Journal of Big Data

SHORT REPORT Open Access

Extending reference architecture of big data systems towards machine learning in edge computing environments

P. Pääkkönen* and D. Pakkala

*Correspondence:
pekka.paakkonen@vtt.fi
VTT Technical Research
Centre of Finland, Kaitoväylä
1, 90570 Oulu, Finland

Abstract

Background: Augmented reality, computer vision and other (e.g. network functions, Internet-of-Things (IoT)) use cases can be realised in edge computing environments with machine learning (ML) techniques. For realisation of the use cases, it has to be understood how data is collected, stored, processed, analysed, and visualised in big data systems. In order to provide services with low latency for end users, often utilisation of ML techniques has to be optimized. Also, software/service developers have to understand, how to develop and deploy ML models in edge computing environments. Therefore, architecture design of big data systems to edge computing environments may be challenging.

Findings: The contribution of this paper is reference architecture (RA) design of a big data system utilising ML techniques in edge computing environments. An earlier version of the RA has been extended based on 16 realised implementation architectures, which have been developed to edge/distributed computing environments. Also, deployment of architectural elements in different environments is described. Finally, a system view is provided of the software engineering aspects of ML model development and deployment.

Conclusions: The presented RA may facilitate concrete architecture design of use cases in edge computing environments. The value of RAs is reduction of development and maintenance costs of systems, reduction of risks, and facilitation of communication between different stakeholders.

Keywords: Neural networks, ArchiMate, Edge computing, DevOps, Inference, Machine learning, Reference architecture

Introduction

Many big data systems have been developed and realised to provide end user services (Netflix, Facebook, Twitter, LinkedIn etc.). Also, underlying architectures and technologies of the enabling systems have been published [1–3], and RAs have been designed and proposed [4–6]. Edge/5G computing is an emerging technological field [7], and the first products are being shipped to the markets. However, the utilisation of machine learning (ML) as part of the edge computing infrastructure is still an area for further research [8]. Particularly, it should be understood, how data is collected, and how models are developed, distributed, and deployed for decision making purposes in the edge computing infrastructure.

The goal of this paper is to provide a RA for facilitating the design of concrete edge computing architectures [9], which rely on the utilisation of ML techniques. Also, it has been studied how development and deployment of ML models may be related to actors (e.g. Data scientist, ML engineer) at the business layer. In general, the value of RAs has been reduction of development and maintenance costs of systems [10], facilitation of communication between important stakeholders [10], and reduction of risks [10]. Also, when a system is designed without a RA, an organisation may accumulate technical debt [11] and end up with a complex and non-optimal implementation architecture. The presented reference architecture (RA) has been created inductively based on 16 published implementation architectures developed for edge/distributed environments. The RA has been extended based on an earlier published RA for big data systems [4]. Particularly, model development and deployment has been focused on in edge computing environments. High level and deployment environment views of the RA may facilitate architecture design of new edge computing systems in the future.

The paper is structured as follows: First, related work regarding RA design of big data systems, SW engineering challenges in utilising ML/artificial intelligence (AI) techniques, and implementation architectures utilising ML in edge/distributed environments are reviewed. Then, research context and research method are described. Subsequently, RA design is presented with different architectural views. Finally, the RA is analysed and discussed, future work is presented, and the study is concluded. The Appendix provides detailed mapping of the implementation architectures to the RA, which illustrates how the RA was designed. Also, detailed views are provided regarding the association between the RA, and business actors focusing on development and deployment of the presented architectural elements.

Related work

First, the earlier approaches for RA design of big data systems are reviewed. Then, utilisation of ML/AI techniques in SW engineering is discussed. Finally, different architectural approaches are presented for realising ML utilisation in edge computing environments.

RAs are designed for facilitating design of concrete architectures, reducing of risks with proven components, and improvement of communication within an organisation [10]. Real drawbacks and benefits of RAs have been studied with a case study (company), which utilised RAs in multiple projects [10]. RA facilitated development of concrete SW architectures, and reduced maintenance costs. However, learning curve of RA adoption was seen as a drawback by the application builders. Stakeholders (e.g. architects, application builders) of RAs had different concerns regarding observed benefits and drawbacks. Sang et al. [12] present a RA based on implementation architectures from Facebook, Amazon, Twitter, and Netflix. The RA has four layers (Data collection, processing and loading, Data analysis and aggregation, Interface and visualisation, Job and model specification) into which processing and data stores are mapped to (e.g. LinkedIn [13]). SOLID [14] is an architecture for managing semantic big data in real time. It relies on RDF (Resource Description Framework) for storing, compressing and indexing of semantic data, and has a 3-tier architecture (content, merge, and service tiers). Lambda architecture [15] consists of a batch layer providing pre-computed views of the master data set, a serving layer providing access to batch views, and a speed layer providing a real time view to the latest data. Bolster [16] extends Lambda-architecture with a new semantic-layer, which adds variety, variability and veracity dimensions of big data to architecture modeling. User experiences from three industrial projects indicated that usefulness, and functional appropriateness/correctness were the most highly regarded attributes of the architecture, but there was hesitation regarding semantic exploitation of data [6]. Utilised a decision making framework in the evaluation of several presented RAs. Bolster [16] achieved the highest score, whereas SOLID [14], the RA by Pääkkönen and Pakkala [4], Lambda, and Kappa achieved good results. Also, RAs were systematically studied based on literature, and analysed in terms of architectural requirements, modules, layers, and patterns [5]. Additionally, RAs for big data systems have been developed by including security aspects [17], targeting plant genotyping and phenotyping systems [18], and government enterprise architecture frameworks [19].

Additionally, big data RA design has been addressed in on-going standardization activities. International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) JTC1/SC 42 committee is focusing on RA design [20]. Big Data Value Association (BDVA) has presented a reference model for big data [21]. The model has horizontal layers encompassing aspects of the data processing chain, and vertical layers addressing cross-cutting issues (e.g. cybersecurity and trust). Also, National Institute of Standards and Technology (NIST) Big Data Program is developing a big data RA [22]. The conceptual model is comprised of five functional components: data producer/consumer, system orchestrator, and big data application/framework provider. Data flows, algorithm/tool transfer, and service usage between the components can be described with different types of arrows. Activities and functional component views of the RA can be used for describing a big data system, where roles/subroles, activities, and functional components within the architecture are identified. Edge computing related RAs have been reviewed [23], which include RA developed by H2020 FAR-Edge-project, Intel-SAP RA, Edge Computing RA 2.0 by The Edge Computing Consortium, and Industrial Internet Consortium RA. The authors also presented a new Global Edge Computing Architecture, which is based on components of the reviewed RAs, and suggests encrypting all sensor-based data.

Development and deployment of ML-based models to the edge environment may create new challenges. DevOps is a process, which integrates SW development and delivery for delivering value faster and continuously to customers [24]. DevOps has been applied successfully, when building SW for traditional systems. However, when SW or service engineers build and operate services, which are based on the utilisation of ML/AI techniques, new challenges are met. For example, there may be a shortage of labelling data, quality of data (for modeling) may be low, and continuous improvement/training of models is needed. AIOps-term [25] has been suggested for empowering the SW/service engineers to address the challenges created by utilisation of ML/AI techniques. Additionally, platforms have been created for facilitating AIOps. ModelOps is a cloud-based framework/platform for lifecycle management of AI application artifacts [26]. ModelOps is comprised of pipelines, which generate, monitor, and continuously improve AI models. Pipelines are defined by users as Directed Acyclic Graphs (DAG), where nodes represent tasks, and edges control flows between them.

Computer vision, Natural Language Processing (NLP), network functions, Internet-of-Things (IoT), and virtual/augmented reality applications utilising ML techniques have been developed for the edge computing context [8]. In order to meet computational challenges of the applications, collected data could be moved from end user devices back to the cloud for processing and ML [8]. However, latency, scalability and privacy challenges may need to be addressed for achieving satisfactory performance [8]. In the following, several approaches are described, in which latency of model inference has been optimised for the edge environment.

One alternative is to offload model computation/inference [27–29] to the edge/cloud or partition neural network (NN) computation [29–31] to the edge/cloud. Offloading of NN inference to the edge achieved best performance in the context of image recognition in web browsers [27]. Also, when inferring the first NN layer (of the model) at the client, and offloading the rest of the NN-layers to the edge server achieved best performance (in terms of latency) in partial inference due to the large size of the first NN layer. An offloading system for deep neural network (DNN) computations has been implemented [28], where NN models were experimented in client/edge devices, and the produced statistics were utilised for creating a partitioning plan at the client device for model deployment. The edge server incrementally received and built NN partitions, which were received from the client. The client performed inference of local NN layers, and received results of remote inference from the edge server. Results indicated significantly lower query latencies, when compared to local or all-at-once uploading of NN layers. 5% accuracy improvement in object detection from images was observed, when NN inference was partially performed in the cloud and end devices [30]. A convolutional neural network (CNN) was partitioned and deployed for inference computation in a Kubernetes cluster [31]. The results indicated that parallelization of inference on multiple devices/Kubernetes Pods reduced significantly execution time (44–48%), when compared to local NN execution or pipelining approaches.

Sometimes models need to be designed or compressed for execution on resource-constrained devices [8]. For example, experiments with augmented reality enabling video processing framework [32] indicated that Raspberry Pi couldn't achieve a satisfying performance without optimisation, as only ~ 1 frame per second (FPS) could be processed. Smart execution of big Yolo [33] in an edge device and tiny Yolo in a mobile device led to improved performance in object recognition of video streams [34].

Integration of many optimisation approaches (DNN partitioning, pre-processing, offloading, distributed computing) [8] achieved high accuracy object detection with 60 FPS in edge environment [29]. The solution was to encode only interesting parts of the video [Region of Interest (RoI)], utilise parallel streaming and inference pipeling to reduce offloading latency, send only significantly changed frames (adaptive offloading) to the edge, and utilise a fast object tracking method in the mobile device.

Additionally, model training in a distributed/edge environment has been studied. Often, training performance has been improved by optimising the frequency and size of training updates [8]. AdaComp [35] compressed updates to parameter servers in distributed deep learning by selecting updates to be exchanged (largest gradients are selected), and by utilising staleness mitigation per exchanged parameter. Results in a test bed with Linux containers (LXC) indicated better accuracy and lower ingress traffic, when compared to competitive solutions. 3LC [36] is a lossy compression technique for distributed ML, which is based on 3-value quantization of state changes (with sparsity multiplication), encoding of 3-values with lossless transformation (into bytes), and another lossless encoding. 3LC reduced training time 16–23× while reaching similar test accuracy. Gaia [37] decoupled local communication within a data center from communication between data centers (Wide Area Network (WAN)). Only significant changes between parameters servers across WANs were transferred, and exchanges were limited to WAN bandwidth. Gaia reduced monetary cost of communication by 2.6–59× (when compared to the reference), and achieved ML speed, which was similar to LAN-environments.

Also, decentralised communication protocols have been utilised for optimising training performance [8]. For example, DLion [38] was a distributed network of workers, which utilised a probabilistic model for exchanging gradients between a subset of microclouds (instead of all-to-all), and data size in gradient/weight exchange was reduced by controlling the quality of exchanged data. A lossy compression technique for floating-point gradients was utilised in network FPGA (field-programmable gate array)-chips for compression [39]. The approach reduced gradient communication time (~ 70–80%), and speeded up training in distributed ML with little effect on accuracy. A distributed video surveillance system has been proposed, where vehicle classification/traffic flow prediction models were trained at edge nodes in parallel [40].

DeepCham [41] provided an alternative approach based on fusing of predictions produced by a domain-specific shallow model, and a domain constrained deep model. The domain-specific model was created by labelling training instances (suggested by a deep model) interactively with local end users. Object recognition accuracy was significantly better in comparison to a generic deep model.

In summary, different approaches have been utilised for RA design of big data systems in the literature [10, 16] and standardisation [21, 22]. Also, new SW processes [25] and platforms [26] have been suggested for facilitating utilisation of ML/AI techniques for the SW/service developers. The utilisation of ML/AI techniques in the edge environment has been studied by focusing mostly on either model training [29, 35–39, 41] or inference [27, 28, 30–32, 34, 40] point of view. However, RA design of big data systems utilising ML techniques in the edge environment has not been addressed to the authors' best knowledge, which is the contribution of this paper. Also, a system view is provided regarding SW engineering aspect of big data system realisation based on the proposed RA.

Research context and research method

Research context

Big Data systems, ML components and software (SW) systems in general are increasingly designed and developed for specific application needs within rapidly evolving and changing technology environment. The application needs emerge from new digital products, solutions, systems and services, that organizations design, develop and operate for internal or customer use. Without paying attention into architecture design during the design and development process, the resulting architectures and constructs can easily be under continuous change pressure, caused by changes and rapid evolution of the underlying implementation technologies of Big Data systems, ML frameworks and application SW frameworks (accumulation of technical debt [11]).

DevOps [24] based SW engineering, and its extensions for ML engineering (e.g. ModelOps [26]), target to continuous delivery of value in form of new SW intensive product or service features, and functionalities. If the value driven SW engineering processes are managed purely on feature and functional basis, without paying attention to the end-to-end system and SW architecture, the risk to accumulate technical debt and end up with extremely complex, non-optimal end-to-end system implementations increases. Applying a RA may help to identify the basic functional elements of the overall system, independently of implementation technologies. This can help to manage the complexity and evolution of the end-to-end system considerably. A RA can help to modularize the system architecture into functional elements, enabling feature and functionality driven design, development and deployment on element-to-element basis, instead of across the whole system architecture. In distributed computing environments including edge computing nodes, a RA may also be useful for considering the different deployment options for individual elements, in order to optimize the end-to-end system resources usage and performance of the digital product, system, solution or service under design and development.

Enabling consideration of different design, implementation technology and deployment options on element-to-element basis, instead of for the whole static end-to-end system design, is one of the main drivers for the RA design presented in this paper. This is especially relevant in distributed computing environments, where the computing continuum and deployment options vary from resource rich but high latency public clouds, to resource limited but low latency in-device computing at different user sites. In between there are additional domains for computing resources, such as private enterprise clouds/servers, in network computing nodes (e.g. Multi-access Edge Computing—MEC) and various on-site computing devices. Figure 1 below illustrates the computing continuum as target for deployment of end-to-end SW system with big data and ML components.

It is noteworthy that a computing continuum including edge computing nodes is relational to a physical location, which is the point of use for the end-to-end SW system. Deploying SW elements close to point of use enables minimizing latency of corresponding functionalities of the SW system. The concept of edge computing is loosely defined and used in the literature, and as a result refers to any computing extending the current public cloud computing towards the point-of-use. Accordingly the different types of edge computing nodes present in the computing continuum can include mix of Private cloud (e.g. Enterprise private clouds), MEC/Edge (e.g. [42]), On-site computing (e.g. in premises nodes with fixed installation) and In-device computing (e.g. mobile terminals or computing in wearable and IoT devices).

To make optimal implementation technology selections, and to optimize end-to-end system for the computing continuum, is not a straightforward task. Having an implementation technology independent RA with modularization of system functionalities may be of great help. A RA may facilitate the design of concrete architectures [9], reduce development and maintenance costs of systems [10], facilitate communication between

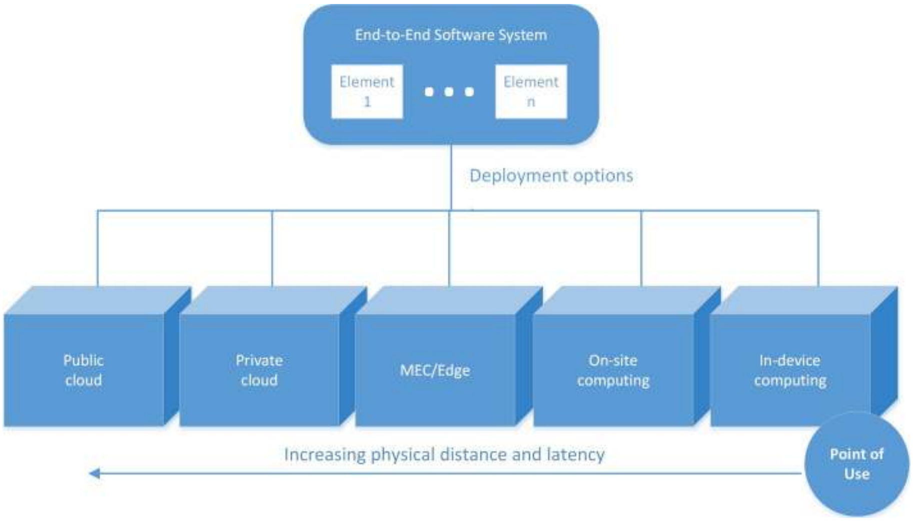


Fig. 1 Deployment options for an end-to-end SWsystem in an edge computing environment

important stakeholders [10], and reduce risks [10]. It may also greatly facilitate communication and shared understanding among the involved design and development process participants with different backgrounds (Product/service/solution business owners, Data scientists, SW engineers and ML engineers).

Research method

The goal of this work is to create a RA for facilitating the realisation of big data systems utilising ML techniques in edge computing environments. The following research questions (RQ) are posed:

- RQ1: Which elements comprise RA of a big data system enabling development and deployment of ML-based models in the edge computing environment?
- RQ2: Where should the main architectural elements of the RA be deployed?

In this work, the earlier big data RA [4] is extended as a differentiated replication study [43]. Particularly, the RA is differentiated by utilising 16 published architecture implementations as new source material in the study. In the following, the differentiated research method is described in detail (similar approach as in [4]):

An empirically-grounded architecture was designed [44] based on the following steps:

Step 1: Defining a type for the RA: A facilitation type of RA (type 3 in [9]) was chosen, which is designed by an independent organization for utilization by multiple organizations.

Step 2: Selection of a design strategy: A practise-driven strategy was used, because the RA was designed based on the realised implementations/prototypes of big data systems.

Step 3: Empirical acquisition of data: 16 published architecture implementations were used as source material [27–32, 34–41, 45, 46]. Only realized prototypes were considered in the selection process of publications.

Step 4: Construction of RA: The implementation architecture of each paper was mapped into the earlier big data RA [4]. Particularly, elements of the RA [4] were reused in the mapping where feasible. When mapping to the original elements was infeasible, new elements (functionality and data stores) were designed based on the architectural elements of the prototypes, and the original RA was extended accordingly. Thus, the RA was designed inductively based on the implementation architectures of the realised prototypes.

The detailed mapping process has been described in Fig. 2. First, implementation architectures of papers focusing on machine learning inference were mapped into functionalities, data flows, and data stores (similar notation used in [4]). Then, the identified elements were mapped into the high level architecture view of the RA [4] (see Figs. 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 and 20 for detailed mapping figures). Subsequently, the mapped implementation architectures of individual papers were merged into an updated RA, which included existing [4] and new elements (Fig. 22 in the Appendix). A similar procedure was performed for papers focusing on model training in edge/distributed environments (Fig. 23 in the Appendix). Then, the updated RAs were merged in order to create an updated overall RA, which included new elements from all the published papers.

Additionally, the elements of the updated big data RAs (Figs. 22, 23) were mapped into different deployment environments (Fig. 1). Public and private cloud, edge, on-site, and in-device were considered as deployment environments, which were targeted for execution of the architectural elements identified in the papers. Computing devices (e.g. servers, PCs, and laptops) closest to the data source were considered as edge devices. Computing devices at least one hop closer from the primary edge device towards the point of use, were considered as on-site devices. Computing devices containing an application or user interface for the end user (point of use of the SW system), were considered as in-device computing devices (e.g. wearables or mobile terminals).

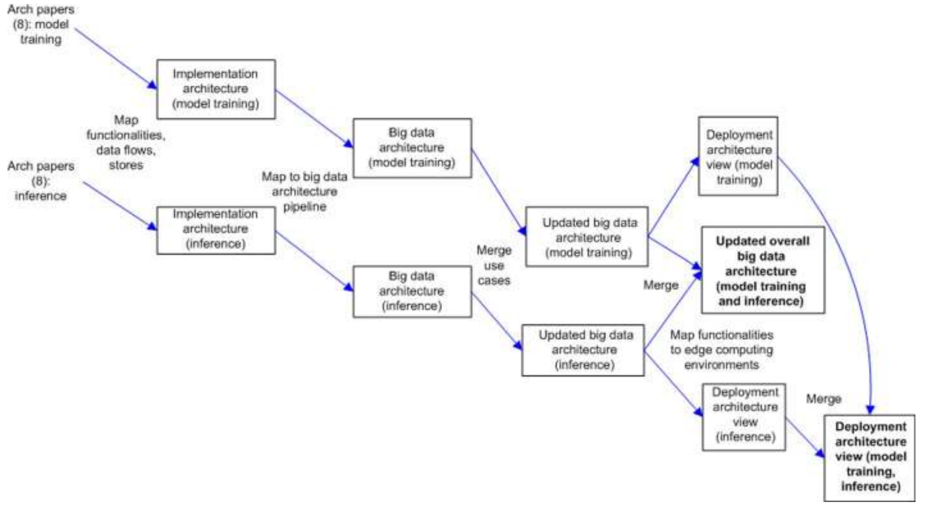

Arch papers (8): model training
Implementation architecture (model training)
Map functionalities, data flows, stores
Arch papers (8): inference
Implementation architecture (inference)
Big data architecture (model training)
Map to big data architecture pipeline
Big data architecture (inference)
Merge use cases
Updated big data architecture (model training)
Updated big data architecture (inference)
Deployment architecture view (model training)
Updated overall big data architecture (model training and inference)
Merge
Map functionalities to edge computing environments
Deployment architecture view (inference)
Merge
Deployment architecture view (model training, inference)

Fig. 2 Construction of RA based on realised implementation architectures

Finally, the deployment architecture views were merged for creating an overall deployment environment figure (Fig. 4) based on all the studied papers/prototypes. The deployment figure is a matrix between deployment environments, and functional areas of the big data RA.

Step 5: Enabling variability: Variability was not defined to the RA.
Step 6: Evaluation of the RA: Evaluation of the RA is left for future work.

RA design

Architectural views

Updated high level view of the RA has been presented in Fig. 3. The reader is referred to the initial version of the RA [4], where all original architectural elements are presented in detail. Functionality in the RA is described with rectangles, data stores are presented with ellipsis, and arrows indicate data flows. Similar functionality/data stores have been grouped into functional areas (FA). The original RA has been extended by including new functionality/data stores (depicted as bold in Fig. 3) in this paper. The name of the original 'Data analysis'-FA [4] has been modified into 'Model development and inference'. Functionality and data stores of the original 'Job and model specification'-FA [4] have been moved to the new FA. Also, the name of the original 'Data loading and transformation'-FA has been modified to 'Data transformation and serving'. Finally, a new FA called 'Model deployment' has been introduced.

The new functionality and data stores have been described in Table 1, while the reader is referred to the original paper [4] for unchanged architectural elements.

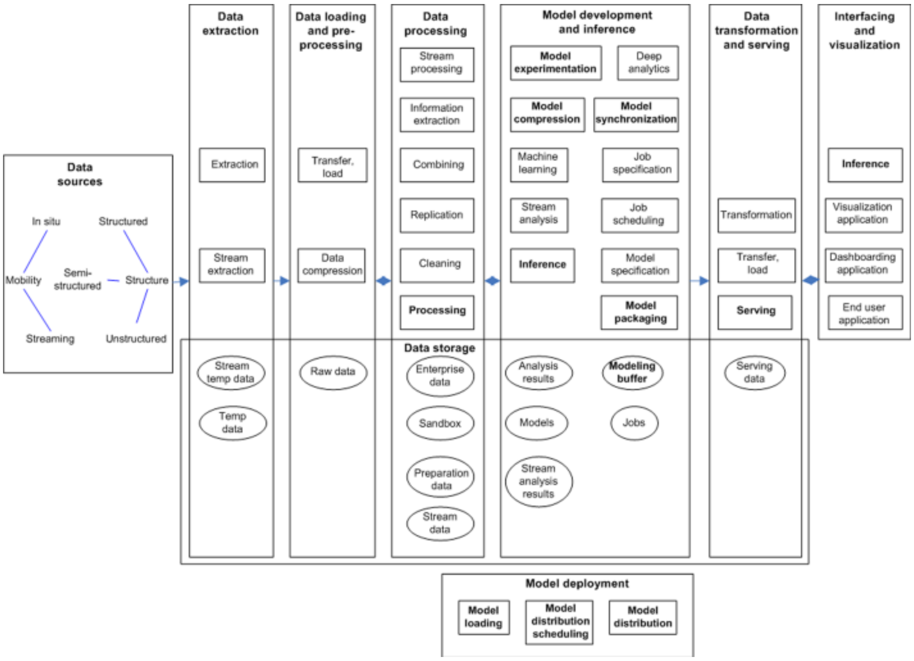
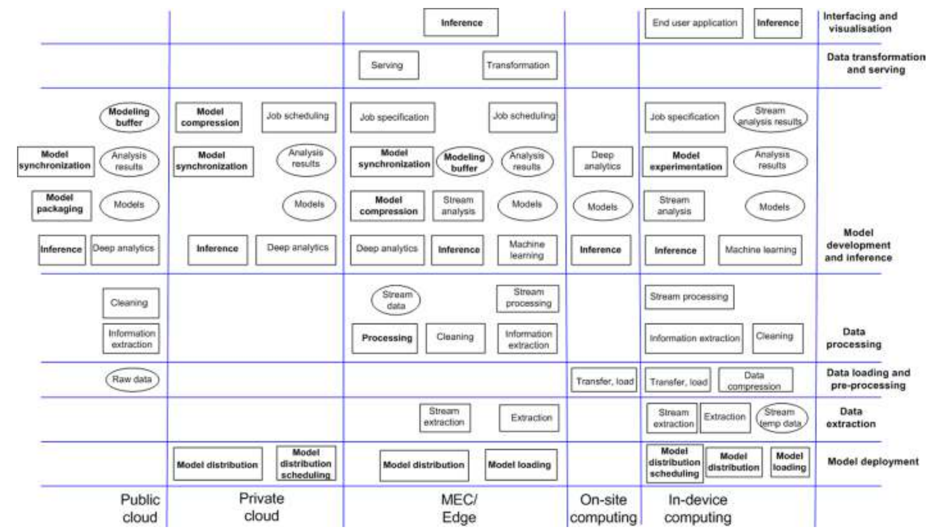


Fig. 3 Updated high level view of the big data RA (based on [4]). Data stores are presented as an ellipsis, functionality as a rectangle, and data flows with arrows. New architectural elements are described in bold

Table 1 Descriptions of new architectural elements of the RA

Architectural element	Description
Processing	Processing applied to in situ data
Inference	Making predictions by applying the trained model [53]
Model compression	Compression applied in model training [53] (e.g. compression of gradients [8] in neural network [53] training)
Modeling buffer	Short-term storage of modeling data
Model synchronization	Synchronization between modeling processes for updating of model's parameters [53]. For example, a parameter server [53] may be used for synchronization in a distributed network
Serving	Serving functionality for interfacing and visualisation (e.g. a server or a proxy)
Model experimentation	Execution of experiments with a model [53]
Model packaging	Packaging model(s) into executable/loadable file(s)
Model loading	Loading a model into memory for inference
Model distribution	Transfer and deployment of model(s) into node(s)
Model distribution scheduling	Scheduling of model distribution



	Public cloud	Private cloud	MEC/ Edge	On-site computing	In-device computing
Interfacing and visualisation			Inference		End user application Inference
Data transformation and serving			Serving Transformation		
Model development and inference	Modeling buffer Model synchronization Analysis results Model packaging Models Inference Deep analytics	Model compression Job scheduling Model synchronization Analysis results Models Inference Deep analytics	Job specification Job scheduling Model synchronization Modeling buffer Analysis results Model compression Stream analysis Models Deep analytics Inference Machine learning	Deep analytics Models Inference	Job specification Stream analysis results Model experimentation Analysis results Stream analysis Models Inference Machine learning
Data processing	Cleaning Information extraction		Stream data Stream processing Processing Cleaning Information extraction		Stream processing Information extraction Cleaning
Data loading and pre-processing	Raw data			Transfer, load	Transfer, load Data compression
Data extraction			Stream extraction Extraction		Stream extraction Extraction Stream temp data
Model deployment		Model distribution Model distribution scheduling	Model distribution Model loading		Model distribution scheduling Model distribution Model loading

Fig. 4 Deployment environment view of the big data RA. Edge deployment environments are presented on the horizontal axis, and FAs of the high level view of the big data RA are provided on the vertical axis

Figure 4 presents a deployment environment view to the big data RA. Elements of the implementation architectures (Figs. 22, 23) were identified and mapped into deployment environments. Most of the new architectural elements are located at the 'Model development and inference' and 'Model deployment' FAs.

Development and deployment view of the RA

For describing development and deployment of models within a big data system, a system view has been created with ArchiMate [47] (Fig. 5). First, the association of the elements in the big data RA to model development and deployment process was identified (Fig. 24). Then, the identified elements of the RA were added to ArchiMate's technology

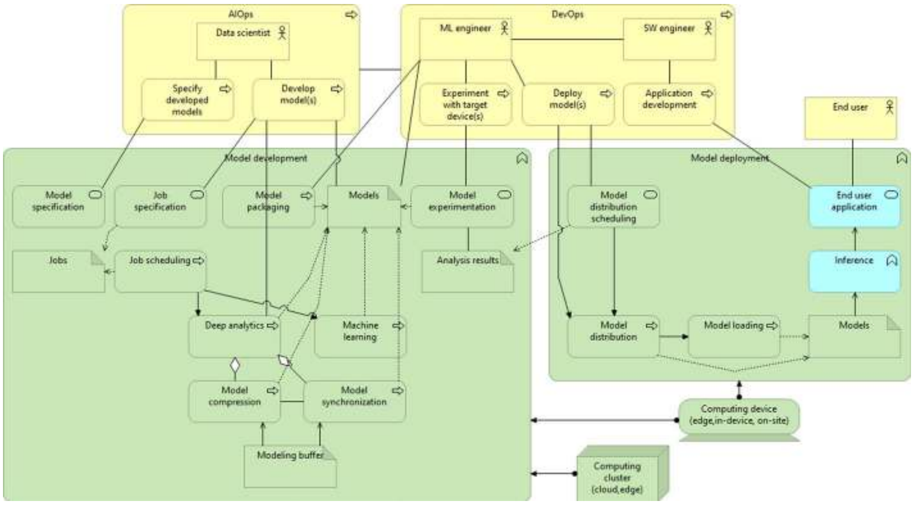


Fig. 5 Model development and deployment view to the big data RA

layer [47], and their relationships were modelled. Subsequently, actors and business processes related to the development and deployment of models were identified, and added to ArchiMate's business layer [47]. In the following, a possible workflow for development and deployment of models is described.

Usually a data scientist (as part of AIOps [25]) has the required competence for development of models. The models to be developed may be specified in the UI [4]. The data scientist may also specify batch processing jobs to be executed for development of the models. The jobs are typically scheduled to a computing cluster by triggering Deep analytics or other ML tasks. When DNN models are developed, ML between the nodes of a distributed system is often synchronized, and communication is compressed. When the model(s) have been developed, a ML engineer may need to experiment with the model(s) in the target device environment. The results of the experiments may be utilised, when scheduling distribution of the model(s). Particularly, it should be decided how to deploy the developed model(s) on the target infrastructure (edge, in-device, on-site). The developed model(s) may be packaged into executable/loadable file(s) for deployment. Finally, the model(s) are distributed and deployed on the target devices. Also, the model(s) are loaded into memory for providing inference to end user applications. When deploying model(s) for providing inference services to an end user application, a SW engineer may need to be involved in the deployment process with the ML engineer.

The Appendix provides an additional figure (Fig. 25), which illustrates how model development and deployment activities could be reflected in the deployment view of the big data RA.

Analysis

The original RA of big data systems [4] was modified for inclusion of new functionality and data stores (Table 1). Most of the new architectural elements were added to the 'Model development and inference'-FA. The purpose of the FA is to comprise all functionality, which is needed for development of ML models and inference. The purpose of the 'Model deployment'-FA is to comprise functionality needed for distributing and deploying developed model(s) to the target infrastructure. Also, new functionality was added to other FAs. Inference was included to the 'Interfacing and visualization'-FA based on a prototype, where a pre-trained model was used for inference in a web browser [27]. Model packaging comprises adding of model(s) into executable or loadable file(s) (e.g. adding models into Docker images [45]). Serving was added to 'Data transformation and serving'-FA based on an another prototype, where a web proxy served end users with augmented streaming video [32]. Processing was added to comprise any general processing (e.g. image resizing in [30]), which is applied to in situ data.

When the deployment view of the RA (Fig. 4) is analysed, it can be seen that architectural elements may be placed on almost all deployment environments (depending on the use case). A few reviewed prototypes included sensor-based devices, which were located between the primary edge-device, and the final point of use. In those cases [45, 46], architectural elements were identified in on-site computing devices.

When the location of architectural elements is analysed in detail, it can be seen that Data transformation and Serving functionality was executed in the edge server (transformation and serving of end users with augmented video streaming content [32]). Models were trained mainly in private/public clouds [36, 37, 39] or in edge devices [35, 46] while the results were inferred from models in all environments [27, 28, 30, 31, 45]. Also, resource demanding Deep analytics and Machine learning, and Job scheduling tasks were executed mostly in private/public clouds, and/or edge environments. Similarly, the associated DNN synchronization and packaging tasks (Model synchronization, Model compression, Modeling buffer, Model packaging) were executed in the same environment, where the DNN training (Deep analytics) was executed. An exception was clustering a subset of images on client devices (machine learning in-device) [41]. Streaming data was analysed and inferred in edge [29, 32, 34] and in-device [29, 34] environments. Models were experimented in in-device environments [28].

Data processing functionality was executed either in end user devices [29, 30], in edge servers [40, 41], or in the public cloud [45, 46]. Data loading and pre-processing was performed in end user devices [29, 34] or in on-site devices [45]. Data was extracted either from end-user devices [29], or from the edge environment [30, 40]. Deployment of models required functionality in different environments. Distribution of models was scheduled within the private cloud [31] or end user device [28]. Subsequently, models were distributed either from edge [27], from private cloud [31], or from end user devices [28]. Finally, models were loaded to memory in mobile devices [27] or in edge servers [28, 32].

Figure 5 provides a vision how development and deployment of models may affect AIOps [25] and application SW development processes at the business layer. Also, it provides a more detailed description regarding relationships of the architectural elements, which were presented in the earlier architectural views (Figs. 3, 4). Also, Figs. 24 and 25 illustrate model development and deployment related elements in the RA views.

In the following, the developed RA is compared to related work. First, this paper extended the earlier RA [4] based on published implementation architectures of 16 prototypes, which were developed for edge/distributed computing environments. The earlier RA [4] was designed based on 7 realised implementation architectures of big data systems (e.g. Facebook, LinkedIn, Twitter etc.). The high level view was extended with new architectural elements (Fig. 3). Also, a new architectural view was created (Fig. 4), which focused on deployment of the architectural elements to different edge-related environments. Finally, a vision was provided regarding the role of the actors at the business layer i.e. development and deployment of ML-based model(s) and applications (Fig. 5). Thus, this paper provides a significant extension to the earlier RA [4].

The RA presented in this paper can be compared to other published RAs, which have been developed for big data systems. The main difference to the earlier RA by Sang [12, 13] is fewer amount of processing layers considered (in [12]), and a bit more fine-grained notation utilised for describing architectural elements (in [12]). Sang included 'Job and Model specification' related elements to the RA on a high level quite similarly as we did [4]. Lambda architecture [15] focuses on separation of data handling with batch and speed layers, and serving of batch views in the serving layer. Thus, Lambda architecture provides a much more simplified approach to architecture design. Lambda's batch layer could be considered as a pipelined chain of processing in our RA (Extraction → Transfer/Load → Data processing → Deep analytics/Machine learning). Similarly, Lambda's speed layer could be considered as a similar processing chain in our RA (Stream extraction → Stream processing → Stream analysis). Our data 'Loading and transformation'-FA may correspond to Lambda's serving layer. SOLID [14] differentiates by focusing on semantic big data, and it is based on the Lambda architecture [15]. SOLID's online layer could be considered as extraction of streaming data into a big data system, while merge layer could correspond to processing of data, which is stored into an Enterprise/Raw data store (data layer in SOLID) (Fig. 3). SOLID's service tier could serve a similar role as 'Loading and transformation'-FA in our RA (Fig. 3). Bolster [16] extended Lambda-architecture with a semantic layer/repository, which may be depicted with an Enterprise store in our RA. It can be concluded that none of the published RAs [12–16] include detailed aspects of model development and deployment especially in the edge computing context. Also, we considered deployment environment view of the architecture, and a vision regarding the role of the actors at the business layer.

Regarding the standardisation work of RAs for big data systems, NIST [22] seems to have the most evolved architecture (ISO/IEC [20] is not open to the public for free). The processing phases of the big data application provider, data provider, and data consumer in NIST RA [22] may be considered as a subset of the high level view of the RA (Fig. 3) we have proposed in this paper. NIST RA has additionally separated processing, platforms, and infrastructures as part of the big data framework provider. We consider processing as part of the RA, and platforms/infrastructures as part of the deployment view of the architectures (deployment environments).

The edge computing RA by H2020 FAR-Edge project [23] differs from our approach by focusing on high-level placing of system components to different scopes (plant, enterprise ecosystem). Intel-SAP RA [23] is based on Intel and SAP platforms, and divides functional components into edge endpoint, edge gateway, and cloud. The RA may be used for understanding how to integrate edge-related SW projects into the technology platforms (provided by Intel and SAP), while we focus on providing a technology independent RA. Edge Computing RA 2.0 [23] is comprised of horizontal layers: Smart services, Service Fabric, Connectivity and Computing Fabric, and Edge Computing Node. The model-driven Smart Services-layer of the RA, which includes a model-driven service development framework, is related to our development and deployment view (Fig. 5). However, we have focused on DevOps [24] /AIOps [25] approach in the design of our system view. Industrial Internet Consortium RA [23] includes three tiers (edge, platform, enterprise). Decision support system or user interface of the enterprise layer generates control commands, which are analysed in the platform layer, and finally relayed to the edge layer. The horizontal layer structure of the RA may be considered as a simplification of the vertical structure in the high level view (Fig. 3) we provided. The Global Edge Computing Architecture [23] consists of IoT, Edge, and Business solution layers. IoT layer comprises collection and storing of data from IoT-devices based on blockchain-technology. The edge-layer is responsible for filtering and pre-processing of collected data, which is passed to the cloud-layer consisting of services and business applications. The horizontal layers of the RA could be considered as a simplification of our high level view of the RA consisting of vertical layers (Fig. 3).

In overall, the proposed RAs are different from ours. Industrial Internet Consortium RA and The Global Edge Computing Architecture may be considered as horizontally layered more simplified architectures (3 layers) of our high level architecture view (Fig. 3), which consists of vertical processing layers. Smart services-layer of the Edge Computing RA 2.0 is related to our system view (Fig. 5), which focuses on development and deployment of services based on DevOps [24]/AIOps [25]. Finally, none of the proposed RAs focus specifically on the utilisation of ML in the edge computing environments, which is the contribution of our work.

Finally, Chen and Ran [8] provided a comprehensive review of deep learning with edge computing. Particularly, literature was reviewed regarding implementation architectures for optimising inference and training in edge environment. However, the review didn't focus on reference architecture design, which is the contribution of this paper.

Discussion

First, only realised prototypes were considered, when publications were selected for the design of the RA. Publications focusing on simulations were not considered, because important implementation details regarding edge environments might be neglected in such studies. Additionally, we focused on ML and big data-based decision making on providing end user related services (e.g. image classification [27], augmented reality [29], object recognition [34], vehicle traffic flow prediction [40]), and didn't consider other problems (e.g. end-to-end connectivity and navigation between sensor nodes [48]), which might also be important within the context of edge computing environments. In the following, the lessons learnt from the design process are discussed.

The design process of the RA was not straightforward, because of the heterogeneity of the implementation architectures. Some of the reviewed prototypes contained only a subset of functionality in few computing domains. For example, most of the model training prototypes (e.g. [36–39]) focused on distributed training in edge computing devices. On the contrary, most of the inference-related prototypes (e.g. [27–29, 32, 34]) focused on improving inference performance in edge devices for providing services with end user devices. Only a few prototypes [41, 46] had realised a full pipeline of data extraction and processing, model training and inference for visualising results to the end user.

Implementation architectures of the reviewed publications were mapped to most of the elements of the original RA [4]. However, the RA had to be extended, and modified based on more detailed design regarding model development and deployment, which were not considered in the original design [4].

In the design of the RA, different processing stages were identified in model development and deployment. When DNNs are trained, synchronization is needed between distributed nodes [36–39] in order to reduce communication cost and training time. The communication between distributed nodes is usually optimized by utilising lossy/lossless compression techniques [35, 36, 39]. An intermediary data store [Modeling buffer (Fig. 3)] may be needed in the synchronization process. After training, the models are packaged [45] and scheduled for distribution. Particularly, it has to be determined how to partition DNNs to be executed on edge/mobile devices to achieve optimal performance in terms of latency. Prior to partitioning, models may need to be experimented on edge/mobile devices [28] in order to determine a plan for model distribution [27, 28, 31]. After the distributed models have been loaded [27, 28, 32] for inference at the target devices, data may need to be handled intelligently in order to optimize performance of inference [29].

In general, the deployment view of the RA may be considered as a guideline for concrete architecture design regarding the needed architectural elements, and deployment environments where the elements may be deployed. Also, the relationship between architectural elements and model development and deployment process has been described (Fig. 25). Finally, the system view (Fig. 5) illustrates the possible role of SW engineering actors in the development and deployment of different architectural elements.

Future work

In general, the value of RAs has been related to facilitation of concrete system design, reduction of development costs and risks [10], and facilitation of communication between important stakeholders within organisations [10]. In the future, the value of the presented views of the RA should be studied (in more detail) in the development of concrete edge computing systems, which utilise ML techniques in providing services to end users. Particularly, the provided system view (Fig. 5) indicating SW engineering relationship to the elements of the RA, should be evaluated in a real SW project, which focuses on model development and deployment within an edge environment.

The RA was designed based on implementation architectures where aspects of distributed model development, and deployment/inference were not integrated in a single prototype. Instead, the architectural elements of the implementation architectures of prototypes were integrated into a common RA. Thus, in the future the proposed RA should be evaluated with a prototype, which covers model development and deployment within an edge environment of a single prototype system (step 6 of the RA design process [44]).

Security, monitoring, and management layers are among the aspects [5], which are missing from the developed RA. Also, a lacking metadata layer has been seen as a drawback of the developed RA [6]. Previously a metadata layer has been developed for management of quality in social media data, and the original RA was extended with an overlay [49, 50]. Similarly, the RA could be extended with additional layers (e.g. monitoring) in the future.

Various use case scenarios have been designed for 5G networks, where ML/AI techniques could be utilised for optimizations [51, 52]. Selforganizing Networks (SONs) have been developed for reducing the cost of installation and management of 5G networks by enabling capability to configure, optimize and heal itself [51]. Resource elasticity has been designed by embedding AI-functionality to the 5G network (European Telecommunications Standards Institute's (ETSI) Experiential Network Intelligence (ENI) architecture) [52]. However, it has not been addressed how to manage ML/deep learning flows [8] within the 5G architecture.

Conclusion

The first research question was related to the elements comprising a RA of a big data system, which enables development of ML-based models in edge computing environment. The RA was developed inductively based on 16 published implementation architectures, which were focused on ML model development and inference. The previously published RA for big data systems was utilised as a starting point, which was extended in this work. In the design, new architectural elements were added to the high level view of the RA (Fig. 3). Thus, the presented RA and the elements within provide an answer to the research question.

The second research question focused on the deployment location of the architectural elements. Based on the high level view of the architecture, the architectural elements were mapped into deployment environments, which were utilised in the prototypes of the reviewed papers. As a result, a new deployment view of the architecture was provided, which illustrated the architectural elements as a matrix between the deployment environments and the functional areas of the high level view of the RA. The deployment view provides an answer to the second research question.

Additionally, the role of the actors in the business layer, who would participate to the development and deployment of ML-based models was identified with a system view.

Abbreviations

AI: Artificial intelligence; BDVA: Big Data Value Association; CNN: Convolutional neural network; DAG: Directed Acyclic Graph; DNN: Deep neural network; ENI: Experiential Network Intelligence; ETSI: European Telecommunications Standards Institute; FA: Functional area; FIFO: First in, first out; FPGA: Field-programmable gate array; FPS: Frames per second; IEC: International Electrotechnical Commission; ISO: International Organization for Standardization; LXC: Linux containers; ML: Machine learning; NIST: National Institute of Standards and Technology; NLP: Natural Language Processing; NN: Neural network; RA: Reference architecture; RDF: Resource Description Framework; RoI: Region of Interest; RQ: Research question; SW: Software; VM: Virtual machine; WAN: Wide area network.

Acknowledgements

The authors acknowledge researchers, who have created and published the source material utilised in this study. Juha-Pekka Soininen (VTT) is acknowledged for providing feedback to the development of the architectural views.

Authors' contributions

DP suggested the initial approach for the research (use a similar approach as in [4]). The detailed research plan (utilise a specific set of implementation architectures/publications to design a RA) was created by PP. PP studied implementation architectures of the publications, performed detailed design of the RA based on the source material, and produced the architectural views. DP participated to the RA design by suggesting the initial approach (different deployment environments) to the design of the deployment view (Fig. 4), and to study the relationship between model development roles, and the RA (in Fig. 5 designed by PP). PP wrote the article with the exception of 'Research context'-section in 'Research context and research method'-chapter (written by DP). DP provided feedback to the article. All authors read and approved the final manuscript.

Funding
Finnish Government provided funding for the research.

Availability of data and materials
Implementation architectures of the prototypes, which were utilised as source material in this study are presented in the respective publications.

Competing interests
The authors declare that they have no competing interests.

Appendix

Mapping of implementation architectures focusing on inference

Figure 6 describes mapping of computation offloading architecture in web application environment [27] to the big data RA. Codes, libraries, web browser, and NN-model are distributed to the edge server in a Virtual Machine (VM)-overlay (Model distribution), which is used for creating a running instance of the NN-model. Machine learning model is loaded (with CaffeJS) for inference in the client's and server's browsers. In the end user application images are partially classified (Inference) in the web browser. A snapshot including execution state of the web browser (including image to be analysed) is transmitted to the edge server, where full/partial inference is executed, and results are transmitted back to the web browser.

Figure 7 presents the mapping of distributed DNNs over cloud, edge, and end devices [30] to the big data RA. Initially, camera images are extracted, and resized (Processing). Then, convolutional and fully connected NN layers are utilised for inference of processed images at the edge server, and results are locally aggregated. Output from the

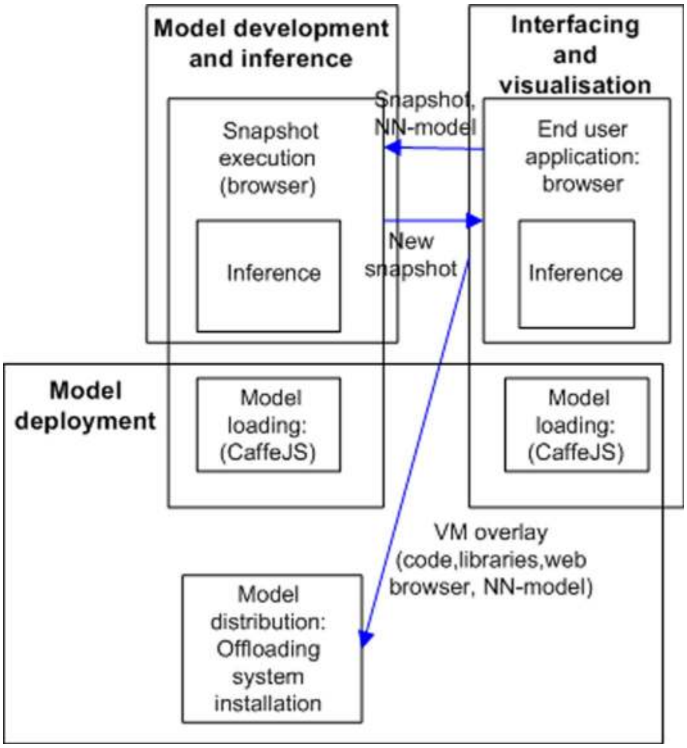

Model development and inference
Interfacing and visualisation
Snapshot, NN-model
Snapshot execution (browser)
End user application: browser
New snapshot
Inference
Inference
Model deployment
Model loading: (CaffeJS)
Model loading: (CaffeJS)
VM overlay (code,libraries,web browser, NN-model)
Model distribution: Offloading system installation

Fig. 6 The mapping of computation offloading architecture [27] to the big data RA

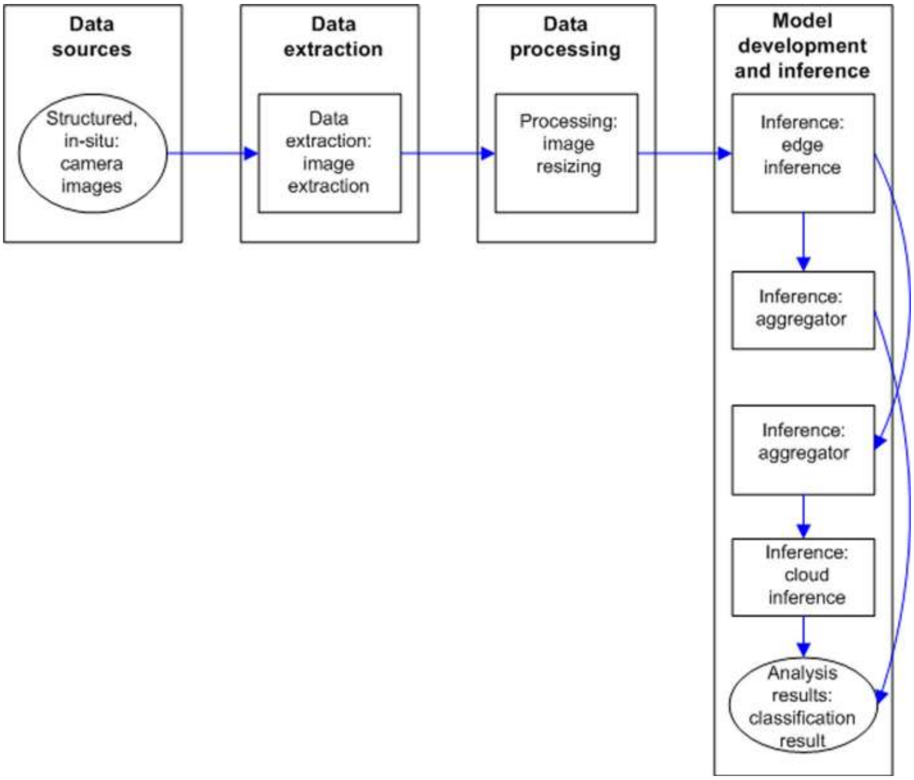

Data sources
Structured, in-situ: camera images
Data extraction
Data extraction: image extraction
Data processing
Processing: image resizing
Model development and inference
Inference: edge inference
Inference: aggregator
Inference: aggregator
Inference: cloud inference
Analysis results: classification result

Fig. 7 The mapping of distributed DNNs over cloud, edge, and end devices [30] to the big data RA

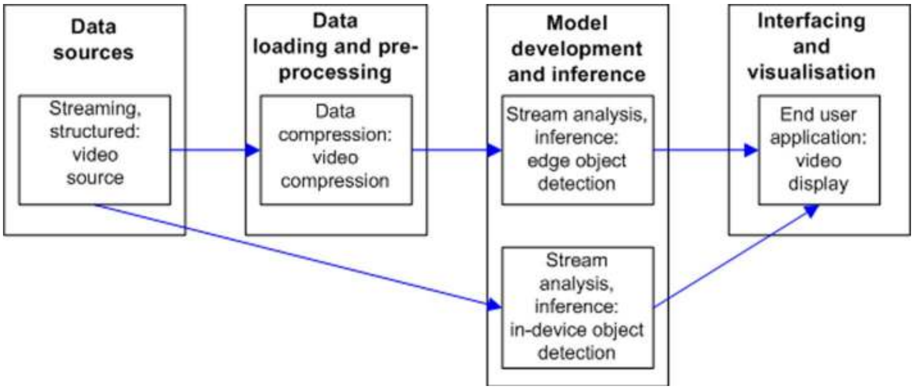

Data sources
Streaming, structured: video source
Data loading and pre-processing
Data compression: video compression
Model development and inference
Stream analysis, inference: edge object detection
Stream analysis, inference: in-device object detection
Interfacing and visualisation
End user application: video display

Fig. 8 The mapping of DeepDecision architecture [34] to the big data RA

convolutional network is transmitted to the cloud for aggregation, and further inference for providing the final classification result (Analysis results).

Figure 8 presents mapping of DeepDecision [34] architecture to the big data RA. Initially, video was compressed (Data compression) at the mobile device, and streamed to the edge server for object detection (Stream analysis/inference). Object detection was executed also at the mobile device.

Figure 9 presents the mapping of incremental offloading of NN computations from mobile devices to edge servers [28] to the big data RA. When a DNN model is

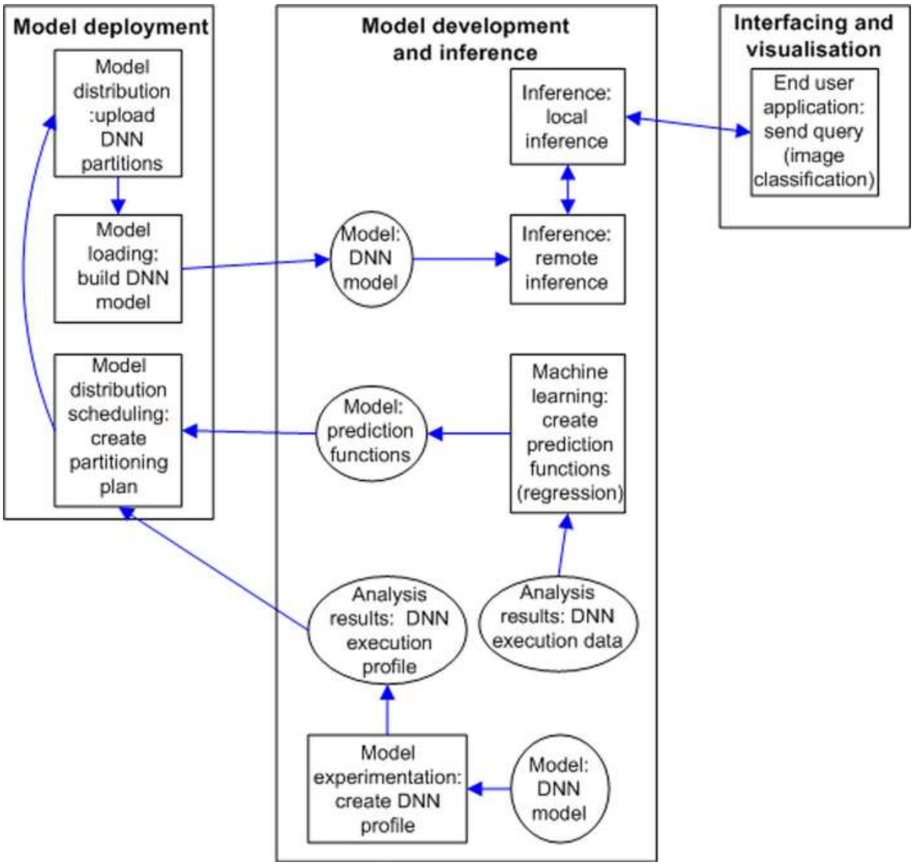

Model deployment
Model distribution :upload DNN partitions
Model loading: build DNN model
Model distribution scheduling: create partitioning plan
Model development and inference
Inference: local inference
Model: DNN model
Inference: remote inference
Model: prediction functions
Machine learning: create prediction functions (regression)
Analysis results: DNN execution profile
Analysis results: DNN execution data
Model experimentation: create DNN profile
Model: DNN model
Interfacing and visualisation
End user application: send query (image classification)

Fig. 9 The mapping of incremental offloading of NN computations [28] to the big data RA

experimented at a mobile client (Model experimentation), execution times of NN layers are recorded, and saved into a DNN profile (Analysis results: DNN execution profile). The edge server uses execution data of DNN layers (Analysis results: DNN execution data) in the creation of prediction functions (Machine learning). The prediction functions are transferred to the client for creating a partitioning plan (Model distribution scheduling). DNN partitions are uploaded to the edge server based on the plan (Model distribution), which are used for loading/building a DNN model (Model loading). The end user application sends queries for image classification, which are partly inferred at the client (Local inference). The client sends the result of local inference, and indices of DNN layers to the edge server, which are utilised for remote inference. Finally, the edge server provides the output results back to the client.

Figure 10 presents the mapping of distributed DNNs in containerized edge servers [31] to the big data RA. Edge device information and performance data (Analysis results) were used for partition point calculation (Model distribution scheduling) of models. Particularly, execution of partitions was solved as a multi-variate optimisation problem. Then, the partitions were parallelized for execution (Model distribution) in Kubernetes Pods, where inference was executed.

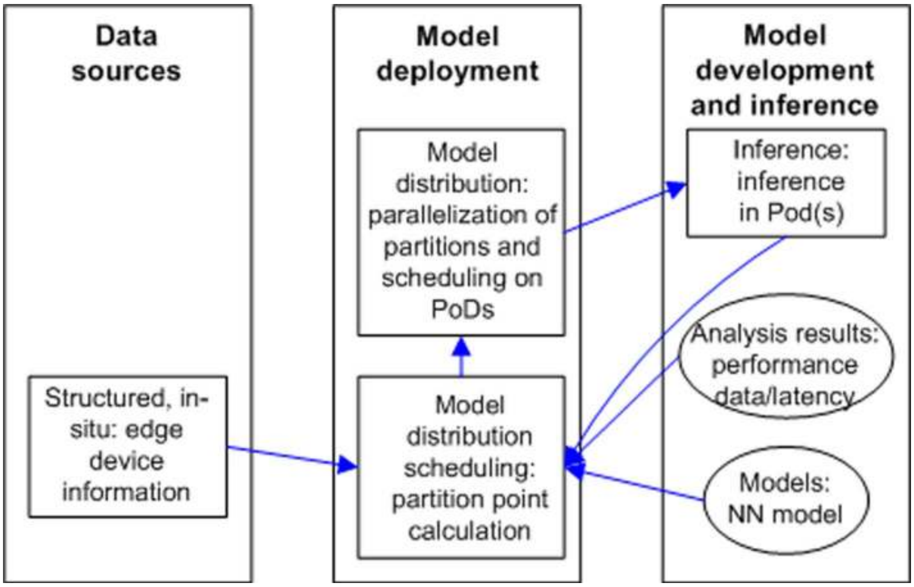

Data sources
Structured, in-situ: edge device information
Model deployment
Model distribution: parallelization of partitions and scheduling on PoDs
Model distribution scheduling: partition point calculation
Model development and inference
Inference: inference in Pod(s)
Analysis results: performance data/latency
Models: NN model

Fig. 10 The mapping of distributed DNNs in containerized edge servers [31] to the big data RA

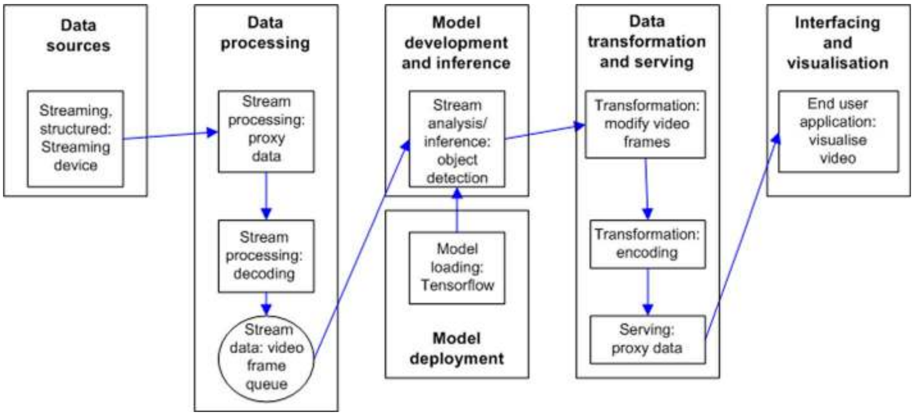

Data sources
Streaming, structured: Streaming device
Data processing
Stream processing: proxy data
Stream processing: decoding
Stream data: video frame queue
Model development and inference
Stream analysis/ inference: object detection
Model loading: Tensorflow
Model deployment
Data transformation and serving
Transformation: modify video frames
Transformation: encoding
Serving: proxy data
Interfacing and visualisation
End user application: visualise video

Fig. 11 The mapping of augmented reality acceleration at the edge [32] to the big data RA

Figure 11 presents the mapping of augmented reality acceleration at the edge [32] to the big data RA. Initially, video is streamed, proxied and decoded (Stream processing) at the edge server. Processed video is queued (Stream data), and objects are detected/inferred (Stream analysis/inference). A pre-trained Tensorflow-model is loaded and utilised for object detection. Subsequently, video frames are modified (with OpenCV) and encoded (Transformation), and the resulting stream is proxied (Serving) for visualisation at the end user application.

Figure 12 presents the mapping of "Edge Assisted Real-time Object Detection for Mobile Augmented Reality" [29] to the big data RA. First, frames were captured from a video stream (Stream extraction), and cached (Stream temp data) at the client. Then, RoIs were detected (Information extraction), macroblocks with overlapping RoIs were

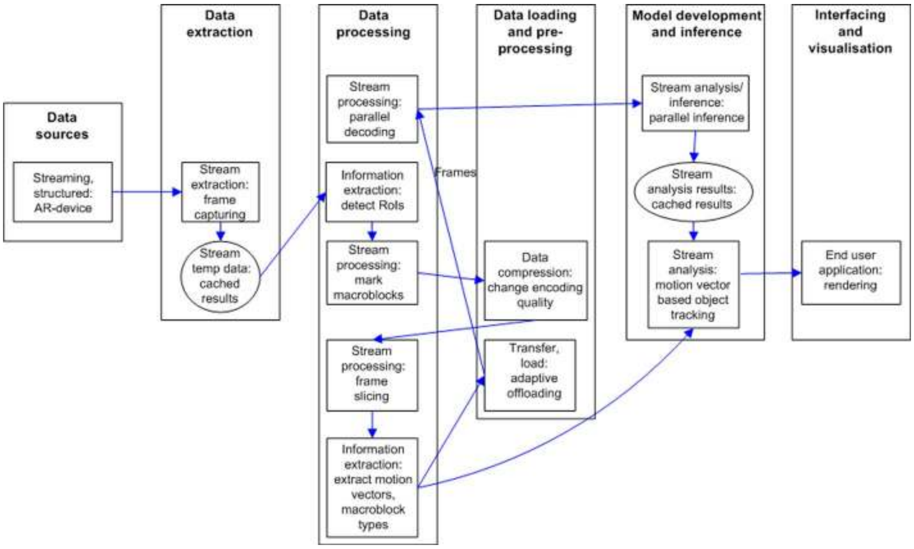

Data sources
Streaming, structured: AR-device
Data extraction
Stream extraction: frame capturing
Stream temp data: cached results
Data processing
Stream processing: parallel decoding
Information extraction: detect RoIs
Stream processing: mark macroblocks
Stream processing: frame slicing
Information extraction: extract motion vectors, macroblock types
Frames
Data loading and pre-processing
Data compression: change encoding quality
Transfer, load: adaptive offloading
Model development and inference
Stream analysis/ inference: parallel inference
Stream analysis results: cached results
Stream analysis: motion vector based object tracking
Interfacing and visualisation
End user application: rendering

Fig. 12 The mapping of "Edge Assisted Real-time Object Detection for Mobile Augmented Reality" [29] to the big data RA

identified (Stream processing), and lossy compression (Data compression) was applied for changing encoding quality of frames. The frames were sliced (Stream processing), and motion vectors and macroblock types were extracted (Information extraction). Then, the frames were selectively offloaded to the edge server (Transfer), where the frames were decoded (Stream processing). After parallel inference at the edge server (Stream analysis/inference), the results were sent back to the client and cached (Stream analysis results). Finally, the client performed motion vector based object tracking based on the results, and augmented video was rendered in the end user application.

Figure 13 presents the mapping of smart surveillance system on edge and cloud computing [45] to the big data RA.

First, faces were detected (Information extraction) based on an extracted (Stream extraction) video stream of an IP camera. The detection was made based on a database of encoded (Information extraction) photos of faces (Analysis results: Face DB). The detected faces were load balanced (Transfer, load), recognized (Inference), and the results were saved to a database (Analysis results) in the cloud. The recognized faces were saved to a face database (Analysis results). Face recognition (Inference) was performed again in the cloud. Inference models were packaged to Docker images, which were distributed to the nodes.

Mapping of implementation architectures focusing on model development

Figure 14 presents the mapping of DLion [38] implementation architecture to the big data RA. Image data set was used for training of models with distributed workers (Deep analytics). The workers synchronized gradients and weights in modelling via a messaging broker (Modeling buffer).

Figure 15 presents the mapping of Adacomp [35] to the big data RA. A DNN was trained based on an image data set (Deep learning). Model training was synchronized

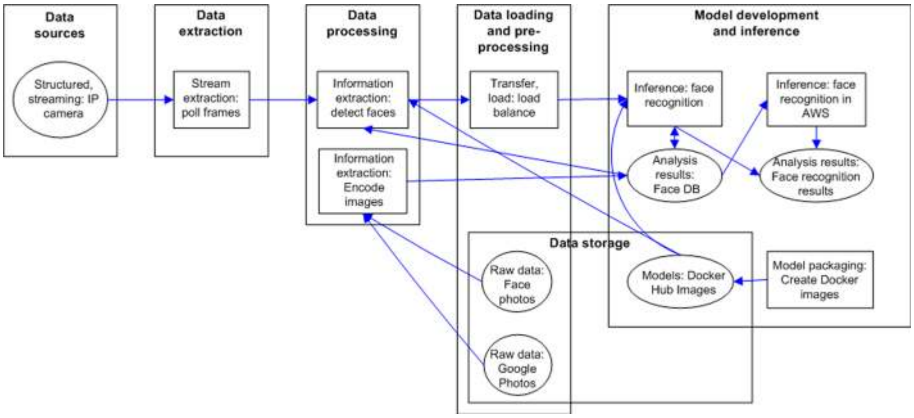

Data sources
Structured, streaming: IP camera
Data extraction
Stream extraction: poll frames
Data processing
Information extraction: detect faces
Information extraction: Encode images
Data loading and pre-processing
Transfer, load: load balance
Model development and inference
Inference: face recognition
Inference: face recognition in AWS
Analysis results: Face DB
Analysis results: Face recognition results
Data storage
Raw data: Face photos
Raw data: Google Photos
Models: Docker Hub Images
Model packaging: Create Docker images

Fig. 13 The mapping of smart surveillance system on edge and cloud computing [45] to the big data RA

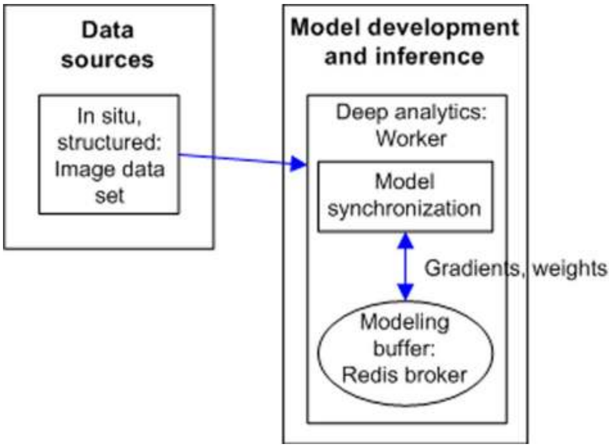

Data sources
In situ, structured: Image data set
Model development and inference
Deep analytics: Worker
Model synchronization
Gradients, weights
Modeling buffer: Redis broker

Fig. 14 The mapping of DLion [38] to the big data RA

via a parameter server, and learning updates were compressed (Model compression) for faster training.

Figure 16 presents the mapping of 3LC [36] implementation architecture to the big data RA. Image data was used for distributed training of DNNs (Deep analytics). In the distributed training steps, gradients were calculated (Model synchronization) and compressed (Model compression) at the worker. The parameter server decompressed the gradients (Model compression), which were aggregated (Model synchronization) to a global model. The parameter server calculated model deltas (Model synchronization), which were compressed (Model compression) and decompressed (Model compression) at the worker, and applied to the local model.

Figure 17 presents the mapping of network accelerated distributed DNN learning [39] to the big data RA. An image data set was used for model training by distributed workers (Deep analytics). Each worker maintained a local copy of the trained model. Gradients were compressed and decompressed (Model compression) in the FPGA-chip. A virtual

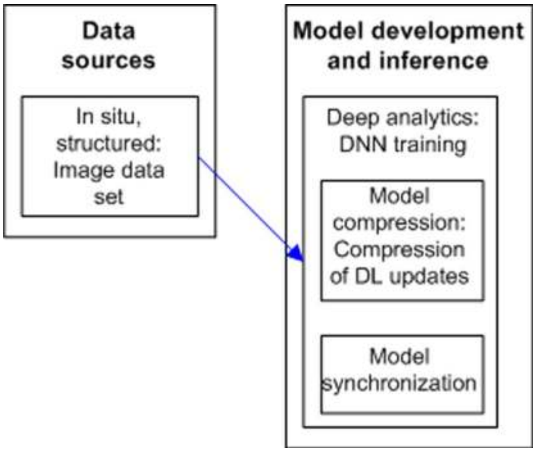

Data sources
In situ, structured: Image data set
Model development and inference
Deep analytics: DNN training
Model compression: Compression of DL updates
Model synchronization

Fig. 15 The mapping of Adacomp [35] to the big data RA

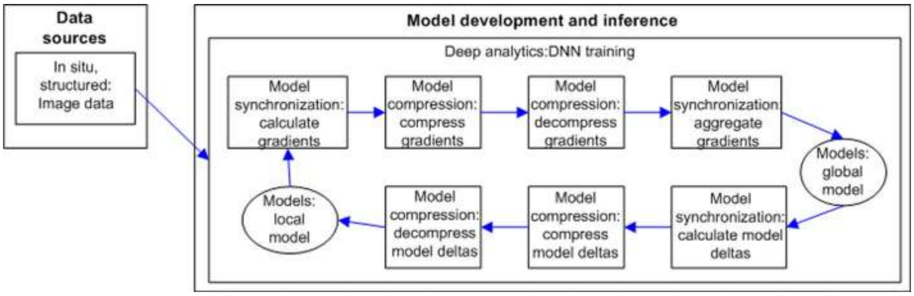

Data sources
In situ, structured: Image data
Model development and inference
Deep analytics:DNN training
Model synchronization: calculate gradients
Model compression: compress gradients
Model compression: decompress gradients
Model synchronization: aggregate gradients
Models: global model
Model synchronization: calculate model deltas
Model compression: compress model deltas
Model compression: decompress model deltas
Models: local model

Fig. 16 The mapping of 3LC [36] to the big data RA

first in, first out (FIFO) (Modeling buffer) was used as an intermediate buffer between workers, which synchronized by exchanging gradients.

Figure 18 presents the mapping of Gaia [37] implementation architecture to the big data RA. Three different types of source data sets (Image data set for image classification, news data set (New York Times) were used for topic modeling, and Netflix matrix (for matrix factorization) was used in distributed deep learning (Deep analytics). The worker maintained a local copy of the trained model, and transferred modelling updates (Model synchronization: synchronize local updates) to the parameter server at the local data center. The parameter server maintained a global model. Global updates between parameter servers across a WAN in different data centers (Model synchronization: synchronize global updates) were exchanged via modelling buffers. A significance filter was used for controlling the amount of updates to be exchanged. Detailed operation of the global updating process is described in [37].

Figure 19 presents the mapping of distributed deep learning with video surveillance systems using edge servers [40] to the big data RA. Initially, traffic monitoring video was extracted from cameras (Extraction). The stream was divided into data frames (Stream processing), and data blocks were extracted (Information extraction). Then, model(s) were trained for vehicle classification and traffic flow prediction among distributed

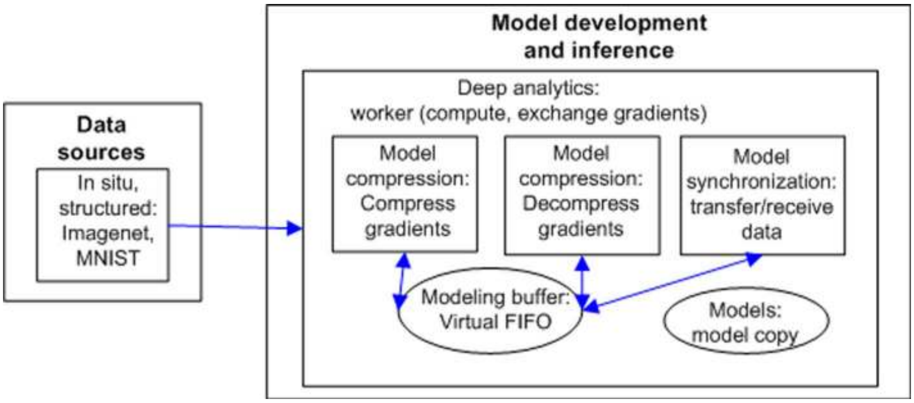

Data sources
In situ, structured: Imagenet, MNIST
Model development and inference
Deep analytics: worker (compute, exchange gradients)
Model compression: Compress gradients
Model compression: Decompress gradients
Model synchronization: transfer/receive data
Modeling buffer: Virtual FIFO
Models: model copy

Fig. 17 The mapping of network accelerated distributed DNN learning [39] to the big data RA

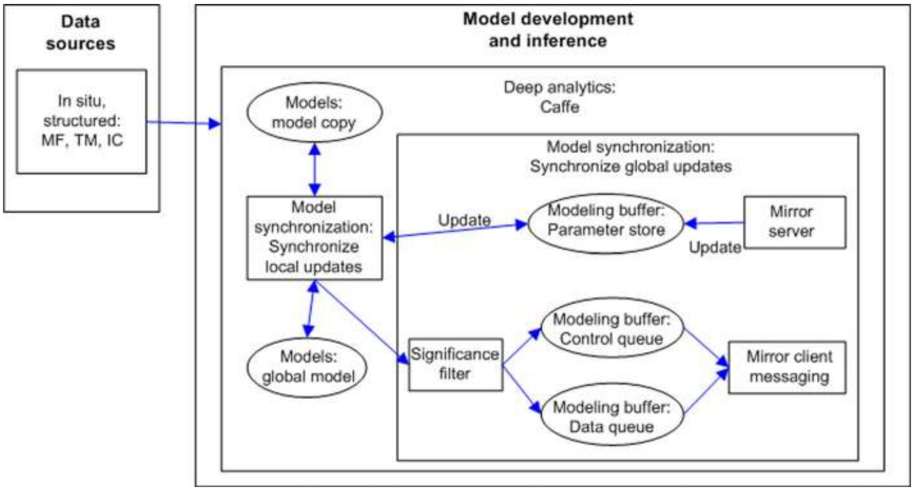

Data sources
In situ, structured: MF, TM, IC
Model development and inference
Deep analytics: Caffe
Models: model copy
Model synchronization: Synchronize local updates
Models: global model
Model synchronization: Synchronize global updates
Update
Modeling buffer: Parameter store
Update
Mirror server
Significance filter
Modeling buffer: Control queue
Modeling buffer: Data queue
Mirror client messaging

Fig. 18 The mapping of Gaia [37] to the big data RA

edge servers (Deep analytics). Task and model level parallel training was synchronized between edge nodes (Model synchronization).

Figure 20 presents the mapping of DeepCham [41] to the big data RA. First, a person specified an adaptation task for recognizing of objects (Job specification). The task was forwarded to the mobile device, where images from the device were scanned and pruned (Cleaning). Also, candidate images were clustered (Machine learning) for finding subsets of images with a large visual difference. The mobile clients sent image features to the master, where another clustering task (Machine learning) was executed, and the results were sent back to the mobile clients. At the mobile clients training object images were created (Information extraction), and labels/features were suggested based on a pre-trained deep learning model (Inference). Then, the end users labeled images with the mobile clients. The training instances were utilised for training a shallow adaptation model (Machine learning). The model was used for recognizing the objects (Inference), which was finally fused with the results (Inference: fuse results) produced by a deep model.

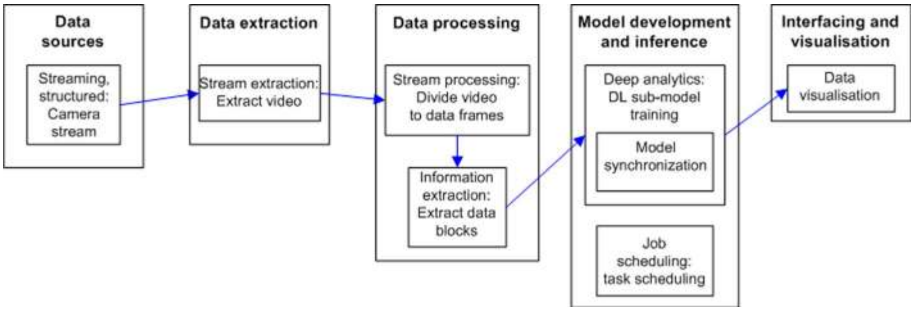

Data sources
Streaming, structured: Camera stream
Data extraction
Stream extraction: Extract video
Data processing
Stream processing: Divide video to data frames
Information extraction: Extract data blocks
Model development and inference
Deep analytics: DL sub-model training
Model synchronization
Job scheduling: task scheduling
Interfacing and visualisation
Data visualisation

Fig. 19 The mapping of distributed deep learning with video surveillance systems using edge servers [40] to big data RA

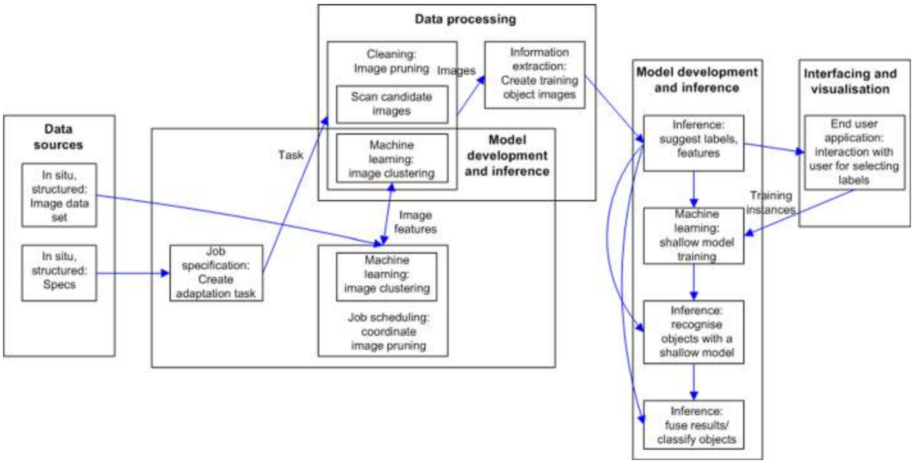

Data processing
Cleaning: Image pruning
Images
Scan candidate images
Information extraction: Create training object images
Machine learning: image clustering
Model development and inference
Task
Data sources
In situ, structured: Image data set
In situ, structured: Specs
Job specification: Create adaptation task
Image features
Machine learning: image clustering
Job scheduling: coordinate image pruning
Model development and inference
Inference: suggest labels, features
Machine learning: shallow model training
Inference: recognise objects with a shallow model
Inference: fuse results/ classify objects
Training instances
Interfacing and visualisation
End user application: interaction with user for selecting labels

Fig. 20 The mapping of DeepCham [41] to the big data RA

Figure 21 presents the mapping of HealthFog system [46] to the big data RA. Health-related sensor data was extracted (Extraction) from patients. The managed data (Processing) was delivered via a gateway device (mobile phone) and broker (laptop) to edge devices (RPi) or to an instance in the cloud (AWS). The data was preprocessed (Cleaning), and trained (Deep analytics). The trained model predicts (Inference) heart disease of patients. The gateway included an application, which was used for submitting prediction jobs (Job specification) to the system. An edge-device (a laptop) scheduled (Job scheduling) the received jobs to edge devices or to the cloud (AWS).

Integrated views of use case mapping

Figure 22 presents an integrated view, which includes mapping of all inference-related implementation architectures to the big data RA.

Figure 23 presents an integrated view, which includes mapping of all modeling-related implementation architectures to the big data RA.

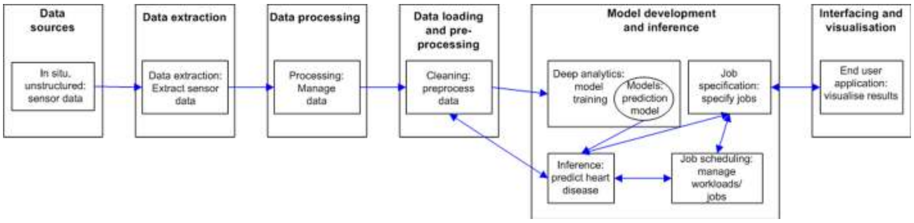

Data sources
In situ, unstructured: sensor data
Data extraction
Data extraction: Extract sensor data
Data processing
Processing: Manage data
Data loading and pre-processing
Cleaning: preprocess data
Model development and inference
Deep analytics: model training
Models: prediction model
Job specification: specify jobs
Inference: predict heart disease
Job scheduling: manage workloads/ jobs
Interfacing and visualisation
End user application: visualise results

Fig. 21 The mapping of HealthFog [46] to the big data RA

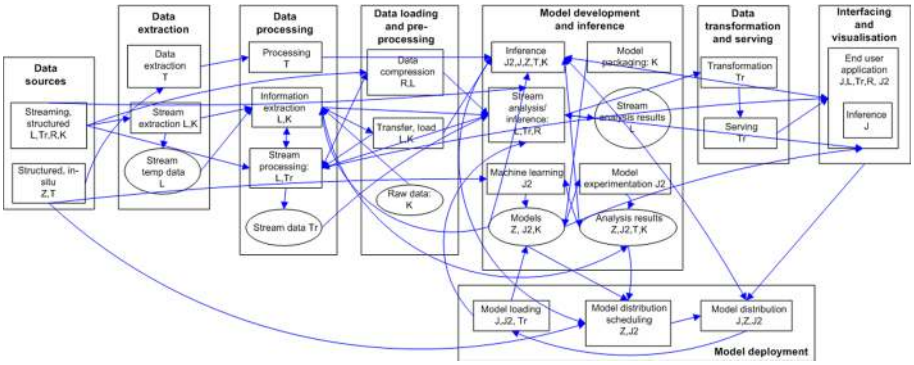


Fig. 22 Mapping of all inference-related implementation architectures to the big data RA. Publications are referred to from the architectural elements as follows: L: Liu [29]; Tr:Trinelli [32]; R:Ran [34]; Z:Zhou [31]; T:Teera [30]; J:Jeong [27]; J2:Jeong [28]; K:Kascavalci [45]

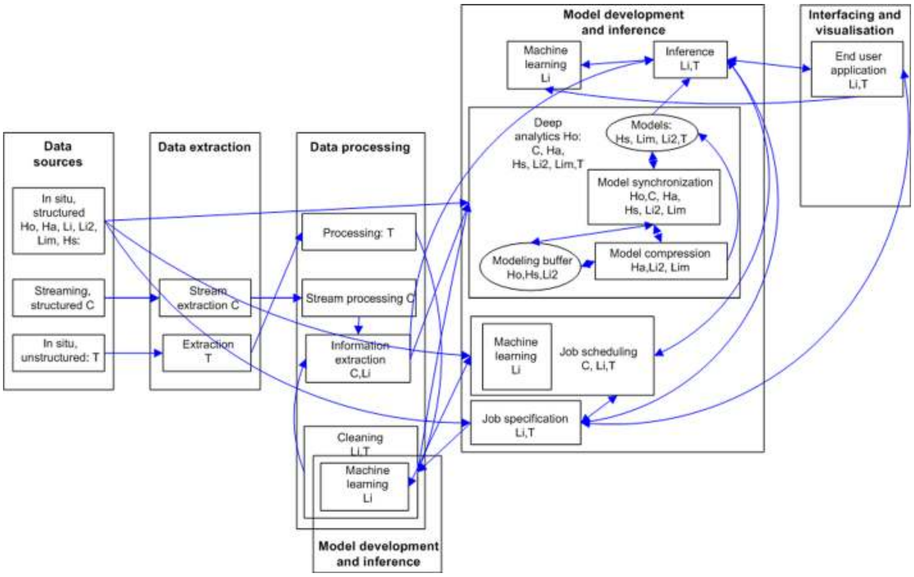

Data sources
In situ, structured Ho, Ha, Li, Li2, Lim, Hs:
Streaming, structured C
In situ, unstructured: T
Data extraction
Stream extraction C
Extraction T
Data processing
Processing: T
Stream processing C
Information extraction C,Li
Cleaning Li,T
Machine learning Li
Model development and inference
Model development and inference
Machine learning Li
Inference Li,T
Deep analytics Ho: C, Ha, Hs, Li2, Lim,T
Models: Hs, Lim, Li2,T
Model synchronization Ho,C, Ha, Hs, Li2, Lim
Modeling buffer Ho,Hs,Li2
Model compression Ha,Li2, Lim
Machine learning Li
Job scheduling C, Li,T
Job specification Li,T
Interfacing and visualisation
End user application Li,T

Fig. 23 Mapping of all modeling-related implementation architectures to the big data RA. Publications are referred to from the architectural elements as follows: Ho:Hong [38]; Ha:Hardy [35]; Li:Li [41]; Li2:Li2 [39]; Lim:Lim [36]; Hs:Hsieh [37]; C:Chen [40]; T:Tuli [46]

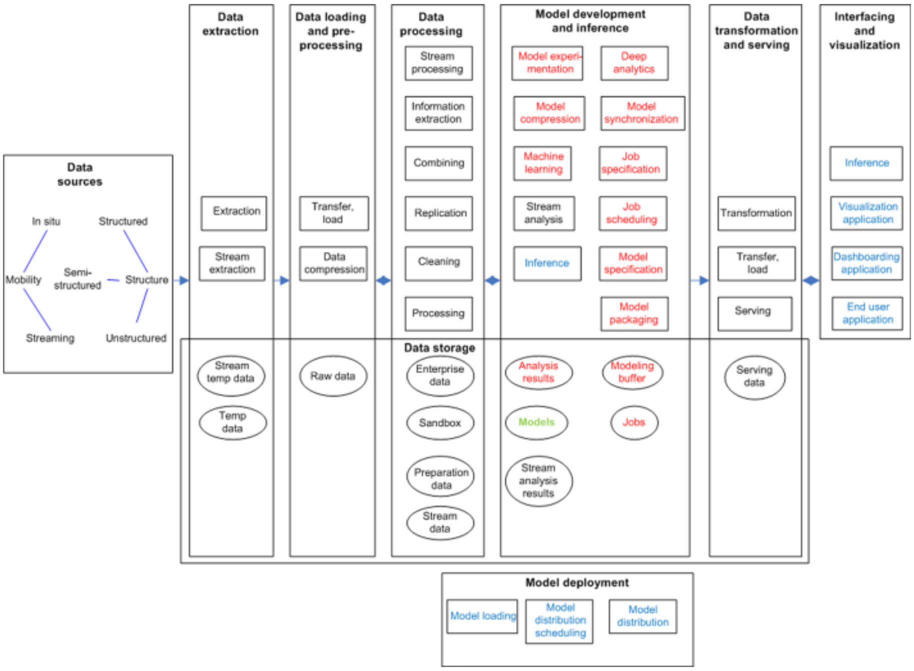


Fig. 24 Architectural big data RA elements associated with model development (red) and model deployment (blue). Green indicates association to model development and deployment

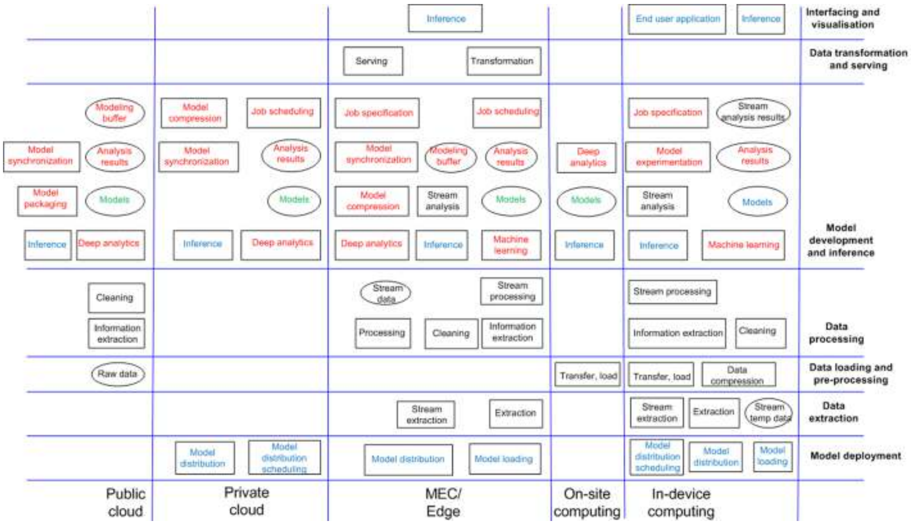


Fig. 25 Architectural elements of the deployment environment view of the big data RA associated with model development (red) and model deployment (blue). Green indicates association to model development and deployment

Model development and deployment: detailed views

Figure 24 describes the elements of the big data architecture, which are associated with model development or deployment (or both) activities.

Figure 25 describes the architectural elements of the deployment environment view of the big data RA, which are associated with model development or deployment (or both) activities.

Received: 27 November 2019 Accepted: 30 March 2020
Published online: 06 April 2020

References

1. Chen GJ et al. Realtime data processing at facebook. In: ACMSIGMOD international conference on management of data, San Francisco, CA, USA, 16 June–01 July, 2016.
2. Noghabi SA, et al. Samza: stateful scalable stream processing at LinkedIn. VLDB Endowment. 2017;12:1634–45.
3. Fu M et al. Twitter heron: towards extensible streaming engines. In: IEEE 33rd international conference on data engineering, San Diego, CA, USA, 19–22 April, 2017, p. 1167–72.
4. Pääkkönen P, Pakkala D. Reference architecture and classification of technologies, products and services for big data systems. Big Data Res. 2015;4:166–86.
5. Sena B, Allian AP, Nakagawa EY. Characterizing big data software architectures: a systematic mapping study. In: The 11th Brazilian symposium on software components, architectures and reuse (SBCARS), Fortaleza, Brazil, 18–19 September, 2017.
6. Volk M, Bosse S, Bischoff D, Turowski K. Decision-support for selecting big data reference architectures. Business Information Systems (BIS 2019), lecture notes in business information processing, vol. 353. Cham: Springer; 2019.
7. Li R, Zhifeng Z, Xuan Z, Guoru D, Yan C, Zhongyao W, Honggang Z. Intelligent 5G: When Cellular Networks Meet Artificial Intelligence. IEEE Wirel Commun. 2017;24:175–83.
8. Chen J, Ran X. Deep learning with edge computing: a review. Proc IEEE. 2019;107:1655–74.
9. Angelov A, Grefen P, Greefhorst D. A framework for analysis and design of software RAs. Inf Softw Technol. 2012;54:417–31.
10. Martínez-Fernández S, Ayala CP, Franch X, Marques HM. Benefits and drawbacks of software reference architectures: a case study. Inform Softw Tech. 2017;88:37–52.
11. Martini A, Besker T, Bosch J. Technical Debt tracking: current state of practice A survey and multiple case study in 15 large organizations. Sci Comput Program. 2018;163:42–61.
12. Sang GM, Xu L, de Vrieze P. A reference architecture for big data systems. In: The 10th international conference on software, knowledge, information management & applications (SKIMA), Chengdu, China, 15–17 December, 2016, p. 370–5.
13. Sang GM, Xu L, de Vrieze P. Simplifying big data analytics systems with a reference architecture. In: The 18th IFIP WG 5.5 working conference on virtual enterprises, Vicenza, Italy, 18–20 September, 2017, p. 242–9.
14. Martínez-Prieto MA, Cuesta CE, Arias M, Fernández JD. The Solid architecture for real-time management of big semantic data. Future Gener Comp Sy. 2015;47:62–79.
15. Marz N, Warren J. Big data Principles and best practises of scalable real-time data systems. New York: Manning Publications Co.; 2015.
16. Nadal S, Herrero V, Romero O, Abelló A, Franch X, Vansummeren S, Valerio D. A software reference architecture for semantic-aware big data systems. Inform Software TecH. 2017;90:75–92.
17. Moreno J, Fernandez EB, Serrano MA, Fernandez M. Secure development of big data ecosystems. IEEE Access. 2019;7:96604–19.
18. Roy B, Mondal AK, Roy CK, Schneider KA, Wazed K. Towards a reference architecture for Cloud-based plant genotyping and phenotyping analysis frameworks. In: IEEE international conference on software architecture, Gothenburg, Sweden, 03–07 April, 2017, p. 41–50.
19. Lnenicka M, Komarkova J. Developing a government enterprise architecture framework to support the requirements of big and open linked data with the use of cloud computing. Int J Inform Manage. 2019;46:124–41.
20. ISO/IEC JTC1/SC 42 committee. 2020. https://www.iso.org/committee/6794475.html. Accessed 18 Feb 2020.
21. Big Data Value Association. BVA SRIA—European big data value strategic research and innovation agenda. 2017. http://bdva.eu/sites/default/files/BDVA_SRIA_v4_Ed1.1.pdf. Accessed 18 Feb 2020.
22. Boid D, Chang W. NIST Big Data Interoperability Framework: Volume 6, RA Version 2. NIST Big Data Program. 2018. https://bigdatawg.nist.gov/_uploadfiles/NIST.SP.1500-6r1.pdf. Accessed 18 Feb 2020.
23. Sittón-Candanedo I, Alonso RS, Corchado JM, Rodríguez-González S, Casado-Vara R. A review of edge computing reference architectures and a new global edge proposal. Fut Gener Comput Syst. 2019;99:278–94.
24. Wiedemann A, Forsgren N, Wiesche M, Gewald H, Krcmar H. Research for practice: the DevOps phenomenon. Commun ACM. 2019;62:44–9.
25. Dang Y, Lin Q, Huang P. AIOps: Real-World challenges and research innovations. In: IEEE/ACM 41st international conference on software engineering: companion proceedings (ICSE-Companion), Montreal, Canada; 2019.
26. Hummer, Muthusamy V, Rausch T, Dube P, El Maghraoui K, Murthi A, Oum P. ModelOps: Cloud-based lifecycle management for reliable and trusted AI. In: IEEE international conference on cloud engineering (IC2E), Prague, Czech Republic, 24–27 June, 2019.
27. Jeong H, Jeong I, Lee H, Moon S. Computation offloading for machine learning web apps in the edge server environment. In: IEEE 38th international conference on distributed computing systems, Vienna, Austria, 02–05 July, 2019, p. 1492–9.

28. Jeong H, Shin CH, Lee H, Moon S. IONN: Incremental offloading of neural network computations from mobile devices to edge servers. In: ACM symposium on cloud computing, Carlsbad, CA, USA, 11–13 October, 2018, p. 401–11.
29. Liu L, Li H, Gruteser M. Edge assisted real-time object detection for mobile augmented reality. In: The 25th annual international conference on mobile computing and networking, Los Cabos, Mexico, 21–25 October, 2019.
30. Teerapittayanon S, McDanel B, Kung HT. Distributed deep neural networks over the cloud, the edge and end devices. In: IEEE 37th international conference on distributed computing systems, Atlanta, GA, USA, 05–08 June, 2017, p. 328–339.
31. Zhou L, Wen H, Teodorescu R, Du DHC. Distributing deep neural networks with containerized partitions at the edge. In: 2nd usenix workshop on hot topics in edge computing, Renton, WA, USA, 09 July, 2019.
32. Trinelli M, Gallo M, Rifai M, Pianese F. Transparent AR processing acceleration at the edge. In: 2nd international workshop on edge systems, analytics and networking, Dresden, Germany, 25 March 2019, p. 30–35.
33. Redmon J, Farhadi A. YOLO9000: better, faster, stronger. In: The IEEE conference on computer vision and pattern recognition, Honolulu, HI, USA, 21–26 July, 2017, p. 7263–71.
34. Ran X, Chen H, Chu X, Liu Z, Chen J. DeepDecision: A Mobile deep learning framework for edge video analytics. In: IEEE conference on computer communications, Honolulu, HI, USA, 16–19 April, 2018, p. 1421–9.
35. Hardy C, Merrer EL, Sericola B. Distributed deep learning on edge-devices: feasibility via adaptive compression. In: IEEE 16th international symposium on network computing and applications (NCA), Cambridge, MA, USA, 30 October 30–01 November, 2017.
36. Lim H, Andersen DG, Kaminsky M. 3LC: Lightweight and effective traffic compression for distributed machine learning. In: SysML conference, Stanford, CA, USA, 31 March–02 April, 2019.
37. Hsieh K, Harlap A, Vijaykumar N, Konomis D, Ganger GR, Gibbons PB. Gaia: Geo-distributed machine learning approaching LAN speeds. In: The 14th USENIX symposium on networked systems design and implementation (NSDI'17), Boston, MA, USA, 27–29 March, 2017, p. 629–47.
38. Hong R, Chandra A. DLion: Decentralized distributed deep learning in micro-clouds. In: 11th USENIX workshop on hot topics in cloud computing, Renton, WA, USA, 08 July, 2019.
39. Li Y, Park J, Alian M, Yuan Y, Qu Z, Pan P, Wang R, Schwing AG, Esmaeilzadeh H, Kim NS. A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In: 51st annual IEEE/ACM international symposium on microarchitecture, Fukuoka, Japan, 20–24 October, 2018, p. 175–188.
40. Chen J, Li K, Deng Q, Li K, Yu PS. Distributed Deep Learning Model for Intelligent Video Surveillance Systems with Edge Computing. IEEE Ind Inform. 2019;19(23):5324.
41. Li D, Salonidis D, Desai NV, Chuah MC. DeepCham: collaborative edge-mediated adaptive deep learning for mobile object recognition. In: IEEE/ACM symposium on edge computing, Washington, DC, USA, 27–28 October, 2016, p. 64–76.
42. ETSI. Multi-access edge computing (MEC). 2020. https://www.etsi.org/technologies/multi-access-edge-computing. Accessed 18 Feb 2020.
43. Lindsay RM, Ehrenberg ASC. The design of replicated studies. Am Stat. 1993;47:217–28.
44. Galster M, Avgeriou P. Empirically-grounded reference architectures: a proposal. In: Joint ACM SIGSOFT conference on quality of software architectures and ACM SIGSOFT conference on quality of software architectures and ACM SIGSOFT symposium on architecting critical systems, Boulder, Colorado, USA, 20–24 June, 2011.
45. Kascavalci HC, Gören S. A deep learning based distributed smart surveillance architecture using edge and Cloud-Computing. In: International conference on deep learning and machine learning in emerging applications (Deep-ML). Istanbul, Turkey, 26–28 April, 2019.
46. Tuli S, et al. HealthFog: an ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments. Fut Gener Comput Syst. 2020;104:187–200.
47. The Open Group. ArchiMate 3.1 Specification. 2020. https://pubs.opengroup.org/architecture/archimate3-doc/. Accessed 18 Feb 2020.
48. Wei W, Song H, Li W, Shen P, Vasilakos A. Gradient-driven parking navigation using a continuous information potential field based on wireless sensor network. Inform Sci. 2017;408:100–14.
49. Immonen A, Pääkkönen P, Ovaska E. Evaluating the quality of social media data in big data architecture. IEEE Access. 2015;3:2028–43.
50. Pääkkönen P, Jokitulppo J. Quality management architecture for social media data. J Big Data. 2017;4:6.
51. Moysen J, Giupponi L. From 4G to 5G: self-organized network management meets machine learning. Comput Commun. 2018;129:248–68.
52. Gutierrez-Estevez DM, Gramaglia M, De Domenico A, Dandachi G, Khatibi S, Tsolkas D, Balan I, Garcia-Saavedra A, Elzur U, Wang Y. Artificial intelligence for elastic management and orchestration of 5G networks. IEEE Wirel Commun. 2019;26:134–41.
53. Google. Machine learning glossary. 2020. https://developers.google.com/machine-learning/glossary. Accessed 18 Feb 2020.

Publisher's Note