# Creek: Leveraging Serverless for Online Machine Learning on Streaming Data

Nazmul Takbir*[a], Tahmeed Tarek*[b] and Muhammad Abdullah Adnan[c]
*Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh*

Keywords: Online Machine Learning, Machine Learning, Serverless Computing, Stream Analytics.

Abstract: Recently, researchers have seen promising results in using serverless computing for real-time machine learning inference tasks. Several researchers have also used serverless for machine learning training and compared it against VM-based (virtual machine) training. However, most of these approaches, which assumed traditional offline machine learning, did not find serverless to be particularly useful for model training. In our work, we take a different approach; we explore online machine learning. The incremental nature of training online machine learning models allows better utilization of the elastic scaling and consumption-based pricing offered by serverless. Hence, we introduce Creek, a proof-of-concept system for training online machine learning models on streaming data using serverless. We explore architectural variants of Creek on AWS and compare them in terms of monetary cost and training latency. We also compare Creek against VM-based training and identify the factors influencing the choice between a serverless and VM-based solution. We explore model parallelism and introduce a usage-based dynamic memory allocation of serverless functions to reduce costs. Our results indicate that serverless training is cheaper than VM-based training when the streaming rate is sporadic and unpredictable. Furthermore, parallel training using serverless can significantly reduce training latency for models with low communication overhead.

## 1 INTRODUCTION

Serverless computing is a cloud computing model that has garnered a lot of interest recently due to its advantageous features, including consumption-based pricing, elastic scalability for fluctuating workloads, and low startup times. Leveraging these features, researchers have effectively employed serverless in a wide range of applications, including video processing (Ao et al., 2018), data science workflows (Patel et al., 2022), data analytics (Müller et al., 2020), machine learning inference (Ishakian et al., 2018; Yu et al., 2021), and machine learning training (Carreira et al., 2018; Wang et al., 2019; Jiang et al., 2021).

However, serverless has many drawbacks (Hellerstein et al., 2018; Jonas et al., 2017), such as limited execution times, a stateless nature, a data-shipping architecture, and a lack of direct communication between serverless functions. Serverless solutions are suboptimal for scenarios with long-running stable workloads, where VM solutions are more cost-effective, or for tasks demanding substantial computational power due to serverless's limited RAM and lack of GPU support (Hellerstein et al., 2018; Jiang et al., 2021).

Several researchers have explored the usage of serverless in training ML models (Carreira et al., 2018; Wang et al., 2019; Jiang et al., 2021). Yet, given that training ML models, particularly deep learning ones, is a resource-intensive, long-running task that iterates over the same dataset multiple times with predictable workloads, serverless has not had much success in this application. Training ML models using serverless is often slower and costlier than traditional methods using dedicated virtual machines (VMs) (Jiang et al., 2021; Hellerstein et al., 2018). In contrast, serverless has shown great potential for providing ML model inference services (Ishakian et al., 2018; Yu et al., 2021). This is because inference tasks generally demand fewer resources compared to training, exhibit dynamic workloads that are dependent on real-time user requests, and have a more straightforward data flow due to the non-iterative nature of the process.

[a] https://orcid.org/0009-0005-7440-0407
[b] https://orcid.org/0009-0007-8621-7460
[c] https://orcid.org/0000-0003-3219-9053
*These authors contributed equally to this work.

Takbir, N., Tarek, T. and Adnan, M.
Creek: Leveraging Serverless for Online Machine Learning on Streaming Data.
DOI: 10.5220/0012619100003711
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 38-49
ISBN: 978-989-758-701-6; ISSN: 2184-5042
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

With the aforementioned observations in mind, we investigate the use of serverless for training Online Machine Learning (Online ML) models on real-time streaming data. Online ML (Benczúr et al., 2018) is an approach to machine learning where models are updated incrementally in real-time, adapting to continuously streaming training data. This online incremental training differs from traditional offline batch training. In fact, training an online ML model is more akin to the inference tasks in traditional ML: it has unpredictable workloads because training occurs on real-time streaming data; the models are typically lightweight with relatively lower computational demands; and the data flow is simpler since each data point is used for training just once without any iterations. These characteristics suggest that a serverless solution might be an ideal choice for training online ML models. Hence, we introduce Creek, a proof-of-concept system that utilizes serverless to train online ML models on streaming data. Through Creek, we identify the factors that determine the suitability of serverless for training online ML models: data streaming rates, model complexity, model parallelizability, and data dimensionality. We also compare Creek with traditional VM-based solutions in terms of cost and training latency to identify when the serverless solution is superior.

The architecture of Creek has four components: the data ingestor, trainer, invoker, and model store. We identify the requirements for each component and explore the various options available on AWS for their implementation, weighing their respective advantages and disadvantages in Section 3.2. Furthermore, we detail how an automated end-to-end training pipeline can be set up on AWS using these components. Depending on the specific implementations chosen, there can be multiple variants of this pipeline. We highlight and analyze three such variants: Push-based Kinesis, Push-based SQS, and Pull-based SQS in Section 3.3.

In our experiments, we also vary the online ML models used, the data streaming rate, and the data dimensionality. For each scenario, we report the training cost and latency. Additionally, we explore how the unique characteristics of a model can influence the suitability of a serverless solution. Our experiments employ two different online ML models: Adaptive Random Forest (ARF) (Gomes et al., 2017) and Continuously Adaptive Neural Networks for Data Streams (CAND) (Gunasekara et al., 2022). The primary objective of our experiments is to conduct a comparative analysis between Creek and traditional VM-based solutions, evaluating the aptness of serverless for online ML training. Our contributions are summarized as follows:

1. We have implemented a system that trains online ML models using serverless while being cost-effective and having low training latency.
2. We have identified three variants of our system, compared them in terms of cost and latency, and found the Pull-based SQS variant to be optimal. It is 3x more cost-effective than Push-based Kinesis and offers at least 8x lower latency than Push-based SQS during data surges.
3. We studied the effect of data streaming rates and observed that at lower rates, a serverless solution saves at least 87% more than a VM-based solution for ARF and at least 17% for CAND.
4. We have investigated and found that it reduces latency (3x for ARF and 1.5x for CAND), but at a greater cost.
5. We have implemented dynamic memory allocation for Pull-based SQS, and it reduced costs by 80% when training ARF.

Our paper begins with a section that describes a motivating use-case scenario where Creek would excel. This is followed by the *System Design* section, listing Creek's components, describing the end-to-end pipeline's implementation, and elaborating on further optimizations like dynamic memory allocation and model parallelism. Then, in the *Experimental Setup* section, we specify the models, datasets, metrics, and streaming patterns used in our experiments. The *Experimental Results* section presents findings from our experiments with ARF and CAND. Lastly, the *Related Works* mentions notable previous studies in serverless computing and online ML and highlights our work's novelty.

## 2 MOTIVATION

Online ML excels in scenarios where the statistical nature of the training data changes frequently, and thus the ML model needs to be adaptable. Several industry giants have reported utilizing online ML: ByteDance for recommendation systems (Liu et al., 2022), Facebook, and Google for predicting clicks on ads (He et al., 2014; McMahan et al., 2013). However, continuous training of online ML models can be expensive and infeasible. Some challenges related to building systems for training online ML models are identified in (Huyen, 2022).

Yet smaller companies could benefit from online ML. Consider an e-commerce start-up making thousands of daily deliveries. Predicting delivery times

Table 1: Use Case Properties and Associated Challenges.

| Challenge | Description |
|---|---|
| Fluctuating data rates | Allocating resources based on peak data rates causes over-provisioning and extra costs, while using average rates risks inadequate handling of data surge. |
| Sequential training | When learning from a data stream, the order in which the model receives the training samples must be maintained in order to adapt to trends and concept drifts (Tsymbal, 2004). |
| Low latency training | The trainer must quickly update the model when concept drift alters the data stream's statistical distribution. Acceptable update latency varies from seconds to minutes, depending on the use case. |
| Synchronization with inference system | Decoupling training and inference systems enhances fault tolerance and specialization, but in continuous training, quickly updating the inference system with the latest model is challenging. |

relies on current conditions, such as supply chain disruptions, warehouse issues, unexpected weather, or traffic congestion. Online ML would be suitable for such a use case where real-time data takes precedence over historical data. Similar use cases include personalized product recommendations, time series predictions, and much more. Unfortunately, smaller companies can't afford the cost and complexity of building online ML systems.

This is where Creek has a promising solution. It provides a training system for online ML models that is cost-effective and scalable, thanks to features of serverless computing such as consumption-based pricing and elastic scalability. It is easy to set up since the serverless platform abstracts away all server management, and the developers only have to worry about the business logic. Thus, by minimizing the infrastructural cost and overhead, Creek makes online ML training more accessible and seamless to integrate.

## 3 SYSTEM DESIGN

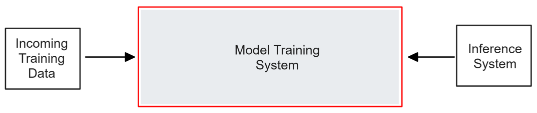


Figure 1: Overview of an Online Learning System.

In this section, we describe how Creek's architecture utilizes serverless computing for the training of online machine learning models on streaming data. Figure 1 gives a high-level overview of an online machine learning system. We focus on the model training system, whose goal is to update the model using new training data and make the updated model available for the inference system. To implement this *trainer* Creek uses a serverless-based solution, which has a potentially lower cost and training latency compared to a traditional dedicated VM-based solution. Besides the trainer, several other components are required to implement an end-to-end system that handles everything from the ingestion of new training data to post-training model availability for inference. We detail this end-to-end architecture in three steps: identifying challenges in designing a trainer online machine learning models, outlining the required components and their AWS implementation, and explaining how the components work together as a single system.

### 3.1 Challenges

Creek uses serverless computing to tackle the challenges of training online machine learning models with streaming data. This involves minimizing training latency and ensuring sequential data processing, while also adeptly managing fluctuating data rates and maintaining synchronization with the inference system. These challenges, detailed further in Table 1, are addressed by Creek in a cost-effective manner.

### 3.2 Components

Considering the challenges identified in Table 1, we propose Creek's architecture in Figure 2. Creek has four major components which work together to form an end-to-end pipeline for online training on streaming data.

To implement Creek, we used Amazon Web Services (AWS) since it is a mature cloud computing platform and allows easy integration between the components of Creek. However, Creek can be implemented using any other cloud platform.

Next, we discuss the components of Creek in detail.

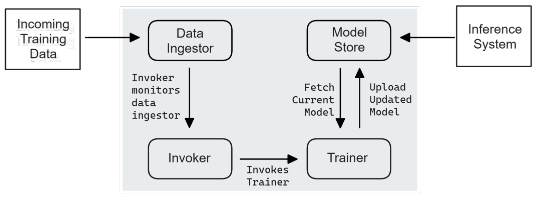


Figure 2: Main Components of an Online Model Training System.

#### 3.2.1 Data Ingestor

- Purpose: Collects training data from various sources, maintaining temporal order.
- Challenge: Handling fluctuating data streams efficiently. Traditional services like AWS MSK and Kinesis Data Streams provide high throughput but their fixed hourly costs are not economical for streams with variable rates and idle times.
- Implementation: Amazon SQS FIFO queues offer a cost-efficient solution with their usage-based pricing, reducing expenses during idle times and balancing cost with data handling efficiency.

#### 3.2.2 Trainer

- Purpose: Updates the model with data from the Ingestor using serverless functions.
- Challenge: Prior research suggests serverless trainers are ill-fitted for traditional ML, notably deep learning, because of no GPU support, statelessness, and brief execution times (Hellerstein et al., 2018; Jiang et al., 2021), resulting in increased costs and latency.
- Implementation: We use AWS Lambda. Despite the challenges, serverless trainers are effective for online machine learning because:
  - Online ML uses lightweight models that do not require GPUs for incremental training.
  - Unlike traditional training that iterates over the same dataset multiple times, online ML processes each data sample once, aligning with the stateless and ephemeral nature of serverless functions.
  - Serverless functions can scale to zero during idle periods, eliminating the idle costs associated with VM-based solutions.
  - Enables concurrent training across multiple functions, essential for quick response to data influxes.

#### 3.2.3 Model Store

- Purpose: Aims to persistently maintain the model and other stateful data, addressing the Trainer component's statelessness.
- Challenge: Ensuring efficient state management across training invocations without incurring excessive load times or data transfer costs.
- Implementation: We utilize AWS S3 for the Model Store, where the Trainer loads the model before training and saves it afterward.

#### 3.2.4 Invoker

- Purpose: Invokes the Trainer when new training data arrives.
- Challenge: Needs to handle both data surges and periods of low data rate.
- Implementation: Although the ingestor, such as AWS Kinesis and SQS, can serve as the invoker, we consider alternatives like AWS Cloudwatch Alarm to bypass SQS (FIFO) limitations and achieve more refined control over the invocation mechanism.

Next, we describe an end-to-end pipeline that seamlessly integrates all system components.

## 3.3 Pipeline

An end-to-end pipeline automates everything from data ingestion to making the updated model available for inference. Depending on the ingestor and invoker choices, there are three pipeline variants, as shown in Table 2. We will explore these variants, highlighting their pros and cons.

Table 2: Implementation Variants of End-To-End Pipeline.

| Ingestor | Invocation | Variant Name |
|---|---|---|
| Kinesis | Kinesis | Push-based Kinesis |
| SQS | SQS | Push-based SQS |
| SQS | Cloudwatch | Pull-based SQS |

1. **Push-based Kinesis:** AWS Kinesis directly invokes the Lambda trainer with a batch of training data. The trainer retrieves the model from S3, trains it incrementally, and uploads it back to S3. If more data remains in Kinesis, another trainer is triggered. Multiple invocations happen sequentially.

   Each invocation can handle up to 10,000 data samples. However, we set a batch window time of five seconds such that an invocation occurs the time limit is reached, even if the batch is not full. This is because:

- While increasing batch window time reduces state management overhead, it increases training latency. Besides, if a higher latency is acceptable, an SQS-based solution is more effective anyway, as our results will illustrate.
- A smaller batch window means fewer data samples per invocation, increasing state management overhead.

When order across all data is required, we are restricted to the throughput of a single shard of Kinesis. However, our results show that at rates where serverless is more effective than VM-based solutions, this constraint is not an issue

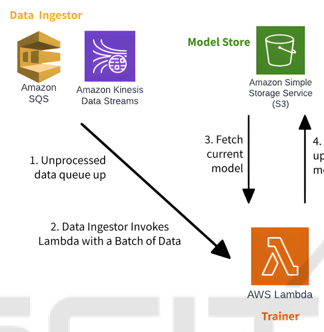

Data Ingestor
Amazon SQS
Amazon Kinesis Data Streams
Model Store
Amazon Simple Storage Service (S3)
1. Unprocessed data queue up
3. Fetch current model
4. Upload updated model
2. Data Ingestor Invokes Lambda with a Batch of Data
AWS Lambda
Trainer

Figure 3: Implementation of Push-Based Kinesis/SQS Pipeline.

2. **Push-based SQS:** SQS triggers Lambda trigger when new data arrives. This method is simple, responsive, and cheap, but it is limited by the FIFO queue's limitation of 10 data samples per invocation, making the system vulnerable to data surges.
3. **Pull-based SQS:** A Cloudwatch Alarm monitors the FIFO SQS queue's message count. When unprocessed data is detected, the Lambda trainer is triggered via AWS SNS. As shown in Figure 4, the trainer first disables the alarm to prevent concurrent invocations, ensuring consistent training. Then it retrieves and processes data from SQS in batches of 10 until the queue is empty. Finally, it reactivates the alarm and terminates.

   Unlike Push-based SQS, Pull-based SQS can process unlimited training samples per Lambda invocation, limited only by execution time. This reduces state management overhead, lowers costs, and provides greater robustness to data surges. However, Pull-based SQS has an average invocation delay of 30-seconds since the minimum trigger interval for Cloudwatch Alarm is one minute.

Some additional challenges need to be addressed:

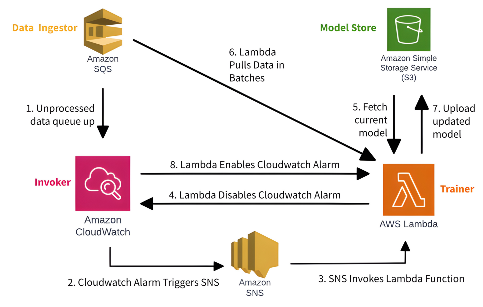

Data Ingestor
Amazon SQS
Model Store
Amazon Simple Storage Service (S3)
6. Lambda Pulls Data in Batches
1. Unprocessed data queue up
5. Fetch current model
7. Upload updated model
Invoker
8. Lambda Enables Cloudwatch Alarm
4. Lambda Disables Cloudwatch Alarm
Amazon CloudWatch
Trainer
AWS Lambda
2. Cloudwatch Alarm Triggers SNS
Amazon SNS
3. SNS Invokes Lambda Function

Figure 4: Implementation of Pull-Based SQS Pipeline.

- AWS Lambda's 15-minute limit requires careful batch size management for Push-based Kinesis and SQS to avoid exceeding this limit. For Pull-based SQS, training times over 14 minutes trigger an additional trainer.
- To prevent the S3 model from becoming outdated during lengthy training sessions, it is updated at predefined intervals.

## 3.4 Dynamic Memory Allocation

The Pull-based SQS pipeline can be optimized by utilizing the extra level of control on the invocation mechanism provided by Cloudwatch Alarm. Since the memory needed for the trainer Lambda depends on the model and data dimensionality, and can change over time, blindly allocating memory according to the maximum requirement is wasteful. Therefore, we use a dynamic memory allocation strategy, where a Lambda function adjusts memory for its subsequent invocations, ensuring cost-effective and usage-based resource allocation.

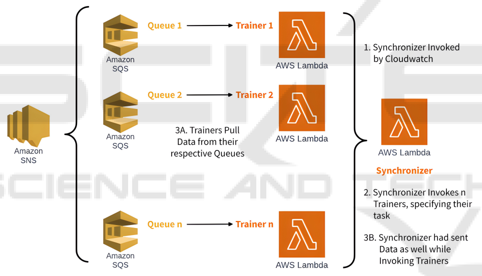
When training, if the trainer's memory usage exceeds the *max_threshold* of 70%, the training is halted. At this point, the trainer triggers another Lambda function. This function determines the future memory allocation for the trainer based on the maximum memory used during the previous three invocations of the trainer, multiplied by the *mem_scaling_factor* of 1.3. It then updates the trainer's memory, enables the Cloudwatch Alarm, and exits, allowing the training to resume.

On the other hand, if a trainer completes training with memory usage below a *min_threshold* of 50%, it invokes a Lambda function to adjust the memory allocation for the trainer in a similar manner, thus preventing resource wastage due to excessive memory allocation.

## 3.5 Parallelism

There are two main approaches for distributed machine learning training: data parallelism (Jiang et al., 2021), which partitions the dataset across nodes with each using the same model, and model parallelism, which splits the model across nodes, each training on the same dataset (Ooi et al., 2015).

Models chosen in our experiments favor model parallelism using concurrent Lambda invocations. As depicted in Figure 5, our system utilizes a *synchronizer* function and a *worker* function. The synchronizer invokes the required number of workers and assigns a model partition to each worker invocation. Each worker then pulls its partition from the model store and trains it using the data from the ingestor. Each worker can either independently fetch data from its own SQS FIFO queue in parallel (3A in Figure 5) or receive data from the synchronizer during invocation (3B in Figure 5). Since model loading, training, and saving are performed in parallel by the workers, training latency decreases. There is also no chance of data races as each worker updates a different model partition.


Amazon SNS
Amazon SQS
Queue 1
Trainer 1
AWS Lambda
Amazon SQS
Queue 2
Trainer 2
AWS Lambda
3A. Trainers Pull Data from their respective Queues
Amazon SQS
Queue n
Trainer n
AWS Lambda
1. Synchronizer Invoked by Cloudwatch
AWS Lambda
Synchronizer
2. Synchronizer Invokes n Trainers, specifying their task
3B. Synchronizer had sent Data as well while Invoking Trainers
Figure 5: Parallelization Implementations.

## 4 EXPERIMENTAL SETUP

### 4.1 Model Selection

We chose two different online machine learning models for evaluating Creek:

1. **Adaptive Random Forest (ARF)** (Gomes et al., 2017): ARF is an enhanced version of the Random Forest algorithm tailored for classification on evolving data streams. ARF updates the trees in an incremental manner, and it is designed to handle concept drift by replacing existing trees with new ones. These new trees are preemptively trained through ”background learning” when a potential future drift is detected.
2. **Continuously Adaptive Neural networks for Data streams (CAND)** (Gunasekara et al., 2022): It maintains a pool of neural networks with different sets of hyperparameter values. This ensures robustness against concept drifts. For each mini-batch, a subset of networks is selected for training. During predictions, CAND picks the network that is performing best in terms of training loss.

We use these models for the following reasons:

- Neural networks excel in traditional batch-learning scenarios and are now being explored for online ML scenarios. (Zheng and Wen, 2022; Gunasekara et al., 2022; Sahoo et al., 2017b).
- ARF is superior for low-dimensional data, whereas CAND excels with high-dimensional data (Gunasekara et al., 2022). So using them allows us to explore the effect of data dimensionality on Creek.
- We used ARF from River (Montiel et al., 2021), a Python library for online ML. On the other hand, CAND is available within MOA (Bifet et al., 2010) which was built using Java. This allowed us to evaluate Creek in two different development ecosystems.
- ARF and CAND support model parallelism, allowing us to investigate parallel training using concurrent serverless functions.

### 4.2 Datasets Used

We use two datasets in our experiments for the two different models:

- **Airlines Dataset:** The Airline on-time performance dataset (Expo, 2009) consists of US commercial flight details from October 1987 to April 2008. We used a variation of this dataset for our experiments with ARF, as used by the authors of ARF (Gomes et al., 2017), where the goal is to predict whether a given flight will be delayed or not: a classification problem.
- **Random Radial Basis Function Generator:** It is a synthetic data generator used in (Gunasekara et al., 2022) to evaluate CAND. Since the data is synthetic, we can control data dimensionality precisely. So we can explore the effect of data dimensionality on our system. We use a random RBF generator in our experiments with CAND.

### 4.3 Data Streaming Patterns

We based our experiments' data streaming rates on the data stream patterns of the Airlines dataset. Although the dataset does not provide exact timestamps

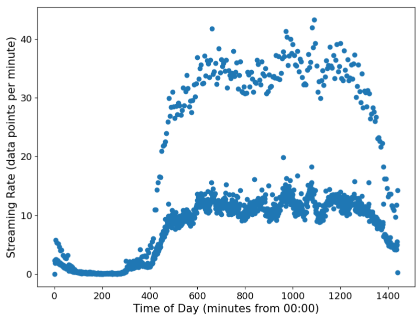


Figure 6: Minute-wise Data Rate (Airline Dataset).

for each data point, we estimated the timestamps using scheduled arrival times. Specifically, we assumed that flight delay status would be available at the scheduled arrival time. We then studied the variation in data production rate throughout the day. The results are shown in Figure 6. The rate of data samples for a particular minute of the day, denoted by $r_t$, is defined as:

$$r_t = \frac{\sum_{i=1}^{N} \mathbb{I}(a_{i,t} = 1)}{D} \quad (1)$$

where $N$ is the total number of data points. $\mathbb{I}$ is the indicator function with $a_{i,t}$ which is a binary variable indicating whether the flight corresponding to the $i$-th data point was scheduled to arrive at minute $t$. Finally, we divide by $D$, the total number of days in the dataset, to calculate the mean rate for a typical day. The results indicate a higher data streaming rate during the day than at night, with significant surges at specific times of the day. At night, there are also some idle periods with no data. The streaming rate peaks at 43.28 samples per minute, drops to a minimum of 0.005 samples per minute, and averages around 10 samples per minute during most of the day. Thus, the rate is variable, with occasional peaks and idle times.

## 4.4 Evaluation Metrics

We evaluated Creek using two metrics:

- **Monetary Cost:** The estimated training expense includes costs from various AWS services.
- **Training Latency:** The system's reaction time to new training data and involves everything from invoking the trainer upon data arrival to updating the model and making it ready for inference.

## 4.5 Experiment types

To evaluate Creek's performance, we conducted several experiments. These fall into two main categories:

1. **Single Trigger:** For these experiments, data is preloaded into the ingestor, and then the trainer is manually triggered. While not mimicking a production environment, these experiments allow us to evaluate individual components and system performance in extreme scenarios, like measuring latency during a data surge.
2. **Time Bound:** These experiments are run for a fixed duration during which we simulate a data stream by uploading samples with varying inter-arrival times generated from an exponential distribution. These experiments simulate the performance of a production system. The goal is to compare the cost-efficiency of serverless solutions against VM-based solutions to determine their feasibility and limitations.

## 5 EXPERIMENTAL RESULTS

We experimented with ARF and CAND to compare Creek's cost and training latency across serverless variants and against VM-based solutions.

### 5.1 Experiments with ARF

Our Adaptive Random Forest (ARF) model is configured with 100 base learners in accordance with the "immediate setting" described in the original paper for ARF (Gomes et al., 2017)

#### 5.1.1 Memory Requirements of ARF

The model size, and thus the memory requirement, of Adaptive Random Forest changes as trees in the forest grow and are replaced by new ones to handle concept drift. This is illustrated in Figure 7. For the VM-based solution, a T2-Medium EC2 instance with 4096 MB memory is sufficient. But for serverless, the Lambda functions are allocated 5120 MB because of some state management overhead in the Boto3 SDK of Python.

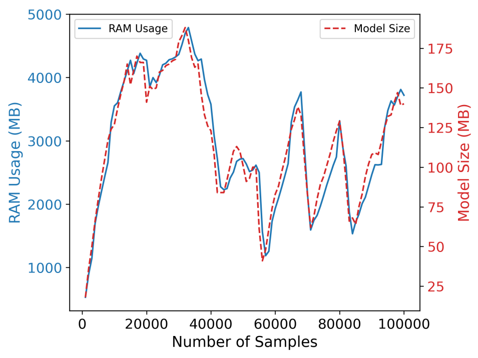


Figure 7: Model Size & Memory Usage of ARF.

### 5.1.2 Training Cost: Push-based-Kinesis vs Push-based-SQS vs Pull-based-SQS

In Figure 8, we have the monetary cost comparison of three implementation variants of our system against the log mean time gap for one hour time-bound experiments. In Table 3, we map the mean time gaps to log mean time gaps. In Figure 8, we observe that the cost decreases for all three variants as we increase the mean time gap, reflecting the consumption-based pricing nature of Lambda since a greater mean time gap means a lower data rate and a lower training requirement. However, we can see that the cost associated with Push-based Kinesis is always higher due to the fixed cost associated with using Kinesis.

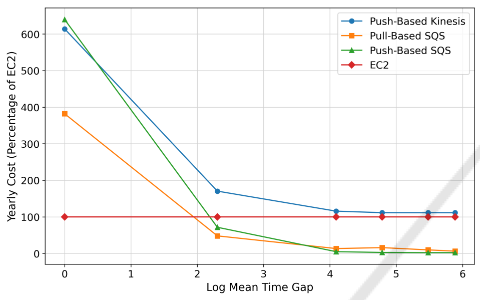


Figure 8: Training Cost of Three Pipeline Implementation Variants at Different Data Rates.

Table 3: Mean Time Gap to Log Mean Time Gap Mapping.

| Mean Time Gap | Natural Log of Mean Time Gap |
|---|---|
| 1 | 0 |
| 10 | 2.3 |
| 60 | 4.09 |
| 120 | 4.79 |
| 240 | 5.48 |
| 360 | 5.89 |

### 5.1.3 Training Latency: Push-Based-Kinesis vs Push-Based-SQS vs Pull-Based-SQS

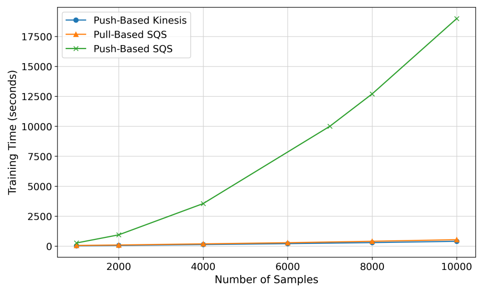


Figure 9: Training Latency of Three Pipeline Implementation Variants for Different Number of Samples.

In Figure 9 we can see how well the three different variants can handle an unexpected data surge. We can see that the Push-based SQS is far worse than the other two methods, with much higher training latency. This is because for Push-based SQS, since each Lambda invocation can train at most 10 data samples, frequent sequential invocations significantly waste time due to overhead.

### 5.1.4 Training Cost: Pull-Based-SQS vs EC2

Figure 10 shows the cost of training using Pull-based SQS variant as a percentage of the cost incurred when training using a dedicated T2-Medium EC2 instance. The experiments performed are one hour time-bound experiments. The percentage of cost compared to EC2 decreases as the log mean time gap increases; a lower data rate means the consumption-based pricing of Lambda incurs a lower cost.

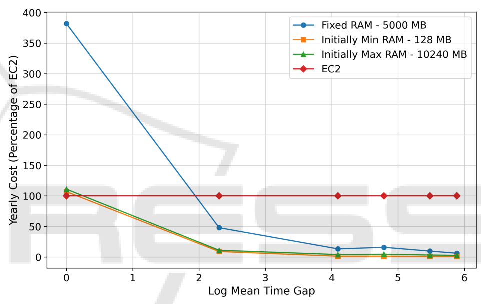


Figure 10: Training Cost of Pull-Based SQS With & Without Dynamic Memory Allocation as % of EC2 Cost.

Now, the performance of Pull-based SQS can be improved further with dynamic memory allocation. In Figure 10, we see that for each log mean time gap, the percentage cost is lower for dynamic memory allocation compared to that of fixed memory. The system self-adjusts to the optimal memory size to minimize cost, regardless of initial allocation.

### 5.1.5 Model Parallelism

In Figure 11, we can see that the Pull-based SQS method is slower than the EC2-based solution in handling unexpected data surges because it has a higher training latency. However, we can easily solve this by utilizing the fact that ARF is embarrassingly parallel and that serverless computing provides us with elastic horizontal scalability. To train ARF in parallel, the synchronizer function invokes a worker function for each tree in the forest, then each tree is trained independently in parallel. As shown in Figure 11, by training each tree in the forest in parallel, we can reduce the training latency.

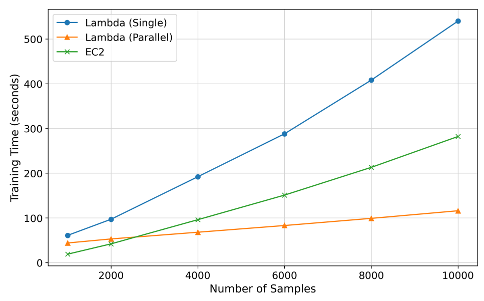


Figure 11: Training Latency of Lambda With & Without Parallelism Compared to EC2.

## 5.2 Experiments with CAND

Neural networks often work with high-dimensional data (a 224 x 224 image has 50,176 attributes) and have large model sizes. Recent works explore the use of concurrent serverless functions for partitioning large models for inferences (Yu et al., 2021). For Creek, we implement model parallelism by invoking a synchronizer for CAND and parallel worker functions for each MLP. We typically use a pool size of 10 MLPs for the experiments, along with a Pull-based SQS pipeline.

### 5.2.1 Revisiting Results from ARF

Firstly, we perform a one hour time-bound experiment to compare our serverless solution with a VM-based solution. From Figure 12 we observe the cost of our serverless solution decreases with the increase in mean time gap and for higher mean gaps the serverless solution is much cheaper than the VM-based solution. This corroborates the results for ARF.

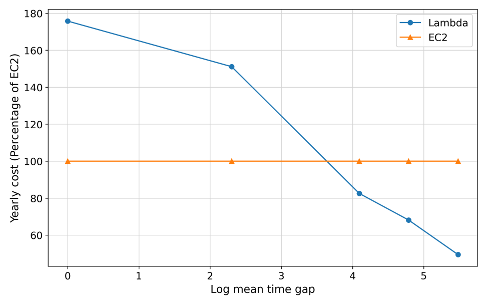


Figure 12: Training Cost of Lambda at Different Data Rates as Percentage of EC2 Cost.

### 5.2.2 Training Latency: CAND-Single vs CAND-Parallel

We conducted single-trigger experiments by varying the number of samples to compare the training latency between CAND-single and CAND-parallel, and the results are shown in Figure 13. We used low-dimensional data (50 attributes), and all Lambda functions are allocated 2048 MB of memory. CAND-single has much lower training latency than CAND-parallel. In CAND-parallel, worker functions need to frequently communicate with the synchronizer function since, for each training batch, CAND needs to choose a new subset of MLPs for training. This leads to high synchronization overhead.

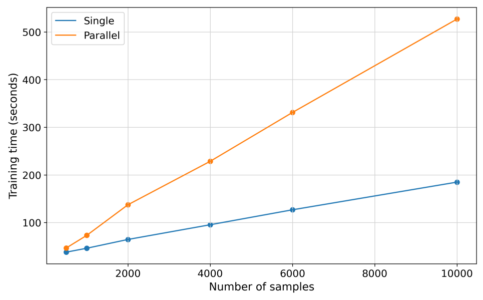


Figure 13: Training Latency of CAND-Single and CAND-Parallel for Different Number of Samples.

We perform further experiments, as shown in Figure 14, where we vary the training batch size and observe its effect on the synchronization overhead. Increasing the batch size reduces the frequency of synchronization, making the overhead less significant and reducing the training latency. However, this might make CAND less effective at reacting to concept drifts. In ARF, this synchronization overhead was less significant because the workers in ARF-parallel could independently decide whether to update themselves with a particular data sample instead of having to communicate with the synchronizer.

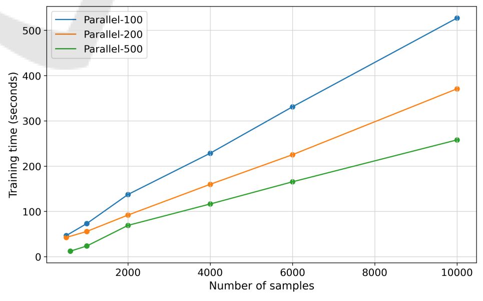


Figure 14: Training Latency of CAND-Parallel for Varying Synchronization Overhead.

### 5.2.3 Data Dimensionality: CAND-single vs CAND-Parallel

We perform single-trigger experiments with varying data dimensionality for both variants of CAND. From the results shown in Figure 15, we can conclude that model parallelism using serverless can yield significant benefits in terms of latency. Again, for higher-dimensional data (above 30,000 attributes), CAND-single fails due to a lack of resources, but CAND-parallel can perform training. Note that all Lambda functions were allocated 10240 MB of memory (which is the maximum possible allocation) and a pool size of 30 MLPs was used for this experiment. In conclusion, using serverless functions, we can perform distributed training for large, parallelizable models that work with high-dimensional data.

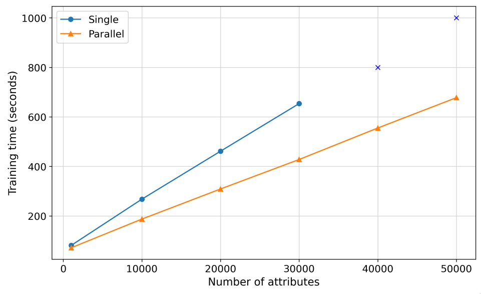


Figure 15: Training Time of CAND-Single & CAND-Parallel for Varying Data Dimensionality.

### 5.2.4 Training Cost: CAND-single vs CAND-parallel vs EC2

In our previous experiments, we established that serverless is more suitable for sporadic data streaming patterns. So accordingly, we chose a low data rate (with a mean delay of 120 seconds between data samples) and perform a one hour time-bound experiment to compare cost of serverless with VM-based systems. Bear in mind that as we increase the number of attributes in our data, we need to upgrade our EC2 server instance to meet memory requirements, which increases the cost of our VM-based deployments.

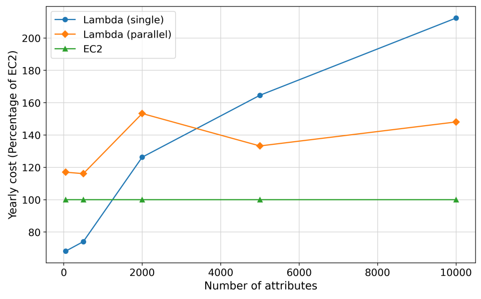


Figure 16: Training Cost of Lambda & EC2 for Varying Data Dimensionality.

From the results in Figure 16, we can conclude that our serverless (CAND-single) solution offers a cheaper alternative compared to VM-based solutions for low-dimensional data. For high-dimensional data, the models grow in size considerably, resulting in some overhead for loading and saving models every time we perform training. CAND-parallel, on the other hand, gives better results for high-dimensional data but does not outperform EC2 in terms of cost.

## 6 RELATED WORKS

### 6.1 Serverless Computing

Previous research on serverless computing falls into two categories: studies improving the serverless platform and studies leveraging serverless for specific workloads. Examples of the first category include Firecracker (Agache et al., 2020), a serverless Virtual Machine Monitor; Hermod (Kaffes et al., 2022), a specialized scheduler for serverless; Pocket (Klimovic et al., 2018), an external storage system for serverless; and OpenLambda (Hendrickson et al., 2016), an open-source serverless platform commonly used by researchers. Examples of the second category include the usage of serverless for workloads such as video processing (Ao et al., 2018), data science workflows (Patel et al., 2022), and data analytics (Müller et al., 2020).

Several studies have used serverless for ML inference tasks (Ishakian et al., 2018; Yu et al., 2021) with promising results in terms of inference latency and cost. Finally, a considerable amount of research has examined the suitability of serverless for training ML models in traditional batch-learning settings (Carreira et al., 2018; Wang et al., 2019; Jiang et al., 2021).

### 6.2 Online Machine Learning

A stream of training data is inherently different from offline training data as it is generated in real time at unpredictable rates. Thus, machine learning on streaming data differs from traditional machine learning and is referred to as online machine learning (Benczúr et al., 2018). Although not widespread in industry (vZliobait.e et al., 2012), industry giants like ByteDance (Liu et al., 2022), Facebook (He et al., 2014), and Google (McMahan et al., 2013) employ online ML. Yet, most companies often lack the financial capacity to deploy the dedicated infrastructure needed for continuous training of online ML models. Despite this, there has been a general interest in the community to explore online ML (Huyen, 2020), and efforts are being made to make the deployment of such systems feasible (Huyen, 2022).

Most research on online ML focuses on developing models suitable for incremental learning on evolving data streams. For example: data stream classification (Gaber et al., 2007), clustering (Montiel et al., 2022), and recommender systems (Pálovics et al., 2017). Other papers explore the challenges in training online ML models: concept drift (Tsymbal, 2004), scalability, and efficiency (Fontenla-Romero et al., 2013). Recent works aim to adapt deep neural networks for use in online ML: (Sahoo et al., 2017a; Gunasekara et al., 2022; Hammami et al., 2020).

### 6.3 Serverless Computing for Online Machine Learning

To the best of our knowledge, no prior research has explored the use of serverless computing for training online ML models on streaming data. While some studies have utilized serverless for streaming data (Hou et al., 2023; Konstantoudakis et al., 2022), their focus is not machine learning. A study in (Jiang et al., 2021) found using serverless for offline ML training to be faster but not necessarily cheaper than VM-based methods. However, no such study exists for online models. In (Carreira et al., 2019) a serverless-based framework was developed for traditional ML, while our research studies the development of a serverless-based framework for online ML.

## 7 CONCLUSION

In recent years, a lot of research has explored serverless computing for diverse applications. Our work concentrates on online machine learning, developing a serverless system for efficient, low-latency training of online ML models. We examine three system variants, comparing their cost and latency. We identify factors that must be considered when designing such a system: data streaming rates and data dimensionality. In our experiments, we found that with unpredictable streaming rates, serverless training offers up to 87% cost savings for ARF over VM-based methods and 17% for CAND. Model parallelism improves training latency by 3x for ARF and 1.5x for CAND with higher data dimensionality.

Our work focuses on two models - ARF and CAND - but exploring additional models could offer deeper insights into which are best suited for serverless systems. Another promising research direction is to extend our work to develop an end-to-end framework that can be used to seamlessly integrate new models. Since previous works have not explored this intersection between online ML and serverless computing, we hope our work inspires further research in this domain. Our in-depth look at system architectures, parameters, and model characteristics offers valuable insights to those choosing between serverless and non-serverless solutions for online ML.

## ACKNOWLEDGEMENTS

We are thankful to Research and Innovation Centre for Science and Engineering-Bangladesh University of Engineering and Technology (RISE-BUET) for funding this project.

## REFERENCES

Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., and Popa, D.-M. (2020). Firecracker: Lightweight virtualization for serverless applications. In *NSDI*, volume 20, pages 419–434.

Ao, L., Izhikevich, L., Voelker, G. M., and Porter, G. (2018). Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 263–274.

Benczúr, A. A., Kocsis, L., and Pálovics, R. (2018). Online machine learning in big data streams. *CoRR*, abs/1802.05872.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). Moa: massive online analysis. *Journal of Machine Learning Research*, 11.

Carreira, J., Fonseca, P., Tumanov, A., Zhang, A., and Katz, R. (2018). A case for serverless machine learning. In *Workshop on Systems for ML and Open Source Software at NeurIPS*, volume 2018, pages 2–8.

Carreira, J., Fonseca, P., Tumanov, A., Zhang, A., and Katz, R. (2019). Cirrus: A serverless framework for end-to-end ml workflows. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 13–24.

Expo, J. D. (2009). *Airline on-time performance*.

Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B., and Peteiro-Barral, D. (2013). Online machine learning. In *Efficiency and Scalability Methods for Computational Intellect*, pages 27–54. IGI global.

Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2007). A survey of classification methods in data streams. *Data Streams: models and algorithms*, pages 39–59.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495.

Gunasekara, N., Gomes, H. M., Pfahringer, B., and Bifet, A. (2022). Online hyperparameter optimization for streaming neural networks. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9.

Hammami, Z., Sayed-Mouchaweh, M., Mouelhi, W., and Ben Said, L. (2020). Neural networks for online learning of non-stationary data streams: a review and application for smart grids flexibility improvement. *Artificial Intelligence Review*, 53:6111–6154.

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al. (2014). Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*, pages 1–9.

Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., and Wu, C. (2018). Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651*.

Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2016). Serverless computation with openlambda. In *8th {USENIX} workshop on hot topics in cloud computing (HotCloud 16)*.

Hou, B., Yang, S., Kuipers, F. A., Jiao, L., and Fu, X. (2023). Eavs: Edge-assisted adaptive video streaming with fine-grained serverless pipelines. In *INFOCOM 2023-IEEE International Conference on Computer Communications*. IEEE.

Huyen, C. (2020). Machine learning is going real-time. Accessed: 2022-12-16.

Huyen, C. (2022). Real-time machine learning: Challenges and solutions. Accessed: 2022-12-16.

Ishakian, V., Muthusamy, V., and Slominski, A. (2018). Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE.

Jiang, J., Gan, S., Liu, Y., Wang, F., Alonso, G., Klimovic, A., Singla, A., Wu, W., and Zhang, C. (2021). Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data*, pages 857–871.

Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., and Recht, B. (2017). Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 symposium on cloud computing*, pages 445–451.

Kaffes, K., Yadwadkar, N. J., and Kozyrakis, C. (2022). Hermod: principled and practical scheduling for serverless functions. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 289–305.

Klimovic, A., Wang, Y., Stuedi, P., Trivedi, A., Pfefferle, J., and Kozyrakis, C. (2018). Pocket: Elastic ephemeral storage for serverless analytics. In *OSDI*, pages 427–444.

Konstantoudakis, K., Breitgand, D., Doumanoglou, A., Zioulis, N., Weit, A., Christaki, K., Drakoulis, P., Christakis, E., Zarpalas, D., and Daras, P. (2022). Serverless streaming for emerging media: towards 5g network-driven cost optimization: A real-time adaptive streaming faas service for small-session-oriented immersive media. *Multimedia Tools and Applications*, pages 1–40.

Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., Zhu, B., Zhu, Y., Wu, P., Wang, K., et al. (2022). Monolith: Real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663*.

McMahan, H., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A., Boulos, T., and Kubica, J. (2013). Ad click prediction: a view from the trenches. pages 1222–1230.

Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., et al. (2021). River: machine learning for streaming data in python.

Montiel, J., Ngo, H.-A., Le-Nguyen, M.-H., and Bifet, A. (2022). Online clustering: Algorithms, evaluation, metrics, applications and benchmarking. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4808–4809.

Müller, I., Marroquín, R., and Alonso, G. (2020). Lambada: Interactive data analytics on cold data using serverless cloud infrastructure. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 115–130.

Ooi, B. C., Tan, K.-L., Wang, S., Wang, W., Cai, Q., Chen, G., Gao, J., Luo, Z., Tung, A. K., Wang, Y., et al. (2015). Singa: A distributed deep learning platform. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 685–688.

Pálovics, R., Kelen, D., and Benczúr, A. A. (2017). Tutorial on open source online learning recommenders. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 400–401.

Patel, D., Lin, S., and Kalagnanam, J. (2022). Dsserve - data science using serverless. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2343–2345.

Sahoo, D., Pham, Q., Lu, J., and Hoi, S. C. (2017a). Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*.

Sahoo, D., Pham, Q., Lu, J., and Hoi, S. C. H. (2017b). Online deep learning: Learning deep neural networks on the fly. *CoRR*, abs/1711.03705.

Tsymbal, A. (2004). The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58.

vZliobait.e, I., Bifet, A., Gaber, M., Gabrys, B., Gama, J., Minku, L., and Musial, K. (2012). Next challenges for adaptive learning systems. *SIGKDD Explorations*, 2:in press.

Wang, H., Niu, D., and Li, B. (2019). Distributed machine learning with a serverless architecture. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1288–1296. IEEE.

Yu, M., Jiang, Z., Ng, H. C., Wang, W., Chen, R., and Li, B. (2021). Gillis: Serving large neural networks in serverless functions with automatic model partitioning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 138–148. IEEE.

Zheng, T. and Wen, Z. (2022). Online convolutional neural network for image streams classification. *Proceedings of the 5th International Conference on Big Data Technologies*.