AI. Anywhere.

Why (I)IoT needs modular AI inference on the Edge

prof. dr. Maurits Kaptein

Scailable
14-01-2021

maurits.kaptein@scailable.net

Scailable.

Artificial Intelligence (AI) is the critical technology enabling next generation (I)IoT systems across industries. To utilize AI inference at scale, system architects need to deploy ML and deep-learning models on embedded systems in a modular fashion.

AI is empowering the next generation of the (Industrial-) Internet-of-Things. The (I)IoT demonstrated its value in the last decade by enabling the real-time collection of data to monitor complex (production) processes. In its next generation, the (I)IoT will not solely consist of passive measuring devices: it will consist of semi-intelligent entities that act upon incoming data directly. This allows us to move beyond monitoring towards real-time proces control. As patterns in the data rapidly become too complex to process using pre-programmed, rule-based systems, AI becomes a necessity. AI in (I)IoT becomes valuable when the feedback loop from data collection, to analysis, to actions is closed efficiently and securely.

The AI in (I)IoT market is forecasted to grow from 7 billion USD in 2020, to over 16 billion USD in 2024.[1] The expected compound annual growth rate (CAGR) of 26 percent will be driven by the need to efficiently process huge volumes of real-time data generated by IoT devices and act upon these data. Given the size of the data and the costs of transferring data from edge devices to the cloud, it is clear that scalable and secure AI in (I)IoT cannot adopt the standard cloud-processing model. Data will need to be processed and acted upon on the edge (i.e., as close as possible to its originating source) to ensure real-time processing, lower energy consumption, and lower maintenance costs.

AI is transformative across industries. Consider the following promising examples:

- *Predictive Maintenance (PdM):* Inspecting the maintenance state of hard-to-reach surfaces, such as harbor silos, is challenging and costly. AI on (I)IoT makes it possible to deploy deep neural networks (DNNs) that detect surface irregularities directly on (autonomous) drones. These drones scan the surface of the silo—without human intervention—and directly analyze the pictures *on device*.[2] When an irregularity is detected, its location is transmitted to the cloud and the repair process can be initiated.

- *Predictive intelligence:* Data collection has made its way into the farm in recent years: cattle is often, even while in the field, equipped with sensors that measure their activity and feeding behavior. Analysis of these complex sensor signals on the edge makes it possible to act in real-time and (e.g.,) ensure optimal feeding.

- *Reducing complexity:* Production processes often span multiple sites and diverse geographical locations. In the water and waste-water industry for example, dozens of diverse sensors (e.g., vibration sensors, mmWave sensors, pressure sensors) paint a picture of the overall process: AI can analyze and summarize these measurements on the edge (e.g., on the gateway) before sending more meaningful signals to the cloud.

[1] https://www.marketsandmarkets.com/Market-Reports/ai-in-iot-market-43388726.html.
[2] https://towardsdatascience.com/silo-surface-maintenance-using-drones-15d6c3f2439b.

- *Repurposing of off-the-shelf components:* Small (e.g., ESP32) devices equipped with cameras can be purchased at very low costs. However, transferring the captured data to the cloud for analysis is costly and therefore infeasible. AI on the edge makes it possible to analyze the incoming images on the device and solely submit the relevant results. Using modular AI deployment, the exact same device can be used to (e.g.,) count the number of products on a shelf, recognize the number of people in a room, or recognize a license plate (ANPR).

- *Anomaly detection:* A set of (e.g.,) vibration, temperature and pressure sensors can jointly be used to detect anomalies in complex industrial processes.[3]. Trained AI models can provide early warning signals that allow for timely intervention.

Despite these intriguing examples, using AI in (I)IoT is challenging. Each step in the feedback loop from data collection, to analysis, to actions, requires effective tools and secure processes. The challenges for AI in (I)IoT are distinct from those encountered by large web-companies who were the first to benefit from AI: AI in (I)IoT cannot rely solely on the cloud. However, the deployment of AI on various edge devices—often with vastly different specifications—is time-consuming and error prone. In this paper we describe our approach to modular (or "containerized") deployment of AI in (I)IoT that makes efficient, consistent, and secure AI inferences in (I)IoT possible. We first unravel the AI development process to identify the unique challenges encountered in every step, and subsequently we motivate why AI at scale needs to be decentralized. Next, we describe the benefits of modular AI deployment: AI in (I)IoT will only be successful when choosing the right tools for the job.

Unraveling the AI development process.

Adding value using AI is challenging since *closing the feedback loop* requires multiple steps to be carried out successfully:

1. *Data collection:* Data is necessary for AI applications. The core behind recent AI applications is the idea that—given sufficient data—a computer can learn a set of complex rules (or hypotheses) that we humans are unable to program ourselves. However, for this process to work, a lot of data needs to be available.

2. *Annotation / labeling:* Often the "raw" data is not sufficient. While a computer can uncover the (extremely complex) rules to map an image of a car onto a string describing its license plate, such supervised learning applications are only possible if appropriate labels are available: we need both the images of the cars and the correct license plate to train an AI model.[4]

3. *AI model training:* When data and labels are available we can train models. The training of AI models such as deep neural networks effectively requires a computer to generate hypotheses that map the input (i.e., the image) to the output (i.e., the license plate) and evaluate the appropriateness of these hypotheses. Iteratively, new hypotheses are proposed and evaluated until a set of hypotheses with an acceptable performance[5] is uncovered. This process requires all the collected data, and many iterations: it requires large cloud computing devices and/or GPUs.

4. *AI inferences:* Once a model has been trained; its usage is often called *inference*. Given a single input (i.e., an image, a reading of a set

[3] https://docs.efpf.linksmart.eu/projects/data-analytics/.
[4] For unsupervised learning, labels are not directly necessary. Thus, the labeling step is not always required.
[5] Obviously acceptable performance should be measured on a test set, not on the training set: a sufficiently complex set of hypotheses will always make it possible to obtain good performance in a training set (overfitting).

of sensors, or an audio-clip) the learned hypotheses should map the input to the desired output (i.e., the license plate string, the probability of an anomaly, or the keyword occurring in the audio-clip). Preferably these inferences take place efficiently on the edge to reduce network latency and energy consumption.

5. *AI model maintenance:* Once an AI model is deployed and inferences can be obtained, it often needs maintenance: issues such as concept drift make it essential to update models as new data is being collected. It is important to be able to safely update models and monitor their performance across a wide range of edge devices.

While the importance of data is well-understood within (I)IoT, the *AI model training* and *AI inference* steps are often confused: these two steps have different requirements as detailed below.

AI Training is offline and batched

Training an AI model effectively entails a (guided) search through a huge space of possible hypotheses that map the desired input to the desired output. The process requires a large batch of data (i.e., a large number of examples) since potentially complex relationships cannot effectively be learned on small datasets. A small dataset all too often leads to the selection of a model (or set of hypotheses) that performs well on the specific dataset at hand but does not generalize to new data. To illustrate, image processing models have only become feasible in recent decades primarily due to the vast amounts of training data available. Next to data, the AI training process also requires computing power: generating the hypotheses is computationally intensive, as is evaluating a new set of generated hypotheses to judge its performance. This process of generation and evaluation needs to be carried out thousands of times[6] to find a reasonable set of hypotheses. Each iteration is computationally demanding and is often carried out offline: it is carried out using an infrastructure that is separated from the business process that is generating the actual data.

Because of the need for large datasets and a large number of iterations, training AI models requires heavy computing. Typically, GPUs are used and the process is parallelized over many cores in a cloud environment. Moving such training to the edge is unlikely to be beneficial. However, the result of all this training is a model whose memory footprint is much much smaller than the training data, and whose evaluation (i.e., the process of generating inference) is much less computationally demanding than the training of the model. Thus, training and inference are two distinct steps.[7]

It is interesting to note that although arriving at the end of step 3—i.e., having access to a trained AI model that performs well—is challenging, by the time one finalizes step 3 the results are easily transferable. Once a model has been trained and its performance is on par, the model can be used without ever looking back to the data used to train the model. Thus, trained models can easily be distributed and even traded: in fact, numerous repositories and marketplaces for trained AI models exists[8], and common standards that describe trained models such as ONNX[9] allow us to easily export trained models. Hence, while steps 1-3 are necessary to develop AI systems, it is not at all necessary that every user of AI in (I)IoT carries out these steps themselves.

[6] For complex models the number of iterations might be even larger.

[7] Reinforcement learning provides a notable exception to this general rule where individual datapoints (or very small batches of data) are used to continuously, online, update a model and make decisions based on the current state of the model (and the value of potential exploration). The majority of uses of AI however is batched and benefits from the offline training of models.

[8] e.g., https://www.ibm.com/products/imaging-ai-marketplace and https://aws.amazon.com/marketplace/solutions/machine-learning.

[9] https://onnx.ai.

AI inference is online, one example at a time

Generating inferences—or predictions[10]—using a trained AI model is a process that is very different from training the model. Inferences are generated based on a single (or a very small number) of input examples. E.g., we would like to known the license plate of the currently approaching car, but we don't need to retain all the images of all the cars we have ever observed. Hence, memory requirements are vastly different between training and inference. Furthermore, once the model is trained, we don't need to repeatedly generate new hypotheses and iteratively check their validity. We simply execute the specific hypothesis which was deemed best during training. Figure 1 illustrates the differences between model training and model inference.

Figure 1: The difference between AI training (left) and AI inference (right): training needs all the data, and many iterations to determine model fit and adjust hypotheses. Inference needs a single example and only the selected hypothesis.

Note that the selected hypothesis—the one resulting from the model training—is often encoded using a number of matrix operations: given a numerical input vector (or matrix/tensor), operate on the input and generate the desired output. Although complex, the number of matrix operations required is limited compared to the often terabytes of data needed to train the AI model. Extremely large deep neural networks—the largest in production today—weigh in at several gigabytes. However, commonly used models, such as effective digit and character recognition models, often do not surpass a memory footprint of several megabytes. Simple, yet effective, regression models can even be expressed in kilobytes or less. These small representations, combined with the appropriate embedded implementation, make AI inferences on edge devices realistic. When the differences between training and inference are appropriately exploited[11], even a small ESP32 device, or an old 486 processor, can execute complex inference tasks. Before the advent of AI theses tasks would have been impossible to create.

Note that after deployment of an AI model, model testing, maintenance, and version control are important features of any AI in (I)IoT platform. The Scailable AI deployment platform[12] ensures that AI models are automatically optimized for inference. Subsequently, deployed models can easily be tested, maintained, and updated.

AI at scale needs to be decentralized

It is clear that optimizing for AI inference brings computational speed and memory advances—these improvements cut the computational costs of generating AI inferences in the cloud by an order of magnitude. However, why would one choose to deploy AI on the edge, instead of in the cloud? The surface maintenance drone mentioned in the introduction of this paper provides a compelling example.

The surface maintenance drone effectively scans the surface of a harbor silo by taking pictures of its surface in small, 50×50 centimeter tiles. A single silo often takes thousands of tiles to scan in full. Each (compressed) image weighs in at about 2Mb. Thus, over a gigabyte of data would have to be send over the—often

[10] The term inference is used in the ML/AI literature as a synonym for predictions. Traditionally, in the statistics literature, the two terms are distinct in the sense that the term inference refers explicitly to making statements about a population quantity.

[11] Often, AI models can be further optimized for inference using *quantization*—reducing size up to a factor 4. Quantization s directly available in the Scailable platform and allows models to run efficiently on small (legacy) devices.

[12] www.scailable.net.

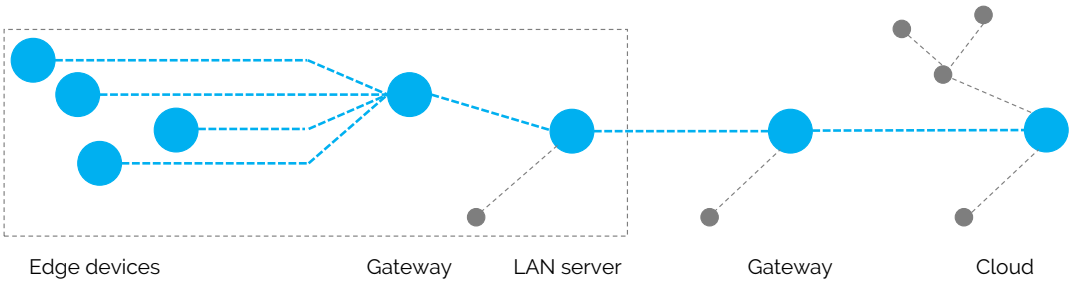

Figure 2: A subgraph depicting part of a typical IIoT infrastructure. It is preferable to move AI inference as much as possible to edge devices (left hand side of the image) as opposed to the central cloud (right hand side of the image).

unreliable—wireless network per silo if crack detection was to be carried out in the cloud. Both image compression and network traffic consume significant energy, limiting the operating time of the drone. Conversely, the actual DNN used to detect cracks can—after training and optimization—be encoded in a little over 10Mb; it needs to be send to the drone only once. Once deployed on the device, the drone can autonomously inspect multiple silos, only sending the locations of the detected cracks to a central server. The significant energy savings and increased reliability obtained by reducing the network traffic allow the drone to remain in flight longer. AI inference on the edge allows the same device to reliably inspect more silo's while reducing the network (and cloud computing) costs.

The advantages of decentralized AI are clear in many contexts: wherever large amounts of data need to be processed using AI inference, moving the processing as much as possible to the edge saves time and resources. Moreover, in many situations cloud processing is simply infeasible: networks might not be reliable, or the data concerned might be subject to security or privacy constraints that inhibit sending the data involved to a central server. To summarize, decentralized AI inference is beneficial over traditional, cloud-based, AI since a) it ensures minimal latency, and b) it results in a low energy consumption. Figure 2 schematically shows a typical IIoT infrastructure: Effective AI at scale will push AI inference as much as possible to distant nodes (i.e., those far away from the central cloud). The most distant node that has access to the necessary input data and has sufficient computational resources should carry out the desired inference.

AI inference in (I)IoT needs to be modular

Although the benefits of decentralized AI inference are clear, the design patterns and (software-) development processes necessary to deploy AI on the edge are only just emerging. Historically, two paths towards deployment of AI models have been common:

1. *Rebuild for each target:* Using this path, the deployment of AI for inference on the edge is done by effectively re-engineering a trained model using a language/platform suitable for the targeted edge device.

2. *Containerized deployment:* Using this path, the trained model—as it is created directly using a training platform/tool—is encapsulated in a container which effectively recreates the training environment (albeit without the training data). The container (which, when implemented in isolation on an edge device is akin a virtual machine[13] prevents having to rebuild the model for the targeted device and ensures modular deployment. However, containerized deployment intro-

[13] A container is often not the same as a virtual machine; multiple containers on the same machine often share their (virtual) OS making containers more efficient than traditional virtual machines (albeit with the added security risks).

duces significant overhead; such overhead is not acceptable for many edge devices.

Containerized deployment seems appealing: since we are dealing with a diverse set of devices, each with its own specifications, containers hold the promise of providing a single unified deployment The alternative of rebuilding a model that was trained using popular AI training tools such as `pyTorch`, `TensorFlow`, or `openbt` is comparatively time consuming and error prone. Rebuilding even a simple linear regression model for an edge device often takes at least a day of work by a skilled (embedded) software engineer. Weeks are however more common when proper testing is included. The rebuilding needs to be redone for every type of device, and for every new release of a model. Containers should be able to save us and provide a unified deployment process: this is exactly why containers have become so popular in an (AI) microservices infrastructure on cloud applications.

However, while current container solutions achieve transportability and a uniform and well-controlled deployment process in cloud environments, they fail on the edge. Popular container solutions, even when optimized, often consume close to 100Mb of memory; a scarce resource that is simply not available in many edge devices. Commonly used MCUs will simply not be able to facilitate the currently popular containers. Thus, while containers work well in theory, in practice they fail on the edge.

One alternative path towards AI deployment could be to ensure that model training is done in an environment that is similar to the deployment environment. Thus, rather than aligning the deployment process to fit AI model training, the training tools that are used are adapted to match the desired deployment target. Various companies are moving into this space. However, this path severely limits users in their choice of model types (and thus ultimately limits performance), and retains the dependence between training and inference that clearly, in an ideal world, does not exist at all.

AI deployment can be modular, transportable, and efficient

Interestingly, the advantages of the two paths to deployment described above can be combined whilst separating AI training form AI inference. It is possible to use *any* tool one would like during training (preventing lock-in to a specific platform), and still deploy to virtually *any* edge device without manually rebuilding the model (and while guaranteeing consitent model performance). WebAssembly[14]—a name poorly chosen as it is neither assembly nor is it dedicated to the Web—provides a portable compilation target that enables easy deployment of AI models on any device. The rebuilding to WebAssembly is automatic: it is possible to move automatically from common trained AI model formats (such as `ONNX`) to a WebAssembly binary using the toolchains offered in the Scailable platform. Akin to simply `pickle`-ing a model—a common way for containerized AI deployment—simply saving the trained model (including its pre- and post-processing pipeline) to `ONNX` is sufficient to enable automatic compilation to WebAssembly. After compilation a trained model can modularly be deployed inside a WebAssembly runtime installed on the targeted edge device.

WebAssembly runtimes can functionally be thought of as (extremely) small containers. However, WebAssembly runtimes are more akin to a virtual CPU as opposed to a full blown (Docker) Container: its memory footprint is only 64Kb—allowing its implementation on virtually any edge device—and its secure sandbox only allows a WebAssembly module to call functions allowed by the host environment thus adding important security guarantees.[15] De-

[14] https://webassembly.org

[15] WebAssembly runtimes can also utilize devices specific hardware acceleration and they can be deployed on GPUs if neccesary.

ployment of AI in (I)IoT using WebAssembly runtimes is quick, safe, portable, and modular. Deploying a new model to a device requires simply swapping the WebAssembly binary: This can be done directly, OtA, from the Scailable platform. Newly trained models can flexibly and safely be assigned to diverse edge devices with the click of a button (or using the Scailable CLI).

Providing the right tools for AI on the Edge

The ecosystem of AI training and inference tools has exploded in recent years. Many vendors aim to facilitate all the steps involved in training and deploying AI. Often these platforms aim to curate your data: in such cases usability comes at the costs of strong vendor lock in. These platforms offer only a small number of available models—a small subset of the models that are created by the extremely active AI/ML research and development community around the globe. Next, these platforms will offer deployment tools for only a small subset of devices. Thus, while these "do it all" platforms can appear appealing due to their initial ease-of-use, they limit innovation and often severely limit the number of devices that can be targeted.

A large (partly open) eco-system of model training tools exist and is directly available for use. These tools—such as TensorFlow, PyTorch, and Cognitive Services— can be used on top of virtually any database system; thus, steps 1-3 in the AI loop described earlier in this paper can be compartmentalized such that new AI training tools can quickly be incorporated to foster innovation. Furthermore, the emergence of open standards to encode trained models allows the effective modular separation of steps 3 and 4: virtually any model can be deployed on virtually any device as long as the right, open, tools are used. The Scailable platform—including its open WebAssembly runtimes available for virtually any edge device—allow for the safe, efficient and modular deployment of AI models for inference on any device large enough to support the trained hypothesis.[16] Furthermore, the Scailable platform allows for model maintenance, distribution, and testing: thus, the platform offers steps 4 & 5 described above, specifically for (I)IoT, and without dependence on the earlier steps or vendor lock-in.

AI on the edge creates value

The operational and technical necessity of pushing AI inference as much to the edge as possible in a modular fashion is well-understood. However, it is useful to emphasize the business value of having the appropriate tools to move AI inference to the edge. The business value can be decomposed into two parts:

1. *Costs savings:* Using modular WebAssembly deployment of AI inference on edge devices cuts development costs and operational cloud expenses.

2. *Accelerated innovation:* Having a seamless AI in (I)IoT deployment process accelerates innovation: AI models can be continuously developed, improved, and deployed. Thus, consumer value is added by (e.g.,): enabling better surface crack detection algorithms, better prediction of cattle behavior, improved context sensing, repurposing of "dumb" devices to smart devices, and better detection of anomalies in production processes.

Efficient AI on the edge cuts costs

Many organizations that operate in a challenging context (e.g., limited network connectivity or small edge devices) currently are forced

[16] Even reinforcement learning implementations on the edge are possible using WebAssembly, see https://towardsdatascience.com/efficient-reinforcement-learning-on-the-edge-331afa979a30.

prof. dr. Maurits Kaptein is the CEO of Scailable and a Professor of Data Science at the Tilburg University. Maurits' his research focusses on (statistical) machine learning and AI with special interest in the computational challenges that emerge when deploying AI models in practice. Maurits has authored over 50 papers appearing in international journals such as the *Journal of Machine Learning Research* and *the Journal of Statistical Software*. With the Scailable team, Maurits is making complex AI models available for inference on the edge. Contact Maurits at maurits.kaptein@scailable.net or +31 6 21262211.

to select a "rebuild for each target" path towards AI deployment on the edge. Putting aside the fact that this path scales poorly and is error prone, we find that the average costs involved are high. Even for relatively small suppliers (or end users) of smart IoT devices we find that it is not uncommon to have more than ten models in production, on at least 5 different types of devices. Due to concept drift—or otherwise changing environments—models are updated several times a year, often monthly. Using an estimate on the lower end of the spectrum—assuming that each rebuild takes approximately one day of work—makes that even moderate users of AI in (I)IoT spend at least $ 500.000 on AI deployment every year. Appropriate tooling and a modular deployment process can reduce these costs significantly.

Efficient AI on the edge adds value

While costs savings alone should motivate one to seriously consider building up an efficient and modular AI deployment pipeline, the actual benefits of having the appropriate tooling vastly outnumber the costs savings. By endowing edge devices with intelligence, new business propositions emerge and customer value can be created. For example, simply improving the performance of a camera from identifying whether a warehouse shelf is empty to providing the exact number of products on the shelf—irrespective of the type of product—brings direct value. Better anomaly detection prevents complex (and expensive) production lines from having to shut down. In such cases each minute of reduced downtime will add value. And, as AI models can be trained and improved continuously, proper tooling for AI deployment on the edge will increase innovation speed and provide a competitive advantage. While hard to quantify numerically without resorting to a specific context, the benefits of having a proper edge AI deployment and maintenance pipeline are orders of magnitude larger than just the cost savings involved.

Related resources:

- Numerous platforms for AI training exists: see https://towardsdatascience.com/top-artificial-intelligence-platforms-for-2020-80570c65c1b4 for a recent overview.
- `ONNX` is an open format built to represent trained ML/AI models and pre- and post-processing pipelines. ONNX defines a common set of operators—the building blocks of machine learning and deep learning models—and a common file format to enable AI developers to use models with a variety of frameworks. See https://onnx.ai.
- WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications. See https://webassembly.org.
- Scailable provides an easy-to-use, open, platform to transpile trained AI models and pipelines to WebAssembly and manage the modular deployment of these models to edge devices. Scailable models can be run safely in runtimes of 64Kb making deployment feasible on a large variety of devices. See. https://www.scailable.net.