ARTHUR: Machine Learning Data Acquisition System with Distributed Data Sensors

Niels Schneider[a], Philipp Ruf, Matthias Lermer and Christoph Reich[b]
Institute for Data Science, Cloud Computing and IT Security, Furtwangen University of Applied Science, Robert-Gerwig-Platz 1, 78120 Furtwangen im Schwarzwald, Germany

Keywords: Distributed Monitoring System, Machine Learning, Cloud, Data Acquisition, IoT-Cloud Integration.

Abstract: On the way to the smart factory, the manufacturing companies investigate the potential of Machine Learning approaches like visual quality inspection, process optimisation, maintenance prediction and more. In order to be able to assess the influence of Machine Learning based systems on business-relevant key figures, many companies go down the path of test before invest.
This paper describes a novel and inexpensive distributed Data Acquisition System, ARTHUR (dAta collectoR sysTem witH distribUted sensoRs), to enable the collection of data for AI-based projects for research, education and the industry. ARTHUR is arbitrarily expandable and has so far been used in the field of data acquisition on machine tools. Typical measured values are Acoustic Emission values, force plate X-Y-Z force values, simple SPS signals, OPC-UA machine parameters, etc. which were recorded by a wide variety of sensors. The ARTHUR system consists of a master node, multiple measurement worker nodes, a local streaming system and a gateway that stores the data to the cloud. The authors describe the hardware and software of this system and discuss its advantages and disadvantages.

1 INTRODUCTION

The competitiveness of manufacturing companies stands and falls with efficiency in production. Collecting relevant production data in combination with Machine Learning (ML) can contribute to increasing efficiency in many areas of manufacturing. When integrating and implementing Artificial Intelligence (AI)-based systems, production facilities often have to be costly adapted in a time-consuming process to enable a systematic data collection. It is a challenge to assess in advance the impact of ML-based systems on business-relevant key figures. However, such predictions must be made at the latest for the release of funds for larger investments. Therefore, test before invest is a good approach to be able to assess the influence of ML-based systems on business-relevant key figures. Especially for small and medium-sized enterprises (SME)s a low-cost solution, like proposed in this paper, is of particular interest (Kaiser et al., 2021).

Typical ML based systems for manufacturing are visual quality inspection, process optimization and maintenance prediction (Cioffi et al., 2020). Implementing such ML solutions requires obtaining the appropriate data, which can be a costly and difficult problem. Production and automation engineers have a lot of experience and knowledge about possible data collection options. However, there is a lack of ML expertise, which is usually provided by external companies with data scientists in a test before invest project. A typical scenario for testing ML approaches in production is to collect the data at the shop floor, and transfer the data to an on-premise or external cloud storage solution. Data scientists work with the provided data to train ML models and show the results and possible improvements to the manufacturing company. While there are many professional solutions for a Data Acquisition System (DAQ) to enable this workflow, these usually involve high costs and the use of expensive predefined hardware which is compatible with the individual ecosystem (Haizad et al., 2016). The work on hand proposes a low-cost, fast, dynamic, easily adaptable Data Acquisition System, that sends the collected data into a Cloud, where it can further be processed by data scientists. The distributed DAQ, dAta collectoR sysTem witH distribUted sensoRs (ARTHUR), collects data from spa-

[a] https://orcid.org/0000-0002-1341-2704
[b] https://orcid.org/0000-0001-9831-2181

Schneider, N., Ruf, P., Lermer, M. and Reich, C.
ARTHUR: Machine Learning Data Acquisition System with Distributed Data Sensors.
DOI: 10.5220/0011747100003488
In *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*, pages 155-163
ISBN: 978-989-758-650-7; ISSN: 2184-5042

tially distributed different sources and ensures the reliability of the system, its scalability and the time synchronization of the individual data collection components. Monitoring ensures the data quality inspection.

The paper is structured as the following: Section 2 gives an overview of the related work. The requirements for a distributed DAQ are defined in Section 3. Section 4 presents and describes the ARTHUR system. After the evaluation in Section 5 a conclusion is drawn in Section 6.

2 RELATED WORK

The main advantages of distributed measurement systems have been known for some time. The authors in (Grimaldi and Marinov, 2001) describe those advantages where they point out that improved scalability, fault tolerance of singular instances, real-time capabilities, and the collaboration of parallel processes will be the enabler for using new and upcoming software technologies in such distributed environments. The advantages which are described build the basis for modern and distributed Industry 4.0 environments.

In most use cases, the biggest entry barrier, when adapting an existing environment to a distributed environment with Industry 4.0 capabilities and going into the direction of a smart factory, is the high cost of retrofitting older machines and infrastructures. It has been shown, in recent years, that cost-effective solutions can be achieved. One of the solutions is to use systems like a Raspberry Pi, which is expected to hit a market size of 385 million dollar by 2026 in the industry according to (IndustryARC, 2021). Raspberry Pis are used in many different domains, since they are cost-effective, suited for prototyping, and performant IoT devices are stated in a survey of (Saari et al., 2017).

The possibility of using a Raspberry Pi cluster as a backbone for a smart factory has been surveyed in (Kim and Son, 2018). The authors identify the basic functions, which can be implemented by using a Raspberry Pi cluster: Data collection, monitoring, preprocessing and processing, time synchronization, and the secure communication of data. In the outlook, they conclude that an experiment for speed and accuracy has to be conducted. The approach proposed in this paper implements all the basic functions to ensure the integrity and accuracy of heterogeneous data and additionally provides an evaluation that includes the speed. ARTHUR is also part of the ML pipeline but concentrates on distributed data collection and expandability.

Another data acquisition showcase is described in (Kamat et al., 2021). It is shown, that in the smart manufacturing domain it is possible and efficient to use low-cost Raspberry Pis in combination with cloud storage for implementing a predictive maintenance and fault detection solution. The successful usage of Raspberry Pis for different problems in the industry has been proven in many inherently different use cases, compare (Ravindran et al., 2021), (Ramalingam et al., 2019), (D., 2017), (Chu and Yap, 2021), (Song and Moon, 2022). The approach proposed in this paper is conditionally used case-independent, as heterogeneous data is supported and the time constraints are clearly defined.

In (Anik et al., 2022), a cost-effective, scalable and portable open-source Internet of Things (IoT) infrastructure for indoor environment sensing is presented. As multiple Raspberry Pi enabled systems are placed in different areas of an apartment, local databases are synchronized with a central server. Although the basic architecture is similar to the one described in the work on hand, ML-related operations were not considered. Furthermore, the individual Raspberry Pi nodes in this work have their own database which is then synchronized with a central SQL database after a certain time. ARTHUR, on the other hand, sends recorded data from the individual sensor nodes directly to a NoSQL database via an event stream.

In (Ferencz and Domokos, 2018) the authors propose a Java based application used for sensor data collection in combination with Raspberry Pis. The backbone is built on the use of a multi-node Apache Cassandra cluster with a replication factor of 3. It has been shown that the Redis database is superior to Apache Cassandra in many use cases, comparing (Seghier and Kazar, 2021), (Reichardt et al., 2021). Additionally, it has been demonstrated in (Parks et al., 2022) that Redis streams are future proof when designing systems with real-time constraints. After seeing those results, it was then decided that our approach should use the Redis database with its streaming capabilities.

3 REQUIREMENTS FOR A DISTRIBUTED MEASURING SYSTEM

Measurement technology must keep up with the ML task for manufacturing in terms of flexibility and adaptability to varying measurement tasks. Above all, this means effortless expandability in the number of sensors to be integrated, as well as the understand-

ing of common protocols in production (e.g., OPC-UA, CAN, Profibus, etc.). These numerous requirements have been investigated during the development of ARTHUR and will be presented and examined in more detail in the following paragraphs.

R1: Easily Extendable. The system must be easily extendable with additional sensor components for collecting data in order to have the freedom of integrating the needed data sources for the ML application. It should be possible to connect a data source to ARTHUR using various technologies such as OPC-UA, CAN, Profibus, MQTT, REST, TCP, a digital signal, an analog signal, a switch, and much more. The ARTHUR system should provide an abstract programming model that supports the implementation of new interfaces and protocols, respectively, without influencing or disturbing existing components of the system.

R2: Data Pre-Processing on the Edge. Data scientists usually like to work on raw data, but in cases of data privacy, amount of data, timing conditions, or noise, the raw data has to be pre-processed. Typical examples of pre-processing are cleaning, normalizing, aggregation, or building the average of a value across a dataset. When pre-processing the data as close to the respective source as possible, the network load can be drastically reduced (Hafeez and Kathirisetty, 2022). The disadvantages are that the amount of resulting data will contain a lower information density and that errors during pre-processing cannot be corrected afterwards.

R3: Data Quality Assessment. The quality of an ML solution correlates strongly with the quality of the dataset that was used for its training. Ensuring the quality of an acquired dataset within a distributed system brings additional complexity to the overall scenario, as unreliable network transactions must be considered. To ensure the quality of the data, threshold values can be used, for example, to check the data stream for anomalies like sensor failures. (Keijzer and Ferrari, 2022). The ARTHUR system should allow for the extension and utilization of further and customizable quality control tests.

R4: Affordable Data Acquisition. The system should be inexpensive, easy to build, and easy to implement in existing environments. For this purpose, the use of open-source technologies should be favored over expensive licensing or proprietary solutions. The overall system should be reasonably stable, but not implemented in terms of a complete industrial solution and therefore be in line with the motto: “test before invest”. This principle has been touted as a strategic means for driving the digital transformation of European SMEs by European Commission Digital Innovation Hubs (Asplund et al., 2021).

R5: Heterogeneous Collaboration. As there is often an absence of expert data scientists in SMEs, the utilization of gathered data within ML systems is not trivial. To overcome such boundaries, a common collaboration platform for the respective domain experts, e.g., mechanics, shopfloor workers and collaborating data scientists, is required. A data scientist may obtain the sensed data by performing Application Programming Interface (API) calls to the cloud environment.

When ML workflows are made configurable for a respective task, domain experts become more independent with respect to the ad-hock training and assessment of models.

R6: Time Constraints. A prompt reaction to events that occur within an ARTHUR system can be very important, which is why a high data throughput must be ensured throughout the system and bottlenecks have to be eliminated. The speed of the system should also not be affected by a growing number of nodes. Further, the Nyquist Sampling Theorem (Shannon, 1984) requires that the sample rate should be greater than twice the amount of analog signals, which has to be taken into account.

4 ARTHUR A DISTRIBUTED DATA ACQUISITION SYSTEM

ARTHUR is a distributed DAQ system, which has been developed during the research project ”DQ-Meister: quality assessment - predictive quality assessment for complex production processes” and is used in the model factory at the Furtwangen University. The following subsections describe the requirements as well as the architectural structure of the system.

4.1 Machine Learning Infrastructure

In order to fulfill the previously described requirements for the distributed DAQ, a hierarchical approach is presented in Fig. 1, where domain experts and data scientists can have access in order to generate models for their specific ARTHUR systems.

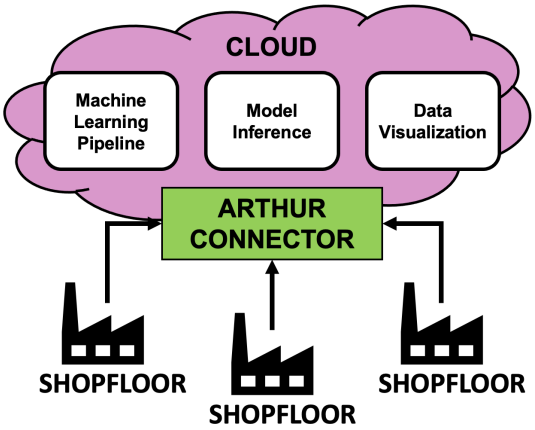

Figure 1: High-level Perspective of the ML Infrastructure.

Data from a shopfloor is collected from a cloud environment using the *ARTHUR Connector* and persisted for future access (see Fig. 1). A web interface as part of the *Data Visualization* module allows visualizing and monitoring the particular datasets. The resource-intense training of a respective model is carried out in a scalable cloud solution, where the *Machine Learning Pipeline* is integrated. Peaks during learning utilization can be tolerated by such a cloud infrastructure in principle. The configuration and implementation of such a ML pipeline is primarily intended for a collaborating ML expert. Trained models are also inferred in this central environment represented in Fig. 1 as *Model Inference*, where the results can also be visualized and compared.

4.2 Architecture

ARTHUR was developed with Python, but its open architecture allows the implementation of components with any other programming language. ARTHUR is made up of distributed computing nodes with predefined roles, typically with one master, several workers, and a streaming system node at the local and global layer (see Fig. 2). A node can take on any of these roles simultaneously if its computing power and local dependence allow it to do so. ARTHURs open architecture makes it easy to add more roles to the system and to increase the number of nodes at will. The following subsections present existing roles and layers of the system in detail.

4.2.1 Redis Database as Middleware

In order to enable a high degree of decoupling of individual roles, the communication between individual nodes was developed on an event-based architecture realized through the streaming system role. ARTHUR uses Redis to implement this role (see Fig. 3). The architectural concept of ARTHUR is not bound to the use of Redis, other streaming systems such as e.g., Kafka can be used instead. Redis is a key value based open source (BSD licensed) multi-model in-memory database with a very high read and write speed. Redis collects events of the same type in a stream defined by a unique key name. A key is structured in a similar way as a file path. It is a string consisting of words and separators e.g., ”MEASUREMENT:START”. All measurement events of the type ”START” can be found under this key. The payload of an event also consists of a combination of key-value pairs. To ensure the order of multiple events in a stream, each event is assigned a UNIX timestamp as well as a sequence number. If the timestamp of an event is lower than the timestamp of the last added event resulting due to the clock drift between different nodes, the largest timestamp of the last added event with an increased sequence number is used. To minimize such occurrences, the streaming system node also includes an NTP server, which all local nodes use for time synchronization.

Local Caching. To ensure fast communication, ARTHUR provides at least one local streaming system node for each shopfloor. When using a single streaming system node instance, a single point of failure occurs. To avoid this, several replication strategies are possible depending on the use case. In order to enable global access to the recorded data of a shopfloor streaming system node, ARTHUR Connectors exists for transferring the local data to a cloud storage (see Fig. 2). For the implementation of an ARTHUR Connector with Redis, Redis consumer groups can be used. With them, it is possible to track the information about already consumed data between several consumers of a group. This makes it possible to use a consumer group to consume the data from the local Redis node and automatically remove what has already been consumed. The local Redis instance thus has similar properties to a limited queue. By persisting individual events with ARTHUR, a historical view of a shopfloor can be made possible and a dataset for a ML task can be acquired.

Stream Processing. Redis supports the use of stream processors with RedisGears. These allow operations to be performed directly on a stream of data. For this, RedisGears runs a python code directly in a Redis instance. For example, a stream processor can be used to check an event stream for a threshold violation. If a threshold violation is detected, another event can be generated to react to it. Stream processors are used on the shopfloor level to process the data

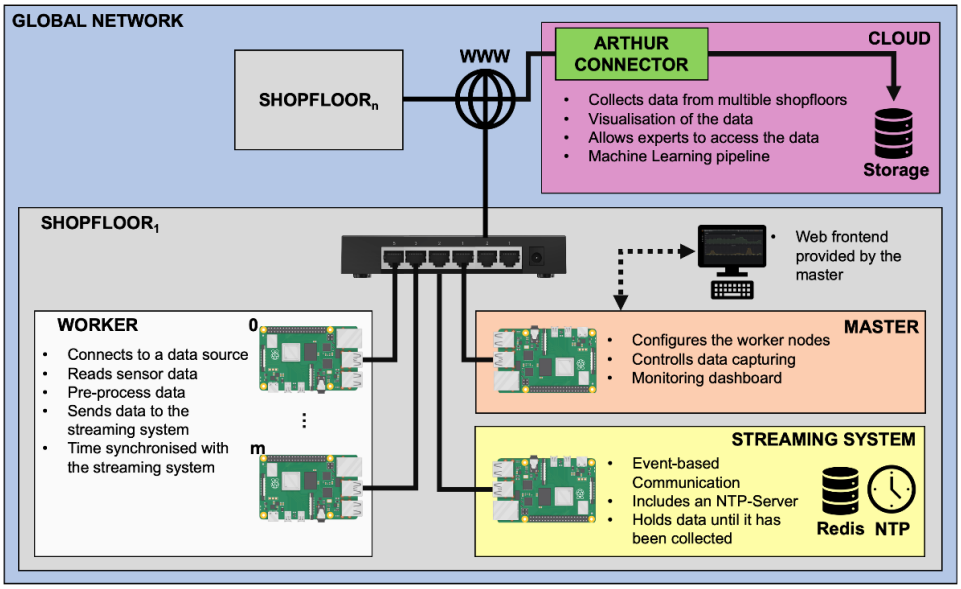

Figure 2: ARTHUR Structure with Nodes (Master, Worker, etc.).

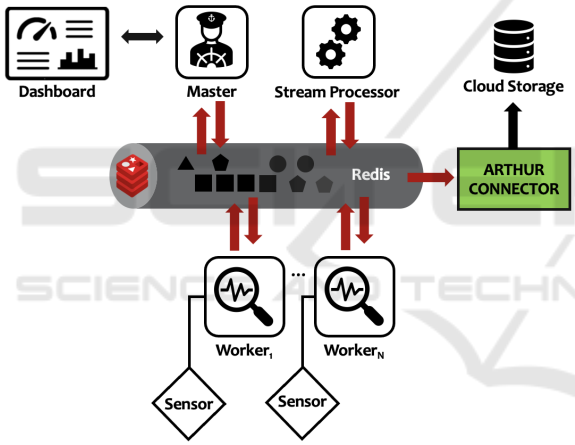

Figure 3: Redis as streaming middleware.

stream as early as possible. Furthermore, Redis also enables the direct deployment of an ML model within the streaming system via RedisML. The proximity to the data enables a fast execution time which can be needed for application use cases with strong time constraints.

4.2.2 Node Event Handler

If a node is interested in a certain type of event, for example, the arrival of an event to start the data acquisition, it observes a corresponding stream for the arrival of this event with an event handler. Event handlers can be used to define the reaction of a node to an event that occurs or fails to occur. If a node crashes and then comes back online, it looks at missed events and decides what to do next. In principle, the information about who created an event and who reacted to it is irrelevant for an individual node. A single node therefore only needs the information about the IP address to connect with the streaming system to be part of ARTHUR. This can be determined as a configuration when setting up a node. This property enables considerable simplification in comparison to a request-response system.

4.2.3 Master Node

The role of the master node provides a central control unit for configuring the individual worker nodes via the following events:

- `StartEvent(startTime: UNIX Nanosecond timestamp)`
- `StopEvent(stopTime: UNIX Nanosecond timestamp)`
- `ConfigChangeEvent(diviceID: Unique device ID, config: JSON)`

An important role of the master is the transmission of a start event for the simultaneous start of the data recording. Furthermore, the master functions as a dashboard for monitoring the system. Grafana, an open-source dashboard, is used for this purpose. Grafana enables the connection of the local Redis instance as a data source from which data is constantly read out. Such a dashboard can be adapted to any use case and is recommended to monitor the status of the database. With the dashboard, it is possible to measure the network load caused by the communication and to observe the available memory of the local Redis instance.

4.2.4 Worker Node

The worker role serves as an adapter between ARTHUR and various data sources. A worker is controlled by several event handlers who constantly wait for events like:

- `MEASUREMENT:START`
- `MEASUREMENT:STOP`
- `WORKER:CONFIG:CHANGE`

To achieve optimal hardware utilization and to guarantee a high data acquisition rate, the worker starts several processes inspired by the Extract Transform Load (ETL) design pattern (see Fig. 4). All processes are started and managed from a main process, called controller and communicate with each other over first in first out (FIFO) queues. Synchronous processing of the data in a queue is necessary to ensure its correct order. The following sections discuss the individual ETL processes in more detail.

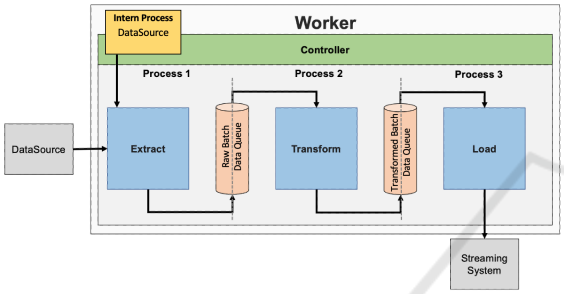

Figure 4: Dataflow in a worker node.

Extract Process. The extract process receives data from a connected data source. This can be done in different ways and is strongly dependent on the use case of the worker. Data can be directly produced by an intern worker process or be obtained from external sources e.g., OPC-UA.

If the master starts data acquisition by sending a start event, all distributed workers react to the event with their corresponding event handlers and inform the controller. The controller of a worker forwards the start signal to the extractor process. During data acquisition, the extractor continuously reads one ore multiple targets and writes the recorded data batches into the *Raw Batch Queue* for further processing.

Transform Process. The transform process reads incoming batches from the *Raw Batch Queue* and processes them further. After the data has been successfully processed, it is transferred to the *Transformed Batch Queue*. This process can be used to enable individual data pre-processing at the edge. For example, averages can be formed from the raw data to minimize the number of data points resulting in a reduced load for the system.

Load Process. The load process is responsible for persisting incoming data batches from the *Transformed Batch Queue* by sending them to the streaming system. The streaming system further takes care of the received data (see section 4.2.1). For this purpose, the load process uses the Redis interface to attach data batches as events to the `MEASUREMENT:DATA` stream.

.

5 EVALUATING THE REQUIREMENTS

In this section, the implemented ARTHUR system is evaluated and discussed according to the requirements stated in section 3.

Description of the Used Hardware. ARTHUR has been successfully used to acquire data from multiple data sources in a grinding process on a CNC machine tool. All worker nodes as well as the master node and the streaming system node were setup on a Raspberry Pi 4 with 8GB RAM. The usage of a Raspberry Pi is not mandatory. ARTHUR can easily be deployed on different hardware if e.g., a use case requires more computational power. The choice of the Raspberry Pi was made for several reasons. The connection of different data sources to a Raspberry Pi is easily possible through already existing expansion cards. The Pi also has an ethernet port that allows a direct network connection.

In the experiments, we used a separate worker node (see Fig. 5), a master node, and a streaming system node for each data source. To evaluate the maximum sample rate, data synchronization, etc. during the experiment we used a signal generator instead of the CNC machine.

The following Table 1 provides an overview of the data sources in the real manufacturing process. Both

Table 1: Overview of all data sources.

Data source	Signal
Dittel AE6001	acoustic emission
Kistler force plate	X-Y-Z force values
Simens SIMATIC IPC227E	machine tool values

the acoustic emission and force values were recorded via the GPIO HAT MCC118 analog-to-digital converter mounted on the Raspberry Pi. The MCC118

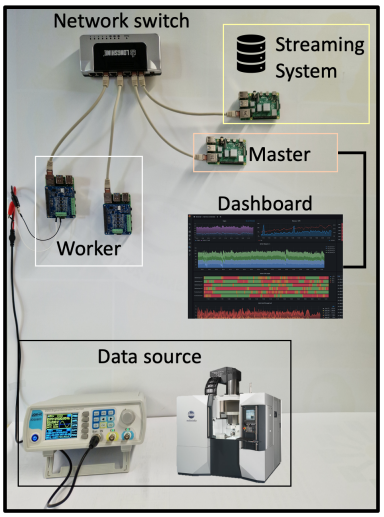

Figure 5: Experimental Setup.

HAT of the Raspberry Pi is able to acquire data at a sample rate of 100 kHz per second with a resolution of 12-bit in a value range of +-10 volts and has 8 input channels. The maximum recording rate of 100 kHz is divided equally between several channels. Thus, when using two channels, a maximum data recording rate of 50 kHz per channel is possible. The MCC118 Python library `daqhat` was used to implement the extract process described in the section 4.2.4. The same extract process implementation was used to read AE and force values.

Evaluation of R1: Easily Extendable. In order to receive data from the SIMATIC IPC, an OPC-UA server was configured on it, which publishes the machine parameters of the machine tool. These were then read out by a worker via OPC-UA, for which the python library `opcua` was used in the reader process.

Connecting additional data sources with OPC-UA or other technologies does not affect the recording of existing workers. If adding a new work node, it is sufficient to configure the IP address of the streaming system node. The requirement for simple expandability of the system was therefore fulfilled. How many worker nodes can be added depends on the amount of data collected, which is related to the sample rate configured in the worker node.

Evaluation of R2: Data Pre-Processing on the Edge. The analog signals recorded by the MCC118 must be further processed for their storage. The MCC118 writes recorded values of all its channels into a single list. The list is then pre-processed by the transform process presented in section 4.2.4 and thus divided into its individual channels. In addition, the recorded force value must be converted from volt to newton.

Pre-processing of the data can be done as desired by the customizable transformation process and s the requirement R2.

Evaluation of R3: Data Quality Assessment. Figure 6 shows the input data of the value of the force plate Z direction, defining two thresholds. If the measurement is outside the $Threshold_{min}$ and $Threshold_{max}$, the worker node of ARTHUR can signal alarms locally or/and sends an event to the cloud to avoid failure in data collection. Due to the simple possibility of extending the system with quality mechanisms, requirement R3 can be considered fulfilled.

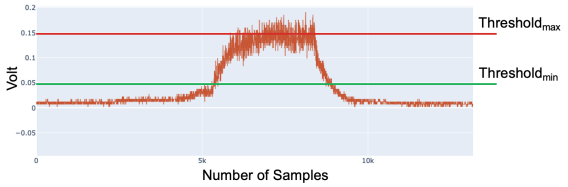

Figure 6: Sensor input with Thresholds.

Evaluation of R4: Affordable Data Acquisition. Although the proposed ARTHUR system consists of multiple Raspberry Pis, other low-cost embedded devices can be utilized. Low-cost sensors can be connected to any device capable of acting as a worker node, e.g., a laptop or stationary PC, where resource-intense operations like pre-processing can be carried out. Of disadvantage is that the operation of Raspberry Pis might be inappropriate for the rough manufacturing environment. The presented system is individually adaptable to the cost requirements and fulfills the demand for a cost-effective system formulated in R4.

Evaluation of R5: Heterogeneous Collaboration. The collaboration of domain experts and data scientists can be accomplished by granting access to datasets in the cloud system. When the resulting ML pipelines are being made available to the cloud environment, the re-training with new datasets and the inference of respective models is made possible. In principle, there can be triggers for starting a particular ML training task in the cloud, using data from a specific ARTHUR system or a fusion of multiple ARTHUR systems data respectively. As the cloud interface can be extended with arbitrary logic, the incremental training of a model can be started for example whenever a certain amount of new data becomes available. At the same time, existing work-

flows/pipelines can dynamically be adapted to current situations, for example reacting to sensor drifts. This enables heterogeneous collaboration and meets the requirement of R5.

Evaluation of R6: Time Constraints. By optimally utilizing the resources of the Raspberry Pi and the MCC118 GPIO HAT, a maximum recording rate of 80 kHz per worker, resulting in a 2 MB/sec network load, could be achieved without causing queue jamming. The CPU load of the local streaming system node implemented with Redis was utilized by 3 workers at an average of 60%, enabling fast processing of events. A reaction to incoming events takes place immediately after their occurrence. If more workers are added in the future, Redis can be scaled both horizontally and vertically to continue ensuring a fast response time. The system met the time requirements in the DQ-Meister project and thus fulfills requirement R4.

6 CONCLUSION

In this work, a Data Acquisition System for distributed sensors was discussed with respect to prototypical ML operations. This hierarchical system consists of multiple embedded sensing devices, which are synchronized. The system has been shown to be easily extendable and it is not dependent on specific technologies like OPC-UA. ARTHUR supports data pre-processing close to the resource to reduce network load and can additionally be tailored to specific data pre-processing use cases. The same goes for specific data quality assessments where thresholds for data values can be defined and acted upon on with customized events. The proposed system was evaluated with several Raspberry Pis and low cost sensors making it affordable, even for early stages of prototyping. The provided interface enables a heterogeneous collaboration between domain experts and data scientists. In order to uphold the time constraints of different use cases, appropriate hardware can be used without a problem as the architecture of the system is hardware agnostic.

For the future, it is planned to integrate cameras for visual surface quality inspection and integrate ARTHUR into non-Raspberry Pi hardware. Additionally we intend to add more complex hardware, like a precision RTP oscilloscope with a max. sample rate of 40 G sample/s in order to further test the limits of the system. Furthermore a thorough security analysis of the system has to be conducted in order to be prepared for future Machine Learning audit requirements. Finally, the system will be evaluated with a complex manufacturing use case.

ACKNOWLEDGMENT

This work as outcome of the project DQ-Meister*in with the project number P2021-01-013 has received funding from the Carl Zeiss Stiftung.

REFERENCES

Anik, S. M. H., Gao, X., Meng, N., Agee, P. R., and McCoy, A. P. (2022). A cost-effective, scalable, and portable iot data infrastructure for indoor environment sensing. *Journal of Building Engineering*, 49:104027.

Asplund, F., Macedo, H. D., and Sassanelli, C. (2021). Problematizing the service portfolio of digital innovation hubs. In Camarinha-Matos, L. M., Boucher, X., and Afsarmanesh, H., editors, *Smart and Sustainable Collaborative Networks 4.0*, pages 433–440, Cham. Springer International Publishing.

Chu, Y. B. and Yap, W. K. (2021). Raspberry pi based wireless interface system for automated microfabrication in the context of industry 4.0. In Zakaria, Z. and Emamian, S. S., editors, *Advances in Electrical and Electronic Engineering and Computer Science*, pages 117–123, Singapore. Springer Singapore.

Cioffi, R., Travaglioni, M., Piscitelli, G., Petrillo, A., and De Felice, F. (2020). Artificial intelligence and machine learning applications in smart production: Progress, trends, and directions. *Sustainability*, 12(2).

D., D. (2017). Industrial automation using iot with raspberry pi. *International Journal of Computer Applications*, 168:44–48.

Ferencz, K. and Domokos, J. (2018). Iot sensor data acquisition and storage system using raspberry pi and apache cassandra. In *2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE)*, pages 000143–000146.

Grimaldi, D. and Marinov, M. (2001). Distributed measurement systems. *Measurement*, 30(4):279–287.

Hafeez, S. and Kathirisetty, N. (2022). Effects and comparison of different data pre-processing techniques and ml and deep learning models for sentiment analysis: Svm, knn, pca with svm and cnn. In *2022 First International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR)*, pages 1–6.

Haizad, M., Ibrahim, R., Adnan, A., Chung, T. D., and Hassan, S. M. (2016). Development of low-cost real-time data acquisition system for process automation and control. In *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 1–5, Ipoh, Malaysia. IEEE.

IndustryARC (2021). Industrial raspberry pi market 2022 - 2027. *IndustryARC*, Report Code: AIR 0372.

https://www.industryarc.com/Report/19454/industrial-raspberry-pi-market.html.

Kaiser, J., Terrazas, G., McFarlane, D., and de Silva, L. (2021). Towards low-cost machine learning solutions for manufacturing smes. *AI & SOCIETY*.

Kamat, P., Shah, M., Lad, V., Desai, P., Vikani, Y., and Savani, D. (2021). Data acquisition using iot sensors for smart manufacturing domain. In Singh, P. K., Polkowski, Z., Tanwar, S., Pandey, S. K., Matei, G., and Pirvu, D., editors, *Innovations in Information and Communication Technologies (IICT-2020)*, pages 393–400, Cham. Springer International Publishing.

Keijzer, T. and Ferrari, R. M. (2022). Threshold design for fault detection with first order sliding mode Observers. *Automatica*, 146:110600.

Kim, C.-S. and Son, S.-B. (2018). A study on big data cluster in smart factory using raspberry-pi. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5360–5362.

Parks, D. F., Voitiuk, K., Geng, J., Elliott, M. A., Keefe, M. G., Jung, E. A., Robbins, A., Baudin, P. V., Ly, V. T., Hawthorne, N., Yong, D., Sanso, S. E., Rezaee, N., Sevetson, J. L., Seiler, S. T., Currie, R., Pollen, A. A., Hengen, K. B., Nowakowski, T. J., Mostajo-Radji, M. A., Salama, S. R., Teodorescu, M., and Haussler, D. (2022). Iot cloud laboratory: Internet of things architecture for cellular biology. *Internet of Things*, 20:100618.

Ramalingam, S., Baskaran, K., and Kalaiarasan, D. (2019). Iot enabled smart industrial pollution monitoring and control system using raspberry pi with blynk server. In *2019 International Conference on Communication and Electronics Systems (ICCES)*, pages 2030–2034.

Ravindran, V., Ponraj, R., Krishnakumar, C., Ragunathan, S., Ramkumar, V., and Swaminathan, K. (2021). Iot-based smart transformer monitoring system with raspberry pi. In *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–7.

Reichardt, M., Gundall, M., and Schotten, H. D. (2021). Benchmarking the operation times of nosql and mysql databases for python clients. In *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–8.

Saari, M., bin Baharudin, A. M., and Hyrynsalmi, S. (2017). Survey of prototyping solutions utilizing raspberry pi. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 991–994.

Seghier, N. B. and Kazar, O. (2021). Performance benchmarking and comparison of nosql databases: Redis vs mongodb vs cassandra using ycsb tool. In *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*, pages 1–6.

Shannon, C. E. (1984). Communication in the presence of noise. In *Proceedings of the IEEE*, volume 72, pages 1192–1201.

Song, Z. and Moon, Y. B. (2022). A cloud-fog continuum computing architecture for cyber-manufacturing systems. *Advanced Engineering Forum*, 45:97 – 102.