Training K-means on Embedded Devices: a Deadline-aware and Energy Efficient Design

Hafsa Kara Achira*†, Camélia Slimani *, Jalil Boukhobza*
*ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, F-29200 Brest. France
† École Nationale Supérieure D'Informatique (ESI), Algiers, Algeria
Email: ih_karaachira@esi.dz, camelia.slimani@ensta-bretagne.fr, jalil.boukhobza@ensta-bretagne.fr

***Abstract*—With the surge in data production, Machine Learning techniques are now commonly used to build intelligent models. Traditionally, powerful platforms process data collected from endpoint devices. However, to address security threats and minimize communication traffic, models can be learned near endpoint devices, despite their resource shortage. K-means clustering is among the most common machine learning tasks used for embedded applications. Because the system is running on scarce resources, the learning process needs to obey a certain time limit. Even if current implementations of K-means have been optimized for embedded devices, they do not consider running within a predefined time budget. In this paper, we propose a deadline-aware and energy-efficient version of K-means called Embedded K-means (EK-means) [1], that relies on two main ideas : (1) smartly select the right subset of data to train on to meet the deadline at the expense of the smallest clustering error possible ; (2) by dropping part of the data, slack times are identified and exploited opportunistically to apply Dynamic Voltage and Frequency Scaling techniques (DVFS) so as to decrease the energy consumption of the learning task. EK-means has been built on top of an I/O optimized version of K-means for embedded devices to maintain a low I/O proportion regardless of memory constraints. EK-means allows to cluster data while meeting more than 98% of the deadlines with a loss of 1.43% of clustering quality, and an energy reduction of up to 84.26%.**

***Index Terms*—K-means, Edge Intelligence, DVFS, Real-Time constraint, I/Os, secondary storage.**

I. Introduction

Internet of Things (IoT) Technology has resulted in the rapid growth of data that need to be processed to extract meaningful knowledge. For this reason, machine learning (ML) techniques are now widely used. Traditionally, data were sent to and processed on powerful platforms to address ML technique calculation and memory requirements. However, moving data to remote platforms exposes them to security vulnerabilities and increases the traffic and energy consumption of the network on the collection devices [15]. Edge Intelligence, is then, emerging to enable learning near data collection devices. [15].

Edge intelligence presents several challenges due to resource constraints of embedded devices. Memory limitations are particularly troublesome when the volume of training data cannot fully reside in memory, leading to swapping on secondary storage [3], [28]–[31]. Meanwhile, the energy limitations inherent in these devices emphasize the need for applications that are both fast and energy efficient [23].

[1]The source code is available on https://github.com/HafsaKaraAchira/EK-means-Embedded-K-means-.git

One of the most popular clustering ML algorithms used in IoT is K-means [2] [24]. It aims to partition similar data together by assigning them to clusters represented by their mean value which is helpful for applications like anomaly detection [22] and image segmentation [8]. Considering the execution deadline in K-means learning phase is important in various application scenarios. In real-time data analysis [4] and interactive systems [6], meeting the execution deadline is crucial for immediate responses and enhanced user experience. Furthermore, obtaining clustering models quickly is essential when K-means is used as a preprocessing step before applying other AI models, such as image processing and pattern recognition tasks preceding convolutional networks [8].

A notable limitation of K-means is its reliance on scanning the entire data set during each iteration, added to the need to execute an indeterminate number of iterations to converge to its final solution. This becomes problematic when dealing with large data sets that exceed available memory, leading to an I/O-bound execution. To address this, the K-MLIO algorithm (for K-means with Low I/O Overhead) [3] adopts a divide-and-conquer approach. It divides the data set into chunks that can fit in the memory workspace, performs independent K-means clustering on each chunk, and combines the results. This way, K-MLIO reads the data set only once, regardless of size, memory, and data set. It reduces the I/O cost while maintaining similar precision with the original K-means [3]. While previous studies investigated how to optimize the K-means inference and training process, to the best of our knowledge, no study focused on training a model in a predefined amount of time on resource-constrained devices on the edge.

In this paper, we designed a version of the K-means that runs within a time budget and that can opportunistically reduce energy consumption with as slight as possible impact on the quality of the clustering. Our approach is based on the assumption that it is possible to consider only a subset of data to meet the time constraint with only a minor increase in clustering error. Inspired from the work done on K-MLIO, we considered the per-chunk version of the K-means to propose a method that incorporates two main strategies: (1) executing an online estimation of the learning process execution time on each chunk in order to estimate the volume of data to drop to meet the deadline and (2) adjusting the processor's frequency by opportunistically reclaiming slack times to reduce the frequency after processing each chunk.

The remainder of the paper is structured as follows. In section II, we provide some background about K-MLIO and energy basic concepts. Section III motivates the study. Section IV describes the proposed solution. Section V presents some experiments. Section VI discusses some threats to the validity of the study. Section VII highlights some related work and Section VIII concludes the paper and gives some perspectives.

II. Background

As our contribution reuses the concept of divide-and-conquer employed by K-MLIO to optimize the K-means from an I/O perspective, in this section we give a brief description of the strategy. In addition, we give some background information about Dynamic Voltage and Frequency Scaling (DVFS) as it was used for energy efficiency sake. Table I summarizes the notations used in the rest of this paper.

TABLE I
NOTATION TABLE

Notation	Description
K	Number of clusters
N	Data set size
M	Chunk size
s	Data set element size
i	Chunk index
it	Number of iterations to converge
it_{max}	Maximum number of iterations
T_{load}	One chunk loading time
T_{init}	K-means initialization time
T_{it}	K-means one iteration time
T_{chunk_i}	Chunk i processing time
T_{alloc}	Time allocated for one chunk
$WCET$	Worst-Case Execution Time
C_{max}	Worst-Case Execution processor cycles
$skip\_chunk$	Number of chunks to drop
f	CPU frequency

A. K-MLIO overview [3]

The main idea behind K-MLIO is to employ a divide-and-conquer approach. It divides the data set into subsets of M elements that can fit into main memory (called chunks), and applies K-means on each of them. The obtained results are combined by grouping similar obtained clusters and forming a final chunk that is representative of the data set. K-means is applied to this representitive chunk to derive the final model.

K-MLIO steps are shown in Fig. 1 and are the following :

(1) Chunk Partial Clustering Step: Each chunk is loaded into memory and undergoes K-means clustering, resulting in K centroids for each chunk. (Figure 1, steps 1 and 2).

(2) Grouping Clusters Step: Similar clusters are grouped together. This aggregation provides valuable information about the distribution of elements that are likely to belong to the same cluster across different chunks. Each partial centroid is associated with a group (Fig. 1, step 3).

(3) Forming the Final Chunk Step: A representative sample chunk of a size that matches the memory workspace, is created by selecting elements based on the overall group composition (Fig. 2, step 4).

(4) Final Chunk Clustering: The K-means algorithm is applied to the final chunk, resulting in the final clustering solution for the entire data set (Figure 1, step 5).

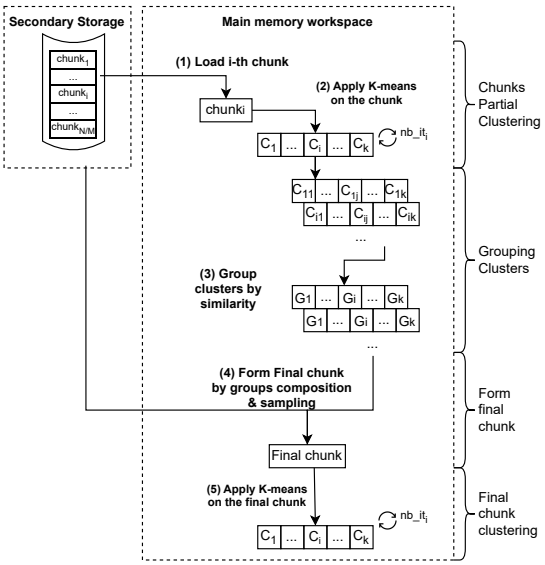

Fig. 1. K-MLIO overview

B. Energy basic concepts

Energy consumption refers to the amount of power used during a specific time interval. It can be calculated by integrating the power over time using eq. (1):

$$E = \int_0^T P(t)\,dt \tag{1}$$

Power, on the other hand, represents the rate at which energy is consumed. Power can be classified into static power, also known as leakage power, which refers to power consumption when there is no circuit activity, and dynamic power, which is the power dissipated by the circuit during the charging and discharging of capacitors.

$$P = P_{static} + P_{dyn} \tag{2}$$

Dynamic power can be expressed as:

$$P_{dyn} = C_{eff} \cdot V^2 \cdot f \tag{3}$$

Where C_{eff} is the circuit effective capacitance, V is the supplied voltage and f the clock frequency.

For CPU-bound instructions, the execution time T_{on} spent by the processor represents the time required to perform instructions CPU cycles C with a frequency f :

$$T_{on} = C/f \tag{4}$$

This model is simple as it does not account for cache hierarchy, but it is commonly used for simple analysis.

DVFS is a power management technique used to adjust the operating voltage and frequency of the processor dynamically [13]. DVFS adjusts voltage levels using fixed discrete voltage steps, corresponding to specific frequency values. These pairs of voltage and frequency are referred to as P-states. The processor frequency is considered proportional to the voltage [16]. The dynamic energy is then :

$$E_{dyn} = K \cdot f^3 \cdot t \tag{5}$$

With K being a constant parameter. Scaling down the frequency reduces power consumption but increases execution time while boosting the frequency enhances performance but increases power dissipation.

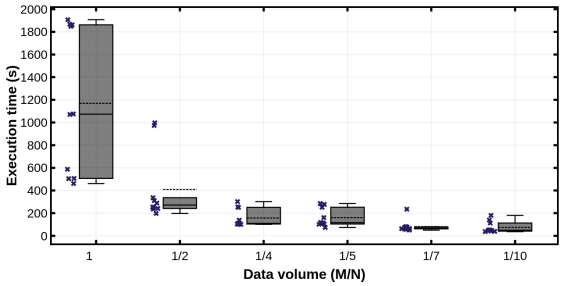

Fig. 2. K-MLIO chunk execution time per memory constraint

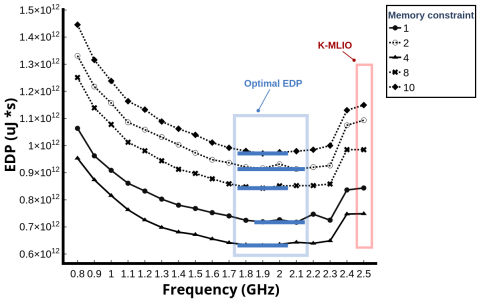

Fig. 3. K-MLIO EDP metric analysis

Several metrics are used to assess the energy efficiency of an algorithm design. One of the most commonly used is the EDP (Energy-Delay Product) metric that takes into account both energy consumption and processing delay, aiming to find the optimal frequency that lowers $EDP(f_{opt})$ value to indicate a more efficient design [13]. It can be further adapted to platform constraints or specific needs, such as the $EDPC(freq_{opt})$ (Energy-Delay Product with delay constraint C) metric [13], which selects the most energy-efficient operating point that still completes the process within a given time constraint C.

III. Motivation

To motivate our study, we performed an experimental analysis of K-MLIO to outline two phenomena: 1) one cannot simply reduce the data set to bound the training time, 2) DVFS can actually help reduce the consumed energy.

First, we measured the average execution time of K-MLIO under different memory constraints (N/M), which stands for the data set volume as compared to the available memory work-space, applied on a synthetic data set (see Section V-A for experimental setup). To observe the effect of data volume on the convergence speed, we ran K-MLIO with decreasing volumes of data. In the experiment, we used the following proportions of data as compared to the overall data-set size: 1, 1/2, 1/4, 1/5, 1/7, 1/10. We repeated the measures 10 times to average execution time values. The results are given in Fig. 2.

Second, we measured K-MLIO execution time and energy consumption using PyJoules API [21] with different memory constraints and frequencies. The results are shown in Fig. 3

From the two experiments, we can draw two observations : 1) The execution time is not simply correlated to the data set size (see boxplot on Fig. 2), and it does not necessarily decrease with the reduction of data to process. In fact, it is due to the versatile character of centers' initialization quality of K-means which impacts the number of iterations necessary for partial clustering steps to converge. This unpredictability can hinder meeting deadlines for a strategy solely based on the data set size. 2) For different memory constraint executions, minimal EDP metric measurements were obtained with execution frequencies between 1.8Ghz and 2Ghz that are inferior to the maximum frequency 2.6Ghz which is systematically used with K-MLIO. Thus, frequency can be leveraged to achieve a favorable energy-delay trade-off (see Fig. 3).

IV. Embedded K-means (EK-means)

In this section, we describe EK-means. We start by formulating the energy and execution time of K-MLIO to highlight the levers on which we can act. Then, we give an overview of the proposed method before describing it in more details.

A. Preamble: K-MLIO energy and delay analysis

The overall K-MLIO execution time can be expressed as :

$$T_{K-MLIO} = \sum_{i=1}^{\lceil N/M \rceil + 1} (T_{chunk_i}) \tag{6}$$

It can be broken down into the sum of chunks processing times. The latter can, in turn, be split into the sum of chunk loading time (T_{load}), the initialization time of the K-means (T_{init}), and the iteration time of the K-means, where T_{it} stands for one iteration time of the K-means and it_i the number of iterations required for convergence for this chunk. it_i is bound by the user-defined maximum number of iterations that K-means can perform, it_{max} ($it_i \leq it_{max}$).

$$T_{chunk_i} = T_{load} + T_{init} + T_{it} \cdot it_i \tag{7}$$

For the final chunk processing ($T_{chunk_{\lceil N/M \rceil + 1}}$), in addition to chunk processing time, all chunks are read once again to sample points.

$$T_{chunk_{\lceil N/M \rceil + 1}} = T_{load} \cdot \lceil N/M \rceil + T_{init} + T_{it} \cdot it_{\lceil N/M \rceil + 1} \tag{8}$$

So, the worst-case execution time $WCET$ for a given chunk is reached when the number of iterations before convergence reaches its maximum it_{max}.

$$WCET_{chunk_i} = T_{load} + T_{init} + T_{it} \cdot it_{max} \tag{9}$$

The WCET of K-MLIO happens when the maximum number of iterations is reached for each chunk. It can be expressed from equations (6) to (9), as:

$$WCET_{K-MLIO} = \lceil N/M \rceil \cdot WCET_{chunk_1} + WCET_{chunk_{\lceil N/M \rceil + 1}} \tag{10}$$

$$WCET_{K-MLIO} = 2 \cdot T_{load} \cdot \lceil N/M \rceil + (\lceil N/M \rceil + 1)(T_{init} + T_{it} \cdot it_{max}) \tag{11}$$

Eq. (11) shows that two levers can exploited to reduce the execution time or the energy consumption of K-MLIO : (1) Reduce the time by reducing the number of chunks $\lceil N/M \rceil$: if the number of chunks is decreased, the overall execution time decreases at the expense of clustering error. (2) Reduce energy by reducing the frequency when processing a chunk: frequency modulation allows one to control the execution time of an

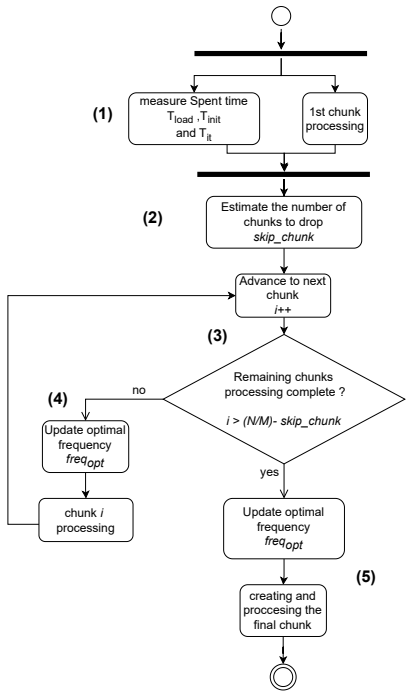

Fig. 4. EK-means overview

iteration T_{it}. Lowering the frequency decreases the energy consumption at the expense of a larger execution time.

Our contribution is based on these two principles. It ensures deadline satisfaction first and tries, whenever possible, to reduce the frequency to decrease energy consumption.

B. EK-means Overview

In this section, we present an overview of EK-means (see Fig. 4). Our approach relies on these two principles :

(1) Dropping a certain number of chunks to meet the deadline : it consists in estimating the necessary time to process one chunk, which allows to estimate the WCET (see Eq. 10) necessary to complete K-MLIO at the maximum frequency. The number of chunks to drop to meet the deadline is, then, deduced (see (1) and (2) in Fig. 4).

(2) Opportunistic adjustment of the processor frequency: If the number of iterations effectively performed during a chunk processing is less than it_{max},the chunk processing is expected to complete before the predicted time limit. This slack time is exploited to reduce the frequency to an optimal frequency f_{opt}, thus reducing energy consumption. Otherwise, the execution continues at the maximum frequency f_{max} to ensure that the deadline is met. (see (3) and (4) of Fig. 4). Note that the final chunk is excluded from the chunks candidate to be dropped since it plays a key role in calculating the final results of K-MLIO (see Fig. 4).

C. EK-means Description

In what follows, we describe each EK-means step.

(1) Chunk delay online analysis (step 1 in Fig. 4): we start by measuring the first chunk execution time with the maximum frequency, say $T_{chunk_1}(f_{max})$. This step allows us to deduce two information:

1− The time spent by the processor waiting for chunk to be loaded T_{load}. As stated in Section II, this time is assumed to be independent from the processor frequency as it represents the needed I/O to load the data chunk ;

2− The number of CPU cycles performed during one iteration C: the rest of the time spent on the first chunk, say T_{on}, after removing T_{load} (see eq. 7), is related to the initialization T_{init} and processing for each iteration T_{it}:

$$T_{on} = T_{init} + T_{it} \cdot it_i \tag{12}$$

By measuring T_{load}, T_{init} of the first chunk and the number of iterations effectively performed, and according to eq. (4) we can deduce the number of cycles required for one iteration of chunk processing.

(2) Estimation of the number of chunks to drop (step 2 in Fig. 4): we use the information inferred from step 1 to estimate the WCET of the remaining chunks $WCET_{chunk_i}$ according to eq. (9). We aim to meet a deadline $T_{deadline}$:

$$T_{K-MLIO} \leq T_{deadline} \tag{13}$$

Combined with eq. (10), to guarantee that EK-means execution will meet the deadline, we must find the minimum number of partial chunks to drop, noted $skip\_chunk$, it should satisfy:

$$\begin{aligned}(\lceil N/M \rceil - skip\_chunk) \cdot WCET_{chunk_i} + \\ WCET_{chunk_{\lceil N/M \rceil + 1}} \leq T_{deadline}\end{aligned} \tag{14}$$

Where $0 \leq skip\_chunk \leq \frac{N}{M} - 1$.
$WCET_{chunk_{\lceil N/M \rceil + 1}}$ stands for the final chunk's WCET
We deduce that :

$$skip\_chunk \geq \lceil N/M \rceil - \frac{T_{deadline} - WCET_{chunk_{\lceil N/M \rceil + 1}}}{WCET_{chunk_i}} \tag{15}$$

To minimize the clustering quality degradation, we set $skip\_chunk$ to the ceiling value that satisfies eq. (15).

(3) Applying DVFS (steps 3 and 4 in Fig. 4): Once the number of chunks to skip to meet the deadline is known, we opportunistically adjust the processor frequency to reduce energy consumption. To do so, for each chunk, we estimate the time allocated for processing it T_{alloc_i}. This time is calculated by subtracting from the deadline the measured processing time of the already processed chunks and the WCET of the last one (for which we allocate the maximum time) and dividing the whole by the number of remaining chunks to consider :

$$T_{alloc_i} = \frac{T_{deadline} - \sum_{j=1}^{i-1} T_{chunk_j} - WCET_{chunk_{\lceil N/M \rceil + 1}}}{\lceil N/M \rceil - skip\_chunk - i - 2} \tag{16}$$

The idea is to take advantage of the time freed up from previous chunks that converge before reaching the number of iterations it_{max} and reuse the slack time to reduce frequency. The slack time resulting from one chunk processing is noted:

$$slack_i = WCET_{chunk_i} - T_{chunk_i} \tag{17}$$

Essentially, if the evaluation of T_{alloc} shows that it is higher than $WCET_{chunk_i}$, it indicates the existence of an optimal frequency f_{opt}. This frequency allows for reducing energy consumption while still remaining within the allocated time T_{alloc}. In the following, we describe the development that allows us to find this frequency.

Using eq. (12) in eq. (9), we get:

$$WCET_{chunk_i} = T_{load} + T_{on} \tag{18}$$

Using eq. (4), we make the frequency appear in $WCET_{chunk_i}$ expression as follows :

$$WCET_{chunk_i} = T_{load} + \frac{C_{max}}{f} \tag{19}$$

Where C_{max} is the number of CPU cycles for the WCET. Therefore, finding the optimal frequency f_{opt} that reduces energy consumption while chunk execution time does not exceed T_{alloc} is equivalent to solving this inequality :

$$T_{load} + \frac{C_{max}}{f} \leq T_{alloc} \tag{20}$$

Which is :

$$f \geq \frac{C_{max}}{T_{alloc} - T_{load}} \tag{21}$$

The optimal frequency is, then, the operational frequency available on the processor that is immediately greater to the f lower bound, expressed as follows :

$$f_{opt} = \sup_{f_i \in \{f_{min}, \ldots, f_{max}\}} \frac{C_{max}}{T_{alloc} - T_{load}} \tag{22}$$

This calculation is repeated before each chunk processing since T_{alloc} has to be re-evaluated. In fact, if the number of iterations performed for a chunk is lower than it_{max}, the remaining time before reaching the deadline increases, which allows to better adjust frequency for the remaining chunks. **(4) Final chunk processing (step 5 in Fig. 4):** The last step of EK-means consists in constituting and processing the final chunk as in K-MLIO. The optimal frequency is identified and applied using the same process as for the other chunks.

D. Illustrative Example

In this section, we illustrate each step of EK-means through the example shown in Figure 5. Here, we consider a memory constraint $\frac{N}{M} = 10$ (that is the data set contains 10 chunks) performed by a processor having 12 operating points $freq_0 \geq freq_1 \geq \ldots \geq freq_{11}$. The execution time is limited by the deadline barrier shown in the figure.

The clustering begins with the execution of the first chunk with maximum frequency $freq = freq_0$ assuming $skip\_chunk = 0$ (Figure 5 (a) step 1). In this first step, the estimation of the number of chunks to drop to meet the deadline is run. We find that we can only process 3 more chunks plus the final chunk, so $skip\_chunk = 6$. Then, the frequency is adjusted for the remaining chunks to $freq_{opt} = freq_5$ (Figure 5 (a) step 2).

After processing the second chunk which spent all its $WCET_2(freq_5)$, applying DVFS will result in the same optimal frequency value $freq_{opt} = freq_5$ (5 (b) step 3).

The 3rd chunk converges within less than it_{max}. So, a slack time appears as an opportunity to decrease the optimal frequency when applying DVFS before the 4th chunk to $freq_{opt} = freq_7$ (Figure 5 (c) step 4).

The 4th chunk processing is also completed before reaching $WCET_4(freq_7)$. The partial clustering phase is complete by executing only 4 chunks out of 10. The slack time of the 4th chunk is exploited to decrease the frequency for the final chunk processing to $freq_{opt} = freq_{10}$ (Figure 5 (d) step 5).

V. Evaluation

This section presents an evaluation of EK-means. We first describe our methodology, then show and discuss the results.

A. Experimental methodology

1) Evaluation Metrics: For the clustering quality, we used the ARI (Adjusted Rand Index) metric [25], which measures the agreement between the clustering results and the true labels of the data points. A value close to 1 indicates a high agreement between the clustering and the true labels, while a value close to 0 suggests random assignments. The objective of EK-means is to provide the best clustering possible as compared to K-means, while meeting the time deadline.

For energy efficiency, we relied on the frequencies used to infer an estimation based on an analytic model.

2) Experiments : A series of experiments were conducted to assess the effectiveness of EK-means.

Experiment 1, clustering quality and deadline meeting: to estimate the clustering quality of EK-means, we calculated the ARI reduction it incurs as compared to K-MLIO on several execution scenarios. We set the memory constraint $\lceil N/M \rceil$ to 10 (that is the data set is 10x larger than the memory workspace). The deadlines utilized for each test case are relative to their respective K-MLIO worst-case execution time $WCET_{K-MLIO}$ (which is higher than the actual execution time of K-MLIO as it hardly uses the maximum iterations for each chunk). The used deadline proportions relative to $WCET_{K-MLIO}$ are as follows: $\{2/7, 3/7, 4/7, 5/6, 6/7, 1\}$. In order to better observe the impact of skipping chunks on the final clustering quality, we used 3 data sets with different degrees of overlap described hereunder. All measurements were ran 5 times, and the obtained results were averaged. We also measured the execution time of EK-means to compare them to the imposed deadlines, to assess deadlines satisfaction.

Experiment 2, energy consumption : for this experiment, we estimated the energy consumption reduction performed by EK-means as compared to K-MLIO. The deadlines imposed represent the following proportions to the $WCET$: $\{2/7, 3/7, 4/7, 5/7, 6/7, 1\}$. These proportions (especially those that are close to 1) might give deadlines that are actually higher than K-MLIO effective execution time, since WCET is calculated with the assumption that all the chunks run the maximum number of iterations. We knowingly allowed it to occur to observe the frequency modulation behavior of EK-means when not timely constrained.

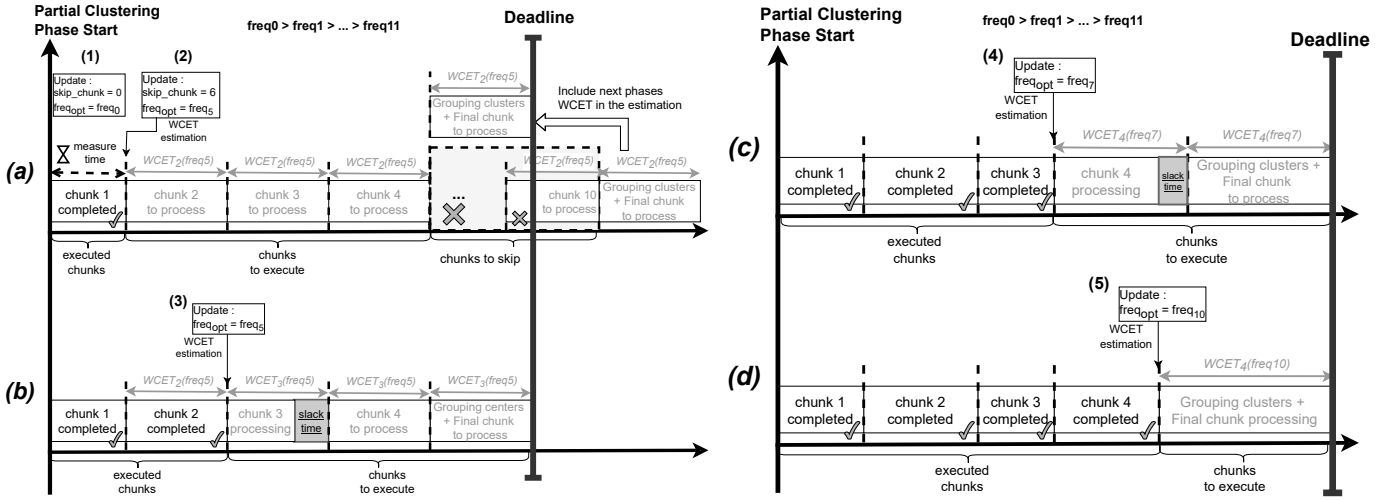

Fig. 5. EK-means illustrative example (at each timeline (a),(b),(c), and (d): the grey boxes represent chunks to be processed in the future with an estimation of its processing time to the value of the WCET. The black boxes with a tick symbol represent chunks processed already.)

3) Experimental data sets: We performed experiments on synthetic data sets generated using the *ClusterGeneration* package [26] of the R platform. This package allows the generation of a data set of N elements that belong to K clusters in a given dimension. We set the number of dimensions to 10 and the number of clusters to 10. We used data sets of size 256MB, and applied a memory constraint of $\lceil N/M \rceil = 10$, by setting the work-space size to 75MB to contain both a chunk of data (the size of the M observation is 25.6MB) and the intermediate data structures necessary to K-MLIO execution.The cluster separation index (sepVal) is a parameter that allows to tune the degree of overlap between data. It is between -1 and 1: The closer to 1, the more separated are the clusters. We set the index to 0.2 for separated clusters, 0.0 for mildly separated clusters, and -0.2 for overlapping clusters.

4) Hardware platform : We used an ODROID XU4 board [27] equipped with a a Samsung Exynos5422 Cortex™-A15 and a Cortex™-A7 Octa core. The main memory is a 2GB LPDDR3. We used a 1 GB partition on an SD card as a swap area. It ran the Ubuntu 22.04 LTS (Jammy Jellyfish).

B. Results and Discussion

Hereafter, we discuss the results of our experiments.

1) Experiment 1, clustering quality and deadline meeting: Fig. 6 shows the ARI values obtained by EK-means and K-MLIO. We observe that the EK-means ARIs are 1.43% lower than K-MLIO ARIs, which indicates that skipping chunks does not considerably affect clustering quality. One could expect to have a higher error with strict deadlines but our experiments did not highlight such a behavior. This is due to two factors: 1) the random nature of the initialization process that makes it hard to compare executions. 2) Our frequency scaling process participates in smoothing executions by increasing frequency in case of strict deadlines (see next experiment), which reduces the overall error for those cases.

The second observation that can be drawn is that the ARIs obtained for overlapping clusters ($sep = -0.2$) are low as compared to mildly and well-separated ones. However, EK-means remains comparable to K-MLIO and the low obtained ARIs are inherent to K-means clustering quality that degrades when clusters are overlapped as discussed in [3].

Table II gives EK-means execution times with respect to the imposed deadlines. The second column gives the deadline while the third gives the minimum and maximum time (from the 5 runs) with the given deadline. The last column highlights the deadline satisfaction percentage. We observe that in all but one case (that is 1 case from 90 experiments, around 1%), the deadlines were met (more than 98% of cases). We observe that we miss the deadline when $sep = -0.2$ and $T\_deadine = 750s$. This is due to K-MLIO calculations that we considered negligible but reveal to be, in some rare cases, considerable when each chunk reaches the WCET.

The variation in the time executions that we notice (difference between minimum and maximum) is mainly due to the initialization process that induces a high variability. Despite this variation, EK-means satisfied most of the deadlines thanks to its conservative estimations.

2) Experiment 2, energy consumption: In Table III, we show the times spent by EK-means on each of the platform operational frequencies with respect to the imposed deadline. K-MLIO for its part uses the default platform frequency, which is the maximum frequency 2GHz. As stated in the experiment description, we allowed for the deadlines to be higher than K-MLIO execution time to observe EK-means frequency modulation when not timely constrained. From Table III, we observe that this occurs when $T_{deadline} \geq 4/7 \cdot WCET$. We observe that the higher the deadline, the more EK-means uses low frequencies. For example, when the deadline is equal to K-MLIO WCET, EK-means spends 41.89% of its overall time on a frequency of 400MHz which is relatively low given the available frequencies, and only 7.8% of its time on the

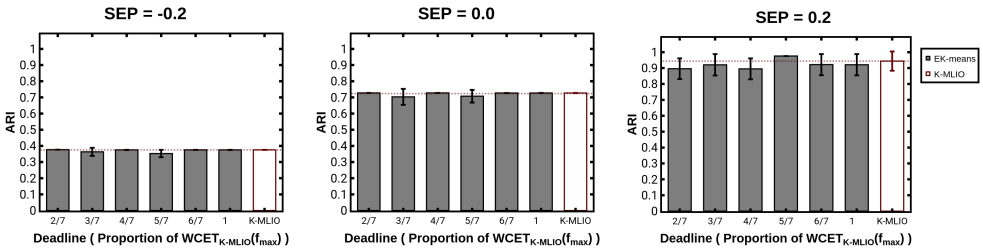

Fig. 6. EK-means and K-MLIO precision by deadline comparison

TABLE II
EK-MEANS EXECUTION TIME FOR DIFFERENT DEADLINES

Sep	$T_{deadline}$	EK-means interval (s)	$T_{deadline}$ satisfaction (%)
-0.2	300	95.82 - 250.6	100
	450	294.7 - 406.01	100
	600	424.38 - 514.37	100
	750	572.38 - 766.69	80
	900	744.19 - 899.37	100
	1050	822.63 - 957.52	100
0.0	300	55.08 - 134.99	100
	450	191.94 - 229.15	100
	600	286.99 - 337.86	100
	750	370.98 - 436.43	100
	900	482.21 - 591.01	100
	1050	578.3 - 660.63	100
0.2	300	56.67 - 248.73	100
	450	192.73 - 324.65	100
	600	247.41 - 561.85	100
	750	358.34 - 529.82	100
	900	404.72 - 821.24	100
	1050	558.44 - 812.79	100

TABLE III
EXECUTION TIME PER FREQUENCY FOR DIFFERENT DEADLINES

f_i	$T_{deadline}$						K-MLIO
	2/7	3/7	4/7	5/7	6/7	1	
2.0	29.29	8.59	26.03	8.66	21.58	47.39	302.05
1.9	0	0	0	21.85	28.83	0	0
1.8	0	20.54	24.74	0	0	21.85	0
1.7	0	0	0	26.96	25.61	0	0
1.6	25.10	0	0	0	28.61	22.97	0
1.5	0	28.78	28.91	25.78	0	30.37	0
1.4	0	0	0	0	0	0	0
1.3	0	0	0	0	28.90	32.33	0
1.2	0	0	32.26	29.29	0	29	0
1.1	0	33.05	0	0	44.07	0	0
1.0	0	0	0	37.12	0	54.15	0
0.9	59.24	0	47.25	0	66.91	0	0
0.8	0	0	0	0	0	50.67	0
0.7	0	0	0	49.88	54.35	0	0
0.6	0	125.70	0	0	0	62	0
0.5	0	0	157.63	0	0	0	0
0.4	0	0	0	206.50	214.60	252.99	0
0.3	0	0	0	0	0	0	0
0.2	0	0	0	0	0	0	0
Σ	**113.63**	**216.66**	**316.81**	**406.03**	**513.45**	**603.90**	**302.05**

maximum frequency. This is because EK-means has enough time to process the chunks and so it reduces the frequency drastically. In contrast, when the deadline constraint is the strongest ($2/7$ of K-MLIO WCET), EK-means spends 52.13% of its time on a frequency of 900MHz and 25.7% of its time on the maximum frequency to be sure not to miss the deadline.

To estimate the impact of frequency modulation, we used a simple analytic estimation of the processor energy consumption of EK-means as compared to K-MLIO. As stated on eq. (5), the dynamic energy of the processor when executing an application is the product of K, f^3 and the execution

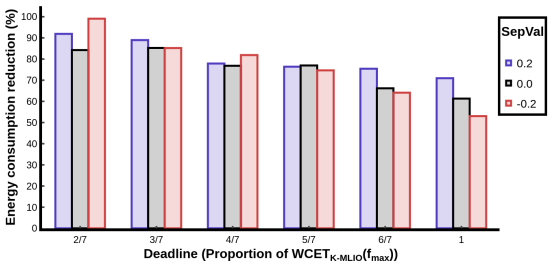

Fig. 7. EK-means processor energy reduction compared to K-MLIO for different deadlines and data set complexities

time of the application t. Thus, given the data of table III, we can infer the energy consumption reduction. The results are given in Fig. 7. It shows that overall, EK-means reduces energy consumption by 61.33% to 84.26%. This reduction is higher when the deadline is strong since EK-means takes up to 5/7 less time than K-MLIO. This energy evaluation does not consider memory and I/Os. Indeed, their energy is also minimized as the overall computation process is shortened.

VI. THREATS TO VALIDITY

Reliance on K-MLIO: EK-means was designed on top of K-MLIO as the latter implements a per-chunk version of K-means that first optimizes the I/O behavior without loss of accuracy, and second makes it possible to run only on part of the data. To builds on top of the original K-means, one way would be to tune the number of iterations, but the overall performance would be degraded.

Use of governors: their use was excluded from this work as we needed to get full control of the time duration of the different operations performed by EK-means. If a governor tends to reduce the frequency of the processor while executing EK-means, there is a risk of missing the deadline. As for now, the frequency needs to be tuned explicitly in EK-means.

VII. RELATED WORK

In recent studies, researchers have focused on reducing computation in real-time AI models to improve their efficiency and responsiveness for both training and inference step. [5] presents a stereo depth estimation with CNN on GPU that performs depth inference in stages. Each stage corrects the depth estimation of the previous one, allowing the model to provide ongoing estimates when queried. [9] proposes a

fast inference framework that learns to speed up inference at run-time by combining a flexible sampling technique with deterministic message passing to reduce computations in general regressors such as random forests. In [10], the authors propose an energy-efficient optimization for Hoeffding trees' online training based on adjusting the minimum number of observations read from a data stream before determining the best attribute for node splitting. This allows better control over the computations required to achieve a desired level of precision within confidence intervals. Another optimization for building Hoeffding trees [14] is based on setting the nodes' splitting criteria depending on the fraction of instances that a particular node has observed so far. These studies aforementioned highlight the need for a predictible learning process on embedded platforms. However, to the best of our knowledge, K-means was not optimized in this direction.

Considerable research efforts have been dedicated to task scheduling for real-time embedded systems, focusing on the use of different DVFS-enabled techniques. Integration of scheduling and DVFS can be achieved in two ways. The first involves independent slack reclamation, where DVFS relies on safe upper bound estimations of WCET to schedule tasks such as in GRUB-PA [17]. Other approaches try to limit the task makespan and energy at the same time, such as [18], [19], and [20]. Our contribution cannot directly use those studies, as it is very specific to the way K-means works. EK-means corresponds to the first type of study as it optimizes the energy opportunistically after ensuring that the deadlines are met.

VIII. Conclusion

This paper presents EK-means, a deadline-aware and energy efficient strategy for training K-means tailored for resource-constrained devices on the edge. EK-means runs on a subset of data that depends on the deadline imposed. This subset is calculated online. Whenever some slack time is available because the algorithm converges faster than expected, the frequency is reduced to save energy. EK-means reduce the accuracy very slightly as compared to traditional K-means while respecting the imposed deadline in more than 98% of cases, in addition to saving energy. We expect to work on another version that makes it possible to determine both a time and energy budget for the K-means to run. Also, we expect to study the interference with the Linux governors.

References

[1] J.-D. Choe, "Memory Technology 2021: Trends & Challenges," 2021 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD), pp. 111–115, Sep. 2021.

[2] A. Fahad et al., "A survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis," IEEE Transactions on Emerging Topics in Computing, vol. 2, no. 3, pp. 267–279, Sep. 2014.

[3] C. Slimani et al., "K -MLIO: Enabling K -Means for Large Data-Sets and Memory Constrained Embedded Systems," 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 262–268, Oct. 2019.

[4] P. Olukanmi et al., "k-Means-Lite: Real Time Clustering for Large Datasets," 2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI), pp. 54–59, Nov. 2018.

[5] Y. Wang et al., "Anytime stereo image depth estimation on mobile devices," arXiv (Cornell University), Oct. 2018.

[6] E. Olivetti et al., "Fast Clustering for Interactive Tractography Segmentation," 2013 International Workshop on Pattern Recognition in Neuroimaging, Jun. 2013.

[7] R. Alguliyev et al., "Constrained k-means algorithm for resource allocation in mobile cloudlets," Kybernetika, pp. 88–109, Mar. 2023.

[8] B. He et al., "Fully convolution neural network combined with K-means clustering algorithm for image segmentation," Tenth International Conference on Digital Image Processing (ICDIP 2018), Aug. 2018.

[9] S. M. A. Eslami et al., "Just-In-Time learning for fast and flexible inference," Neural Information Processing Systems, vol. 27, pp. 154–162, Dec. 2014.

[10] E. García-Martín et al., "Energy modeling of Hoeffding tree ensembles," Intelligent Data Analysis, vol. 25, no. 1, pp. 81–104, Jan. 2021.

[11] S. Kaushik, "An Enhanced Recommendation System using proposed Efficient K Means User-based Clustering Algorithm," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Oct. 2018.

[12] D. Xu et al., "Edge Intelligence: architectures, challenges, and applications," arXiv (Cornell University), Mar. 2020.

[13] R. Cochran et al., "Identifying the optimal energy-efficient operating points of parallel workloads," International Conference on Computer Aided Design, pp. 608–615, Nov. 2011.

[14] E. García-Martín et al., "Green accelerated hoeffding tree," arXiv (Cornell University), May 2022.

[15] N. Kukreja et al., "Training on the Edge: The why and the how," arXiv (Cornell University), Feb. 2019.

[16] Y. Benmoussa et al. "Joint DVFS and Parallelism for energy efficient and low latency software video decoding," IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 4, pp. 858–872, Apr. 2018.

[17] C. Scordino et al., "A resource reservation algorithm for Power-Aware scheduling of periodic and aperiodic Real-Time tasks," IEEE Transactions on Computers, vol. 55, no. 12, pp. 1509–1522, Dec. 2006.

[18] Y. Hu et al., "A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints," Neural Computing and Applications, vol. 32, no. 10, pp. 5681–5693, Aug. 2019.

[19] N. B. Rizvandi et al., "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," Journal of Parallel and Distributed Computing, vol. 71, no. 8, pp. 1154–1164, Aug. 2011.

[20] R. Medina et al., "Work-in-Progress: Probabilistic System-Wide DVFS for Real-Time Embedded Systems," 2019 IEEE Real-Time Systems Symposium (RTSS), Dec. 2019.

[21] "Welcome to pyJoules's documentation! — pyJoules 0.2.0 documentation." https://pyjoules.readthedocs.io/en/latest/index.html (accessed Jun. 06, 2023).

[22] Y. Kitagawa et al., "Anomaly prediction based on machine learning for Memory-Constrained Devices," IEICE Transactions on Information and Systems, pp. 1797–1807, Sep. 2019.

[23] S. Branco et al., "Machine Learning in Resource-Scarce embedded Systems, FPGAs, and End-Devices: A survey," Electronics, vol. 8, no. 11, p. 1289, Nov. 2019.

[24] M. S. Mahdavinejad et al., "Machine learning for internet of things data analysis: a survey," Digital Communications and Networks, vol. 4, no. 3, pp. 161–175, Aug. 2018.

[25] J. M. Santos et al., "On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification," in Lecture Notes in Computer Science, 2009, pp. 175–184.

[26] W. Qiu et al., "Package clusterGeneration." CRAN, 2006.

[27] R. Roy et al., "Ordoid XU4 - User Manuel." Hard Kernel, Ltd, 2015.

[28] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 4, pp. 1595–1623, Apr. 2022.

[29] Q. Wang et al., "Melon: breaking the memory wall for resource-efficient on-device machine learning," The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), pp. 450–463, Jun. 2022.

[30] C. Slimani, C. -F. Wu, S. Rubini, Y. -H. Chang and J. Boukhobza, "Accelerating Random Forest on Memory-Constrained Devices Through Data Storage Optimization," in IEEE Trans. on Computers, vol. 72, no. 6, pp. 1595-1609, June 2023.

[31] A. Laga, J. Boukhobza, F. Singhoff and M. Koskas, "MONTRES : Merge ON-the-Run External Sorting Algorithm for Large Data Volumes on SSD Based Storage Systems," in IEEE Trans. on Computers, vol. 66, no. 10, pp. 1689-1702, Oct. 2017.