ACE: Towards Application-Centric Edge-Cloud Collaborative Intelligence

Luhui Wang
Xi'an Jiaotong University

Cong Zhao
Imperial College London

Shusen Yang*
Xi'an Jiaotong University

Xinyu Yang
Xi'an Jiaotong University

Julie McCann
Imperial College London

ABSTRACT

Intelligent applications based on machine learning are impacting many parts of our lives. They are required to operate under rigorous practical constraints in terms of service latency, network bandwidth overheads, and also privacy. Yet current implementations running in the Cloud are unable to satisfy all these constraints. The Edge-Cloud Collaborative Intelligence (ECCI) paradigm has become a popular approach to address such issues, and rapidly increasing applications are developed and deployed. However, these prototypical implementations are developer-dependent and scenario-specific without generality, which cannot be efficiently applied in large-scale or to general ECC scenarios in practice, due to the lack of supports for infrastructure management, edge-cloud collaborative service, complex intelligence workload, and efficient performance optimization. In this article, we systematically design and construct the first unified platform, ACE, that handles ever-increasing edge and cloud resources, user-transparent services, and proliferating intelligence workloads with increasing scale and complexity, to facilitate cost-efficient and high-performing ECCI application development and deployment. For verification, we explicitly present the construction process of an ACE-based intelligent video query application, and demonstrate how to achieve customizable performance optimization efficiently. Based on our initial experience, we discuss both the limitations and vision of ACE to shed light on promising issues to elaborate in the approaching ECCI ecosystem.

1 INTRODUCTION

In recent years, machine learning, especially deep learning, has been applied to various domains (*e.g.*, computer vision, speech recognition, and video analytics). Emerging *Intelligent Applications* (IAs) such as image classification based on deep Convolutional Neural Networks (CNNs) [23], traffic flow prediction based on deep Recurrent Neural Networks (RNNs) [44], and game development based on deep Generative Adversarial Networks (GANs) [22], are demonstrating superior performance in terms of accuracy and latency. Such performance, however, requires tremendous computation and network resources to deal with the increasing size of Machine Learning (ML)/Deep Learning (DL) models and the proliferation of vast amounts of training data [29].

Cloud computing is indisputably attractive to IA developers as the predominating high-performance computing paradigm [5]. Typically, cloud providers offer services like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) to facilitate application implementation, where resources like high-performance computation, massive elastic storage, and reliable network services are allocated according to user requirements. Intuitively, mainstream IAs are deployed on the Cloud to leverage centralized resources for computationally-intensive Artificial Intelligence (AI) tasks like data processing, ML/DL model training, and inference. For instance, the distributed training of AlphaGo [39] is a typical *'Cloud Intelligence'* (CI) representative.

However, novel challenges to CI emerge when modern IAs rapidly proliferate and are required to be in production in practice, where *high end-to-end service latency*, *high network bandwidth overhead*, and *severe privacy leakage threat* are among the most critical ones [48]. Instead of concentrating on the Cloud, increasing efforts attempt to exploit heterogeneous resources distributed at the network Edge to address such issues. For example, some IAs offload DL tasks to edge servers (*e.g.*, Nvidia Jetson TX2 Board) [47] for privacy preservation and timely responses. Such an edge offloading of relatively simple AI tasks, or *'Edge Intelligence'* (EI) [15, 34], manages to alleviate the controversy between broadened requirements of modern IAs and the conventional CI paradigm.

The rapid development of EI and corresponding prototypes demonstrates that, due to edge devices' heterogeneous resource constraints, the Cloud is still critical to modern production level IAs with multi-faceted performance requirements [48]. Increasing IA developers start to focus on efficiently leveraging edge resources under cloud coordination to collaboratively conduct AI tasks with optimized performance [1, 40], or *'Edge-Cloud Collaborative Intelligence'* (ECCI). ECCI relies on pivotal interdisciplinary technologies of cloud and edge computing (supporting ECCI infrastructure and runtime), and ML/DL-based AI (introducing rich IA workloads).

Existing ECCI applications (*e.g.*, HOLMES [18] for healthcare, EdgeRec [12] for E-commerce, SurveilEdge [43] for urban surveillance, and general solutions like CLIO [19] and SPINN [24]) are individually developed and deployed by either academic researchers or industrial communities, where both the application design and system implementation are highly *developer-dependent* and *scenario-specific*. For example, SurveilEdge [43] is a typical ECCI application for real-time intelligent urban surveillance video query. In its prototypical implementation, the developers depend on relatively higher edge computation capabilities (*i.e.*, X86 PCs) to support system scaling without subtly designing an ECC infrastructure management scheme. For the ease of implementation, they hard-code the load balancing policy with the video query workload for latency

*S. Yang (Corresponding Author) is with Xi'an Jiaotong University and Pazhou Laboratory. Email: shusenyang@mail.xjtu.edu.cn

reduction. Additionally, to achieve intelligent video query, the entire solution is specifically designed to support CNN training and inference workloads, where dedicated service links (*e.g.*, message service links) among all application components are individually configured to achieve edge-cloud collaborations. Without impacting the application performance, such developer-dependent design and implementation, however, are impeding others to migrate the application to general ECC infrastructures (*e.g.*, resource-constrained Industrial IoTs) or pursue customizable performance optimizations (*e.g.*, joint optimization of latency and bandwidth consumption). Moreover, if others want to adopt SurveilEdge (or other existing applications) as the backbone of other applications, driven by different DL models and deployed at different infrastructures, corresponding DL runtimes and different ECC services have to be designed and implemented by the adopters themselves thoroughly. Such a non-generic manner is severely hindering the proliferation of production level ECCI applications.

Therefore, *for the cost-efficient implementation of high-performance production level ECCI applications, it is necessary to construct a unified platform handling both ever-increasing edge and cloud resources and emerging IA workloads with increasing scale and complexity.* Particularly, to construct such a platform, the following **four challenges** need to be explicitly addressed:

Support for unified management of hierarchical and heterogeneous infrastructures. The efficient implementation of ECCI applications requires unified management of not only infrastructures offered by traditional centralized cloud providers but also heterogeneous computation, storage, and network resources geographically dispersed at the edge. The development and deployment of ECCI application components on edge devices are extremely inefficient due to the lack of a unified platform. Furthermore, it is infeasible to directly migrate IaaS and PaaS technologies in cloud computing to the management of inherently distributed edge resources [4].

Support for user-transparent ECC services. ECCI application developers require services providing user-transparent edge-edge and edge-cloud collaborations. In most cases, components of existing ECCI applications are independently deployed on edge nodes, only interacting through services deployed on the Cloud. Such a manner increases both bandwidth cost and response latency. Few existing edge services (*e.g.*, Dapr [9]) can improve edge autonomy and application performance to a certain extent. However, due to the lack of links between edge and cloud services, they cannot provide user-transparent collaborative services to developers.

Support for complex IA workloads. Efficient ECCI application implementations require comprehensive system-level supports to complex IA workloads like ML/DL model training and inference, which cannot be provided by existing cloud and edge computing platforms. For instance, in edge computing systems for IoT data processing, the message-driven communication solution for transmitting KB-level sensor data cannot effectively handle the transmission of DL models as large as hundreds of MBs. Moreover, most existing distributed ML/DL solutions are designed for datacenter networks with high bandwidth and low transmission latency. Such methods are inefficient in ECC systems with inherent constraints like prolonged and unstable End-to-End (E2E) communication latency.

Support for unified optimization of ECCI applications. Unified performance optimization mechanisms are important to efficient ECCI application implementations. For most existing edge computing applications, the efficiency of resource utilization highly depends on the developer's design, where effective optimizations require a profound understanding of system architectures and optimization theories [16]. For existing ECCI applications, except for the multi-component development and cross-device deployment of inherently complex IA workloads, the developers also have to deal with the overall performance optimization across ECC infrastructures by themselves, not to mention the difficulties in application debugging, monitoring, and profiling caused by the distributed and heterogeneous environment. Such a requirement is quite challenging to not only developers of emerging ECCI applications, but also those who want to migrate existing IAs to ECC infrastructures.

2 ECCI APPLICATION PATTERNS

Currently, there exists no commonly-accepted abstraction of general ECCI application patterns, which are critical to improving the efficiency of ECCI application development and deployment. As the foundation of the unified platform, considering the subject of different application tasks, we extract **four** common patterns, *i.e.*, *ECC processing*, *ECC training*, *ECC inference*, and *hybrid collaboration*.

ECC Processing of data is the basis of various ECCI applications. Collaborative data processing applications are often built as *pipelines* or *Directed Acyclic Graphs* (DAGs). For example, the Steel framework [33] deploys a streaming analytic pipeline of different data processing tasks (*e.g.*, filtering, anomaly detection, and storage) for ECC IoT anomaly detection applications.

ECC Training refers to conducting ML/DL model training based on edge-cloud collaborations. Unlike ECC processing, ECC training requires complex interactive and iterative data and control flows between edges and the Cloud (*e.g.*, training data, model, and hyper-parameter exchanges). For instance, Federated Learning (FL) is a typical ECC training application, which conducts ML training across devices to protect data privacy (*e.g.*, Gboard Mobile Keyboard [13] and Apple QuickType Keyboard [3]), and to bridge data silos (*e.g.*, model training for bank fraud detection [45]).

ECC Inference focuses on ML/DL model inference, where only forward propagation is conducted. Generally, ECC inference is achieved through either intra-model or inter-model collaborations. In intra-model solutions, a single DL model is decomposed by layers into two parts (*i.e.*, neural network partitioning) deployed at edges and the Cloud respectively for collaborative inference (*e.g.*, Neurosurgeon [21], SPINN [24], and JointDNN [11]). In inter-model ones, however, multiple DL models with different functionality or performance are deployed at edges and the Cloud respectively for collaborative inference (*e.g.*, VideoEdge [20] and SurveilEdge [43]).

Hybrid Collaboration combines at least two of three ECCI application patterns above or integrates additional CI/EI capabilities into ECCI applications. For example, ShadowTutor [8] enables robust HD video semantic segmentation with significant throughput improvement and network data transmission reduction. Here, cloud servers conduct both the inference of the heavy and general 'teacher' model and the training of the lightweight 'student' model. Mobile edge devices conduct the 'student' model inference.

3 ECCI PLATFORM DESIGN PRINCIPLES

In this article, we aim to construct a unified platform for the efficient development and deployment of ECCI applications. It is required to provide efficient management of heterogeneous ECC infrastructures, user-transparent ECC services, and customizable performance optimizations, supporting scalable, reliable, and robust ECCI application development and deployment. The desired platform should be treated as *ECCI-as-a-Service* (ECCIaaS), similar to the concept of Machine Learning-as-a-Service (MLaaS). Particularly, we extract **five** essential design principles as follows.

Principle One: *an instance of ECCI application should be an integrated entity that can be managed in a scalable manner.* This principle requires the unified management of typical edge and cloud infrastructures, including hardware nodes like edge gateways, clusters like Kubernetes [10], virtual machines, and third-party cloud services like Azure IoT Hub [30]. Any operation of ECCI applications (*e.g.*, deployment and monitoring) should be carried out on large-scale collaborative infrastructures organized as a unity. ECCI applications should be able to provide continuously available services when the infrastructures are scaled or upgraded.

Principle Two: *ECCI application components at edges and the Cloud should be able to operate in both collaborative and autonomous manners.* Unlike the datacenter network on the Cloud, the edge-cloud network has limited capabilities (*e.g.*, bandwidth), and may perform unstably. While supporting collaborations with the Cloud, edges should be able to cache data and provide partial services autonomously to mitigate the impact of network partitioning.

Principle Three: *orchestration is essential to ECCI applications.* Except for edge-cloud separations, modularized ECCI application components have specific requirements of computation and storage resources, as well as deployment locations. Moreover, there can be multiple applications co-located at the same infrastructure. Therefore, component orchestration is necessary to ensure that all applications' resource and user requirements can be satisfied.

Principle Four: *provide in-app control of ECCI applications.* In most cases, offloading computation to edges may not directly improve application performance. Here, in-app control optimization has been demonstrated to be effective in various aspects like bandwidth saving [32] and E2E latency reduction [36], which should be seriously considered for application performance enhancement.

Principle Five: *support as many types of ECCI application workloads as possible.* ECCI application scenarios are ever-increasing, such as federated model training and ECC model inference. It is essential for the platform to support common application patterns and services, facilitating efficient development and deployment of a broadened spectrum of ECCI applications.

4 APPLICATION-CENTRIC ECCI PLATFORM

Driven by all principles above, the explicit design of our Application-Centric ECCI (ACE) platform is as follows.

4.1 Overview

We illustrate the general ECCI application development and deployment procedure based on ACE in Figure 1. For application developers, this procedure comprises **three** major phases, *i.e.*, *user registration*, *application development*, and *application deployment*.

In the *user registration* phase, any ECCI application developer can register at ACE as a *platform user*. The user first requests the registration of an ECC infrastructure at ACE, and registers all his/her edge and cloud *nodes* to form an infrastructure according to operational instructions replied by ACE (see Part 4.3.1). Here, a *node* can be either a physical device or a virtual service like an edge gateway, a cloud server, a private or public cloud, *etc.* The user can also select to deploy different resource-level services based on service components provided by ACE on the infrastructure, which can be shared among all his/her ECCI applications (see Part 4.3.2).

Then, in the *application development* phase, the user implements applications in a modularized manner. Specifically, for each application, different components are separated according to user-defined business logic or functionalities. Meanwhile, requested by ACE, the user deliberately decouples application control flows with workload flows for collaboration optimization and component reuse (see Part 4.4.2). All components are then implemented using the ACE Software Development Kits (SDKs), and encapsulated into containers that can be deployed on edge or cloud according to components' resource and user requirements. For each application, the user constructs a *topology file* describing component relations and resource and user requirements of each component. All component images and corresponding topology files are then submitted to ACE.

Finally, in the *application deployment* phase, ACE determines a *deployment plan* for all components of a specific application according to the topology file, guaranteeing that all resource and user requirements are satisfied (see Part 4.4.3). According to the plan, the application can be deployed on the user's ECC infrastructure through ACE. All deployed applications are continuously monitored by ACE for maintenance, and corresponding users can upgrade, monitor, and remove their applications at any time.

To achieve the procedure above, we construct our ACE platform in a hierarchical manner with **three** layers, *i.e.*, *platform layer*, *resource layer*, and *application layer*. The general architecture of ACE is illustrated in Figure 1. Details of each layer are as follows.

4.2 Platform Layer

This layer manages the ACE platform, all registered users, and users' nodes and applications. It also offers platform-level services for users and their applications.

4.2.1 Platform Management. Our platform-layer manager comprises *controller*, *orchestrator*, *API server*, *Pub/Sub service*, *monitoring service*, and *user interfaces*:

Controller manages platform users, their nodes and applications, *e.g.*, registers and deletes users, shields failed nodes, and controls node component deployment.

Orchestrator determines a deployment plan for all components of each application based on the topology file (see Part 4.4.3), ensuring resource (*e.g.*, computing) and user (*e.g.*, location) requirements of all components are satisfied.

API Server provides uniform APIs for querying and manipulating the status of ACE entities (*e.g.*, users, nodes, applications) to other platform manager components (*e.g.*, orchestrator, controller).

Pub/Sub Service provides a bi-directional data transmission channel between ACE and user nodes and applications (*e.g.*, delivering deployment instructions from the controller to user nodes).

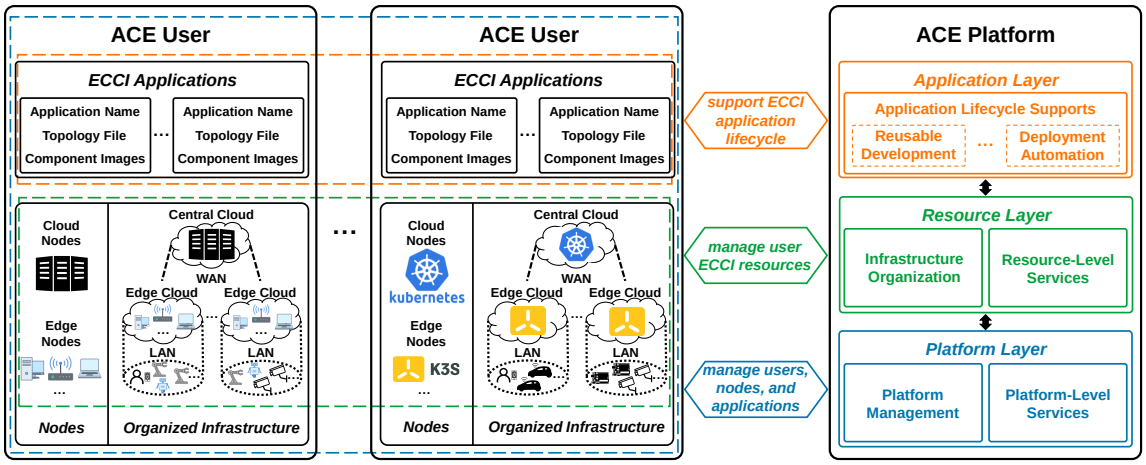

Figure 1: The general architecture of ACE.

Monitoring Service collects the status, performance metrics, and runtime logs of ACE, user nodes and applications.

User Interfaces enhances ACE's user-friendliness with Command Line Interface (CLI) and web-based dashboard. For example, the dashboard with a 'drag-and-drop' visual application editor can be used for handy application development.

4.2.2 Platform-level Services. Platform-level services are not ACE's internal features. They can be implemented as requested to improve the efficiency of ECCI application development and deployment based on ACE. Following are two typical examples:

Image Registry hosts ACE-provided images (*e.g.*, controller, orchestrator), generic runtime images (*e.g.*, Python runtime), and user-provided customized application images.

Validation Testbed allows users to develop, debug, and monitor ECCI applications on an SDN-based application validation testbed. For example, the impact of edge-cloud channel dynamics (*e.g.*, bandwidth, delay, jitter) on the testbed can help users understand the actual performance of an ECCI application in real-world networks.

4.3 Resource Layer

This layer manages the ECC infrastructure of each user. It also provides resource-level services shared among applications deployed on the same infrastructure.

4.3.1 Infrastructure Organization. Considering *Principles One* and *Two*, ACE requires all nodes of each user to be organized as several *Edge Clouds* (ECs) and one *Central Cloud* (CC) to host scalable ECCI applications, and to enable autonomous operations of edge components. For a specific user, all his/her edge nodes are divided into several groups according to the user's preferences (*e.g.*, in terms of nodes' geographical locations or resources). Each group is organized as an EC, serving all end nodes (*e.g.*, IoT sensors and cameras) that can access the EC through Local Area Network (LAN). All cloud nodes of the user are organized as a single CC, and it can interact with each EC through Wide Area Network (WAN). For each EC and the CC, internal nodes are organized as a cluster (similar to Kubernetes ideally, or a node pool for simple implementation).

Treating each EC and the CC as a resource-level operational unit allows ACE to effectively manage the infrastructure and deploy applications on such an infrastructure, without considering the explicit management of potentially massive underlay nodes. Moreover, when there is no cloud coordination caused by either CC or edge-cloud communication failure, each EC as a cluster remains (partially) functional, enabling local area collaborations among edge components based on corresponding edge services.

Receiving the user's registration request, ACE assigns a unique infrastructure ID to the user, and establishes a node information record for infrastructure organization. Meanwhile, ACE assigns a unique second layer ID affiliated to the infrastructure ID to each EC and the CC claimed by the user, where corresponding node registration instructions are generated automatically. Replied from ACE, such instructions are executed by the user on nodes. An agent is deployed on each node, informing ACE of the node information and the EC or CC the node belongs. ACE assigns a unique third layer ID affiliated to the EC or CC's ID to each node. The agent is also used for application deployment and application status collection.

4.3.2 Resource-level Services. For ECCI applications with the typical patterns discussed in Section 2, essential services like small/big packet communication and data caching/storage are commonly required [16, 25, 31]. In a specific ECC infrastructure, existing services supporting ECCI applications are conventionally deployed on both ECs and the CC, serving EC and CC clients (*i.e.*, application components) respectively to ensure the autonomy of ECs. Each service is accessible to all applications deployed on the same infrastructure. However, due to the lack of links between edge and cloud services, conventional services require application developers to handle complex edge-cloud interactions. Treating conventional message service for small packet communication as an example, as shown in Figure 2, for edge-cloud unicast communications, all EC clients have to directly access the message service at CC (*i.e.*, ①) to communicate with CC clients. Here, the developer has to handle the CC message service authorization to each EC client individually, which is quite expensive for large-scale ECCI applications.

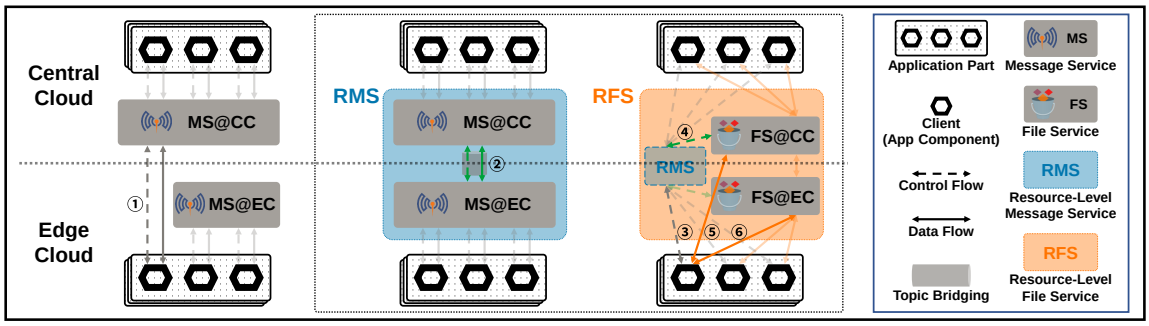

Figure 2: Illustration of ACE provided resource-level services.

Considering *Principle Five*, to facilitate efficient application development, ACE prefers to provide E2E resource-level services with unified interfaces to EC and CC clients, respectively. Therefore, *long-lasting links* between services on ECs and the CC need to be established. Some conventional services support the direct establishment of such links (*e.g.*, service bridging for the message service). Specifically, as shown in Figure 2, ACE implements a *resource-level message service*, where the long-lasting link between EC and CC message services (*i.e.*, ②) is established using MQTT topic-bridging [27]. Here, edge-cloud interactions are conducted by ACE provided SDK, and each client only needs to interact with its local service with a dedicated interface. For other services, directly establishing long-lasting links is expensive or even infeasible. For example, the link between edge and cloud file services could be established using file synchronization, which induces additional requirements on network condition, computation, and access authorization. Instead, ACE uses the resource-level message service to establish long-lasting links for other services. ACE implements a *resource-level file service*, whose control flow (*e.g.*, ③,④) is separated from the data flow and handled by the resource-level message service. Furthermore, the proverbial object storage service is used to handle the data flow (*e.g.*, ⑤,⑥) for transmission simplification.

Note that, as shown in Figure 2, three types of links are used in resource-level services, *i.e.*, ad-hoc links (grey) for repetitive interactions, ad-hoc links (orange) for one-off interactions, and long-lasting links (green) established once the service is deployed. Besides, resource-level services should provide basic operations for applications through their lifecycle (*e.g.*, temporary storage for intermittent models and data, and permanent storage for final trained models in the file service).

4.4 Application Layer

This layer supports user applications through the entire lifecycle.

4.4.1 ACE Supported ECCI Application Lifecycle. As a unified platform, ACE supports each application through its entire lifecycle (*i.e.*, designing, coding, building, testing, deploying, and monitoring). For designing, ACE provides a standard specification (*i.e.*, the topology file) to achieve modularized development for ACE-organized ECC infrastructures. For coding, ACE provides the SDKs with access to resource-level services for application components and the user interface to access the essential Integrated Development Environment (IDE). For building, ACE provides the image registry for efficient image management and distribution. For testing, ACE provides the validation testbed for application verification and evaluation. For deployment, ACE provides the orchestrator and the controller for automatic deployment. For monitoring, ACE provides the monitoring service collecting the status of application components and nodes where they are deployed. Such supports from ACE enable users to develop and deploy basic ECCI applications efficiently. For applications with specific performance requirements (*e.g.*, the minimal E2E latency), or with advanced architectures (*e.g.*, large-scale components with complex topology), ACE provides **two** extra supports, *i.e.*, *reusable development* and *deployment automation*.

4.4.2 Reusable Development. Considering *Principles Four* and *Five*, ACE requires developers to decouple and separate control and workload planes of all application components. The control plane conducts in-app control operations, component monitoring, and policy execution (*e.g.* decide the best partition point for intra-model inference solutions [11, 21]). The workload plane conducts computation, storage, and transmission instructed by the control plane (*e.g.*, deep feature compression module [7] or hybrid collaboration pipeline for data processing and inference tasks [41]). Such a separation allows ACE to construct a reusable in-app controller, enabling developers to concentrate on implementing ECCI workloads and efficiently contribute to the ACE based ECCI ecosystem.

For the reusable in-app controller, ACE constructs a series of general in-app control operations (*e.g.*, start, filter, aggregate, and terminate), component monitoring operations, and a basic control policy. Determined by the ECC infrastructure, the controller is constructed at the resource level in an ECC manner (see Part 4.3.1). The CC controller conducts global coordination related operations, and the EC controller coordinates components within the EC. Resource-level services support interactions between CC and EC controllers. For applications with specific performance requirements, developers can inherit the general in-app controller and override optimization methods as advanced control policies (*e.g.*, the rate control based optimal edge-cloud bandwidth allocation [2]).

4.4.3 Deployment Automation. Considering *Principle One*, ACE needs to support efficient application deployment regardless of the topology complexity and the infrastructure scale. To achieve this, ACE constructs an automatic application deployment method only requiring the application topology file containing information like application specification, component clarifications, parameters,

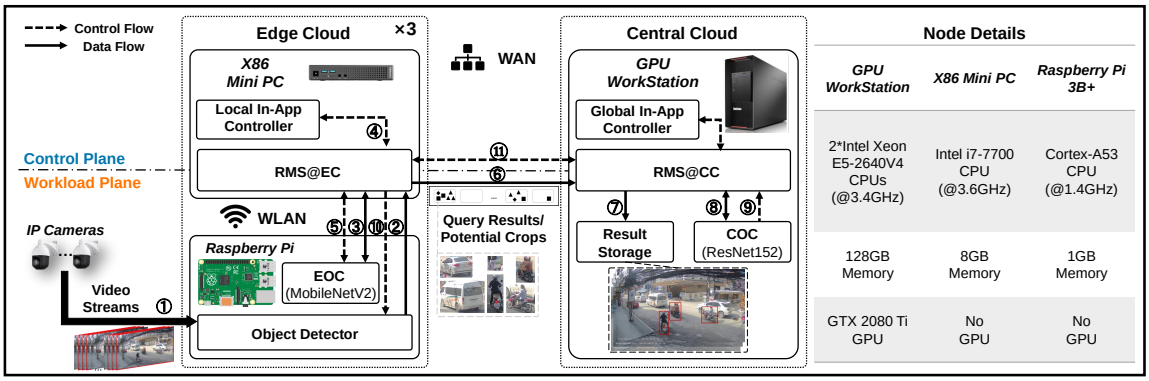

Figure 3: ACE based intelligent video query workflow.

relations, and deployment requirements. Such a manner prevents users from handling complex component-infrastructure mapping.

Specifically, to deploy an application, the user submits the topology file through the user interface to ACE, and triggers the orchestration process. According to component deployment requirements, the ACE platform-layer orchestrator binds each component with specific nodes in the infrastructure and resource-level services required, generating the deployment plan. When the user triggers the deployment process, the ACE platform-layer controller generates the instruction to deploy each component instance on the corresponding node according to the deployment plan, and sends the instruction to the node agent for execution. Note that users can manage applications (*e.g.*, update and delete) by modifying the topology file. For example, for updates, submitting an updated topology file, the user can trigger a thorough update, *i.e.*, ACE deletes the previous application and repeats the entire deployment process. An incremental update can also be triggered, *i.e.*, ACE automatically deploys updated components according to the new topology file.

5 HOW IT WORKS: INTELLIGENT VIDEO QUERIES USING ACE

To validate our platform in supporting efficient and high-performing ECCI application development and deployment, we first present the entire development and deployment process of an intelligent video query application based on ACE, then compare the performance of the application implemented with ACE, CI, and EI, respectively.

5.1 Application Development and Deployment

Video query [26, 43] is one of killer ECCI applications. To fulfil latency-sensitive user-specific video query requests (*e.g.*, query about the existence of a type of objects in video streams from a geographic area), it generally uses edge and cloud resources to retrieve targeted objects from the video streams with a proper tradeoff between query accuracy and response latency under practical edge-cloud bandwidth limitations. Based on Subsection 4.1, we developed and deployed a video query application (based on [43]) using ACE.

5.1.1 User Registration. As an ACE user, we first mounted all our nodes and conducted the organization of our ECC infrastructure instructed by ACE. Our infrastructure comprised a CC (one node, *i.e.*, a GPU workstation), and three ECs (each with four nodes, *i.e.*, an X86 mini PC and three Raspberry Pis). Node details are in Figure 3. For each EC, all edge nodes connected to an individual 100Mbps WLAN. Each EC connected to CC through WAN (campus network) with software-limited bandwidth (*i.e.*, 20Mbps uplink and 40Mbps downlink) and one-way delay (*i.e.*, 0ms and 50ms as ideal and practical networks, respectively). Let each Raspberry Pi receive the real-time video stream from a surveillance camera. We deployed the resource-level message service on the infrastructure.

5.1.2 Application Development. Our application [42] aimed at fulfilling user-specific video query requests accurately and rapidly through edge-cloud collaborations under practical network limitations (*i.e.*, bandwidth and delay). We developed the following components: Data Generator (DG) providing the real-time video stream to the edge node, Object Detector (OD) rapidly extracting video frame crops potentially containing moving objects from the video stream, Edge Object Classifier (EOC) conducting lightweight query-specific binary object classification, Cloud Object Classifier (COC) conducting accurate multi-class object classification, In-app Controller (IC) executing the control policy, and Result Storage (RS) saving all query results. OD on edge nodes was implemented using frame differencing [43] (*i.e.*, cropping regions with salient pixel differences across frames) instead of accurate but complex object detector like YOLOv3 [35] for rapid crop extraction on resource-limited edge nodes. COC on CC was a ResNet152 [17] pre-trained on ImageNet ILSVRC15 [37] (4.49% Top-5 error rate). EOC was a MobileNetV2 [38] rapidly trained on-the-fly by CC whenever there were user-specific queries. To form its query-specific training set, video frame crops containing different classes of objects were extracted on CC by a YOLOv3 pre-trained on COCO [28] (57.9% mAP measured at 0.5 IOU) from historical video data uploaded by cameras at (or nearby) the queried area at leisure time, then labelled by COC. The trained EOC (training details are in [43]) was then deployed on edge nodes in the queried area. We used real video clips from Youtube Live [46] (30 fps, 1920×1080 resolution, various durations) as historical video data and real-time video streams to query. For a motorcycle query task, EOC's training set had 14,000 crops extracted from clips (170 hours total duration) from 14 surveillance cameras at or nearby the queried area (*i.e.*, historical video data). Another 6433 'motorcycle' and 68749 'non-motorcycle' crops

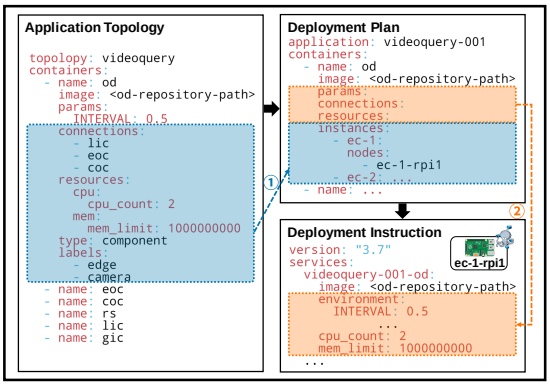

Figure 4: Automatic application deployment.

were extracted as EOC's test set, where EOC achieved 11.06% error rate under 80% object identification confidence, tending to be less accurate than COC. Another three video clips with 5 minutes duration were used as real-time video streams. Each node in the three ECs had one of the three clips.

The video query workflow after EOC's deployment is shown in Figure 3. For each edge node receiving the real-time video stream from DG (*i.e.*, ①), OD selected three consecutive frames with a specific interval (*e.g.*, 0.5 seconds), and rapidly extracted crops potentially containing moving objects. Such crops were classified by EOC (*i.e.*, ②,③), and the results were used by IC for crop scheduling based on the *Basic Policy* (BP) (*i.e.*, ④,⑤). If the object identification confidence of a crop was above 80%, a targeted object was identified (predicted as positive due to the lack of ground truth of the real-time video), and its metadata were sent to RS (*i.e.*, ③,⑥,⑦). If the confidence was below 10%, the crop was dropped. Otherwise the crop was sent to COC (*i.e.*, ③,⑥,⑧). If the Top-5 classification results of the crop on COC contained the targeted label, a targeted object was identified (*i.e.*, predicted as positive), and its metadata was sent to RS (*i.e.*, ⑧,⑦). Since BP may induce queue backlog at EOC and frequent reprocessing at COC, we constructed an *Advanced Policy* (AP) (*i.e.*, ④,⑩) based on BP as a customized IC to further reduce E2E Inference Latency (EIL). AP collected and estimated EILs of EOC (*i.e.*, ⑤,④) and COC (*i.e.*, ⑨,⑪,④) to guide crop uploading from OD (*i.e.*, load balancing [43], always sent to the one with a lower estimated EIL, ②, ⑥,⑧). Then, AP reduced crops uploaded from EOC to COC by shrinking the identification confidence thresholds when either EOC's or COC's EIL got deteriorated.

5.1.3 Application Deployment. As shown in Figure 4, we submitted a topology file to ACE, which was an extended YAML file containing meta information of both the application and all components. We illustrate the deployment of component OD as an instance. Receiving the topology file, as Step ①, the orchestrator determined the node(s) (*e.g.*, Raspberry Pi 'ec-1-rpi1' on edge cloud 'EC-1') satisfying all requirements of OD (*i.e.*, 'connections' implying OD's dependencies with components Local In-app Controller (LIC), EOC, and COC, 'resources' implying CPU and memory required by OD, and 'labels' implying that OD should be deployed on edge nodes connected to cameras). Such decisions were recorded in the deployment plan (*i.e.*, 'instances'), a topology replica modified by the orchestrator. Note that, to manage nodes in an EC as a cluster, ACE can delegate node-level orchestration to the EC. Receiving the deployment plan, as Step ②, the controller transformed information of OD instances into specific deployment instructions in a standard Docker-compose YAML file, which was distributed to the node agent (*e.g.*, the container engine at 'ec-1-rpi1') for OD deployment.

5.2 Impact of Implementation Paradigm on Intelligent Application Performance

We compared the performance of our application implemented with different paradigms. For CI, COC was deployed on CC. For EI, EOCs were deployed on ECs. For ECCI, based on ACE, two versions of the application with BP (ACE) and AP (ACE+) were deployed. Different system loads were simulated by varying the sampling interval of frame differencing in OD from 0.5 to 0.1 seconds. Since all comparatives used the same OD, we compared their video query performance using their object classification performance. Particularly, we used *F1-score* [14] [1], *edge-cloud BandWidth Consumption* (BWC), and *E2E Inference Latency* (EIL) [2] as evaluation metrics. We conducted the motorcycle query task under different system load and edge-cloud network delay (*i.e.*, 0ms for ideal and 50ms for practical) settings. Results are illustrated in Figure 5.

When the system load increases, F1-scores of CI and EI basically remain the same, where CI using COC only and EI using EOC only achieve the highest and lowest F1-scores under all system loads, respectively. ACE and ACE+ using COC and EOC collaboratively manage to achieve F1-scores slightly lower than CI but significantly higher than EI. Unlike EI, in ACE and ACE+, many crops that cannot be confidently classified by EOCs (with a confidence from 10% to 80% and dropped by EI) are uploaded to COC. Compared with CI, few crops are dropped by EOCs (with a confidence below 10%) in ACE and ACE+. Besides, the higher the system load, the better ACE+ performs compared with ACE. Under higher system loads, more crops are directly uploaded from ODs to COC by IC with AP for load balancing in ACE+, reducing crops dropped by IC with BP in ACE. Furthermore, when the system load increases, ACE+ achieves higher F1-scores under practical than ideal network delay. In ACE+, under practical network delay, fewer crops are uploaded from EOCs to COC to avoid higher EILs by shrinking the confidence thresholds, and more are from ODs to COC for load balancing.

When the system load increases, BWCs of all except for EI increase. ACE and ACE+ induce significantly lower BWCs than CI since considerable objects are identified by EOCs. Furthermore, the higher the system load, the higher BWCs of ACE+ compared with ACE. In ACE+, some crops (increase with system load) are directly uploaded by IC with AP for load balancing, where, however, only some of them are uploaded by IC with BP in ACE (with identification confidence from 10% to 80%), inducing higher BWCs.

When the system load is low, CI induces the lowest EIL under different network delay settings benefiting from COC's fast processing (*i.e.*, the inference time of COC is about 32.3ms on CC, and that of EOC on edge node is above 44ms). When the system load

[1] Since real-time video streams to query were not labelled, we classified all crops extracted by OD during the entire query task with COC after the task was finished, and treated COC's predicted labels as the query ground truth for F1-score calculation.
[2] Time from a crop is transmitted by OD to its predicted label is given by EOC or COC.

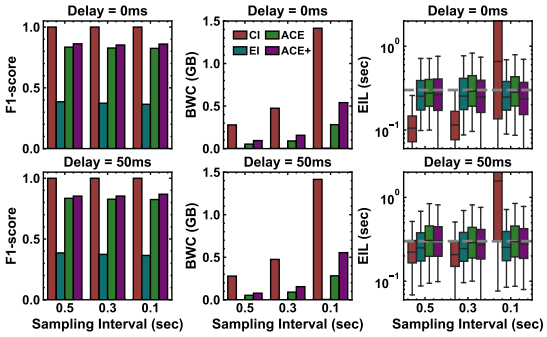

Figure 5: Intelligent video query performance.

increases, different from EI, ACE, and ACE+, CI's EIL increases significantly due to the large queue backlog aggregated from all ODs (normal in large-scale edge-cloud systems). Besides, the practical network delay also enlarges CI's EIL more obviously (significantly higher than the 50ms network delay). Compared to CI, EILs of EI, ACE, and ACE+ are not obviously impacted by both system load (*i.e.*, low queue backlog at EOCs) and network delay (no/low uploading). ACE's EIL is slightly higher than EI since EOCs manage to identify most objects, and only a few crops are uploaded to COC. Furthermore, the higher the system load, the lower EIL of ACE+ compared with ACE. Some crops (increase with system load) are directly uploaded to COC for load balancing by IC with AP in ACE+.

Compared with CI and EI, ACE-based video query manages to better fulfill query requests accurately and rapidly with efficient bandwidth consumption. ACE also facilitates developers for customized optimization (*i.e.*, EIL reduction with customized AP).

6 FUTURE OF ACE

As a prototype for cost-efficient ECCI application development and deployment, ACE is still in its infancy. The construction of ACE reveals fundamental challenges to address and sheds light on the vision of an ACE-based ECCI ecosystem deserving explorations.

6.1 Challenges

Agile ECCI application orchestration is critical, but challenging, to improve the performance of ACE-based applications. ACE's platform-layer orchestrator manages to allocate application components to proper nodes satisfying basic (node-level) resource and user (*i.e.*, edge/cloud deployment) requirements. However, fine-grained orchestration under more explicit constraints is still hard to achieve, which is significant to fully infrastructure utilization. Furthermore, ACE's static application orchestration cannot adjust to application or infrastructure changes. A dynamic orchestrator is also necessary.

Resource contention prevention has to be further investigated to ensure the performance of ECCI applications co-located at the same infrastructure. Currently, ACE manages to achieve component-level resource isolation through containerization, and support inter-component resource allocation optimization through the customized in-app controller, where, however, application-level resource isolation and allocation is still an open issue. Critical resources like edge-cloud bandwidth should be allocated appropriately to co-located applications under ACE's coordination. It is also promising to integrate the serverless technology [6] for elastic resource allocation that cannot be directly achieved by container-based solutions.

Security is another critical issue. ACE now contains no security module, where state-of-the-art encryption and authentication techniques can be directly integrated for fundamental secrecy. The actual challenge, however, is access control. In our design, an ACE user has full access to his/her infrastructure and ECCI applications, where no user collaboration is currently supported. For specific applications (*e.g.*, federated learning) that have to be developed and deployed by multiple users collaboratively, ACE is required to provide a fine-grained access control mechanism. It needs to ensure that each collaborator has limited access to the shared application and infrastructure without jeopardizing others' privacy.

6.2 Vision

ACE demonstrates the potential in supporting closed-loop DevOps of ECCI applications. ACE manages to facilitate the cost-efficient development and deployment of ECCI applications effectively. Taking a step further, we believe it is viable to integrate proper operation and maintenance modules into ACE, aiming at the close loop of continuous ECCI application development, deployment, monitoring, delivering, and testing. Such full DevOps supports will enable ACE to act as the foundation of the approaching ECCI ecosystem.

ACE is promising in promoting a broad spectrum of production level ECCI applications. ECCI applications, especially high-performing ones, are difficult to design, develop, and deploy, which hinders such a paradigm from contributing to the rapidly expanding IA market. ACE manages to provide supports along the entire ECCI application lifecycle, facilitating general users to conduct unified and user-friendly application development and deployment. Besides, ACE can also ease the migration of existing IAs based on CI and EI to ECCI applications satisfying specific practical requirements.

7 CONCLUSION

ML/DL-based IAs with harsher practical requirements cast challenges on conventional CI implementations. The emerging ECCI paradigm can support proliferating IAs that, however, are currently developed and deployed individually without generality. We envision systematic designs of a unified platform for cost-efficient development and deployment of high-performing ECCI applications, guiding us to construct the ACE platform handling heterogeneous resources and IA workloads. Our initial experience shows that ACE manages to help developers and operators along the entire lifecycle of ECCI applications, where customizable optimizations can be conducted efficiently. Further research is still required, and we discuss both the challenges and visions of the newborn ACE.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFA0713900; the National Natural Science Foundation of China under Grants 61772410, 61802298, 62172329, U1811461, U21A6005, 11690011; the China Postdoctoral Science Foundation under Grants 2020T130513, 2019M663726; and the Alan Turing Institute.

REFERENCES

[1] Tarek F. Abdelzaher, Yifan Hao, Kasthuri Jayarajah, Archan Misra, Per Skarin, Shuochao Yao, Dulanga Weerakoon, and Karl-Erik Årzén. 2020. Five Challenges in Cloud-enabled Intelligence and Control. ACM Trans. Internet Tech. 20, 1 (2020), 3:1–3:19.

[2] Saeed Ranjbar Alvar and Ivan V. Bajic. 2021. Pareto-Optimal Bit Allocation for Collaborative Intelligence. IEEE Trans. Image Process. 30 (2021), 3348–3361.

[3] Apple. 2019. Private Federated Learning. https://neurips.cc/ExpoConferences/2019/schedule?talk_id=40.

[4] Saurabh Bagchi, Muhammad-Bilal Siddiqui, Paul Wood, and Heng Zhang. 2019. Dependability in Edge Computing. Commun. ACM 63, 1 (2019), 58–66.

[5] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. 2020. Toward ML-Centric Cloud Platforms. Commun. ACM 63, 2 (2020), 50–59.

[6] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The Rise of Serverless Computing. Commun. ACM 62, 12 (2019), 44–54.

[7] Hyomin Choi and Ivan V. Bajic. 2018. Deep Feature Compression for Collaborative Object Detection. In Proc. of IEEE ICIP. 3743–3747.

[8] Jae-Won Chung, Jae-Yun Kim, and Soo-Mook Moon. 2020. ShadowTutor: Distributed Partial Distillation for Mobile Video DNN Inference. In Proc. of ACM ICPP. 8:1–8:11.

[9] Dapr Community. 2020. Dapr. https://dapr.io/

[10] Kubernetes Community. 2020. Kubernetes. https://kubernetes.io/

[11] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2021. JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services. IEEE Trans. Mob. Comput. 20, 2 (2021), 565–576.

[12] Yu Gong, Ziwen Jiang, Yufei Feng, Binbin Hu, Kaiqi Zhao, Qingwen Liu, and Wenwu Ou. 2020. EdgeRec: Recommender System on Edge in Mobile Taobao. In Proc. of ACM CIKM. 2477–2484.

[13] Google. 2017. Federated Learning Collaborative. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[14] Cyril Goutte and Éric Gaussier. 2005. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In Proc. of Springer ECIR, Vol. 3408. Springer, 345–359.

[15] Samuel Greengard. 2020. AI on Edge. Commun. ACM 63, 9 (2020), 18–20.

[16] Yotam Harchol, Aisha Mushtaq, Vivian Fang, James McCauley, Aurojit Panda, and Scott Shenker. 2019. Making Edge-Computing Resilient. Master's thesis. EECS Department, University of California, Berkeley.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In Proc. of IEEE CVPR. 770–778.

[18] Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. 2020. HOLMES: Health OnLine Model Ensemble Serving for Deep Learning Models in Intensive Care Units. In Proc. of ACM KDD. 1614–1624.

[19] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. 2020. CLIO: Enabling Automatic Compilation of Deep Learning Pipelines across IoT and Cloud. In Proc. of ACM Mobicom. 58:1–58:12.

[20] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In Proc. of IEEE/ACM SEC. 115–131.

[21] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor N. Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In Proceedings of ACM ASPLOS. 615–629.

[22] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. 2020. Learning to Simulate Dynamic Environments with GameGAN. In Proc. of IEEE CVPR. 1231–1240.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM 60, 6 (2017), 84–90.

[24] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud. In Proc. of ACM MobiCom. 37:1–37:15.

[25] Shijing Li and Tian Lan. 2020. HotDedup: Managing Hot Data Storage at Network Edge through Optimal Distributed Deduplication. In Proc. of IEEE INFOCOM. 247–256.

[26] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In Proc. of ACM SIGCOMM. 359–376.

[27] Roger Light. 2020. Mosquitto man page. https://mosquitto.org/man/mosquitto-8.html

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and Larry Zitnick. 2014. Microsoft COCO: Common Objects in Context. In Proc. of ECCV.

[29] Ruben Mayer and Hans-Arno Jacobsen. 2020. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. ACM Comput. Surv. 53, 1 (2020), 3:1–3:37.

[30] Microsoft. 2020. Azure IoT Hub. https://azure.microsoft.com/en-us/services/iot-hub

[31] Sumit Kumar Monga, Sheshadri K. R, and Yogesh Simmhan. 2019. ElfStore: A Resilient Data Storage Service for Federated Edge and Fog Resources. In Proc. of IEEE ICWS. 336–345.

[32] Vinod Nigade, Lin Wang, and Henri Bal. 2020. Clownfish: Edge and Cloud Symbiosis for Video Stream Analytics. In Proc. of ACM/IEEE SEC.

[33] Shadi A. Noghabi, John Kolb, Peter Bodik, and Eduardo Cuervo. 2018. Steel: Simplified Development and Deployment of Edge-Cloud Applications. In Proc. of USENIX HotCloud. 1–7.

[34] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2020. Enabling AI at the Edge with XNOR-Networks. Commun. ACM 63, 12 (2020), 83–90.

[35] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. CoRR abs/1804.02767 (2018). http://arxiv.org/abs/1804.02767

[36] Pei Ren, Xiuquan Qiao, Yakun Huang, Ling Liu, Schahram Dustdar, and Junliang Chen. 2020. Edge-Assisted Distributed DNN Collaborative Computing Approach for Mobile Web Augmented Reality in 5G Networks. IEEE Netw. 34, 2 (2020), 254–261.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. IJCV 115, 3 (2015), 211–252.

[38] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. CoRR abs/1801.04381 (2018). http://arxiv.org/abs/1801.04381

[39] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 7587 (2016), 484–489.

[40] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. 2018. In-Situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems. In Proc. of IEEE HPCA. 92–103.

[41] Mateen Ulhaq and Ivan V. Bajic. 2021. ColliFlow: A Library for Executing Collaborative Intelligence Graphs. https://yodaembedding.github.io/neurips-2020-demo/

[42] Luhui Wang. 2022. ACE-Evaluation. https://github.com/IoTDATALab/ACE-Evaluation

[43] Shibo Wang, Shusen Yang, and Cong Zhao. 2020. SurveilEdge: Real-time Video Query based on Collaborative Cloud-Edge Deep Learning. In Proc. of IEEE INFOCOM. 2519–2528.

[44] Zhumei Wang, Xing Su, and Zhiming Ding. 2020. Long-Term Traffic Prediction Based on LSTM Encoder-Decoder Architecture. IEEE Trans. Intell. Transp. Syst. (2020), 1–11.

[45] WeBank. 2019. WeBank and Swiss Re signed Cooperation MoU. https://www.fedai.org/news/webank-and-swiss-re-signed-cooperation-mou/.

[46] Youtube. 2019. Youtube Live. https://www.youtube.com/live

[47] Daniel Zhang, Nathan Vance, Yang Zhang, Md. Tahmid Rashid, and Dong Wang. 2019. EdgeBatch: Towards AI-Empowered Optimal Task Batching in Intelligent Edge Systems. In Proc. of IEEE RTSS. 366–379.

[48] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proc. IEEE 107, 8 (2019), 1738–1762.