Developing an AI-enabled Industry 4.0 platform - Performance experiences on deploying AI onto an industrial edge device*

Holger Eichelberger
eichelberger@sse.uni-hildesheim.de
University of Hildesheim
Software Systems Engineering
Universitätsplatz 1
31141 Hildesheim
Germany

Gregory Palmer
gpalmer@l3s.de
University of Hannover
L3S Research Center
Appelstraße 9a
30617 Hannover
Germany

Claudia Niederée
niederee@l3s.de
University of Hannover
L3S Research Center
Appelstraße 9a
30617 Hannover
Germany

Abstract

Maximizing the benefits of AI for Industry 4.0 is about more than just developing effective new AI methods. Of equal importance is the successful integration of AI into production environments. One open challenge is the dynamic deployment of AI on industrial edge devices within close proximity to manufacturing machines. Our IIP-Ecosphere[1] platform was designed to overcome limitations of existing Industry 4.0 platforms. It supports flexible AI deployment through employing a highly configurable low-code based approach, where code for tailored platform components and applications is generated.

In this paper, we measure the performance of our platform on an industrial demonstrator and discuss the impact of deploying AI from a central server to the edge. As result, AI inference automatically deployed on an industrial edge is possible, but in our case three times slower than on a desktop computer, requiring still more optimizations.

1 Introduction

Artificial intelligence (AI) is one of the main pillars of Industry 4.0, e.g., to support automated decision making processes, such as AI-based quality control [5]. Although many libraries for development are available, other steps in the data science process for Industry 4.0 are less well supported, in particular the deployment of AI close to the target machines, e. g., on industrial edge devices, to avoid latencies.

While several Industry 4.0 software platforms do exist, e. g., the Siemens MindSphere, they usually fall short in AI aspects like functional encapsulation, remote deployment, openness and heterogeneous edge support [7]. In research on IIoT platforms, approaches typically cover only selected topics, e.g., application of specific AI methods in [3, 4] or evaluations with micro computers, e.g., Raspberry Pi [1], rather than industrial devices. With our platform, we follow a radically different approach. We ease the development of applications by generating large parts of the code (low code) including performance probes based on a model-based configuration. This speeds up development and creates more robust code. Using this low-code approach, we realize a flexible and open platform in the IIP-Ecosphere project that enables easy-to-use AI and supports relevant industry standards. The main contributions of this paper are (a) a novel dynamic AI deployment mechanism and (b) a performance characterization for AI deployment on the edge.

As context for this paper, we use the IIP-Ecosphere platform (Section 2) and a visual quality inspection use-case (Section 3). We measure the processing time of the AI when deployed on an edge device (Section 4), observing a performance difference of factor 3. We conclude with further options for optimization.

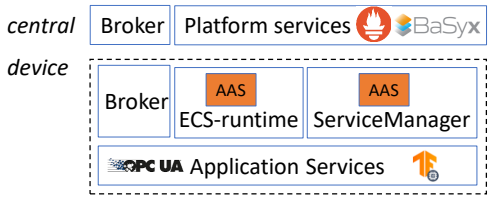

Figure 1: Platform components for deployment.

2 Platform

The Open Source IIP-Ecosphere platform[2] aims at augmenting existing production environments with easy-to-use AI services, operating on data transported through (standardized) protocols. Applications for the platform are realized in terms of services, which operate on data streams. In our configuration model, we link services to applications consisting of data sources such as protocol connectors, e.g., for OPC UA or MQTT, data processors and data sinks. Then

*Supported by the German Ministry of Economics and Climate Action (BMWK), grants 01MK20006A/D.

[1] Intelligent Industrial Production Ecosphere, https://www.iip-ecosphere.de/

[2] https://github.com/iip-ecosphere

a model-based process integrates the services to executable applications packaged as deployable artifacts.

Our platform can deploy services to (edge) devices - a demand of our industrial partners. Then, our platform operates in a distributed fashion, where some components run centrally, while others must be installed on target devices, e.g., to deploy services.

A central installation (see Figure 1) includes a global communication `Broker` and platform services. Such services are a monitoring facility (e.g., Prometheus) and an information model representing all devices/services, their runtime properties and operations in terms of an Industry 4.0 Asset Administration Shell (AAS) realized using Eclipse BaSyx.

On a device, we need a device abstraction (`ECS-Runtime`) and a `ServiceManager`, offering container/service deployment operations and providing access to the runtime state [6]. A `Broker` enables local communication among services running on the same device. If supported by the device, these components can be containerized (the dashed box in Figure 1), easing the installation of services and their technical dependencies such as Python or TensorFlow (TF).

3 Visual Inspection Use Case

As an initial demonstration and validation of our platform, we realized an AI-based visual quality inspection application for individualized products[3]. For a realistic setting, we use industrial components, such as a 5 axis Universal Robots UR5e cobot, a Robotiq wrist cam and a Phoenix Contact AXC F 3152, a combined Programmable Logic Controller (PLC) and edge device[4]. A PC serves as central installation and a tablet presents the application user interface.

As products we are using small aluminum car models with configurable properties, e.g., wheel color, number of windows or engravings. A car configuration is captured as an AAS and made available to the platform/application. Quality inspection is performed based on three position images taken by the wrist camera mounted on the cobot arm. Based on the AI results, the quality is measured in two aspects: a) conformance with production configuration and b) quality of production (existence of scratches).

A first version of the application imposed several integration challenges [8], e.g., excessive container sizes due to TF. Based on these experiences, we improved the platform, which allows now for automated deployment and runtime-monitoring of services.

As shown in Figure 2, the application employs four services and two connectors: In three rounds, the cobot arm is repositioned, the `Cam Source` takes a picture and the `Python AI` extracts relevant properties. First, the QR code of the car is detected. Then, for the two pictures taken from the sides of the car, the AI detects wheel color, engraving, number of windows and

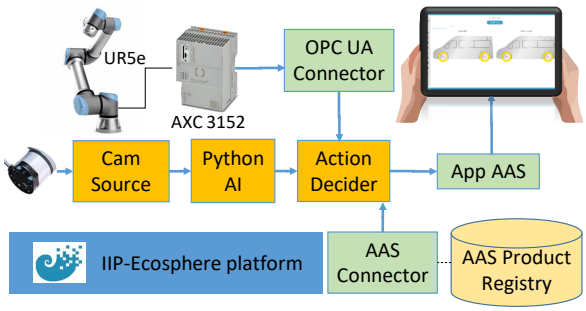

Figure 2: Application components and data flows.

scratches. The AI results are received by the `Action Decider`, which compares them to the AAS car configuration (identified by the QR code) and delivers the final inspection result to the `App AAS`. Two connectors integrate the edge (`OPC UA connector`) and the car configuration AAS (`AAS connector`). Based on its modelling in the platform configuration the deployable application was generated through model-based integration, which also inserts Micrometer probes.

We briefly describe now the Python-based AI service as core subject of our performance measurements, which is based on generally available AI frameworks that can be executed on both, the edge and the PC. The tire color detection is solved using a range-based threshold with ranges for the four configurable colors. For the remaining tasks we utilized deep convolutional neural networks (CNNs) with separable convolutional layers, a light-weight alternative (smaller and less compute intensive) to normal convolution without significantly impacting performance [2]. Given the limited capacity of the target edge device, we train a distinct model for each task that can be loaded when required, instead of training one big model for all three tasks. For each task we apply thresholding for preprocessing to eliminate colors within the range of the car's color. For a detailed description of the preprocessing steps, loss functions and illustrations of the AI approach, please refer to [8].

4 Performance Experiment

For the initial application, a first measurement of the AI response time indicated a performance drop of factor 7-12 between edge and PC [8]. From the improved solution, we expect better results.

W. r. t. hardware, as in [8] we use a Phoenix Contact AXC F 3152 edge with a 2 core Intel Atom processor, 2 GByte RAM, and 32 GByte SD-based hard drive. For the central installation, we use a Lenovo ThinkCentre with an Intel Core i5-7400 3.00 GHz, 32 GByte RAM and 2 TByte SDD. Our software setup[5] consists of the IIP-Ecosphere platform on Linux (Debian 11 on the PC, custom on the AXC). We compare two variants, running the AI a) on the edge and b)

[3] Illustrating video: https://youtu.be/36Xtw1L2XkQ.
[4] https://www.phoenixcontact.com
[5] https://doi.org/10.5281/zenodo.7157607

variant	a) no broker	a) broker	b) PC
first	2.13	2.20	0,61
min	1.79	1.80	0,38
avg	1.83	1.84	0,51
stddev	0.03	0.02	0,14
max	1.89	1.91	0,83

Table 1: AI inference response times (in seconds)

on the PC (as "compensation", with the camera and OPC UA connector running on the edge). As in a live demonstration, the PC also runs the AAS server, the central broker and Prometheus. As the AI in variant a) is the only service on the edge, we can devise two sub-variants, with and without local broker. The AI service is realized in Python 3.8.10 using TF-lite 2.8.0.

We measure the response time by directly instrumenting the Python script. Further, we take the memory usage reported by Micrometer/Prometheus into account. One run of the quality inspection process takes around 50 seconds and delivers three response time values, one per picture. We repeat the process 10 times leading to 30 measures per (sub-)variant.

Table 1 summarizes the measured response times. For each (sub-)variant, we consider the first measurement as an outlier as it includes AI model loading. While the two sub-variants on the edge do not differ much, the PC runs the AI in average 3.6 times faster. However, it exposes a higher deviation, an effect probably caused by the platform services on the PC. While the edge device operates at 95 % memory usage probably due to swapping, the PC utilizes around 30 % of its memory. Running the same AI script on the same device without platform services leads to an execution time of around 1s, which we consider as baseline.

The results are positive compared to our initial measurement [8]. In our context, the measured times are acceptable, as this is just a fraction of the time that the robot arm takes for moving among two positions. However, in a factory environment with a typical machine pace of 8 ms, the AI would be a bottleneck outside the context of our robot arm demonstrator.

Optimization options include using a *different AI library*, e.g., TF instead of TF-lite, or optimized device-/vendor-specific builds of the respective library. However, these are not available for all devices.

A *smaller container* could positively influence execution time as well as deployment time (which is about 3 minutes for the application services). Here, modularized libraries containing only the AI inference (in contrast to a monolithic TF with a size of more than 2 GByte) could be helpful, but are often not provided.

Code optimization is mostly promising for the platform components since more than 89 % of the execution time in the AI script is spent on the AI inference (AI optimization would be required here).

Finally, *hardware acceleration* through graphic (GPU) or tensor (TPU) processors, which slowly become available for industrial edge devices, could significantly improve the response time.

5 Conclusion & Future work

In this paper, we provided initial performance insights into automatically deploying AI functionality onto an industrial edge device. The AI originates from a visual quality inspection use case and was intentionally developed without upfront optimizations. We deployed the AI service utilizing the standard-based infrastructure capabilities of the IIP-Ecosphere platform.

Our results indicate that a general purpose CNN AI can be successfully deployed on an industrial edge, here a device for real time control applications in control cabinets. With more powerful devices or industrial PCs, which nowadays may include GPUs or TPUs, better performance is possible. Further optimizations are outlined in the previous section.

As future work, we plan to further optimize the AI execution performance on resource-limited industrial devices. In particular, we are interested in the extent to which building TensorFlow from source for the instruction set of the target system and with architecture specific compiler options or hardware co-processors, can improve the execution performance. Further, we plan to equip our platform with ready-to-use AI services, e.g., for federated learning.

References

[1] P. Charalampidis, E. Tragos, and A. Fragkiadakis. "A fog-enabled IoT platform for efficient management and data collection". In: *Computer Aided Modeling and Design of Communication Links and Networks*. 2017, pp. 1–6.

[2] A. G. Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv:1704.04861* (2017).

[3] S. Chen et al. "An IoT Edge Computing System Architecture and its Application". In: *International Conference on Networking, Sensing and Control (ICNSC)*. 2020, pp. 1–7.

[4] F. Foukalas. "Cognitive IoT platform for fog computing industrial applications". In: *Computers & Electrical Engineering* 87 (2020), p. 106770.

[5] G. Palmer et al. "The Automated Inspection of Opaque Liquid Vaccines". In: *ECAI 2020*. 2020, pp. 1898–1905.

[6] M. G. Casado and H. Eichelberger. "Industry 4.0 Resource Monitoring - Experiences With Micrometer and Asset Administration Shells". In: *Symposium on Software Performance*. 2021.

[7] C. Sauer et al. *Current Industrie 4.0 Platforms – An Overview*. doi:10.5281/zenodo.4485756. 2021.

[8] H. Eichelberger et al. "Developing an AI-enabled IIoT platform - Lessons learned from early use case validation". In: *SASI4 @ ECSA'22*. 2022.