AdaEdge: A Dynamic Compression Selection Framework for Resource Constrained Devices

Chunwei Liu
MIT CSAIL
chunwei@csail.mit.edu

John Paparrizos
The Ohio State University
paparrizos.1@osu.edu

Aaron J. Elmore
University of Chicago
aelmore@cs.uchicago.edu

***Abstract*—With the Internet of Things (IoT), a vast number of connected devices generate significant data, necessitating efficient compression techniques to manage storage costs and enhance query performance. However, "one-size-fits-all" approach to data compression is ineffective due to diverse applications, which vary in data characteristics, workloads, and hardware limitations. This paper introduces AdaEdge, a dynamic, hardware-conscious compression selection framework tailored for resource-constrained devices. AdaEdge is a best-effort compression selection framework designed to preserve application-critical information as much as possible within system constraints. It enhances the use of limited system resources through a dynamic data compression policy that considers the staleness and the significance of the data. AdaEdge applies a multi-armed bandit algorithm to assist compression selection, optimizing workload targets such as compression ratio, compression throughput, workload accuracy, or their weighted combinations. It supports both lossy and lossless compression selection, adapting to hardware constraints. It operates in both online and offline modes, addressing network constraints for edge nodes and evolving data policies to preserve workload-specific information. AdaEdge improves machine learning task accuracy by up to 30% over baseline within the same storage budget and by up to 20% in scenarios where lossless methods fall short due to low compression ratios. AdaEdge also shows robustness against data shifts and hardware variability.**

I. Introduction

The Internet of Things (IoT) encompasses billions of devices globally, connected to the internet and sharing vast data amounts. For instance, an oil platform with 80,000 sensors can generate up to 2 TB of data daily. However, the high cost and unreliability of satellite communications for offshore operations make transmitting this data to centralized locations impractical. [3], [20]. This issue is mirrored in Renewable Energy Systems (RES) such as wind turbines, which, with their multitude of high-frequency sensors, produce data volumes that far exceed the limited bandwidth available for cloud transfer [25]. This challenge is further compounded in the realms of deep-sea and deep-space exploration [53], where unstable networks, severe data transfer, and storage limitations demand highly efficient data management strategies.

Given the high costs of storage, limited storage capacity, and query performance benefits, compression plays an important role in IoT systems [26], [29], [42], [43]. Data systems typically employ different compression approaches to reduce the storage cost and improve bandwidth utilization on the cloud [5], [27], [36], [38], [44], [45] where the compression happens after data is centralized. Close to the IoT sensors, edge devices are usually the entry points for collecting and preprocessing the IoT data, which is a critical component of an IoT system. As edge devices gain more computational power and popularity, supporting compression, query, and analytics on the edge side is a good option to reduce the cost of IoT systems. By implementing compression closer to the data source, the volume of data requiring transfer is reduced, conserving network bandwidth and sparing edge or cloud storage resources, often leading to enhanced query performance. Nonetheless, edge devices are typically more resource-constrained than cloud servers in terms of network bandwidth, storage, and computational power, presenting novel challenges and constraints for compression design and selection. While edge deployments can vary in terms of resource capacity, many edge applications are built on static allocations and lack elastic-resources, which we consider as resource-constrained.

Modern data systems support a variety of lossless and lossy compression approaches to store and query data. Due to well-known problems of distribution drift in data streams, a single compression solution may fail to encode data of a given signal effectively. Compression approaches differ in terms of compressed size and query performance. Significantly, task accuracy is also affected in the case of lossy methods. Because of dynamic data features and various workloads across extended IoT applications, providing a one-size-fits-all compression solution for all kinds of data and tasks is impossible. Therefore, an adaptive lossless and lossy compression selection strategy is necessary to select the optimal compression for the incoming IoT data based on the data statistics, query workloads, and limited resources available on the host.

Compression selection is an optimization task accounting for the features of the data, hardware, and workload. Compression selection can optimize for different performance targets, including space usage, compression throughput, and query performance. Prior work presents lossless compression selection solutions based on a decision tree [6], regression model prediction [15], [16], or neural network models [27]. Open-source columnar formats [37] like Apache Parquet [11], ORC [10] and Arrow Feather [9], along with commercial systems like Vertica [40] and MySQL [18] choose to hard-coded compression selection based on the column data type. Traditional compression selection solutions rely mainly on lossless compression approaches and optimize for either storage or SQL query performance with assumptions of adequate space for the compressed data and sufficient computation power for compression tasks and workload. Such assumptions are not always valid in real-world applications as the storage, bandwidth, and computation resources can be expensive.

Many devices lack sufficient hardware resources. Edge devices, in particular, often face stringent constraints and limitations stemming from their hardware and system environment, including restricted storage capacity, limited bandwidth, and reduced computational power. Traditional lossless compression comes with a compression limitation defined by the entropy of the data [48]. So it is impossible to always keep 100% accurate data representation with limited system storage resources. In contrast, many time-series databases widely use lossy compression to control storage consumption. To handle the limited storage, many prior works either remove the old data periodically [2], [51] or keep a different level of data summary depending on the workload and accuracy requirement [7], [8], [23]. Systems such as ModelarDB and SummaryStore allow user-defined compression methods to be added by implementing an interface. What is often overlooked is a comprehensive adaptive selection strategy that encompasses a wide range of both general lossy and lossless compression techniques. Furthermore, previous works mainly focused on the lossy impacts on the accuracy of dashboard queries and some SQL operations. However, machine learning tasks are poorly studied as an optimization target.

Different lossy compression methods vary in accuracy and efficiency for various tasks, and no single method fits all needs in IoT systems with varying data statistics, workloads, and hardware constraints. A lightweight compression selection framework supporting both lossy and lossless compression approaches and multiple optimization targets has been overlooked. This paper formulates the compression selection in IoT systems as a multi-armed bandit problem and proposes AdaEdge as a hardware-conscious encoding selection framework for resource-constrained devices. In addition to the conventional lossless compression selection support, AdaEdge provides a trade-off between space and workload accuracy through its auto-recoding component. AdaEdge distinguishes itself as a robust, best-effort compression selection framework, offering the following key contributions:

- Multi-armed bandit-assisted lossy and lossless compression selection based on system constraints and workload target.
- Online and offline compression solutions for resource-limited devices, preserving mission-critical information.
- Multiple optimization targets supported, including compression ratio, throughput, aggregation queries, and machine learning task accuracy.
- Compression policy to selectively compress data according to its importance.

AdaEdge adaptively selects the optimal compression solution for edge signals, considering network, storage, signal features, and workloads. Our experiments show AdaEdge achieves $10\% - 20\%$ higher accuracy in ML tasks than baselines for online cases needing low compression ratios (e.g., 0.1) where lossless compression is not viable, and up to 30% accuracy gains within the same storage constraints posted to the system.

For the rest of the paper, we review related work in Section II. We then formulate the compression problem in Section III and introduce the AdaEdge framework in Section IV. We evaluate AdaEdge in Section V and conclude in Section VI.

II. Related work

Compression selection has been studied for decades. To get an accurate data representation, many data systems use lossless compression selection optimizing for compressed size or query performance. For example, Abadi's decision tree [6] is one of the first projects to explore compression selection for in-situ query execution on compressed data. Such an approach introduces a rule-based encoding selection approach that relies on the global knowledge of the dataset features. LEA [16] is a compression selection framework that builds difference regression models to predict the compression and query performance, then selects the optimal compression according to the optimization target. CodecDB [27] builds a neural network model to predict optimal compression based on the data features. It also provides specialized operators operating on encoded columns directly. Hyrise [15] uses separate models to predict compressed size and query operator performance, then uses linear programming or greedy heuristics to get the best encoding for each data chunk. BtrBlocks [30] offers a sample-based compression selection solution to achieve a better compression ratio and throughput. In addition, many open-source data formats and some commercial data systems choose hard-coded compression strategies for each data type.

All those conventional compression selection solutions assume adequate storage space capable of compressing the file by the optimal lossless compression approach. However, such an assumption on the system resources does not always hold. Lossy compression is necessary for systems with limited storage resources. Some systems directly remove data ingested earlier than a given time or exceeding a storage threshold. RRDtool [51] and InfluxDB [2] remove old data exceeding a certain period to take back space for new data. ModelarDB [24] trades query accuracy for storage space by selecting the best linear model to approximate the data with user-defined error bounds according to the storage budget. SummaryStore [7] reclaims the storage space by replacing the data with aggregate summaries with a specific compression ratio. TVStore [8] uses a time-varying compression framework to bound storage. These projects mainly rely on a single lossy compression approach to compress the data according to the storage budget.

To the best of our knowledge, AdaEdge is the first framework providing both best-efforts lossless and lossy compression selection with multiple optimization goals supported.

III. Problem Statement

In this section, we introduce several compression approaches. We then formulate the compression selection problem in the IoT system setting and frame it as a multi-armed bandit problem. Finally, we introduce the multi-armed bandit Problem, its variations, and its applications as a solution.

A. Lossless and Lossy Compression

Compression falls into two categories: lossless, where the original data can be completely restored from the compressed file, and lossy, where some data is permanently lost and cannot be fully recovered.

1) Lossless compression: Popular lossless compression approaches include byte-compression techniques, such as Gzip, Snappy and Zlib, and lightweight encodings, such as dictionary encoding [13], [31], [38], Gorilla [45] (and its optimized variation CHIMP [33]), Sprintz [14], and BUFF [36] (along with the subsequent variation Elf [32]). Lossless compression is widely used in data systems to persist historical data, yet its effectiveness is hard-bounded by the entropy of the input data. The entropy defines the minimal bits required to represent the corresponding information losslessly. This implies a lower bound for a 100% accurate representation. When storage is insufficient, all the existing data compression selection frameworks fail to get an encoding solution. However, such scenarios are common in IoT systems, where dynamic resource and hardware limitations often demand highly aggressive compression to meet system constraints. In this case, further data size reduction necessitates accepting partial information loss, with the aim to minimize its impact on task accuracy.

2) Lossy compression: AdaEdge incorporates basic lossy compression to address these scenarios, which offers significantly higher compression ratios than lossless methods by sacrificing some information. To better serve our use case, all lossy compression methods in AdaEdge are customizable to reach the desired compression ratio. Given the storage budget, AdaEdge chooses suitable lossy compressions for tasks with minimal accuracy loss. They are designed for easy recoding, minimizing the time typically required for decompression and re-compression with new settings. Currently, AdaEdge supports various lossy compression techniques.

BUFF [36] is a lossless compression method for float types, using a byte-oriented layout and a defined precision, but it can act as lossy compression by reducing float precision. In AdaEdge, we use a lossy version of BUFF to aggressively compress by discarding insignificant bits. BUFF-lossy minimally alters input values, benefiting tree-based machine learning tasks sensitive to changes in feature values.

Piecewise Linear Approximation (PLA) [49] is a linear generalization method that enables linear approximation in each data segment, with the number of segments defined by the resource budget. Largest Triangle Three Buckets (LTTB), a variant of the Visvalingam-Whyatt (VW) algorithm [52], is a line generalization method used in TVStore [8] and TimeScaleDB [4]. LTTB excels in maintaining visual signal integrity, making it ideal for dashboard queries.

Piecewise Aggregate Approximation (PAA) [28], [54] reduces time series dimensionality by segmenting them into mean values, serving as a form of lossy compression. By adjusting the window size, PAA can vary the level of approximation. PAA is effective in maintaining the accuracy of Sum and Avg queries due to its focus on mean values.

Fast Fourier Transform (FFT) [21], based on the Fourier transform, converts signals between their original and frequency domains. This conversion facilitates compression with minimal distortion by eliminating less significant frequencies, particularly high-frequency components. FFT is advantageous for measuring distances in high-dimensional data.

RRD-sample. We include a random sampling method to simulate the RRDTool compression logic which bounds storage by deleting data when the storage quota is reached. Instead of deleting old data, AdaEdge saves one random value from it for future queries and replicates this value across a segment as needed for specific workloads.

We acknowledge that IoT protocols [46], like MQTT and OPC-UA are key for messaging in industrial IoT devices but lack built-in data compression. These protocols aim to minimize device code footprint and network bandwidth, not compress data. In contrast, AdaEdge focuses on data compression at storage and processing levels, offering lossy compression options for edge computing constraints. This approach complements the messaging protocols in the IoT ecosystem.

B. Compression Selection

Data compression is essential for conserving network and storage resources, especially in resource-limited devices and systems. It must balance resource constraints with task accuracy, using lossless compression when possible to allow full data recovery. In cases of severe storage limitations, aggressive lossy compression becomes necessary to save resources with minimal task accuracy loss. The lossless compression selection will mainly focus on the compressed data size and compression throughput, while the lossy compression selection will focus more on the accuracy loss of the workload.

Resource constraints stem from aspects of the system, including data ingested rate, limited memory, storage and network bandwidth. The constraints can also come from the requirements of the downstream application and workload: the compression should help speed up the workload and diminish the workload accuracy loss. These constraints impose requirements on the compression ratio, compression throughput, query throughput, and task accuracy.

We formulate compression selection as an optimization problem for the resource-constrained system. The constraints are denoted as I_{bgt} for a given signal ingestion rate (assuming no control on the signal generating rate), B_{bgt} for network bandwidth, and S_{bgt} for local storage capacity. The data sequence is organized in segments with a fixed number of consecutive data points (original size denoted as U). Only one compression scheme is selected for each segment. $v_{ij} = \{0, 1\}$ indicates whether compression c_j is selected for segment x_i, then we have $\sum_{j=1}^{k} v_{ij} = 1$ for any given segment i assuming k compression approaches in the compression candidate set C. In terms of the compression performance, we also use r_{ij} to indicate the estimated compression ratio of compression

c_j on segment x_i. t_{ij} indicates the estimated compression throughput of compression c_j on segment x_i. e_{ij} indicates the estimated query error of lossy compression c_j on segment x_i. For n segments, we can get the total compressed size:

$$S = U \times \sum_{i}^{n} \sum_{j}^{k} (r_{ij} * v_{ij})$$

And we set the average egress rate as follows:

$$B = \frac{I_{bgt}}{n} \times \sum_{i}^{n} \sum_{j}^{k} (r_{ij} * v_{ij})$$

So when we optimize for the given network bandwidth B_{bgt}, we formulate the lossless compression selection as

$$\arg\min_{V} S(V) \quad s.t. \quad I_{comp}(V) \geq I_{bgt} \ and \ B(V) \leq B_{bgt}$$

where V is the compression assignment for each segment comprised by v_{ij} and $I_{comp}(V)$ indicates the compression throughput with given V. The lossless compression selection should handle the signals and generate compressed data with an egress rate under the network capacity. If no feasible solutions, a lossy compression selection is needed to meet the harsh constraints. To optimize for the given network bandwidth B_{bgt}, we formulate the lossy compression selection as

$$\arg\min_{V} E_T(V) \quad s.t. \quad I_{comp}(V) \geq I_{bgt} \ and \ B(V) \leq B_{bgt}$$

where $E_T(V)$ represents the accuracy loss for task T under compression V. The objective is to minimize this loss while adhering to system constraints.

For systems without a constant network for data egress, the focus shifts to continuously ingesting new data while minimizing accuracy loss during data shedding. We define the compression selection problem as

$$\arg\min_{V} E_T(V) \quad s.t. \quad I_{comp}(V) \geq I_{bgt} \ and \ S(V) \leq S_{bgt}$$

we keep evolving the data to meet storage budget while ensuring the chosen compression can handle the given signal.

The input signals in AdaEdge are organized into fixed-size segments. The compression thread keeps compressing the ingested segments to meet the system constraints. The system refines its estimations by observing how each approach compresses, leading to improved compression recommendations. The compression selection module autonomously chooses the best method by considering changing workloads, data inflow rates, and system limitations. When faced with shifts in data distribution or resource constraints, our solution is poised to unearth the most advantageous strategy rather than defaulting to a simplistic, greedy algorithm that merely leverages historical data patterns. This process necessitates a lightweight framework to summarize past compression outcomes and make recommendations based on recent system feedback. Thus, we cast the challenge of selecting IoT data compression within the framework of a multi-armed bandit problem.

C. Multi-Armed Bandit Problem

The Multi-Armed Bandit Problem (MAB), also called K-armed bandit problem [39], [50], involves an agent choosing from a set number of options or actions, each offering a reward based on different, unknown probability distributions. The goal is to maximize the total expected reward over a certain period.

Assuming a bandit with k arms, each arm has an expected reward $q(a)$ when action a is chosen. The arm chosen at time t is A_t, with its reward being R_t. If action a is chosen at time t, the expected reward is

$$q(a) = \mathbb{E}[R_t | A_t = a]$$

Without knowing the reward for each action in advance, we require an accurate estimate, denoted as $Q_t(a)$ for action a at time t. The more $Q_t(a)$ aligns with the true reward q(a), the better the decision. Ideally, if $Q_t(a)$ is perfectly accurate, we'd always choose the action with the highest value to maximize return. The action selection at time t can be described as:

$$A_t = \arg\max_{a} Q_t(a)$$

A greedy action involves the agent choosing the arm with the highest estimated reward, thereby exploiting its current understanding of the bandit system. However, relying solely on exploitation can limit learning to previously acquired knowledge, making it necessary to explore non-greedy actions to refine reward estimations. While exploitation maximizes immediate rewards, exploration can lead to greater overall rewards over time.

There are many sophisticated variations for balancing exploration and exploitation for particular mathematical formulations of the MAB problem. ϵ-greedy is a basic solution of balance exploration and exploitation for the MAB problem, with ϵ possibility to choose the non-greedy action. The Optimistic ϵ-Greedy Algorithm is a simple modification by setting the initial action estimates to high values in a regular greedy algorithm to push it to explore the whole candidate action, searching for optimal action. Rather than performing exploration by simply selecting an arbitrary non-greedy action, chosen with a probability ϵ that remains constant, the Upper Confidence Bound (UCB) algorithm shifts from prioritizing exploration of less-tried actions to focusing on exploitation, choosing actions with the highest rewards, as it learns more about the environment. Other bandit algorithm variations, such as Gradient Bandit [22] and Contextual Bandits [17], are not the focus of this work.

MAB is efficient in both computation and space ($O(K)$) with K arms. In AdaEdge, we associate each arm of MAB with a specific compression, and we return the optimization target as its reward. Sections IV-C and IV-D detail how we map the compression selection to the MAB problem.

IV. AdaEdge Overview

AdaEdge is an adaptive hardware-conscious compression selection framework. AdaEdge selects the optimal compression according to system resource limitations, data statistics, and query workload. Figure 1 shows the overview of AdaEdge.

A. AdaEdge Framework Constraints

We first introduce the basic constraints in AdaEdge.

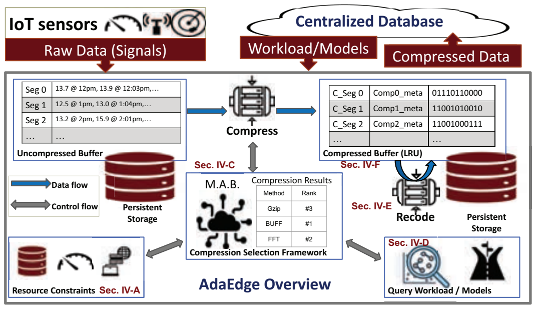


Fig. 1: AdaEdge operates in both online and offline modes.

1) Signal rate: Signal generation rate from various sensors can range from zero to millions of data points per second. In AdaEdge, we assume no control to back-pressure the sensors regarding the data generation rate, so the ingestion rate is a hard constraint for our framework. AdaEdge should ingest all incoming data and process it with proper compression, or downsampling when necessary. In our compression selection step, all selected compression must be able to handle the given ingestion rate from the sensors.

2) Network bandwidth: While broadband networks are common, low bandwidth networks are still popular and play critical roles in the modern IoT ecosystem. The network bandwidth varies depending on the network connection technology. In practice, the bandwidth changes for a cellular network from 0.01 Mbps to 200 Mbps. It also varies significantly in cases of network congestion or edge device movement. Network disconnection is typical for IoT edge devices in some environments, such as the agriculture, aerospace exploration, and mining industries. AdaEdge can handle all different network connection cases and provide proper data ingestion and compression solutions accordingly.

3) Storage space: An edge node has very limited storage space. Thus, storage devices are carefully accessed on the edge device. To extend device lifetime and minimize power consumption, some industrial systems refrain from using storage for routine operations, reserving it solely for offline use.

4) Power: Many edge devices are deployed in remote areas without a sustainable power supply, so all tasks must account for power consumption. AdaEdge mainly focuses on other constraints and leaves power constraints as future work.

AdaEdge supports such constraints and prioritizes the retention of critical information through effective compression strategies rather than attempting to eliminate them entirely.

B. AdaEdge Working Modes

Based on those constraints, we classify the IoT use case into two modes. In terms of network connection, we introduce online and offline modes for AdaEdge.

1) Online mode: The edge node serves as a continually connected data hub, aiming to maximize the signal data transfer within given constraints, including signal rate and network bandwidth. In online mode, AdaEdge keeps all data ingested and compressed in memory before transmitting through network protocol. The selected compression should be able to

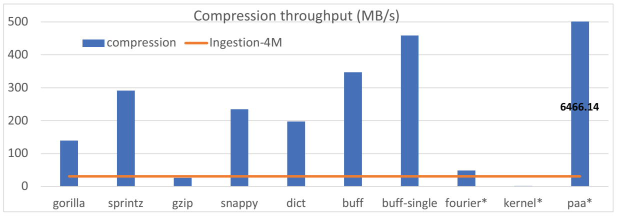


Fig. 2: Compression should be able to handle the given signal generation rate. An example shows a signal with 4 million data points generated per second (* marks lossy compression).

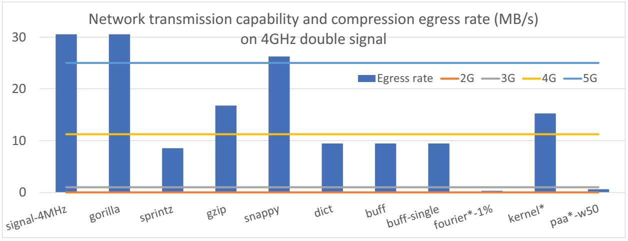


Fig. 3: The compressed file size should be within the network transmission capacity. Bars show the compression egress rate on a 4GHz signal with double type. Lines show the network transmission capacity per second (MB/s).

handle the given signal generation rate, and the compressed file size should be within the network transmission capacity.

Figure 2 shows an example with 4 million data points generated per second, a data generation rate for a typical oil well platform. The line shows the ingestion rate (the size of data generated per second), and the bars are the ingestion rate (the size of data compressed per second at full speed) for different compression approaches. Most compression methods are qualified to compress the signal example except for Gzip and Kernel methods, which are usually slow in compressing the data. In addition to the ingestion rate, we also investigate the compressed egress performance. Figure 3 shows the egress rate of the 4GHz signal without compression and with different compression methods applied. A lower egress rate on the same signal means better compression performance. Thus, less network bandwidth is needed to transmit the compressed data. Without compression before data transmission, it is impossible to transfer the ingested data to the cloud with the given network setting. Under a 4G network, many lossless compression such as Sprintz, BUFF, dictionary encoding, and lossy compression can send out the compressed data. However, in a 3G network environment, none of the prevailing lossless compression methods meet the requisite standards for effective data transmission. At this juncture, many conventional compression selection solutions encounter failure. In contrast, AdaEdge stands out by intelligently selecting the appropriate lossy compression methods under a scarce bandwidth to minimize the query error rate, considering the specific demands of the query workload. This adaptive approach ensures optimal performance under any network constraints.

For the online mode, we keep all data ingested and compressed in memory before we send out those segments through a network protocol. We omit saving data to the local storage to save storage space for future offline cases, save power and prolong the lifespan of the storage device. We choose the best lossless compression by default. In the particular case of no qualified lossless compression, we chose lossy approaches optimized for the target workload.

2) Offline mode: The edge node serves as a local computation and storage node for scenarios with poor network or intermittent connection. The goal switches to keep ingested data as much as possible for data offloading if a future network connection is expected. If information loss is inevitable because of a tight storage budget, it uses lossy compression with the smallest negative impact on the workload. Essentially, maintaining the accuracy of the data is crucial for making informed decisions in the given task.

Figure 1 also shows the components related to the offline module. Unlike the online mode, which consistently prioritizes minimizing compressed size, the offline mode employs load shedding when local storage capacity is exceeded. AdaEdge dynamically manages ingested data within the allocated storage budget for devices operating in offline mode. All ingested data is temporarily stored on the local disk. When there is insufficient space, AdaEdge applies more aggressive compression (even lossy compression) on less valuable or less informative segments instead of removing them.

The informativeness can be measured by the query usage of the segment, such as a query counter for each segment. However, it should also reflect the contribution of each segment to the query. For example, a segment with 1% qualified entries is less informative than one with 99%. The ratio of qualified entries for a segment is defined by the number of entries qualified for a given query divided by the total number of entries in the segment. Since there are many definitions of informativeness, AdaEdge builds a dedicated segment management component with standard *GET* and *PUT* APIs for different policies. AdaEdge uses an LRU-based compression policy by default, discussed in Section IV-F.

C. AdaEdge Workflow

AdaEdge allows the collection and aggregation of data from multiple device clients. It handles time series data collected at uniform, regular intervals—a setup commonly defined by sensor configurations. AdaEdge ingests data points generated by remote sensor clients, caches them into fixed-size arrays, assigns a timestamp to each segment, and then pushes it into the uncompressed buffer. As the uncompressed buffer is being filled, compression threads offload data from the uncompressed buffer. The compression selection threads are adaptively configured to use different compression approaches based on storage capacity, network bandwidth, ingestion rate, and specified analytical tasks. If the uncompressed buffer exceeds its capacity, which may happen when the ingestion rate exceeds the compression speed, the data is flushed to the disk. The compression threads push the compressed data into a compressed buffer pool, which can also flush to the disk. AdaEdge can execute queries or analyses (e.g., aggregation queries, clustering) over the compressed data or the raw time-series segments in the uncompressed buffer. Each segment in the framework is associated with metadata describing its compression configurations, which can be used for the codec to do further compress or decompress the downstream workload.

AdaEdge currently supports byte-compression techniques, such as Gzip, Snappy and Zlib, and lightweight encodings, such as dictionary encoding, Gorilla [45], Sprintz [14], and BUFF [36] for numeric data. AdaEdge also supports specialized lossy time-series representation methods (i.e., lossy compression), such as Piecewise Aggregate Approximation (PAA) [28], [54], Fourier transform [21], Piecewise Linear Approximation (PLA) [49], and BUFF-lossy [36]. These approaches differ in terms of compression ratio, throughput, query efficiency, and workload accuracy. There is no one-size-fits-all approach for any time series or task. With system resource limitations defined by edge devices, AdaEdge can adaptively switch the compression approach according to the optimization target. AdaEdge currently supports optimization targets, including space (data compression ratio), time (compression runtime), and accuracy (machine learning model and aggregation query accuracy).

The compression selection component in AdaEdge can automatically select the compression approach, given the workload, data arrival rates, and resource capacity. AdaEdge compression selection requires a lightweight framework to efficiently summarize the past compression performance and provide compression suggestions based on the latest observation from the system. The compression selection component of AdaEdge is built based on the MAB problem. As the signal is ingested into the system, AdaEdge compresses the uncompressed segments and puts them into the compressed buffer. Once compression is finished for segments, AdaEdge runs an evaluation process to monitor the compression ratio, compression throughput, and target workload accuracy. This information is used for the MAB component to estimate the potential reward by applying each compression candidate. In the MAB greedy algorithm logic, the compression selection for AdaEdge is selecting the compression arm with the highest estimated value of the optimization target. So the $Q_t(a)$ from section III-C will be the optimization target estimation for compression action a at time step t.

In AdaEdge, the compression selection may consist of multiple MAB instances tailored to specific use cases.

1) Online selection: In the online mode of AdaEdge, system constraints determine a target compression ratio. Consider an edge device receiving a continuous signal composed of double-precision floating-point numbers – taking up 8 bytes for each value – at an ingestion rate of I points per second, with a network bandwidth of B bits per second. To facilitate the transmission of this signal via the given network infrastructure, we can calculate a provisional target compression ratio, $R = B/(64 \times I)$, temporarily omit the bit overhead from network packet headers across different network layers. AdaEdge initially applies lossless compression to the data segments as they are ingested. Should it become apparent that the target compression ratio R is impossible with lossless

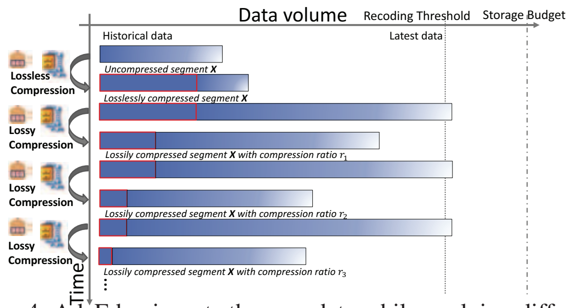

Data volume
Recoding Threshold
Storage Budget
Historical data
Latest data
Lossless Compression
Uncompressed segment *X*
Losslessly compressed segment *X*
Lossy Compression
Lossily compressed segment *X* with compression ratio r_1
Lossy Compression
Lossily compressed segment *X* with compression ratio r_2
Lossy Compression
Lossily compressed segment *X* with compression ratio r_3
Time

Fig. 4: AdaEdge ingests the new data while applying different levels of compression on earlier segments to free space for incoming data. The red rectangle denotes an arbitrary segment, with the rectangle length showing its current size.

methods, the framework seamlessly transitions to employing lossy compression techniques.

For lossless encoding selection, task accuracy remains unaffected, thus the focus for the MAB optimization is solely on minimizing the compressed segment size. Upon successful compression of a data segment, the MAB component is updated with the new compressed size—this update refines the reward estimation for MAB, enhancing the decision-making process for subsequent compressions. Should lossless compression prove inadequate for achieving the desired target compression ratio R, a dedicated MAB instance is spawned to oversee the lossy compression selection. Lossy compression inherently implies an imperfect restoration of the original data upon decompression, which can detrimentally influence task accuracy. However, AdaEdge's current suite of lossy compression techniques can be finely tuned to meet any specified target compression ratio, ensuring uniformly sized compressed segments across varying methods. Consequently, the compressed file size becomes secondary in importance for lossy compression scenarios, shifting the primary optimization objective to the maintenance of task accuracy, which becomes the key reward criterion for MAB. AdaEdge regularly evaluates task accuracy for segments compressed using lossy methods, updating the MAB with the most recent reward values to enhance the quality of future compression recommendations. The methodology for calculating the reward value in each context is further elaborated in Section IV-D.

2) Offline selection: The offline mode for AdaEdge is more complicated than the online mode, as there is no clear target compression ratio derived from the system constraints. An actual use case could be an offline edge device without any network connection, so there is no way to egress the ingested data. The data must keep evolving within the limited storage budget the system gives. The goal of AdaEdge compression selection in the offline mode is to keep mission-critical information as much as possible to minimize the potential loss of task accuracy under the given storage budget. The data evolving here is achieved by applying more aggressive compression on the data with less critical information. Figure 4 shows the cascade compression for the offline mode. For a given segment in the uncompressed buffer, AdaEdge applies lossless compression by default to save space for future segments. When a predefined threshold is reached, the recoding process is waked up to further compress the segments in the compressed buffer. As the system keeps ingesting more and more data points, more space is needed; thus, more aggressive compression will be applied to the old data segments. The compression sequence of the segments depends on the cache maintenance policy, which we discuss in Section IV-F.

AdaEdge creates multiple MAB instances for different compression levels for offline mode. Each MAB instance corresponds to a specific compression ratio range. This design is based on the observation that the optimization target changes significantly across different compression ratio ranges, and a single MAB instance for lossy compression selection is hard to reflect the compression ratio impact on the optimization target. For a given uncompressed segment, the lossless compression selection MAB instance suggests the optimal lossless compression and compresses the segment to save space. Consequently, the size of the compressed segment becomes the optimization target for this MAB instance, and the resulting compressed size is fed back into the MAB for improved future estimations. AdaEdge processes incoming data points until a predefined storage threshold, denoted as (θ), is reached. When additional space is required for new segments, AdaEdge initiates lossy compression on the already compressed segments. By default, the size is reduced to half of the original, from which a target compression ratio for the current segment is derived. The lossy compression selection mechanism then consults the MAB instance that corresponds to the identified range of the target compression ratio. The action recommended by this instance is considered the optimal compression method for achieving the desired optimization target within that specific compression ratio range. Periodically, AdaEdge evaluates the performance of each MAB instance against the target task, focusing on the compressed segments that fall within its designated compression ratio range. This continuous process of compression and recoding ensures that segments are managed within the storage budget constraints until an option to egress the data becomes available. Planning for bandwidth usage during reconnection presents an intriguing area of exploration that we intend to pursue in future iterations of AdaEdge.

D. Optimization Target

AdaEdge supports both single and complex optimization targets to meet system requirements. AdaEdge provides users with the flexibility to specify their optimization targets, which can encompass a variety of objectives, such as the accuracy of aggregation queries (including minimum, maximum, sum, and average calculations), the precision of machine learning tasks, or the efficiency of compression throughput. AdaEdge offers flexibility to accommodate additional workload targets readily, provided they can be assessed through a well-defined quantitative metric.

1) Machine learning task accuracy: For tasks related to classification or clustering within AdaEdge, we adopt accuracy evaluation metrics from the domain of approximate inference, as detailed in the work by [41]. The machine learning task accuracy evaluation on a lossy compressed segment is defined as follows: with lossy compression, we can get its compressed representation R_l. Before feeding it into the machine learning task, we need to decompress it into X_l. Given the task T and a pre-trained model M, we get the prediction results in $Y_l = M(X_l)$. For the machine learning task evaluation on lossy compressed data, we define the accuracy as the ratio of matched prediction labels compared with predicted labels on the original uncompressed data:

$$ACC_{ml} = \frac{|\{x|x \in X \ AND \ M(x_l) = M(x)\}|}{|X|}$$

Here, we assume the task model M is the pre-trained model on the original dataset X and is provided to the compression task as a given input model. AdaEdge incorporates a specialized module for serialization and deserialization to manage instances of machine learning models. When a machine learning task is specified as the optimization target, AdaEdge is equipped to load and deserialize the corresponding binary file, converting it into a machine learning model instance ready for performance evaluation. *In other words, we operate under the assumption that the model undergoes centralized training on the raw data format, and we regard any output from the model—be it a prediction or a cluster assignment—as the ground truth.* Note that while for some tasks, training and inferring on a compressed representation can improve task accuracy [19], we do not consider this here as we assume model tuning and configuration are handled by downstream tasks. By treating the provided model as the ground truth, we enable local experimentation with various compression techniques and parameters without altering the underlying model. This approach aligns with the evolving landscape of edge computing, where federated learning and the deployment of pre-trained models on resource-constrained devices are becoming increasingly prevalent [35].

2) Aggregation accuracy and throughput: In addition to the machine learning tasks, AdaEdge defines relative loss for aggregation queries as other work does for approximate query processing evaluation [34]:

$$Acc_{agg} = 1 - \frac{|V_{true} - V_{lossy}|}{|V_{true}|}$$

Where V_{true} is the real value we get from the aggregation operator on the original data, while V_{lossy} is the estimated value on the lossy decompressed data. The compression throughput is defined as $C_{thr} = S_o/T_c$, where S_o is the original file size and T_c corresponds to the compression runtime. A fast compression usually means fewer instructions for the codec, which consumes less power [12]. Therefore, fast compression may correspond to a power-efficient compression approach, which is very important for IoT applications.

3) complex optimization targets: Beyond catering to a single optimization target, AdaEdge is equipped to handle complex optimization targets that involve a hybrid of weighted objectives. This functionality is particularly useful for workloads encompassing multiple queries and machine-learning tasks. Users are afforded the flexibility to assign distinct weights to each task, allowing for the customization of optimization targets to meet the specific needs of their system. This feature ensures that AdaEdge can adapt to diverse performance criteria, providing tailored support for various workload scenarios. The complex target is defined as

$$target_c = w_1 \times \overline{ACC}_{agg} + w_2 \times \overline{ACC}_{ML} + w_3 \times \overline{C}_{thr}$$

where $\overline{ACC}$s and $\overline{C}_{thr}$ are normalized, $w_1 + w_2 + w_3 = 1$.

E. Recoding Optimization

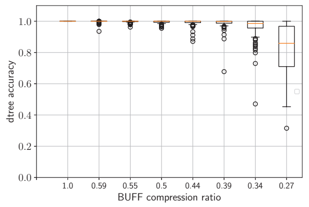
AdaEdge is designed to perform recoding on the same data segment multiple times if necessary. Unlike traditional compression methods, which typically require complete decompression before recoding can occur, AdaEdge employs a strategy akin to ”virtual decompression” [8] to minimize unnecessary decoding steps. This approach allows for more efficient recoding processes, as all lossy compression techniques supported by AdaEdge can be reapplied for recoding without the need for prior decompression. When more aggressive compression is required, AdaEdge evaluates the original and intended compression methods. If feasible, it performs direct recoding on the already compressed data, bypassing the decompression stage altogether.

AdaEdge currently supports ”virtual decompression” for recoding tasks with identical source and destination compression approaches. For instance, to achieve a higher level of compression, we can truncate data compressed with BUFF by discarding the less significant bits. Similarly, we can apply PAA compression to data already compressed with PAA to obtain a larger window size, further compress the FFT-encoded segments by removing additional high-frequency components, and apply PLA compression to PLA-encoded segments to reduce the number of key data points, thereby conserving more space. Similar work can be done by enabling direct transcoding between different compression approaches, which need specific compression optimization for each compression pair, so we keep it as future work.

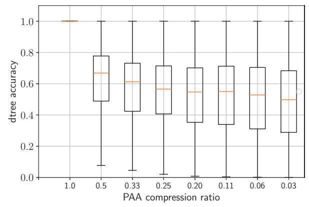
F. LRU-Based Compression

The sequence in which compression and recoding are applied to data segments is crucial for minimizing adverse effects on the target workload. For instance, newly ingested data is often more valuable than older data [7]. Consequently, when space constraints necessitate it, older data should be subjected to more aggressive compression while preserving the most recent data in a lossless format whenever feasible. Previous systems, such as RRDTool [51] have employed a round-robin approach to discard older data. Similarly, TVStore [8] and SummaryStore [7] implement more aggressive compression on segments with earlier timestamps.

AdaEdge adopts an LRU-based compression policy, prioritizing the compression of the least recently accessed segments. This strategy ensures that segments frequently accessed for queries are less likely to undergo lossy compression. The segment management component within AdaEdge operates on a list-based mechanism. It provides the compression or recoding threads with the least recently used segments at the forefront of the list, while newly compressed segments are

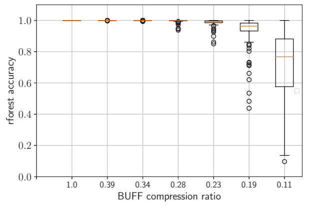


(a) BUFF

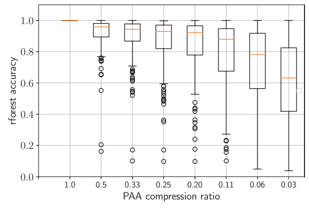


(b) PAA

Fig. 5: Decision tree model accuracy on UCI



(a) BUFF



(b) PAA

Fig. 6: Random forest model accuracy on UCR

appended to the end. Additionally, any segments accessed by active queries are relocated to the end of the list, reducing their likelihood of being lossy compressed. The LRU-based compression policy works well for AdaEdge, where ingesting the signal and some aggregation queries are the main tasks.

The segment management component is designed to be flexible, allowing for easy adaptation to alternative compression sequencing policies. While further exploring different policies could yield interesting insights, we have chosen to focus on the LRU-based approach for simplicity in this work.

V. Evaluation

We evaluate AdaEdge on aggregation and machine learning tasks with varying compression approaches. The MAB-assisted component for selecting compression in AdaEdge includes lossless compression options such as Gzip, Snappy, Gorilla, Zlib, BUFF, and Sprintz, using default settings for the first three and variable compression levels for Zlib. For BUFF and Sprintz, precision is tailored to four digits for the CBF dataset, five for UCR, and six for UCI, according to dataset specifications. For lossy compression, options include PAA, PLA, FFT, BUFF-lossy, and RRD-sample. AdaEdge dynamically adjusts parameters like window size, data point budget, frequency channels, and bit count to meet specific compression ratio targets. We use CodecDB as a baseline, adapted to support lossless compression for double data types, originally designed for integers and strings. We also demonstrate TVStore's approach to lossy compression with PLA. However, it's important to note that neither CodecDB nor TVStore supports real-time selection of lossy compression based on system constraints and its effect on workload accuracy.

We employ an optimistic ϵ-greedy strategy in our experiments. Specifically, for offline mode, ϵ is set to 0.1, allowing MAB to explore sub-optimal actions more thoroughly. Conversely, for online mode experiments, ϵ is adjusted to 0.01 to enhance the exploitation of the optimal MAB action.

We conducted simulations to test system limitations on servers equipped with dual Intel(R) Xeon(R) CPUs E5-2670 2.30GHz, 128GB RAM, and 250GB HDDs, running Ubuntu 18.04 and Rust 1.49.0. Additionally, we ran experiments on an Intel NUC [1] using the same software setup with an i7-5557U CPU, 16GB RAM, and a 250GB SSD. Given that these resources exceed AdaEdge's requirements, we imposed stricter constraints on our experiments. We set hard limits in the experiments, a fixed storage budget with a threshold for recoding, and varying egress rates corresponding to different bandwidth conditions. The experiments fail if any of these constraints are breached. We limit CPU resources with 4 threads by default: one for ingestion, one for compression, one for recoding, and one for task evaluation. MAB related operations are managed by the above threads. In the scalability experiments, we employ multiple threads for the ingestion, compression, or recoding components as needed.

A. ML Accuracy with Lossy Compression

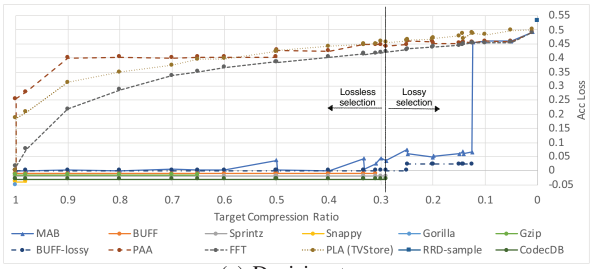
We applied prominent lossy compression techniques (BUFF, PAA, FFT, PLA) to the UCR and UCI datasets, which are widely recognized in the machine learning community and encompass approximately 250 datasets from diverse domains.

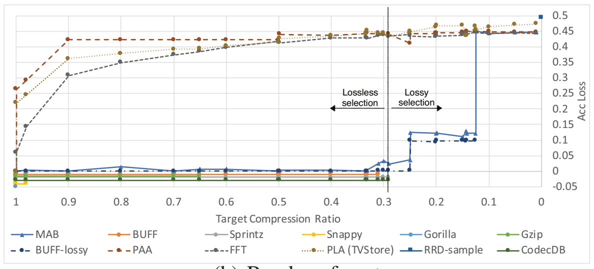
We train the models on the original dataset and then apply them to the lossy compressed data to evaluate the predicted label changes after decompressing. Due to space constraints, we present two illustrative machine learning workloads with two compression representatives. The horizontal axis represents the achieved compression ratio by the respective methods, while the vertical axis displays the relative accuracy as defined in Section IV-D. Figure 5 illustrates that tree-based models are particularly sensitive to lossy compression, where minor data alterations can lead to different branching and, consequently, altered predictions during inference. Generally, BUFF-lossy demonstrates the best performance across most compression ratios due to its minimal distortion from the original data. However, its effectiveness diminishes at lower compression ratios, as evidenced in Figure 6, where BUFF-lossy underperforms compared to FFT, PAA, and PLA at a compression ratio of approximately 0.12 in decision tree and random forest evaluations on the UCR dataset. Moreover, BUFF-lossy cannot compress our dataset when the ratio falls below 0.11.

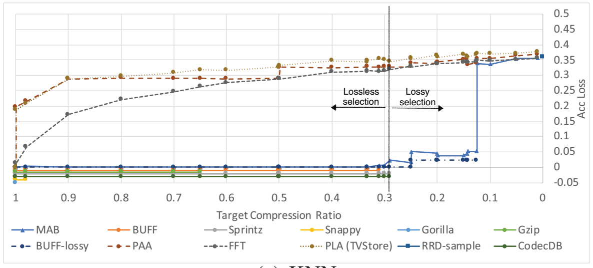
The evaluations of machine learning tasks reveal that performance significantly fluctuates with different lossy compression methods, depending on the target compression ratio and the characteristics of the input data. This variability challenges the effectiveness of simple rule-based, greedy, or heuristic compression selection approaches, which fail to accommodate the changing data features and varying workloads. Consequently, a dynamic compression selection model is essential—one that provides optimal compression recommendations tailored to the incoming data statistics, host system requirements, and the specific demands of downstream workloads.

B. Adaptive Compression Selection

Next, we assess AdaEdge on streaming data with an ongoing ingestion process. AdaEdge receives data from a dummy client that generates data points from the CBF dataset [47], a


(a) Decision tree


(b) Random forest


(c) KNN

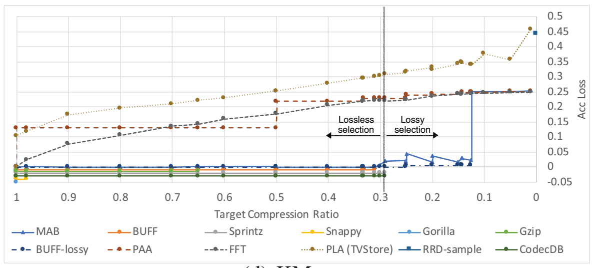

(d) KMeans

Fig. 7: Machine learning model accuracy on target compression ratio

simulated dataset with a controlled distribution. By default, the client produces data at a rate of $200,000$ data points per second. The incoming data is ingested as uncompressed segments. AdaEdge then applies compression selection to these segments based on system constraints. For comparison, we include baselines with predetermined compression combinations and, when applicable, equivalents from TVStore and CodecDB.

1) Online mode: AdaEdge online mode has a clear target compression ratio derived from the ingestion data rate and network bandwidth. AdaEdge adaptively chooses the optimal compression for a given workload. The compression process fetches the segments from the uncompressed buffer and applies compression based on the MAB component estimation.

Machine learning tasks: Figure 7 shows the machine learning accuracy loss across different target compression ratios. The horizontal axis is the target compression ratio derived by system constraints, and the vertical axis shows the corresponding relative accuracy loss. A smaller accuracy loss signifies better compression for the dataset. The MAB line represents AdaEdge's compression selection component. We show the lossless compression performance with solid lines and lossy baselines with dash lines. For lossy compression baselines, PAA, PLA, and FFT can handle any target compression ratio between range $[1.0, 0)$, while BUFF-lossy does not support a compression ratio below 0.125 on the CBF dataset. Lossless compression incurs no accuracy loss within its limited workable range, consistently resulting in zero accuracy loss. In the figure, we depict negative values for various lossless compressions to enhance visual distinction. *AdaEdge uses optimal lossless compression when possible and falls back to best lossy compression otherwise.* From Figure 7, *MAB-based compression selection always selects the optimal compression for each target compression ratio.* It selects BUFF-lossy for compression range above 0.125 and choose either PAA or FFT for other compression ratio range. The variation on the MAB line comes from the exploration cost of MAB, in which sub-optimal action is taken, thus inflating the accuracy loss. Conversely, CodecDB is effective only within a narrow range of target ratios for lossless compression and is otherwise ineffective. KVStore's PLA typically underperforms.

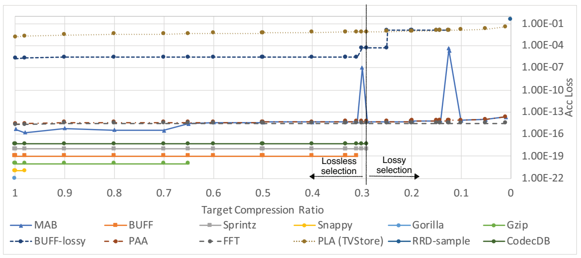


Fig. 8: Sum query over target compression ratio

Aggregation tasks: We also evaluated AdaEdge's encoding selection performance on aggregation queries. Figures 8 and 9 display the relative aggregation accuracy loss for each compression method. Due to the minimal accuracy loss, we employ a logarithmic scale for the vertical axis. As in the previous experiment, we represent all lossless baselines with values less than $1.00e-18$ for clarity, noting that lossless compression incurs no accuracy loss within its limited workable range. *AdaEdge effectively selects PAA or FFT as the optimal compression for sum aggregations, aligning with the ground truth.* The spikes in the MAB lines result from the MAB exploration steps, accentuated by the logarithmic scale which exaggerates the impact of suboptimal actions on a small number of segments. *For max aggregation queries, AdaEdge consistently chooses PLA due to its superior performance.* CodecDB, however, is only effective for lossless compression within a limited range of target ratios and fails outside this range. KVStore is suitable solely for max aggregation queries.

Complex workload targets: AdaEdge readily optimizes for a single target. In subsequent experiments, we explore

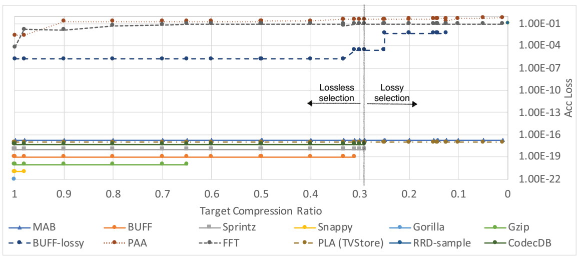


Fig. 9: Max query over target compression ratio

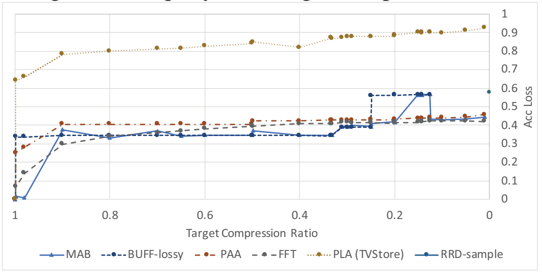


Fig. 10: Sum aggregation query and random forest complex optimization target over target compression ratio with weighted value $w_1 = 0.625$ and $w_2 = 0.375$.

compression selection for complex workload targets with multiple objectives. For time series data systems, both aggregation query accuracy and machine learning task accuracy are often crucial. AdaEdge accommodates complex optimization targets that account for both, offering a knob to balance the two.

$$A = \arg\max_a \left(w_1 \times Acc_{agg}(a) + w_2 \times Acc_{ML}(a)\right)$$

Figure 10 shows the sum aggregation query and random forest complex optimization target for the online mode where a clear target compression ratio can be derived from the system constraints. According to the baselines, there are two crossover points for the complex optimization target: the first is a compression ratio of around 0.8, and the second is around 0.25. FFT is the ground truth optimal compression in the range 1 to 0.8, followed by BUFF-lossy from 0.8 to 0.25, after which FFT regains its optimality from 0.25 to 0. According to Figure 10, *AdaEdge adapts to optimal compression across most ranges.* The outlier around compression ratio 0.15 is because of the variation from the sum aggregation query. Alternatively, the PLA implementation by KVStore exhibits the least favorable performance in this scenario.

Accuracy is not the only performance dimension for system evaluation. Data systems care about compression speed in addition to task accuracy. AdaEdge also supports the balanced optimization target between those two. Figure 11 shows the compression selection performance with the optimization target combining compression speed and task accuracy.

$$A = \arg\max_a \left(w_1 \times C_{thr}(a) + w_2 \times Acc_{ML}(a)\right)$$

Figure 11 shows a complex target of speed and accuracy, with higher values being preferable. *Notably, AdaEdge's MAB effectively selects the optimal compression.* We can see a crossover point at 0.25 between PAA and BUFF-lossy, and AdaEdge handles those cases well. In contrast, KVStore's PLA implementation underperforms in this context.

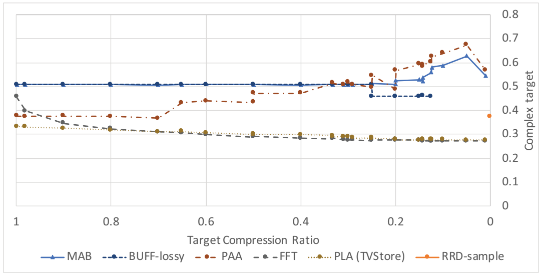


Fig. 11: Compression speed and random forest complex optimization target over target compression ratio with weights $w_1 = 0.524$ and $w_2 = 0.476$.

2) *Offline mode:* The offline mode of AdaEdge aims to preserve workload-critical information to the greatest extent possible within the constraints of the system's storage budget. Thus, the compression process continuously retrieves segments from the uncompressed buffer and applies varying levels of compression based on each segment's importance.

In this experiment, we allocate 10MB of space for AdaEdge to ingest 80MB (10 million) data points with a recoding threshold set at 0.8, meaning recoding is triggered to free up space when space usage reaches 0.8. An LRU-based compression policy is employed in the experiments. The horizontal axis in Figure 12 and Figure 13 represents the ingestion timestamp, while the two vertical axes show space usage and machine learning task accuracy loss respectively. During offline ingestion, there are essentially two phases of encoding selection: lossless compression, which aims to minimize the compression ratio and maximize space savings, and lossy compression, which focuses on optimizing accuracy for machine learning tasks. We also include baselines comprising all possible compression pairs, denoted as $lossless\_lossy$. In total, 25 baselines cover all compression pair combinations. We display only top-performing representatives to maintain readability. The term mab_mab represents our solution, which employs MAB-based encoding selection strategies. Space usage is periodically monitored, indicated by the red lines.

All compression pairs in Figures 12 and 13 manage to keep space usage within the safe limit set by the recoding threshold, shown by the red lines. The MAB space usage curve has a gentler slope compared to the Gzip, Snappy, or Gorilla baselines in Figure 13. This is because the MAB lossless compression selects Sprintz, which yields the smallest file size. CodecDB also selects Sprintz for lossless compression but fails upon reaching the recoding budget, lacking support for lossy compression to further free up space with minimal impact on task accuracy. The blue line to the bottom shows the KMeans task accuracy loss, which increases when the space usage line reaches the recoding threshold where the lossy recoding process is initialized. The slower the task accuracy loss line increases, the better compression it is. *AdaEdge MAB-based lossy compression selection is always the best to choose the optimal compression to minimize task accuracy loss.*

In this experiment, MAB-based lossy compression selection initially picks BUFF-lossy as optimal lossy compression and

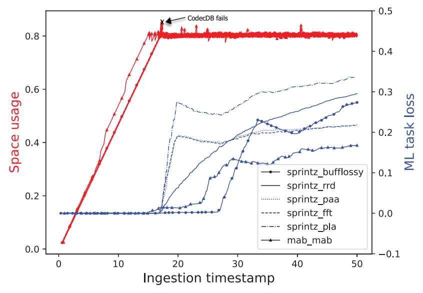


Fig. 12: KMeans accuracy loss over ingestion time for baselines $Sprintz\_X$

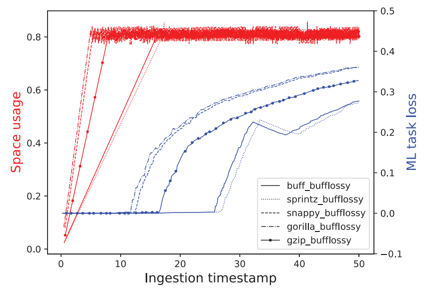


Fig. 13: KMeans accuracy loss over ingestion time for baselines $X\_bufflossy$

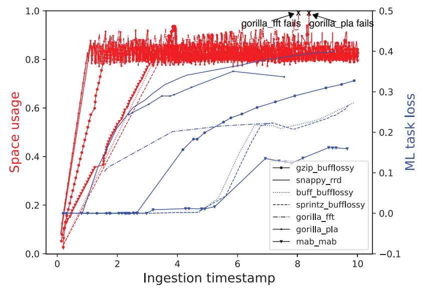


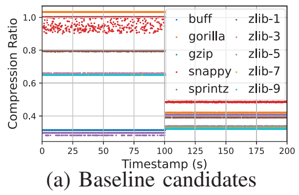
Fig. 14: KMeans accuracy loss over high-frequency signal (1 million points/s)

switches to PAA when BUFF-lossy fails to compress the segment further. Baselines in Figure 12 show that the accuracy loss stays low when recoding first starts and increases quickly when more data is ingested, requiring more aggressive compression for old data. BUFF-lossy fails and falls back to RRD-sample to further compress the segments in the late phase of the ingestion experiment. In addition to KMeans, we also test AdaEdge offline mode on other aggregation and ML tasks. *The experiments demonstrate AdaEdge's adaptability in finding the best solution for aggregation queries and various machine learning tasks, including KNN, random forest, and decision tree.* Beyond general accuracy improvements, we consistently evaluate the task accuracy of the recent data segments. Due to the LRU-based compression policy, AdaEdge consistently delivers 100% accuracy for these fresh data segments.

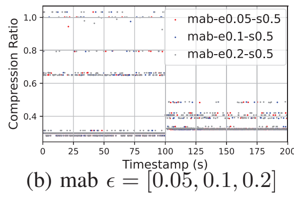
Previous experiments showcase scenarios with relatively modest ingestion rates, where all compression baselines successfully managed the ingested signals within the given storage budget, and AdaEdge delivered the optimal compression solution. Figure 14 shows the ingestion experiments with high-frequency signal (1 million points per second). We include the top compression combinations (e.g., $gzip\_bufflossy$, $buff\_bufflossy$ and $sprintz\_bufflossy$), which perform similarly to their counterparts' performance on a lower frequency signal in Figure 12 and 13 but in a smaller time scale. AdaEdge consistently selects the most appropriate compression method that minimizes accuracy loss while staying feasible. However, several compression pairs could not keep space usage within the storage budget threshold of 0.8, ultimately exceeding the total storage capacity. For instance, combinations like $gorilla\_fft$ and $gorilla\_pla$ exceeds the storage budget at 8.0 seconds and 8.4 seconds, respectively, failing to complete the ingestion task. The Gorilla-based pairs underperform because Gorilla decompression was more time-consuming than other baselines, delaying the recoding process. Such bottlenecks could potentially be alleviated by assigning additional threads to the recoding component.

C. Robustness against data shifts and hardware variability

Here, we test AdaEdge on smaller edge-class hardware. We use a synthetic dataset consisting of half high-entropy data from the CBF dataset and half randomly generated low-entropy data. We set the optimization goal to minimize space usage. We also doubled the decision space by including more



(a) Baseline candidates



(b) mab $\epsilon = [0.05, 0.1, 0.2]$

Fig. 15: AdaEdge choose the best compression solution with shifting workload at a rate 100k points per second

compression candidates shown in Figure 15a. Figure 15b shows AdaEdge still converges to the optimal solution that starts with Sprintz and then switches to gzip or zlib-9 for the second half, even with a large decision space. The ϵ value directly affects the exploration rate, but it does not prevent AdaEdge from finding the best solution. A sub-optimal exploration may result in space overhead, which can be fixed or alleviated later by a recoding step if applicable. We also ran the experiments by tuning the nonstationary step value, where the result shows that a larger step value results in a more swift change of choice with data distribution. To better balance the overall performance, we used MAB $\epsilon = 0.1$ and $step = 0.5$ as the default in AdaEdge for the cases with data shift.

This demonstrates AdaEdge's capability in handling ingestion tasks with a single compression and recoding thread, consistently selecting the most effective compression approach. AdaEdge exhibits strong scalability when ingesting multiple signals, supporting concurrent compression and recoding processes. In our scalability tests, AdaEdge successfully managed an ingestion rate of approximately 8 million points per second using 8 threads while adhering to the system's constraints.

VI. Conclusion

AdaEdge offers a robust compression selection framework tailored for data systems with defined compression ratios or storage budgets. It accommodates singular and multifaceted optimization objectives, allowing users to customize targets specific to their system needs. AdaEdge compresses data based on usage frequency and selects between lossy and lossless compression methods, accounting for hardware limitations and workload requirements.

Acknowledgments This work was in part supported by NSF Award IIS-2048088, Google DANI Award, and Cisco Systems.

References

[1] Breakthrough performance at the edge. https://cdrdv2-public.intel.com/780985/nuc-13-compute-element-product-brief.pdf. (Accessed on 02/25/2024).

[2] Influxdb - open source time series, metrics, and analytics database. https://www.influxdata.com/.

[3] Moving to the edge is crucial for oil and gas companies to make better use of data. https://www.forbes.com/sites/markvenables/2019/05/31/moving-to-the-edge-is-crucial-for-oil-and-gas-companies-to-make-better-use-of-data/?sh=854c34c59bd9. (Accessed on 11/11/2023).

[4] Time-series data simplified. https://www.timescale.com.

[5] D. Abadi, S. Madden, and M. Ferreira. Integrating Compression and Execution in Column-oriented Database Systems. In *SIGMOD*, pages 671–682, 2006.

[6] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682, 2006.

[7] N. Agrawal and A. Vulimiri. Low-latency analytics on colossal data streams with summarystore. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 647–664, 2017.

[8] Y. An, Y. Su, Y. Zhu, and J. Wang. TVStore: Automatically bounding time series storage via Time-Varying compression. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 83–100, Santa Clara, CA, Feb. 2022. USENIX Association.

[9] Apache Foundation. Apache Arrow Feather. https://arrow.apache.org/docs/python/feather.html.

[10] Apache Foundation. Apache ORC. https://orc.apache.org.

[11] Apache Foundation. Apache Parquet. https://parquet.apache.org/.

[12] K. C. Barr and K. Asanović. Energy-aware lossless data compression. *ACM Transactions on Computer Systems (TOCS)*, 24(3):250–291, 2006.

[13] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *ACM SIGMOD*, pages 283–296. ACM, 2009.

[14] D. Blalock, S. Madden, and J. Guttag. Sprintz: Time series compression for the internet of things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–23, 2018.

[15] M. Boissier. Robust and budget-constrained encoding configurations for in-memory database systems. *Proceedings of the VLDB Endowment*, 15(4):780–793, 2021.

[16] L. Cen, A. Kipf, R. Marcus, and T. Kraska. Lea: A learned encoding advisor for column stores. In *Fourth Workshop in Exploiting AI Techniques for Data Management*, pages 32–35, 2021.

[17] W. Chu, L. Li, L. Reyzin, and R. Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

[18] P. DuBois. *MySQL*. Pearson Education, 2008.

[19] A. Dziedzic, J. Paparrizos, S. Krishnan, A. Elmore, and M. Franklin. Band-limited training and inference for convolutional neural networks. In *International Conference on Machine Learning*, pages 1745–1754. PMLR, 2019.

[20] O. Evensen and D. Womack. How ai can pump new life into oilfields. *IBM Corporation. Available online: https://www. ibm. com/downloads/cas/5BNKGNLE (accessed on December 2021)*, 2020.

[21] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[22] A. D. Flaxman, A. T. Kalai, and H. B. McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004.

[23] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Modelardb: modular model-based time series management with spark and cassandra. *Proceedings of the VLDB Endowment*, 11(11):1688–1701, 2018.

[24] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Scalable model-based management of correlated dimensional time series in modelardb+. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1380–1391. IEEE, 2021.

[25] S. K. Jensen and C. Thomsen. Holistic analytics of sensor data from renewable energy sources: a vision paper. In *European Conference on Advances in Databases and Information Systems*, pages 360–366. Springer, 2023.

[26] H. Jiang, C. Liu, Q. Jin, J. Paparrizos, and A. J. Elmore. Pids: attribute decomposition for improved compression and query performance in columnar storage. *Proceedings of the VLDB Endowment*, 13(6):925–938, 2020.

[27] H. Jiang, C. Liu, J. Paparrizos, A. A. Chien, J. Ma, and A. J. Elmore. Good to the last bit: Data-driven encoding with codecdb. In *Proceedings of the 2021 International Conference on Management of Data*, pages 843–856, 2021.

[28] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

[29] S. Krishnan, A. J. Elmore, M. Franklin, J. Paparrizos, Z. Shang, A. Dziedzic, and R. Liu. Artificial intelligence in resource-constrained and shared environments. *ACM SIGOPS Operating Systems Review*, 53(1):1–6, 2019.

[30] M. Kuschewski, D. Sauerwein, A. Alhomssi, and V. Leis. Btrblocks: Efficient columnar compression for data lakes. *Proceedings of the ACM on Management of Data*, 1(2):1–26, 2023.

[31] W. Lang, M. Morse, and J. M. Patel. Dictionary-based compression for long time-series similarity. *IEEE transactions on knowledge and data engineering*, 22(11):1609–1622, 2009.

[32] R. Li, Z. Li, Y. Wu, C. Chen, and Y. Zheng. Elf: Erasing-based lossless floating-point compression. *Proceedings of the VLDB Endowment*, 16(7):1763–1776, 2023.

[33] P. Liakos, K. Papakonstantinopoulou, and Y. Kotidis. Chimp: efficient lossless floating point compression for time series databases. *Proceedings of the VLDB Endowment*, 15(11):3058–3070, 2022.

[34] X. Liang, Z. Shang, S. Krishnan, A. J. Elmore, and M. J. Franklin. Fast and reliable missing data contingency analysis with predicate-constraints. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 285–295, 2020.

[35] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

[36] C. Liu, H. Jiang, J. Paparrizos, and A. J. Elmore. Decomposed bounded floats for fast compression and queries. *Proceedings of the VLDB Endowment*, 14(11):2586–2598, 2021.

[37] C. Liu, A. Pavlenko, M. Interlandi, and B. Haynes. A deep dive into common open formats for analytical dbmss. *Proceedings of the VLDB Endowment*, 16(11):3044–3056, 2023.

[38] C. Liu, M. Umbenhower, H. Jiang, P. Subramaniam, J. Ma, and A. J. Elmore. Mostly order preserving dictionaries. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1214–1225. IEEE, 2019.

[39] A. Mahajan and D. Teneketzis. Multi-armed bandit problems. In *Foundations and applications of sensor management*, pages 121–151. Springer, 2008.

[40] Micro Focus International plc. Vertica Encoding Types. https://www.vertica.com/docs/9.2.x/HTML/Content/Authoring/SQLReferenceManual/Statements/encoding-type.htm.

[41] S. Nakandala, A. Kumar, and Y. Papakonstantinou. Incremental and approximate inference for faster occlusion-based deep cnn explanations. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1589–1606, 2019.

[42] J. Paparrizos, I. Edian, C. Liu, A. J. Elmore, and M. J. Franklin. Fast adaptive similarity search through variance-aware quantization. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2969–2983. IEEE, 2022.

[43] J. Paparrizos and M. J. Franklin. Grail: efficient time-series representation learning. *Proceedings of the VLDB Endowment*, 12(11):1762–1777, 2019.

[44] J. Paparrizos, C. Liu, B. Barbarioli, J. Hwang, I. Edian, A. J. Elmore, M. J. Franklin, and S. Krishnan. Vergedb: A database for iot analytics on edge devices. In *CIDR*, 2021.

[45] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[46] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll. Opc ua versus ros, dds, and mqtt: Performance evaluation of industry 4.0 protocols. In *2019 IEEE International Conference on Industrial Technology (ICIT)*, pages 955–962. IEEE, 2019.

[47] N. Saito. *Local feature extraction and its applications using a library of bases*. Yale University, 1994.

[48] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[49] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545. IEEE, 1996.

[50] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[51] Tobi Oetiker. Rrdtool: round robin database tool. http://oss.oetiker.ch/rrdtool/.

[52] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The cartographic journal*, 30(1):46–51, 1993.

[53] W. Weber, R. Cesarone, R. Miller, and P. Doms. A view of the future of nasa's deep space network and associated systems. In *SpaceOps 2002 Conference*, page 33, 2002.

[54] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. VLDB, 2000.