Towards a Serverless Platform for Edge AI

Thomas Rausch
TU Wien

Waldemar Hummer
IBM Research AI

Vinod Muthusamy
IBM Research AI

Alexander Rashed
TU Wien

Schahram Dustdar
TU Wien

Abstract

This paper proposes a serverless platform for building and operating edge AI applications. We analyze edge AI use cases to illustrate the challenges in building and operating AI applications in edge cloud scenarios. By elevating concepts from AI lifecycle management into the established serverless model, we enable easy development of edge AI workflow functions. We take a deviceless approach, i.e., we treat edge resources transparently as cluster resources, but give developers fine-grained control over scheduling constraints. Furthermore, we demonstrate the limitations of current serverless function schedulers, and present the current state of our prototype.

1 Introduction

Edge AI is one of the major emerging technology trends [19], driven by recent advancements in AI algorithms, edge computing systems, and miniaturized AI accelerators [21]. Despite the growing demand for low-latency and privacy-aware AI at the edge, developing, deploying and operating edge AI applications at scale is still hard and requires immense manual effort, which we attribute to the lack of platform support. Although cloud providers and researchers are developing comprehensive solutions for building and operating AI applications in the cloud [13], they currently fall short of providing programming models and tools that make it easy to seamlessly integrate edge resources into the end-to-end AI workflow.

In this paper, we present a serverless platform to build and operate edge AI applications in edge cloud systems. Based on scenarios adapted from real-world use cases, we first outline the challenges for building and operating AI applications in such edge cloud systems. We then present our approach for a serverless edge computing platform that provides appropriate support for defining AI workflow functions, programming models for synchronizing data and model artifacts between cloud and edge devices, as well as tools for serving, training, refining, and monitoring AI models at the edge. Specifically, our approach elevates concepts from AI lifecycle management into the established serverless programming model, and thereby enables straight-forward development of end-to-end edge AI pipelines. We take a deviceless approach [11], i.e., we treat edge resources transparently as cluster resources, but give developers fine-grained control over scheduling constraints via additional language constructs. Through experimental analysis we uncover some of the major limitations of current state-of-the-art serverless platforms for supporting edge AI applications, and then present our approach for building a prototype of our platform.

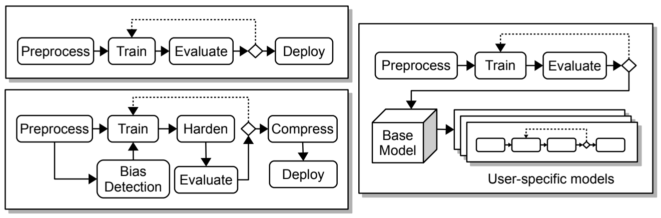


Figure 1: Prototypical AI pipelines [13]: (1) simple model training flow, (2) extended pipeline with custom steps, (3) hierarchical pipelines with transfer learning.

In summary, the contributions of this paper are: (i) a requirements elicitation for edge AI systems from existing real-world use cases, (ii) a design for a serverless platform that treats concepts from edge AI workflows as first-class citizens, both in the programming model and in its architecture, and (iii) a preliminary analysis that demonstrates the limitations of current state-of-the-art technologies to support our approach.

2 Edge AI Workflow Challenges

Operationalizing the AI lifecycle, from data preprocessing, to model training, to model validation, serving and runtime monitoring, is one of the major efforts in the current AI systems space, as made evident by recent developments of production-grade AI platforms such IBM AI OpenScale, TensorFlow Extended [5], or ModelOps [13]. AI pipelines are still not as well understood as traditional software pipelines, but in our previous work we have identified several prototypical pipelines from use cases at IBM, shown in Figure 1. However, platforms focus mostly on cloud-based workflows, or only treat edge resources as inferencing devices [21]. We motivate the need for an edge AI platform that includes edge resource in the entire AI lifecycle with additional examples and then highlight the main challenges for realizing such a platform.

2.1 Example Use Cases

In the following we discuss example scenarios, adapted from real-world use cases that currently use cloud-based AI platforms to deliver the applications. We introduce plausible new requirements to each use case that blend characteristics from AI operationalization and edge computing, to highlight the challenges of developing and operating edge AI applications.

Personal Assistants: Cognitive mobile personal assistants continuously monitor health data via bio sensors, and can predict and raise alerts for critical situation like critically low blood sugar levels [7]. Key concerns in this use case are prediction accuracy, inferencing latency, and data privacy, which can be improved by integrating patients' edge devices into the workflow. To optimize accuracy, the training process is split up into two phases: First, the service provider trains a base model on a representative, anonymized [4] sample of the entire population. This is a resource intensive and long-running process that is performed in a cloud environment. Second, in order to account for patient-specific patterns in the data, the individual models need to be adjusted and fine-tuned for each (type of) patient. However, these patient-specific data should be kept private and there should be no way to associate the data points directly with a patient's identity. The base model is transmitted to the edge device and refined using transfer learning techniques and data collected at runtime at the edge. This refined model is then served on the patient's device, thereby enabling low-latency and privacy-aware inferencing.

Field Technicians: Field service technicians often travel to on-site locations, including engineers maintaining power facilities, mechanics fixing industrial equipment, and technicians for an ISP. Mobile devices augmented with AI capabilities help to identify faulty parts, recommend diagnosis paths, or log and validate the technician's actions [8]. Key concerns in this use case are reliability, inferencing latency, bandwidth, and trust. Due to limited network connectivity, AI models need to run on the these edge devices, such as visual recognition models to classify photos taken in the field. Device equipped with AI accelerators could be used for video stream analytics where momentary information is relevant, for example in high-speed manufacturing lines. Being able to predict and preload AI models on devices with limited memory can be useful when managing fleets of devices. Context-aware policies can help facilitate a trusted AI workflow. For example, security constraints related to device ownership and registration may require that data gathered in the field only be transferred when connected to the corporate network.

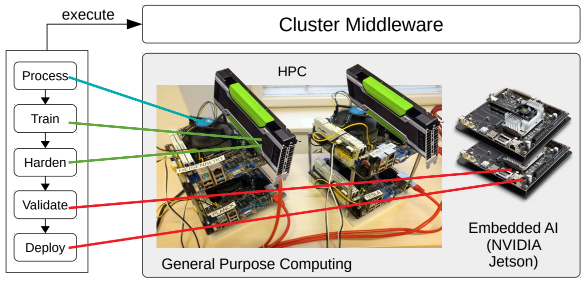


Figure 2: Efficient capability-aware execution of a complex AI pipeline on a multi-purpose edge computer

2.2 Requirements for Edge AI Platforms

Summarizing from the described use cases, we identify the following requirements and high-level design goals:

Operational abstractions: Orchestrating AI pipelines execution on edge resources is more involved than in cloud-based platforms as there are no well-defined APIs and, e.g., homogeneous dedicated learning clusters to submit training jobs to. Figure 2 illustrates how complex workloads are mapped to a multi-purpose edge-compute resources to execute efficiently and make full use of hardware capabilities. A programming model and APIs for edge AI applications needs to hide this operational complexity from the developer. In particular, programmers should not have to worry about the distribution of data and the heterogeneous capabilities of edge resources. The *stickiness* of tasks and resources should be easily configurable, and a scheduler needs to infer scheduling constraints and goals from application and device contexts.

Context-awareness: Being aware of the context of an edge device is essential for enabling seamless end-to-end edge AI pipelines. From a service provider's perspective that manages fleets of edge devices and provides models across several domains, context-awareness can help manage operational complexity, by deploying models on demand to devices based on their context. Context-aware policies can also be used to specify retraining triggers, such as thresholds of new training data, or device battery and charging conditions.

Artifacts as first-class citizens: Locally trained models and data available on the edge magnifies the issues around model and data management. To facilitate this abstraction, our edge AI serverless platform treats models and data as first class citizens. This allows the platform to make decisions on where data can reside or be transferred, based on application contexts and device constraints, and enables the scheduler to satisfy data–function locality objectives.

Fine-grained policy control: Developers should be able to express the *context* in which functions are allowed to execute or data is transferred. However, this requires that the programming model and API are intuitive for developers, but expressive enough to guide the execution platform in its decision on how to schedule functions or replicate data.

3 Related Work

Serverless computing has been proposed in both, the AI and ML [6, 15], as well as edge computing [11, 18] problem space.

Ishakian et al. [15] discuss the advantages and open challenges of serving ML models using serverless functions. In our paper, we go beyond only serving models, and consider all steps in end-to-end AI pipelines. Carriera et al. [6] implement ML workflows in serverless platforms, and outline an approach that includes an API to develop serverless ML functions, stateful-client resource manager, a worker runtime, and distributed data store. Our approach extends this idea to include management of edge resources. We further consider the deployment, serving, and runtime monitoring of models.

Glikson et al. [11] propose an extension of the serverless paradigm for edge computing. Termed deviceless edge computing, devices execute functions are treated as transparently as possible, in the same way that cluster resources are treated in a cloud-based serverless scenario. Nastic et al. [18] build on this idea and propose a programming model for serverless analytics functions, an abstraction layer over edge resources, and a runtime mechanism to deploy, place, and schedule functions on this abstraction layer. In industry, platforms like AWS IoT Greengrass or Azure IoT Edge aim to provide fully-fledged edge computing platforms. Our approach is similar in terms of architecture, but different in that it considers AI workflow concepts as first-class citizens in the programming model and underlying runtime, which affects the overall design.

4 A Serverless Platform for Edge AI

4.1 Programming Model

In serverless computing, the programming model concepts are *functions*, *events*, and *triggers*. The serverless platform executes functions in response to events. Which events lead to a function execution is defined by a trigger. We introduce AI workflow specific extensions to the serverless programming model, by elevating concepts of the AI workflow to first-class citizens in the model to provide common abstractions that make developing edge AI workflow functions easier. Specifically, we add (a) the notion of *artifacts*: data artifacts such as training data sets, and model artifacts, i.e., machine learning models, (b) quality *gates*: AI workflow specific function triggers such as model validators or drift detectors, and (c) *policies*: fine-grained control mechanisms for function scheduling. For workflow *composition*, we rely on systems like ModelOps [13] or OpenWhisk composer [1], which offer different ways to compose AI workflows. Our examples are based on Python and the capabilities of the MXNet ML library [2], but the concepts could be applied to most languages.

Data & Models Current serverless platforms generally only allow JSON documents to be passed between functions. When dealing with large artifacts or streaming data, which is common in AI workflows, developers are required to manually read and write from cloud storage services. In particular, it is common for functions to consume or produce data or model artifacts. Our approach provides ways to annotate such functions, and provides a data API that hides platform data management tasks from the developer. Additional metadata from the annotations allow the platform to transparently handle data transfer and storage and respect data locality policies.

Model Selectors: We allow the injection of models into functions based on *selectors*. This allows developers to define the requirements of a functions without specifying the exact model instance to use. For example, as shown in Listing 1 a selector can specify the type of model (regressor, or classifier), the type of data it was trained on, the type of data it processes, or model performance metrics such as the accuracy or robustness scores [24]. The model metadata collected by the AI lifecycle engine ModelOps [13] facilitates this.

```
@consumes.model(selector={'type': 'image_classifier',
    'data_tags': ['machine_x'], 'accuracy': '>=0.88'}})
def inference(model, request):
  # data prep tasks
  return model.estimate(data)
```

Listing 1: Artifact injection via model selector

Gates: AI pipelines need to express conditions on model validation metrics such as bias or vulnerability validators. Similarly, inferencing functions may be conditionally enabled based on runtime metrics such as concept drift detectors [10]. Some examples of conditional gates are presented in Listing 2. Gates target a specific data or model artifact, and enable the explicit monitoring of certain metrics in the runtime of the respective artifact.

```
@gate.bias(attribute = 'age', predicate = '<0.8')
@gate.drift(metric = 'confidence', predicate = '<0.2')
```

Listing 2: Conditional gates

Policies: Policies allow developers to define additional function execution conditions. The *deadline* policy tells the scheduler how quickly a function should get executed, which is useful for inferencing functions that have low-latency requirements. The runtime can factor in latency incurred by data transfer. A *fn* policy defines properties that a node should or must have for a function to be scheduled on that node. The *data* policy is similar to a role, and defines constraints on data transfer. The *strict* keyword makes the policy a hard constraint. For example, the function may only be executed when the data is accessible from within the given network.

```
@policy.deadline('2s')
@policy.fn(node = 'user_device', capability = 'gpu')
@policy.data(network=['company_network'], strict=True)
```

Listing 3: Different policy annotation

Example Function: A common scenario in the use cases we described are user or device-specific models refined from a base model using device-local data. Listing 4 shows how such a function can be defined using our programming model. To satisfy the constraints, the scheduler runs this function only a user's device (given by the `usr` variable), and when data is available within the specified network.

```
@consumes.model(selector={'urn': 'model:base'})
@consumes.data(batch = 100, selector=...)
@produces.model(type='regressor', urn='model:user:{usr}')
@policy.fn(node = 'local')
@policy.data(network = 'local', strict=True)
def refine(model: ModelArtifact, data: DataArtifact):
  ndarr = data.to_ndarray() # data artifact API
  # transfer learning code
  return refined_model
```

Listing 4: Example function with several constraints

4.2 Execution Platform

By building on an existing serverless platform such as OpenWhisk or Knative, and the AI workflow execution engine ModelOps, we can focus on the necessary extensions to support our programming model. In particular, we leverage the container orchestration system underlying the serverless platform, and AI lifecycle metadata (such as model performance metrics) gathered by ModelOps. We outline the additional components and mechanisms in the remainder of this section.

Function Scheduler: Efficient and effective scheduling of functions to both edge and cloud resources is at the core of our approach. It goes beyond state-of-the art serverless scheduling for cloud-based clusters in that it considers device capabilities, inter-node proximity (e.g., latency between nodes), and data locality (e.g., to enforce privacy constraints or trade off data transfer costs). Figure 3 shows how the scheduler would place a function in one of our examples. Furthermore, to be feasible for the use in a provider setting, the scheduler should scale to handle a high number of multi-tenant edge cloud clusters. The core instance of the scheduler runs in the cloud to handle most requests, but should scale to the edge when necessary, e.g., for large tenant-specific cluster configurations.

Function Preprocessor: The function preprocessor compiles the annotations of a function into soft and hard constraints used as input for the scheduler. A complex composition of policies can potentially generate many constraints that present a serious scalability challenge current for state-of-the-art monolithic schedulers, as we will show in Section 5. Furthermore, the preprocessor interprets the use of variables, and creates functions instances as necessary. For instance, in the example Listing 4, the use of `{usr}` in the `@produces.model` annotation together with the `'local'` function policy, will lead to a fan-out to all user specific devices on record.

Locality-Aware Data Management: Our platform provides a data API that hides platform data management from the user. Data and model artifacts are dynamic proxies that resolve data at runtime using metadata from the annotations

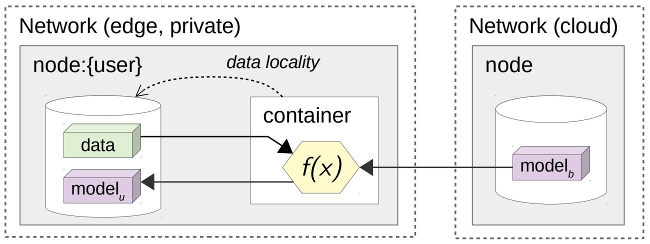


Figure 3: Data locality aware execution of Listing 4

and network topology information gathered by the underlying container orchestration platform. Data is resolved either in batches from local caches replicated from cloud-data stores, or streamed over the network. When functions return an artifact, the runtime finds an appropriate data store that respects data locality and other policies.

5 Prototype & Preliminary Evaluation

This section reports our insights developing a prototype of the system described in Section 4. In particular, we outline the limitations of current state-of-the-art technologies.

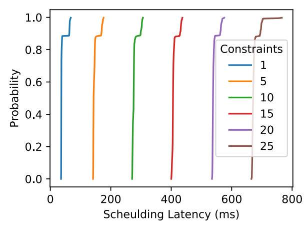
5.1 Testbed & Simulator

We extended the cluster-based edge computer testbed presented in [20], parts of which are shown in Figure 2, adding several Raspberry Pi 3 Model B+ clusters with AI accelerators, and several NVIDIA Jetson TX2 modules. We have distributed clusters across different locations on our university campus, and connected them via the on-premises OpenStack cloud of the CPS/IoT Ecosystem project [14]. Furthermore, we are extending the AI pipeline execution simulator we built for ModelOps to support heterogeneous edge cloud setups. Data from the testbed will serve as simulation parameters.

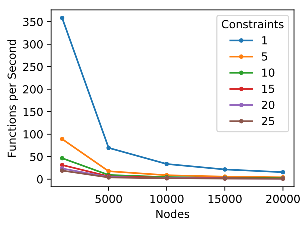
5.2 Prototype

Most serverless technologies such as OpenWhisk, Knative, or OpenFaas, use containers and in particular Kubernetes as an execution platform. We therefore found it useful to explore Kubernetes as a runtime environment for our platform and extend it where necessary. Also, we observe that Kubernetes is increasingly becoming a universal resource scheduler [17].

Constraint Compiler & Function Scheduling: Functions are typically executed in Kubernetes *pods* (the atomic deployable compute units), which are assigned a cluster node at runtime by the scheduler. The scheduler first selects a set of feasible nodes by evaluating *predicate functions*, i.e., hard constraints, for each node, such as exceeding resource limits or required node labels. Second, all feasible nodes are scored by *priority functions*, i.e., soft constraints.The default priority functions score nodes by, e.g., their resource utilization or container image locality. Finally the highest scoring node is



(a) Scheduling delay (Borg)



(b) Scheduling throughput

Figure 4: Experiment results showing (a) CDFs of scheduling delay with the Borg cluster setup, and (b) scheduling throughput for different cluster sizes

selected for provisioning. This approach lends itself well to our programming model where we generate hard and soft constraints from the function annotations. Several constraints from Kubernetes can be re-used for our approach, such as *image locality* that favors nodes where a specific container image is already available, or *taints and tolerations* which allow a pod to specify node preferences in the form of key value pairs. We are developing additional mechanisms for, e.g., considering inter-node proximity and data locality.

5.3 Limitations and Engineering Challenges

Scalable Function Scheduling: We doubt that the monolithic design of Kubernetes will work well for edge computing providers, where a large number of tenants consolidate edge devices to form huge cluster configurations. To assess the task placement latency of the Kubernetes default scheduler, we reconstructed its fundamental parts in Python using *SimPy* [16]. Our implementation is open source [3] and we plan to extend it to provide a Python-based alternative to the default Go implementation. So far we have implemented the queuing mechanism, as well as the main control loop covering node selection, predicate, and priority function execution. We ran two experiments with varying number of scoring functions (soft constraints), which we found to be the largest influence factor on scheduling latency. First, we synthesized a cluster configuration from the Google Borg Cluster trace [22] containing 12583 machines with different characteristics. Second, we evaluated the raw scheduling throughput given different number of nodes. Figure 4 shows the results, which roughly match those of a recent Kubernetes performance evaluation [9].

The results show that the scheduling throughput drastically drops with the number of nodes and soft constraints. Even with 5000 nodes and fewer than 10 constraints, the scheduler struggles to process more than 10-15 functions per second. As our approach potentially generates a large number of constraints for each function, and multi-tenant edge/cloud platforms may include huge node populations, the scheduler must maintain a high throughput under these conditions.

The results highlight the limitations of Kubernetes' queue-based monolithic scheduler. We are hence exploring alternative architectures. Omega [23], a disaggregated shared-state scheduler, has been designed to cope with Google's real-life production workload. Firmament [12] promises to accomplish sub-second latencies for placing more than ten thousand machines by using multiple min-cost max-flow (MCMF) optimizations. Combining these models could be a promising approach for operating edge cloud systems in general.

Hybrid Edge/Cloud Orchestration: Kubernetes manages its own network once set up, but requires all nodes to be publicly addressable. This is a reasonable approach if all cluster nodes are confined in a single data center. However, managing and orchestrating edge resources from the cloud is more complex as nodes are typically behind private networks or firewalls. There is currently no out-of-the-box solution that addresses this issue. Further investigation is needed on how to transparently consolidate cloud and edge resources in a scalable way (without point-to-point integrations like VPNs).

Supporting Heterogeneous Architectures: Clusters with heterogeneous architectures are not a common use case yet, which is why Docker images are typically only available on x86 CPU architectures. Many OpenWhisk components, for example, have no official multi-arch images, requiring significant effort to run on ARM architectures. Furthermore, with the increasing support for additional architectures and growing ecosystem of machine learning libraries, there will be a considerable amount of function images to manage. Edge nodes with limited storage capabilities necessitate intelligent eviction strategies, further increasing operational complexity.

6 Conclusion

It is clear that edge computing will play a critical role in the future of AI applications. However the added complexity from AI workflows and edge AI requirements make it difficult to program and operate such applications. Also, providers are challenged to scale their platforms to a large number of tenants with diverse edge cloud configurations. We proposed a serverless platform that elevates concepts from the AI workflow to first-class citizens, and provides a more approachable way to develop and operate edge AI functions. Our deviceless function scheduling approach respects device capabilities (such as added AI accelerators), and data locality, and can thereby hide complex operational data and model management tasks from the developer. The scheduling approach considers both edge and cloud resources for function execution, and places functions according to contextual policies. In an initial technology evaluation, we found that the main limitations of state-of-the-art serverless scheduling approaches are low throughput when functions have many soft constraints, and the lack of a notion of node proximity in the platforms. Disaggregated schedulers appear to be a promising alternative that we will investigate in future iterations of our system.

Discussion Topics

In keeping with the workshop format, in this section we discuss a) what kind of feedback we are looking to receive b) the controversial points of the paper c) the type of discussion this paper is likely to generate in a workshop format d) the open issues the paper does not address, and e) under what circumstances the whole idea might fall apart

a) We are looking forward to hearing more opinions from edge computing and AI practitioners whether the programming model has the correct level of abstraction, and what could be added or what should be rethought. Furthermore, we hope to learn what evaluation characteristics other systems researchers view as relevant for the platform we are proposing.

b) The additional AI workflow abstraction make the programming model very high-level, and it is possible that it is too inflexible for what we are aiming at. Furthermore, it is still unclear whether the *deviceless* approach (serverless edge computing where devices are treated completely transparently) really works for edge computing applications. We make several assumptions that it does (see Topic d).

c) We hope to spark a broader discussion over i) the usefulness of serverless edge computing and whether it is worth investigating further, in particular intelligent scheduling of workloads to resources with specific capabilities transparently from the user (i.e., what has been termed deviceless computing); and ii) current opportunities to generalize edge AI workflows to a degree that the community can begin developing well-defined methodologies around them (similar to DevOps workflows).

d) A key requirement for fully realizing deviceless programming that has been largely unaddressed in general is transparent distributed data management at the edge. In particular, it is unclear how data replication and consistency issues will be solved. We have yet to look into approaches for solving this, but in this paper we make the assumption that there is one. We hope that, as more edge AI use cases appear, the requirements for data management will become clearer.

e) Our idea builds on the premise that the current "bring your own device" approach of large cloud vendors, where users connect their edge devices with a cloud platform (such as AWS Greengrass, or Google IoT Edge), will prevail. Huge cluster configurations and the need for high function scheduling throughput will be relevant mostly in such a scenario. Furthermore, it is possible that the programming model is too high-level for a translation engine to create good soft and hard constraints.

References

[1] Apache OpenWhisk Composer. `https://github.com/apache/incubator-openwhisk-composer`.

[2] MXNet: a scalable deep learning framework. `https://mxnet.apache.org/`.

[3] sched-sim: serverless scheduler simulation. `https://git.dsg.tuwien.ac.at/serverless-edge-ai/sched-sim`.

[4] Charu C Aggarwal and S Yu Philip. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*, pages 11–52. Springer, 2008.

[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, and Others. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395. ACM, 2017.

[6] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. A Case for Serverless Machine Learning. *Workshop on Systems for ML and Open Source Software at NeurIPS 2018*, 2018.

[7] IBM Corporation. Medtronic builds a cognitive mobile personal assistant app to assist with daily diabetes management, 2017. IBM Case Studies.

[8] IBM Corporation. An AI-powered assistant for your field technician. Technical report, 2018.

[9] Hongchao Deng. Improving kubernetes scheduler performance, 2016. CoreOS Blog. Online. Posted 2016-02-22. Accessed 2019-03-14.

[10] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014.

[11] Alex Glikson, Stefan Nastic, and Schahram Dustdar. Deviceless edge computing: Extending serverless computing to the edge of the network. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, 2017.

[12] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 99–115, Savannah, GA, 2016. USENIX Association.

[13] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, and Kaoutar El Maghraoui. Modelops: Cloud-based lifecycle management for reliable and trusted ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E'19)*, Jun 2019.

[14] Haris Isakovic, Denise Ratasich, Christian Hirsch, Michael Platzer, Bernhard Wally, and Thomas Rausch et al. Cps/iot ecosystem: A platform for research and education. In *Proceedings of the 14th Workshop on Embedded and Cyber-Physical Systems Education (WESE 2018)*, 2018.

[15] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262, 2018.

[16] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009):1–33, 2008.

[17] Janakiram MSV. How kubernetes is transforming into a universal scheduler, 2018. The New Stack. Online. Posted 2018-09-07. Accessed 2019-03-14.

[18] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing*, 21(4):64–71, 2017.

[19] Kasey Panetta. 5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018. *Gartner*, 2018.

[20] Thomas Rausch, Cosmin Avasalcai, and Schahram Dustdar. Portable energy-aware cluster-based edge computers. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 260–272, 2018.

[21] Thomas Rausch and Schahram Dustdar. Edge intelligence: The convergence of humans, things, and ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E'19)*, Jun 2019.

[22] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[23] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.

[24] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. *arXiv preprint*, page arXiv:1801.10578, jan 2018.