EDITOR: Ewa Deelman, deelman@isi.edu

Orchestrating Networked Machine Learning Applications Using Autosteer

Zhenyu Wen and Haozhen Hu, *Zhejiang University of Technology, Hangzhou, 310023, China*

Renyu Yang, *University of Leeds, LS2 9JT, Leeds, U.K.*

Bin Qian, Ringo W. H. Sham, and Rui Sun, *Newcastle University, NE1 7RU, Newcastle, U.K.*

Jie Xu, *University of Leeds, Leeds, LS2 9JT, U.K.*

Pankesh Patel, *University of South Carolina, Columbia, SC, 29208, USA*

Omer Rana, *Cardiff University, CF24 3AA, Cardiff, U.K.*

Schahram Dustdar, *TU Wien, 1040, Vienna, Austria*

Rajiv Ranjan, *Newcastle University, NE1 7RU, Newcastle, U.K.*

A platform for orchestrating networked machine learning (ML) applications over distributed environments is described. ML applications are transformed into automated pipelines that manage the whole application lifecycle and production-grade implementations are automatically constructed. We present* AUTOSTEER*, a software platform that can deploy ML applications on various hardware resources—interconnected using heterogeneous network resources—across cloud and edge devices. Device placement optimization and model adaptation are used as control actions to support application requirements and maximize the performance of ML model execution over heterogeneous computing resources. The performance of deployed applications is continually monitored at runtime to overcome performance degradation due to incorrect application parameter settings or model decay. Three real-world applications are used to demonstrate how AUTOSTEER can support application deployment and runtime performance guarantees.

Machine learning (ML) systems and applications are intrinsically nondeterministic and need to operate in an environment that is constantly evolving, and contains ever-changing data. Typically, a networked ML application consists of a variety of components for data collection, device control, model inference (e.g., speech recognition, object detection), which are deployed and managed at different locations, i.e., either on locally managed servers or remotely in cloud data centers or edge environments.

ML applications executing over a networked platform are arguably complex systems, which have to be continuously updated and maintained. ML applications need to be transformed into automated pipelines that manage the whole application lifecycle and build production-grade ML implementations. A pipeline workflow, typically in the form of a graph representing the component interconnections in an ML application, can comprise: data management, model learning (model selection, training, and hyperparameter selection), model testing, and validation and model deployment. Thereafter, runtime management is responsible for ensuring performance guarantee, i.e., end-to-end model performance optimization and model update,[1] so that the deployed ML applications can be dynamically modified to runtime environment.

Doing so manually is generally unrealistic and not scalable, particularly when thousands of ML applications

1089-7801 © 2022 IEEE
Digital Object Identifier 10.1109/MIC.2022.3180907
Date of current version 23 December 2022.

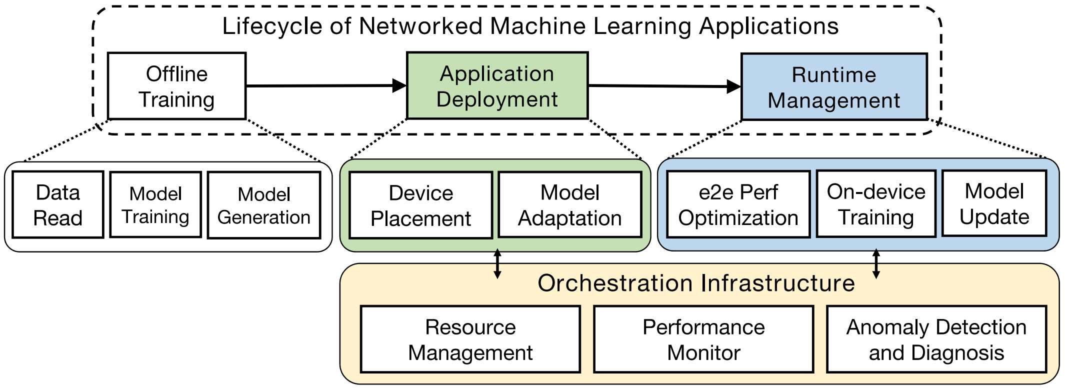

FIGURE 1. Conceptual workflow.

are submitted and maintained in edge and cloud platform that may be composed of hundreds of devices with heterogeneous hardware and software specifications. Continuous and automatic orchestration plays a pivotal role in deploying, managing, and synchronizing models of the ML applications across multiple tiers in a distributed computing environment. For instance, the trained models will be published and delivered to specific cloud servers or edge devices to run inference. Some specific applications, e.g., federated learning tasks require on-device training, indicating more complex device placement and model synchronization. Moreover, model decay arising from changes in data, would inevitably diminish model accuracy over time. Hence, an orchestrator calls for observation of the performance deviation and redeployment of the updated models.

Deploying such networked ML systems, particularly in an IoT and edge environment can be challenging due to the difficulty in managing the complexity of heterogeneous network and hardware resources. A variety of devices are used for data exchange, model training, and data analysis encompassing edge devices (such as IoT gateways and base stations) and servers (such as GPU, CPU, and TPU-based devices). Existing ML model development can be computationally expensive and resource intensive, which impede the effective deployment of applications, particularly those with strict latency requirements to resource-constrained devices.

In this article, we propose a platform solution to deployment and runtime management for the pipelines of networked ML applications. We devise *AUTOSTEER*, a management system that can automatically deploy networked ML applications over heterogeneous network and hardware resources while ensuring their performance through deployment plan optimization and model adaptation. At runtime, *AUTOSTEER* continually monitors the performance of deployed applications and automatically performs model update to mitigate performance degradation caused by obsolete application parameters setting or model decay. Finally, we use three real-world applications that are executed upon *AUTOSTEER* to showcase how the mechanisms are engaged in the application deployment and runtime maintenance.

MOTIVATION

Motivating Examples

We primarily categorize the networked ML applications into a) *centralized off-site* ML applications that can be trained offline or offsite, and b) *distributed on-site/federated* ML applications that must build their models using local dataset on individual device and, in some cases, share and aggregate models with other peer devices.

Centralized off-site learning applications: A smart home application allows users to observe the occupancy of their house, remotely control the smart devices (e.g., LEDs, air conditioner) via smartphone and even automatically control the smart devices. For example, a smart home application can automatically adjust the temperature of air conditioners based on the occupancy, weather, and so on.

Distributed on-site/federated learning applications: A high-quality brain tumor detection application relies on a huge amount of magnetic resonance imaging data that is only locally available and managed within a specific institution domain due to GDPR and other privacy regulations. A shared model is typically distributed to different data owners and trained locally. Locally trained models will be combined into a consensus model.

Research Scope and Overview

In general, the pipeline for such an application can be depicted as the workflow in Figure 1. The pipeline starts with and augments an initial model that has been trained offline along with a reference to metadata and the associated data sources on which the model has been trained. Thereafter, the workflow management platform typically addresses two fundamental problems: planning for device placement and model adaptation in the *deployment* phase and model execution performance guarantee in the *runtime* phase.

Determining the placement of ML components on available resources remains a key challenge—especially due to heterogeneity of resources. In addition, models have to be converted, for example through model pruning,[2] posttraining quantization,[3] and identifying a "focus" for the associated model through *distillation* techniques. This enables the generated models to best fit the target device, balancing the model size with accuracy of prediction. Significant recent efforts in this area include TinyML and EdgeML.

Once the plan of deployment comes into effect, runtime management ensures that the model performance can be monitored and overcomes model staleness. In the automated and continuous pipeline, triggers can be used to update application parameters or retrain the stale model with fresh data when performance observably degrades due to dynamic environment changes, such as network speed drop, workload bursting, model drift, or lack of generalization. For applications of federated learning and distributed training, the platform runtime also needs to enforce efficient on-device training.

A key focus of this work is to devise an orchestration system for supporting multiple ML model development and performance optimization. In addition, the system needs to scale to support both application size and resource heterogeneity. To underpin precise performance monitoring and anomaly detection while measuring platform health and resource utilization, we also need to track and inspect (distributed) system *fingerprints*—consisting of various performance indicators and application metrics, such as drift and prediction scores.

CHALLENGES

We elaborate on these specific challenges facing the ML workflow platform in the following notable aspects.

Complexity of device placement and model adaptation: Planning for a pipeline of a given ML application indicates a mapping procedure between awaiting models and available computing resources on the devices. To accommodate the specific demands of diverse distributed or federated learning applications, infrastructure resources have become increasingly heterogeneous, making the planning a far more intricate task.

1) *Device placement:* Successfully deploying sizeable components of the ML applications served in the platform requires stringent capacity check and optimization solution under numerous constraints. The manifestation of heterogeneity intrinsically stems from the static attributes of the hardware, such as CPU, GPU, memory, SSD, and network bandwidth, and of the software including operating system version, clock speed, and particularly software libraries. The compatibility of a given hardware or library version even becomes a hard constraint, for any violations of such requirements would completely fail the deployment. For example, some components are compiled for ARM Mali cannot be executed on Nvidia GPU. The network constraints, such as bandwidth sharing among colocated components or network latency specified by each individual component, will further exacerbate the planning complexity.

2) *Model adaptation:* The advancement of deep models, such as recurrent neural network and convolutional neural network leads to the substantially increased parameter number and the resultant computational cost, which hinders the real-world model deployment into embedded and edge devices. Hence, model pruning and compression can be used to reduce model size, remove redundant weights, such that pretrained models can better adapt to portable devices with limited resources (e.g., memory, CPU, power, and bandwidth) and be applied into real-time applications.

3) *Enabling dependent components within a pipeline:* Each individual ML model has its own specification and format of input and output data. Dependencies are referred to as the interactions, such as the data flows and remote callings, among interconnected components. This would be problematic and challenging particularly when components deployed on various devices are interconnected via different network types and protocols. Hence, it is imperative to design an effective data messaging system to orchestrate the data flow and manage the network traffic across different models while considering the particular specification and data format.

Optimized runtime management: Improper application parameter setting or model decay could result in poor performance of an ML application and even failures. The first task of runtime management is to perform end-to-end and intraapplication optimization. Application parameters (e.g., model accuracy, task offloading rate) need to be adjusted at runtime to ensure the allocated resource can guarantee the expected performance level. To do so, the orchestration system should be capable of automatically detecting any performance degradation of the deployed applications and then dynamically work out the optimal configuration to rescue the abnormal performance. Second, in the face of any model failures, the orchestration system should automatically perform local on-device training while

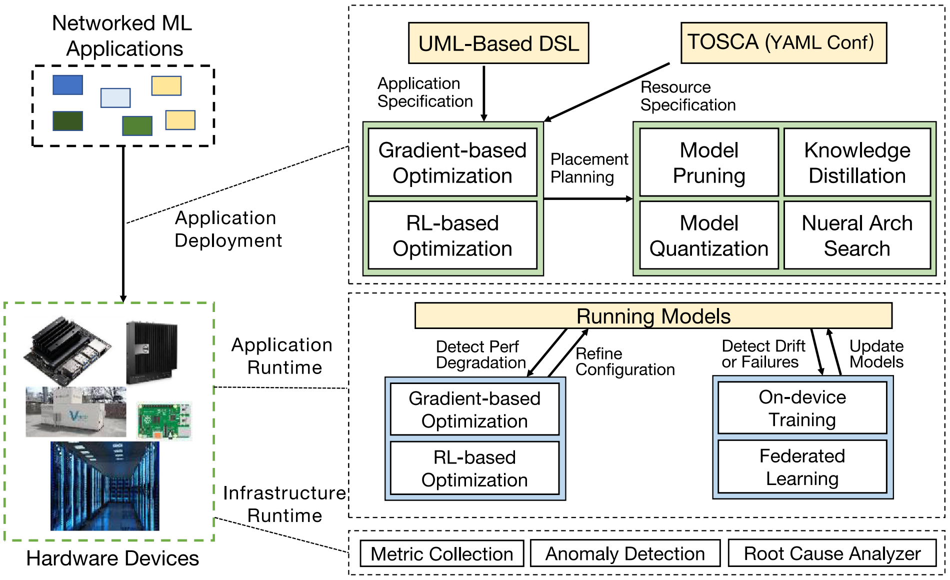

FIGURE 2. Architecture of *AUTOSTEER*.

synchronize and aggregate the up-to-date global models on the fly.

Low-cost platform monitoring and troubleshooting: Monitoring is one of the primary issues in maintaining ML applications and systems; outline or anomaly detection is important to find out unexpected model prediction or any system-wide issues in the early stage. However, anomaly detection and trouble-shooting could be challenging as high-quality labeled data are sparse and difficult to obtain and hence only semisupervised or unsupervised approaches could be applied. The overhead is another non-negligible consideration when designing application instrumentation and metric collection. This usually indicates a tradeoff between the accuracy and granularity of the measured data. Hence, the platform solution of infrastructure monitor should have an overall co-design of metric sampling, storage and real-time analysis.

SYSTEM DESIGN

In response to the aforementioned challenges, we develop *AUTOSTEER*, an orchestration platform for application deployment and runtime management. In this section we mainly highlight a set of key techniques used for implementing the orchestration mechanism. Figure 2 describes the architecture of *AUTOSTEER*.

Automatic Application Deployment

Application and resource specification: The user submits an ML application with execution logic, pretrained models and specifies the pertaining requirements, such as model accuracy and end-to-end latency. To achieve an automatic deployment, we need to translate this knowledge to machine-understandable language. We use a UML-based visual domain specific language[4] that can easily represent the component dependencies within an application and specify the format and source of input and output of each individual component. As a result, the interactions between components, such as data flows and service calls, are loosely coupled through interfaces and agnostic about any model updates. Apart from the application specification, standardized resource specification is the key to automatic and efficient deployment. We exploit[5] for specifying the available underlying computing resources and the hardware and software requirements of each application.

Planning optimization for device placement: To navigate the algorithmic complexity, the orchestrator in *AUTOSTEER* adopts two optimization techniques: gradient-based optimization[6] and reinforcement learning (RL).[7] Gradient-based approaches work upon a realistic model to formalize an optimization problem and usually have relatively low time complexity without the need of *a priori* knowledge or experience, which are therefore suitable for new applications. In contrast, RL-based methods can learn the optimal planning from the experiences and can better support the uncertainties compared the gradient-based solutions.

We also construct an efficient data messaging subsystem where two types of dependencies are defined—*data flow* and *service call*. Since the orchestration system needs to deliver a large volume of data in distributed environments, high system throughput becomes a critical system objective. We employ the publish/subscribe paradigm implemented in Apache Kafka to underpin the data flows. The service call, on the other hand, is implemented through RESTful APIs, as the precise command delivery is the primary goal. Both the *AUTOSTEER* publish/subscribe and RESTful paradigms can be implemented upon a vast majority of network types and protocols, hence capable of supporting most networked ML applications.

Model Adaptation

Computation optimization aims to improve the execution efficiency of different computation units associated with the model (e.g., vector–vector, vector–matrix, and matrix–matrix operations) on various hardware. Optimizing the execution pipeline of the computation graph of a neural network can further improve model performance. We use TensorRT along with the adjustment of weights and numerical precision associated with the activation function (e.g., INT8 and FP16). Model architecture optimization improves the efficiency of on-device computation through well-designed models, such as MobileNetV2, ShuffleNet etc.,—part of the TensorFlow-Lite toolkit). We use *YOLOv3*[8] to strike a balance between computation efficiency and model accuracy.

In addition, more advanced and customizable approaches, such as neural architecture search (NAS)[9] and model compression can be implemented in *AUTOSTEER* further. NAS automates the search of an optimal network structure with the aid of RL or genetic algorithm-based approaches. However, it is computation-intensive and tends to be problematic given the portable devices with limited resources. Model compression is thus extensively studied in three notable aspects: *model pruning* that removes the redundant parameters within the networks; *quantization* that reduces the weights precision, and *knowledge distillation*[10] that trains a new small model based on a larger model. Quantization is the most straightforward approach at the risk of precision degradation and model pruning is the most well-established approach but requires extra calibration process. Integrating mixed techniques in the platform is already underway for building more adaptive and robust models.

End-to-End Application Optimization

In a networked ML system, computational and network resources are dynamically available at different levels. Application parameters, such as input rate and the targeted accuracy need to be adjusted, in response to the ever-changing traffic congestion, to assure the end-to-end latency or system throughput.

We specify model parameters based on extensive benchmarking experiments and transform the problem of finding the "best" setting of parameters into an optimization problem using techniques, such as convex optimization, evolution- and gradient-based methods. RL is an alternative approach that uses statistical or deep learning model where the application parameters are the actions of the agent, and the available computing resources represent the environment. The system performance is represented by the reaction of the environment to the actions. As opposed to the optimization-based approaches that have better interpretability but need extra hand-crafted modeling process, the RL-based approaches have better representation capabilities and can learn to set optimal application parameters from experience.

Model Update

Coping with the drift: During the lifecycle of an ML application, the relationship between the input variables and the performance of the targeting prediction inevitably experiences constant change and drift over time. The model drift usually originates from the following aspects. 1) *Invalid measurement indicator:* the replacement of data collection devices may give rise to different value spaces and a broken device could always deliver nil reading. 2) *Concept drift:* data distribution or statistical characteristics, which is uncertain and frequently varying over time, may lead to concept drift. 3) *Data drift:* the model effectiveness is also prone to inherent changes, such as the seasonal temperature rise and fall. Drifts can be roughly categorized into several classes: sudden drift (sudden change of the data pattern), gradual/incremental drift (new pattern that replaces the old ones within a period of time), and reoccurring drift (old patterns repop up later).

It is imperative to detect such drifts, understand the degree of drift and intervene the model for adapting to changing environments. There are three representative classes of drift detection. 1) Error rate-based approaches focus on the online detection of errors or sudden changes for triggering the model update. 2) Data distribution-based approaches mainly measure the statistical similarities between the original data and the new data and check if the difference is sufficient for model update. 3) Hypothesis test-based approaches, built upon the previous two methods, apply various hypothesis tests to quantify further the severity of model drift. Based on these approaches, our solution can determine *when to intervene* according to the starting and ending points of the drift, *where to intervene*, i.e., localizing the concept/data drift in the feature space, and *how to intervene*, in

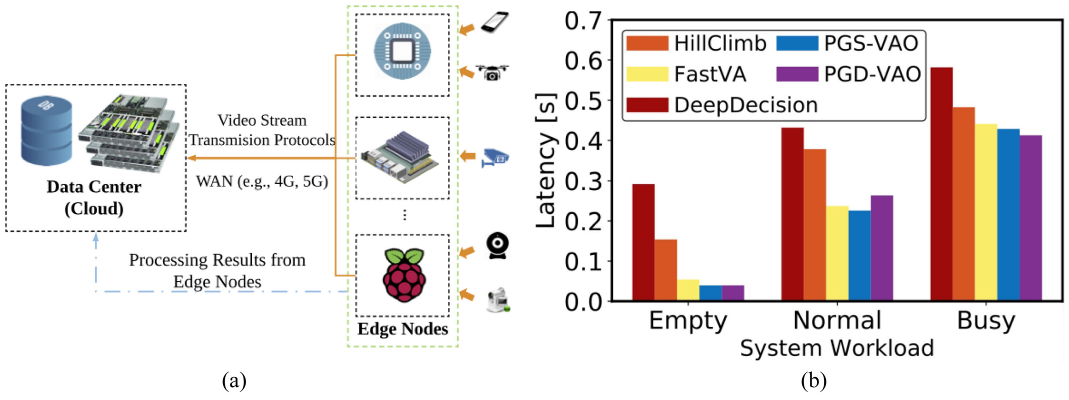

FIGURE 3. Edge-cloud video analysis application and an early performance comparison. (a) Illustration of an edge-cloud video analysis system. (b) Performance of workload optimizer in different system working conditions.

the light of the type and degree of the drift, by adaptively choose model update strategies. The most straightforward approach is the model retraining and updating. For concept drift, we ensemble several base classifiers or utilize knowledge transfer learning for the emerging new target variables.

System implementation: The amount of data engaged in the model update has an impact on the training effectiveness and the system overhead: less data can reduce computation and storage cost but only reflect the latest data distribution; more data are beneficial for reshaping models with higher precision, along with increased overhead. We employ an adaptive window-based solution to select the optimal data amount used for on-device training and/or global model synchronization via ADWIN[11] algorithm: instead of using a fixed time window, the algorithm calculates the drift rate from all possible windows and selects the best cut that reveals the optimal drift level. We modularize and implement the drift detection and alarming system in *AUTOSTEER*. The detection module is responsible for data retrieval and extraction of data statistical properties, and we then leverage hypothesis tests to evaluate the drift degree. Once the alarming system confirms the existence of the model drift, we employ techniques in *Section Model Adaptation* for efficient on-device training. For federated learning applications, once local model has been updated, we also trigger gradient aggregation to keep the global model up-to-date.

Infrastructure Monitor and Maintenance

To learn how the applications perform, we either collect general-purpose telemetry metrics in a black-box manner or instrument, as an integral part of the models, subsystems or system services, in a white-box manner. The metric tracking and tracing system of our orchestration infrastructure collects system logs, model metrics (task execution status, prediction statistics, and evaluation metrics as baselines), system metrics (request latency, error rates, network status, etc.), and resource metrics (CPU utilization, memory utilization, GPU usage, etc.) in real time, and ships them to a centralized analytic platform. We adopt the random sampling mechanism on each agent that is deployed on each physical node, for reducing the overhead of data collection. More advanced technologies, such as sketch[12] can be further added. *Anomaly Detector* comprises real-time event-based processing units, used for identifying per-application performance degradation while *Root-cause Analyzer* is implemented to troubleshoot the causes of performance degradation based on the collected performance indicators.

CASE STUDY: EDGE-BASED REAL-TIME VIDEO ANALYTICS

In this section, we showcase a real-world application backed up by the deployment and runtime management mechanisms in *AUTOSTEER*.

As shown in Figure 3(a), we develop a video analytical application following the edge-cloud paradigm. A set of video generating devices (e.g., traffic surveillance cameras, drones, mobile phones) produce live video streams, which are then processed either on low-power edge devices (e.g., Raspberry pi, Jetson Nano, computing chips), or GPU cluster in cloud datacenters. We prototype the video analytic application via object detection models yolo3 and the wide area network communication between edge devices and the data center is implemented by using the real-time video stream transmission protocol.

The heterogeneity of edge nodes and the interplay among the edge and cloud introduce uncertainties regarding network latency, hardware slowdown, or failures. As discussed in the "End-to-End Application Optimization," section the collected fingerprints and system status are mathematically modeled with a hierarchy queuing model that reveals the relationships between the workload offloading rate (between the edge and cloud) and the system latency and throughput. We then formulate a min-latency optimization problem bounded by a minimal throughput threshold. For model optimization, we implement two gradient-based optimization algorithms (i.e., PGD-VAO and PGS-VAO) to ascertain a solution to minimizing the overall latency. All components are containerized and deployed at both the edge and the cloud side via *AUTOSTEER*.

Figure 3(b) shows the performance of our proposed algorithms under *empty*, *normal,* and *busy* system workloads. Specifically, we insert video chunks into system buffering queues to simulate different workloads. Then, we test our algorithms against the other state-of-the-art task-offloading approaches, i.e., DeepDecision and FastVA. We can see that with the increase of the workload, the system latency is increasing as well. It is also clear that our modeling-based algorithms (e.g., PGS-VAO, PGD-VAO, FastVA) perform better than nonmodeling-based algorithms.

CONCLUSION

Most prior work related to ML applications focuses on algorithm design and optimization for better training ML models.

Although such work is essential for specific applications, there are few studies on the holistic orchestration solution to maintaining the lifecycle of networked ML applications. In this article, we first highlight several key challenges facing the orchestration systems. We then present a set of techniques to deploy ML applications onto resources across cloud and edge devices and assure their runtime performance, making models being served free from model decay and performance degradation due to inappropriate parameter setting. These assist in finding effective pathways to automating the management of networked ML applications at production level, although, admittedly, it still calls for significant effort in large-scale engineering practices and integration with wider domain-specific scenarios.

ACKNOWLEDGMENTS

This work was supported in part by U.K. EPSRC under Grant EP/T01461X/1, in part by U.K. Alan Turing Institute Post-Doctoral Enrichment Award Programme, and in part by U.K. Alan Turing Pilot Project.

REFERENCES

1. B. Qian *et al.*, "Orchestrating the development lifecycle of machine learning-based IoT applications: A taxonomy and survey," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–47, 2020.
2. M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.
3. B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
4. T. Eterovic, E. Kaljic, D. Donko, A. Salihbegovic, and S. Ribic, "An Internet of Things visual domain specific modeling language based on UML," in *Proc. 25th Int. Conf. Inf., Commun. Automat. Technol.*, 2015, pp. 1–5.
5. T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable automated deployment and management of cloud applications," in *Advanced Web Services*. New York, NY, USA: Springer, 2014, pp. 527–549.
6. D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2113–2122.
7. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. Cambridge, MA, USA: MIT Press, 2018.
8. J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018, *arXiv:1804.02767*.
9. T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.
10. J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 4793–4801.
11. A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proc. 2007 SIAM Int. Conf. Data Mining*, 2007, pp. 443–448.
12. T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: A sketch framework for frequency estimation of data streams," in *Proc. VLDB Endowment*, 2017, pp. 1442–1453.

ZHENYU WEN is a professor with the Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, 310023, China. Contact him at zhenyuwen@zjut.ed.cn.

HAOZHEN HU is currently working toward the master's degree with the College of Information, Zhejiang University of Technology, Hangzhou, 310023, China. Contact him at 2112003108@zjut.edu.cn.

RENYU YANG is a research fellow with the University of Leeds, LS2 9JT, Leeds, U.K. Contact him at r.yang1@leeds.ac.uk.

BIN QIAN is a postgraduate research student with the school of computing, Newcastle University, NE1 7RU, Newcastle, U.K. Contact him at b.qian3@ncl.ac.uk.

RINGO W. H. SHAM is a research technician with the school of computing, Newcastle University, NE1 7RU, Newcastle, U.K. Contact him at ringo.sham@newcastle.ac.uk

RUI SUN is a postgraduate research student in the school of computing, Newcastle University, NE1 7RU, Newcastle, U.K. Contact him at r.sun5@newcastle.ac.uk.

JIE XU is a chair professor with the School of Computing, the University of Leeds, LS2 9JT, Leeds, U.K., and chief scientist of BDBC, Beihang University, Beijing, China. Contact him at j.xu@leeds.ac.uk.

PANKESH PATEL is a researcher with AI Institute, University of South Carolina, Columbia, SC, 29208, USA. Contact him at dr.pankesh.patel@gmail.com.

OMER RANA is a full professor with the School of Computer Science and Informatics, Cardiff University, CF24 3AA, Cardiff, U.K. Contact him at ranaof@cardiff.ac.uk.

SCHAHRAM DUSTDAR is a full professor of computer science with TU Wien, 1040, Vienna, Austria. Contact him at dustdar@dsg.tuwien.ac.at.

RAJIV RANJAN is a chair and professor with Newcastle University, NE1 7RU, Newcastle, U.K., and with the China University of Geosciences, Wuhan, China. Contact him at raj.ranjan@ncl.ac.uk.