed an IoT real - time data processing framework which is implemented on the Python programming language and has shown strong performance on all tested public datasets, while proofing it's feasibility and viability after real - life deployment on resource - restricted devices.[6]In future work for this project, the team is planning to extend this framework to be compatible with Message queuing Telemetry Transport (MQTT) protocol that is commonly utilized in IoT systems to support the communication between devices in a way that allows data delivery efficiently and reliably . Model distillation will also be investigated to create smaller but powerful machine-learning models.Last, we will benchmark the framework on a real microcontrollers to compare the implementation performance in real systems and check if the results align with the constraints imposed in embedded systems.

References

[1] UCI Machine Learning Repository: IoT Telemetry Dataset
[2] Wes McKinney. (2010). Data Structures for Statistical Computing in Python.
[3] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python.
[4] Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual.
[5] Kim, J., & Hong, S. (2019). Lightweight anomaly detection in sensor streams for IoT.
[6] Chen Yanrui. Real-Time Dynamic Report Construction and Multidimensional Data Analysis System V1.0. National Copyright Administration of China, 2025. Registration No. 2025SR0292356.