Accelerating K-Means: A Vectorized Approach for AI Engines & Neural Processing Units

Eleonora Cabai, Giuseppe Sorrentino, Marco Domenico Santambrogio, Davide Conficconi
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano
eleonora.cabai@mail.polimi.it, {giuseppe.sorrentino, marco.santambrogio, davide.conficconi}@polimi.it

***Abstract*—K-Means is a clustering technique widely employed in AI workloads, from image processing to data mining. Given its importance, researchers propose different algorithms and hardware-accelerated implementations. While algorithm suitability can depend on the target use case, there is much less doubt about the architecture: FPGAs are the de facto standard, as the design can be perfectly tailored to the target use case. Despite this, AI accelerators such as GPUs and Neural Processing Units (NPUs) are gaining traction. The former attains remarkable performance at the cost of low energy efficiency. The latter, instead, promises to maximize both, but they are strongly underutilized due to the lack of a clear approach for K-Means acceleration. Considering AMD NPU, for example, the main computing cores are AI Engines that require algorithm reshaping and code optimization to harness data parallelism effectively. Thus, this research analyzes different K-Means versions to propose a vectorized algorithm that fully uses AI Engine (AIE) features. We validate our vectorized K-Means on Versal VCK5000, using FPGAs for data movement only, as the Memory Transfer Engines and Shim Tiles of NPUs, and the AI Engine for computation. This design reflects features of modern NPUs, making the validation fair. We attain up to $59.5\times$ speedup against Torch library on GPUs while being comparable but more energy efficient than further optimized GPU solutions.**

***Index Terms*—AI Engine, Versal, FPGA, k-means**

I. Introduction

Clustering [1], [2] is an unsupervised machine learning technique crucial for a wide range of applications, including, but not limited to, image processing [3], [4], big data mining [5], [6], bioinformatics [7], [8], and information retrieval [9]. By grouping data points into distinct clusters based on their similarity, clustering algorithms provide insights into the underlying structure of datasets. Among clustering methods, K-Means [10] is one of the most popular for its simplicity and versatility. The K-Means clustering algorithm iteratively assigns each data point to the nearest centroid and then recalculates it as the mean of the points assigned to it. This process is repeated either for a fixed number of iterations or until centroids converge (i.e., their positions change negligibly between iterations) [11]. The literature distinguishes three main K-Means algorithms, identified by their authors: Hartigan-Wong, MacQueen, and Lloyd. The Hartigan-Wong algorithm [12] enhances clustering accuracy by initially assigning points to clusters and iteratively relocating them to minimize the total within-cluster variance. If such a reassignment reduces the overall variance, the point is transferred to the new cluster. Lloyd's algorithm [11], instead, assigns all the data points to the nearest cluster and then updates the centroids based on these assignments. While this approach reduces synchronization steps and favours low-latency computations, it is highly sensitive to the initial placement of centroids, which can lead to suboptimal clustering results. Lastly, MacQueen [13] proposes a trade-off by introducing a non-random initial assignment while keeping the frequent synchronizations. This approach reduces the number of iterations but mitigates the accuracy errors.

Given the high computational load, multiple accelerated solutions have been proposed to enhance performance or energy efficiency over the years. In particular, Field Programmable Gate Arrays (FPGAs) have become the factor standard for such tasks, improving both [14]–[18]. However, as K-Means is also crucial in multiple AI applications, AI accelerators like Graphical Processing Units (GPUs) and Neural Processing Units (NPUs) have gained popularity. Many libraries provide GPU-acceleration [19]–[21], achieving satisfactory performance, but often at the expense of higher energy consumption. Conversely, NPUs remain underutilized as no approach is tailored for their compute engine: the AIEs. Despite this, NPUs hold promise by offering data parallel capabilities through AIE vector cores while minimizing power consumption [22]–[24].

For these reasons, this research specifically targets NPU compute tiles and compares them against other AI accelerators. We specifically refer to AMD NPUs, but similar considerations can be done for comparable architectures, like the Ascend NPU [25]. Initially, we evaluate existing K-Means implementations to identify those best suited for AIEs. Then we develop a vectorized version of the selected algorithm, employing a scale-out strategy to distribute the workload across multiple compute tiles. This approach leverages both vectorization and spatial parallelization, key features of AIEs. For validation, we implement a multi-AIE version of our vectorized algorithm for validation on the Versal VCK5000 platform, which integrates first-generation AIEs with FPGA fabric [26]. In this setup, FPGA is used exclusively to feed data to the AIEs, mirroring the Memory Tile Engine (MTE) and Shim tiles of the NPUs, ensuring a fair evaluation. Meanwhile, the AIEs handle all computations, as would be done on modern NPUs.

*DOI: 10.1109/fpl68686.2025.00053 © 2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Specifically, this research proposes:

- **An analysis** of the different **K-Means** implementations and their suitability for the acceleration on **AIEs** (§IV-A).
- **A vectorized K-Means** algorithm, tailored on the **AIEs'** vector cores, within a scale-out strategy to split the workload across multiple compute tiles (§IV-B).
- An **open source HW/SW solution**[1] to validate the proposed version on the AIE, using VCK5000 heterogeneity to shape each feature of modern NPUs (§IV-C).

Our implementation achieves a $59.5\times$ speedup over the torch library accelerated through an NVIDIA A5000. Considering more optimized GPU libraries, instead, our solution is comparable, being $1.89\times$ slower than FAISS but consuming $1.49\times$ less energy.

II. Versal Architecture & AI Engines

The target Versal VCK5000 is a heterogeneous platform featuring an FPGA VC1902, integrated through the Network on Chip (NoC) with the first generation of AIE.

The device offers 400 AIEs; each is a statically scheduled Very Long Instruction Word (VLIW) processor with a vector unit, thus offering SIMD operations to benefit from data parallelism. AIE performance comes at the cost of handling constraints imposed by such architecture. For example, optimizations like loop unrolling can lead to a code size increase, which must be mitigated given the limited program memory of $16\ KB$. Additionally, each tile features only $32\ KB$ of local memory. Consequently, manual optimizations are essential to manage both memory and static scheduling constraints [27].

Validation Fairness - To ensure effective validation for modern NPUs that lack FPGA components, our design refrains from offloading any computation to the FPGA. Furthermore, NPUs rely on up to 20 (Phoenix model) or 32 (Strix model) second-generation AIEs (AIE-ML). In contrast to the first generation, AIE-MLs offer $64\ KB$ of memory and native support for the $bfloat16$ data type.

III. Related Work

Literature widely uses hardware accelerators for K-Means acceleration, with remarkable gains in performance and scalability. On the GPU side, the main target is optimizing performance, with multiple parallel K-Means versions striving to reduce latency as much as possible [28], [29]. Despite this, the literature elects FPGAs as the best solution, with a plethora of optimizations and versions proposed over the years. He et al. used the K-Means algorithm to evaluate their HLS library [14], and presented a K-Means operator for hybrid CPU-FPGA databases [15]. Hussain et al. implemented an FPGA-based approach for clustering microarray data [30]. Bhimani et al. explored GPU-based parallel implementations on large datasets [31]. Wang et al. introduced a triangle-inequality-based FPGA accelerator [32]. Perera and Raghavan proposed a scalable FPGA architecture for big data clustering [33]. Deng et al. presented a real-time FPGA-based K-Means implementation for image clustering [34]. Despite being interesting, none of these solutions considers AIEs or NPUs.

[1] https://github.com/necst/AIE-kmeans

TABLE I
K-Means Analysis for AI Engine Acceleration.

Algorithm	AIE Vectorization	AIE Parallelization
Hartigan-Wong [12]	✗	✗
MacQueen [13]	✓	✗
Lloyd [11]	✓	✓

IV. Methodology

The main goal of this research is to explore NPUs' capabilities for K-Means acceleration. To this extent, §IV-A analyzes different K-Means implementations to find the most suitable to benefit from AIE main features: vectorization and spatial parallelization. Then, §IV-B proposes a vectorized version of the most suited algorithm, as well as a scale-out strategy to partition the computation across multiple hardware layers. Finally, §IV-C describes the hardware validation of the presented algorithm, targeting the Versal VCK5000.

A. K-Means Algorithms Analysis

There are three main K-Means implementations: MacQueen's [13], Hartigan-Wong's [12], and Lloyd's version [11]. Literature deeply analyzes them, proposing different algorithmic and hardware optimizations [14]–[17]. However, no detailed analysis exists for vector processors as those of AIE. Thus, we describe the three implementations, highlighting their suitability or unsuitability for the target architecture.

Hartigan-Wong's algorithm [12] randomly assigns points to clusters and gradually moves them from one cluster to another, measuring variance decreases. Despite being very accurate, this algorithm is *unsuitable* for AIEs as it uses sequential updates: before each assignment at step i, the algorithm must consider the previous cluster update at step $i-1$, preventing vectorization. For the same reason, spatial parallelism is *unfeasible*, as it needs a shared state or synchronization among AIEs, to be aware of what happened in the previous iteration.

MacQueen's version [13], instead, relaxes Hartigan-Wong's constraints, allowing iterative cluster updates without point-to-point dependencies, making it *vectorizable*. In other words, an entire vector can be evaluated before updating the clusters. However, spatial parallelization is still *unfeasible*, as data dependencies between iterations i and $i-1$ would require a shared state among compute tiles. As no shared memory exists, this state is feasible through exchanging information among tiles, which is doable but dramatically ineffective. Thus, MacQueen's version is *not suitable* for AIEs.

Finally, Lloyd's K-Means algorithm [11] assigns each point to the nearest centroid, delaying centroid recalculations as much as possible and removing almost all data dependencies. This makes the algorithm very sensitive to the initial cluster position, but it also permits both vectorization and spatial parallelization, making it particularly interesting for AIEs.

Algorithm 1 Vectorized Lloyd's K-Means Algorithm

1: **Input:** *n_clusters, n_points, c_coords, p_coords*
2: **parfor** $i \leftarrow 0$ **to** $(n\_clusters/16) - 1$ **do**
3: *vect_in* $\leftarrow$ **Vect_Read**(*c_coords*)
4: *clusters* $\leftarrow$ **Cluster**(*vect_in*)
5: **end parfor**
6: **parfor** $i \leftarrow 0$ **to** $(n\_points/16) - 1$ **do**
7: *vect_in* $\leftarrow$ **Vect_Read**(*p_coords*)
8: *point* $\leftarrow$ **Point**(*vect_in*)
9: **for** *point* in *points* **do**
10: *d* $\leftarrow$ **Vect_Euclidean_Distance**(*point*, *clusters*)
11: *c_idx* $\leftarrow$ **Min_Dist**(*d*, *n_clusters*)
12: *clusters*[c_idx] $\leftarrow$ **Accumulate_Coords**(*point*)
13: **end parfor**
14: **Output:** *partial_cluster_coords*

However, To the best of our knowledge, no vectorized Lloyd's algorithm exists to leverage AIEs vector core.

B. K-Means Vectorization

Algorithm 1 extends Lloyd's algorithm by introducing a vectorized implementation that fully utilizes SIMD instructions, while Figure 1 depicts the vectorized kernel Ⓑ, integrated with the data dispatcher Ⓐ and collector Ⓒ.

The process begins with the AIE Ⓑ receiving data from the dispatcher (Ⓐ in Figure 1). This dispatcher can be implemented using the Memory Transfer Engine (MTE) of NPUs or through FPGAs, and its primary function is to distribute data from the main memory to the AIEs. Given the statically scheduled nature of the architecture, the dispatcher also provides information about the data volume assigned to each AIE tile, ensuring synchronized and efficient execution.

Initially, the data exchanged are cluster coordinates. At each step $(L3-4)$, Ⓑ loads a vector of 1024 bits — the maximum vector register size for the AIE. Each cluster and point corresponds to an n-dimensional set of coordinates, representing each coordinate as a 32-bit floating-point number. Therefore, the vector unit can fetch at most $1024/(32 \times n)$ elements. Our implementation focuses on two-dimensional coordinates, allowing a vector to contain up to 16 elements. This approach can be generalized to higher dimensions; however, as the number of dimensions n increases, each element (cluster or point) requires more bits, reducing the number of elements accommodated within a single vector.

The cluster coordinates must reside in the AIE's local memory, as they are essential for assigning points to clusters. Once all cluster coordinates have been loaded, Ⓑ fetches points $(L7-8)$. For each vector of points, we identified two possible computation strategies. The first simultaneously computes Euclidean distances from **all points** to **each cluster**. Although straightforward and allowing multiple points to be processed independently, this method requires storing multiple intermediate distance results on the AIE, which is problematic given its memory constraints. To overcome this limitation, our proposed strategy calculates Euclidean distances from **each point to all clusters** in parallel $(L10)$, immediately assigning each point to its closest cluster $(L11)$. This approach optimizes the algorithm specifically for the AIE architecture by eliminating the need for intermediate storage. Lastly, the algorithm computes partial coordinate updates, which are applied to the cluster coordinates after all points have been evaluated $(L12)$.

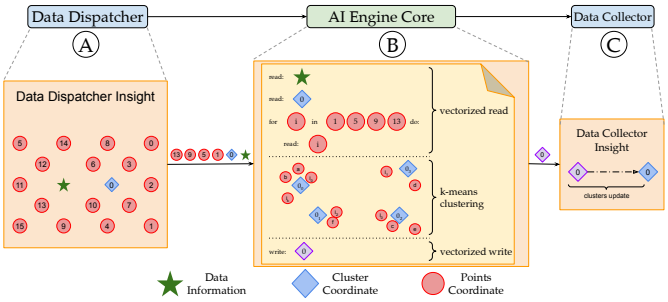

Fig. 1. Architecture of Lloyd's Accelerator, where clusters are updated at the end in the Data Collector and not at each iteration in the AIE.

Line 12 of Algorithm 1 is crucial as the cluster update is delayed up to the end of points evaluation, as Lloyd's algorithm indicates. This cancels the data dependencies between consecutive computation steps, favoring parallelism. Indeed, the computation can now be split on multiple compute cores, each receiving all the clusters - needed for correctly assigning points - but only a fraction of points. Doing so, each core computes its own partial updates $(L12)$ that will be combined in the data collector (Ⓒ in Figure 1). This combination is trivial, just a sum of partial updates to the initial coordinates, and can be implemented either on the CPU, for the NPUs, or in FPGAs for systems such as the VCK5000.

A final consideration is mandatory, as vectorization introduces constraints that must be handled. Indeed, vectorized reads impose limitations on the input size, which must be a multiple of the $vector\_size$. This constraint is even tighter when considering multiple compute cores, as we must ensure an input, from the dispatcher, multiple of $N_{AIE} \times vector\_size$. To handle this issue, our AIE tiles also take as input the amount of data padded, within the data volume to be handled. Thanks to this, AIE can discard the fetched padding data, skipping useless computation.

C. Hardware Validation On Heterogeneous Versal Systems

To evaluate the performance of Algorithm 1 and compare it against existing solutions for AI accelerators, we implemented it on the compute tiles of the VCK5000. This platform embodies features of modern NPUs while ensuring the algorithm's effectiveness within a complex heterogeneous system.

Although the VCK5000 features 400 compute tiles, we limited our usage to align with contemporary NPU architectures, such as the Phoenix and Strix models, which utilize 20 and 32 compute tiles, respectively [36]. To feed the AIE, we implemented dispatcher and collector kernels in reconfigurable logic, connecting these kernels to the AIE layer through PL-AIE connections (PLIO), which offer the highest communication performance. This approach mirrors the functionality of

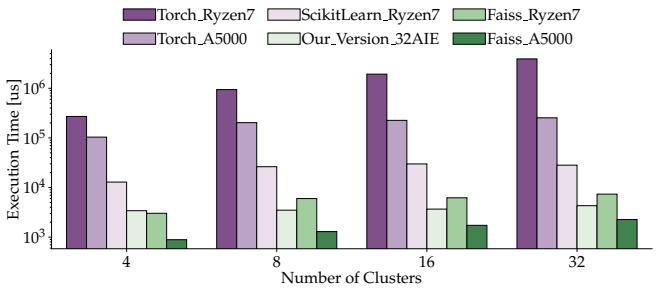

Fig. 2. Execution time comparison of our vectorized K-Means kernel, scaled on 32 AIE, against FAISS [20], Torch [21], and ScikitLearn [19], using an open source dataset [35].

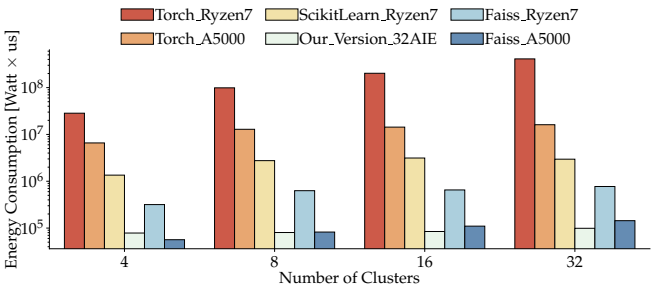

Fig. 3. Energy consumption comparison of our vectorized K-Means kernel, scaled on 32 AIE, against FAISS [20], Torch [21], and ScikitLearn [19], using an open source dataset [35].

modern MTEs and Shim tiles that fetch data from memory and distribute it effectively across multiple compute tiles.

Both kernels optimize memory access through burst transfers and increased word widths, enabling Ⓐ to read multiple enlarged words from memory per access. Additionally, we employed the dataflow paradigm for kernel execution, maximizing the data rate at the AIE' input. These optimizations are essential, given that the FPGA's working frequency is lower than the AIE ones (300 MHz versus 1.2 GHz).

We mapped the entire computational workload onto the AIEs, as would be done in an NPU. Each core algorithm reflects the one described in §IV-B, further optimized through compiler directives for scheduling and software pipelining.

V. Experimental Results

We design the FPGA part using HLS 2022.2 and Vitis while relying on Xilinx Runtime (XRT 2022.2) for the interaction between the C++ CPU-based host and the accelerator card. The targeted hardware platform is the Xilinx VCK5000, paired with an AMD EPYC 7V13 CPU.

We compare our accelerator against existing libraries such as Facebook AI Similarity Search (FAISS) [20], a highly optimized K-Means implementation known for its accuracy and efficiency in handling large datasets, Torch [21], and Scikit-Learn [19]. As a competing AI accelerator, we consider an NVIDIA A5000 paired with a Ryzen 7 5800X. Since our evaluation focuses on NPU validation, **we exclude PCIe transfer time** for both our and literature solutions, thus measuring the elapsed time between data retrieval from the main memory (included) up to the result write-back.

For benchmarking, we use an open-source dataset of bi-dimensional points [35]. We first scale out the design, using up to 32 compute tiles (the maximum in modern NPUs). Then, we compare it against existing solutions, assessing execution time (in microseconds) and energy consumption (measured as $Power \times Execution\_time$), relying on vendors' tools (xbutil and nvidia-smi respectively for FPGAs and GPUs).

A. Literature Comparison

Firstly, to measure the improvement of vectorization and pure hardware acceleration, we compare our single-tile version against the CPU baseline of the proposed algorithm. This analysis showcases the speedup of $1.41\times$, highlighting the need for spatial parallelism. Indeed, scaling the number of AIE employed up to 32 compute tiles equally handling the workload, we attain up to $15.75\times$ speedup against our CPU baseline and $11.69\times$ speedup against our single AIE version.

For the literature comparison, we focus on AI accelerators and compare them against CPU and GPU libraries. Figure 2 depicts the performance analysis, highlighting the FAISS library accelerated on NVIDIA A5000 as the best configuration, while our solution is the second best. Indeed, while being faster than other CPU and GPU-based solutions, with a remarkable speedup of $59.5\times$ against Torch accelerated on NVIDIA A5000, we are still $1.89\times$ slower against FAISS accelerated on the NVIDIA A5000 for the scenario with 32 clusters. On the positive side, looking at trends, our solution scales better than FAISS, opening possible improvements once considering more clusters.

Figure 3 showcases the energy consumption comparison for the same solutions. From this analysis, our implementation stands as the most efficient one, with massive improvements with respect to CPU-based implementations and $160.2\times$ consumption reduction against Torch accelerated on the NVIDIA A5000 in the 32 clusters scenario. Compared with FAISS in the same case, instead, we attain a $1.43\times$ energy consumption reduction. Overall, this analysis proves the effectiveness of accelerating K-Means on NPUs AI Accelerator, as it offers comparable results but higher energy efficiency than its literature counterpart. Furthermore, we highlight that real NPUs would attain even lower energy consumption, as they only consume a few Watts, against the $25W$ needed by our design.

VI. Conclusions

This research leverages VCK5000 features to present a vectorized K-Means implementation. We accelerate the procedure across both FPGA and AIE. In this setup, the FPGA implements logic to emulate the MTE and Shim Tile—components responsible solely for data movement—while the computational workload is entirely handled by the AIEs. Our validation demonstrates comparable execution times but reduced energy consumption compared to existing GPU-based libraries.

ACKNOWLEDGMENTS

This work has financial support from ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by European Union – NextGenerationEU. This work was supported in part by AMD under the Heterogeneous Accelerated Compute Clusters (HACC) program. The authors are grateful to AMD University Program support.

REFERENCES

[1] L. Rokach and O. Maimon, *Clustering Methods*. Boston, MA: Springer US, 2005, pp. 321–352. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_15

[2] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217311815

[3] F. U. Siddiqui and A. Yahya, *Clustering techniques for image segmentation*. Springer, 2022.

[4] S. K. Dubey, S. Vijay, and A. Pratibha, "A review of image segmentation using clustering methods," *International Journal of Applied Engineering Research*, vol. 13, no. 5, pp. 2484–2489, 2018.

[5] S. Arora and I. Chana, "A survey of clustering techniques for big data analysis," in *2014 5th International Conference-Confluence The Next Generation Information Technology Summit (Confluence)*. IEEE, 2014, pp. 59–65.

[6] H. Wang, W. Wang, J. Yang, and P. S. Yu, "Clustering by pattern similarity in large data sets," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 394–405.

[7] M. R. Karim, O. Beyan, A. Zappa, I. G. Costa, D. Rebholz-Schuhmann, M. Cochez, and S. Decker, "Deep learning-based clustering approaches for bioinformatics," *Briefings in bioinformatics*, vol. 22, no. 1, pp. 393–415, 2021.

[8] S. Datta and S. Datta, "Comparisons and validation of statistical clustering techniques for microarray gene expression data," *Bioinformatics*, vol. 19, no. 4, pp. 459–466, 2003.

[9] E. M. Rasmussen, "Clustering algorithms." *Information retrieval: data structures & algorithms*, vol. 419, p. 442, 1992.

[10] N. Memarsadeghi, *Classified Information: The Data Clustering Problem*. Society for Industrial and Applied Mathematics, 2009, ch. 11, pp. 149–155. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/9780898717723.ch11

[11] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[12] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 28, no. 1, pp. 100–108, 12 2018. [Online]. Available: https://doi.org/10.2307/2346830

[13] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967. [Online]. Available: https://api.semanticscholar.org/CorpusID:6278891

[14] Z. He, D. Korolija, and G. Alonso, "Easynet: 100 gbps network for hls," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 197–203.

[15] Z. He, D. Sidler, Z. István, and G. Alonso, "A flexible k-means operator for hybrid databases," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 368–3683.

[16] Y. Lu, D. Wang, M. Liu, H. Wang, and J. Zhang, "Design and implementation of an fpga-based k-means clustering algorithm for large-scale data," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Budapest, Hungary: IEEE, 2016, pp. 221–224.

[17] J. Choi, J. Lee, and S. Hong, "A high-performance fpga-based implementation of k-means clustering algorithm for large-scale data," in *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Timișoara, Romania: IEEE, 2019, pp. 233–236.

[18] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on fpgas?" in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020, pp. 991–1010. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/roscoe

[19] Scikit-learn Developers, "Clustering — Scikit-learn Documentation," 2025. [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html

[20] FAISS, "FAISS Official Website," 2025. [Online]. Available: https://faiss.ai/index.html

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[22] P. S. Galfano, G. Sorrentino, E. D'Arnese, and D. Conficconi, "Co-designing a 3d transformation accelerator for versal-based image registration," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*. IEEE, 2024, pp. 219–222.

[23] C. Heinz, T. Kalkhof, Y. Lavan, and A. Koch, "Tapasco-aie: An open-source framework for streaming-based heterogeneous acceleration using amd ai engines," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 155–161.

[24] G. Sorrentino, P. S. Galfano, E. D'Arnese, and D. Conficconi, " Soaring with TRILLI: An HW/SW Heterogeneous Accelerator for Multi-Modal Image Registration ," in *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 56–65. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/FCCM62733.2025.00040

[25] L. Huawei Technologies Co., "Cann: Compute architecture for neural networks," https://www.hiascend.com/en/software/cann, 2025, accesso: 30 marzo 2025.

[26] G. Sorrentino, P. S. Galfano, E. D'Arnese, and D. Conficconi, "VOTED: Versal optimization toolkit for education and heterogeneous systems development," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2025. [Online]. Available: https://2025.ieee-iscas.org/

[27] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach 6th edition*. Elsevier, 2018.

[28] "K-means clustering algorithm - nvidia," *NVIDIA Developer Blog*, 2023. [Online]. Available: https://www.nvidia.com/en-us/glossary/k-means/

[29] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," ser. DaWaK '08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 166–175. [Online]. Available: https://doi.org/10.1007/978-3-540-85836-2_16

[30] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 248–255.

[31] J. Bhimani, M. Leeser, and N. Mi, "Accelerating k-means clustering with parallel implementations and gpu computing," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 2015, pp. 1–6.

[32] Y. Wang, B. Feng, G. Li, G. Tzimpragos, L. Deng, Y. Xie, and Y. Ding, "Tiacc: Triangle-inequality based hardware accelerator for k-means on fpgas," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 133–142.

[33] R. Raghavan and D. G. Perera, "A fast and scalable fpga-based parallel processing architecture for k-means clustering for big data analysis," in *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2017, pp. 1–8.

[34] T. Deng, D. Crookes, F. Siddiqui, and R. Woods, "A new real-time fpga-based implementation of k-means clustering for images," in *Intelligent Computing and Internet of Things*, K. Li, M. Fei, D. Du, Z. Yang, and D. Yang, Eds. Singapore: Springer Singapore, 2018, pp. 468–477.

[35] A. Ferretti, "Public Dataset for K-means Clustering," 2025. [Online]. Available: https://github.com/andreaferretti/kmeans/blob/master/points.json

[36] AMD, "Leveraging the iron ai engine api to program the ryzen ai npu," 2024. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/products/processors/ryzen/ai/iron-for-ryzen-ai-tutorial-micro-2024.pdf