

# KAN-SAs: Efficient Acceleration of Kolmogorov-Arnold Networks on Systolic Arrays

Sohaib Errabii, Olivier Sentieys, Marcello Traiola  
University of Rennes, CNRS, Inria, IRISA, Rennes, France

**Abstract**—Kolmogorov-Arnold Networks (KANs) have garnered significant attention for their promise of improved parameter efficiency and explainability compared to traditional Deep Neural Networks (DNNs). KANs’ key innovation lies in the use of learnable non-linear activation functions, which are parametrized as splines. Splines are expressed as a linear combination of basis functions (B-splines). B-splines prove particularly challenging to accelerate due to their recursive definition. Systolic Array (SA)-based architectures have shown great promise as DNN accelerators thanks to their energy efficiency and low latency. However, their suitability and efficiency in accelerating KANs have never been assessed. Thus, in this work, we explore the use of SA architecture to accelerate the KAN inference. We show that, while SAs can be used to accelerate part of the KAN inference, their utilization can be reduced to 30%. Hence, we propose KAN-SAs, a novel SA-based accelerator that leverages intrinsic properties of B-splines to enable efficient KAN inference. By including a non-recursive B-spline implementation and leveraging the intrinsic KAN sparsity, KAN-SAs enhances conventional SAs, enabling efficient KAN inference, in addition to conventional DNNs. KAN-SAs achieves up to 100% SA utilization and up to 50% clock cycles reduction compared to conventional SAs of equivalent area, as shown by hardware synthesis results on a 28nm FD-SOI technology. We also evaluate different configurations of the accelerator on various KAN applications, confirming the improved efficiency of KAN inference provided by KAN-SAs.

**Index Terms**—Kolmogorov-Arnold Networks, hardware accelerator, systolic array

## I. INTRODUCTION

The Kolmogorov-Arnold Network (KAN) is a neural network architecture [1] that has garnered interest and been adopted in various applications, such as time series analysis [2], recommender systems [3], and medical image segmentation [4]. The main interest of KANs lies in their improved parameter efficiency and explainability compared to conventional Deep Neural Networks (DNNs). This is enabled by replacing the conventional scalar weights with learnable spline-based activation functions, which are parameterized in a basis function (*B-spline*). However, this leads to an increase in computational complexity proportional to the basis size. Indeed, to compute KAN inference, instead of a single scalar multiply, each basis function must first be evaluated at the input, and then a linear combination of all the basis functions is performed. Moreover, the B-spline functions are evaluated recursively through the Cox-de Boor formula (See Eq. 3), which makes their acceleration challenging.

Recent artificial intelligence (AI) accelerators increasingly rely on spatial architectures, which have become the *de facto* standard for AI acceleration. Such architectures efficiently execute general matrix multiplication (GEMM), the core operation underlying

many AI workloads. Prominent examples include Google’s Tensor Processing Unit (TPU) [5] and NVIDIA’s Tensor Cores, first introduced in the Volta GPU microarchitecture [6]. The efficiency of spatial architectures stems from their ability to maximize data reuse and minimize data movement, which dominates the energy cost [7]. Building on this foundation, numerous works have proposed further optimizations tailored to specific workloads. For instance, Eyeriss [8] introduced dataflow optimizations to improve the efficiency of convolutional neural networks. Other efforts, such as SCNN [9], explored computation on compressed weights and activations to exploit zero weights stemming from model pruning and zero activations that occur in ReLU-based networks. The unique design of KANs limits the effectiveness of existing solutions in accelerating KAN inference compared to other AI applications. Specifically, the recursive nature of B-spline function computation creates a considerable bottleneck, making it challenging to take full advantage of the efficient GEMM executions in spatial architectures such as modern GPUs or Systolic Arrays (SAs) [10], which are the foundation of modern TPUs.

Therefore, research efforts have been focusing on new approaches to accelerate KANs (details in Sec. II-B). Among recent studies, compute-in-memory (CIM) approaches have been proposed [11], [12]. In such studies, different ways to approximate the learned non-linear KAN functions or their basis are utilized, such as piece-wise linear (PWL) approximation. However, no insights are offered into how spatial (non-CIM) architectures may be optimized for KANs. A recent approach, ArKANe [13], focuses on the B-spline evaluation bottleneck. It proposes an efficient dataflow acceleration methods for the Cox-de Boor recursive formula, achieving a considerable speedup compared to CPU and GPU implementations. While all these efforts considerably improve knowledge of KAN acceleration, unfortunately, the current literature lacks solutions to efficiently accelerate KANs on spatial architectures, and particularly on SAs. To address this gap, this paper analyzes and utilizes the KAN properties to enhance SAs, enabling efficient end-to-end inference acceleration of KAN while maintaining the generality of the accelerator for non-KAN DNN workloads.

As shown in Section III, once the B-splines have been evaluated, their linear combination is nothing more than a GEMM operation, which can be accelerated on SAs. Regarding the B-spline computation, we observe that a direct floating-point implementation of the recursive evaluation of B-splines is quite costly. For inference-only acceleration, an efficient tabulation strategy of B-splines is possible, thanks to their properties [12]. Furthermore, B-spline computation results in an  $N:M$  sparsity

pattern in the values processed by the SA, thereby leading to low Processing Element (PE) utilization. Designing a PE of the SA that can handle inputs with a KAN-specific  $N:M$  sparsity pattern is needed to improve the overall efficiency of the SA [14]. To the best of our knowledge, there is currently no SA-based accelerator that, in addition to standard DNN workloads, can accelerate KANs by utilizing non-recursive B-spline computation and achieving high PE utilization. In summary, this work makes the following contributions:

- We show how a KAN layer can be transformed into a GEMM formulation for execution on a systolic array, and analyze the causes of the inefficiencies that lead to low throughput and poor PE utilization.
- Building upon our analysis, we include in KAN-SAs the needed architectural modifications to handle such inefficiencies. These consist of a PE (i) enhanced with a non-recursive B-spline unit and (ii) providing efficient handling of the  $N:M$  sparsity pattern generated by B-splines.
- We synthesize KAN-SAs on a 28nm FD-SOI technology and evaluate its improvements compared to a conventional systolic array and to the state-of-the-art approach for B-spline acceleration. Our results show that the proposed KAN-SAs achieves 39.9% average improvement in PE utilization and 50% clock cycle reduction on average compared to conventional SAs. Moreover, the B-spline evaluation approach utilized in KAN-SAs achieves more than  $72\times$  improvement compared with the state-of-the-art approach.

KAN-SAs source code is available at <https://github.com/sohaiberrabii/kansas>.

## II. BACKGROUND

In this section, we present the core operation of a KAN layer and the implications for execution on a systolic array.

### A. KAN Layer

The main difference between a layer in a Multi-Layer Perceptron (MLP) and a KAN layer is illustrated in Fig. 1. It essentially involves replacing the weights at the connections with learnable activation functions  $\phi_i$ . In general, a KAN layer as originally proposed [1] can be expressed as follows:

$$\text{KANLayer}(x) = \sum w_i \phi_i(x) + w_b b(x) \quad (1)$$

In the first term, the  $w_i$  scales suggested by [1] offer better control of the magnitude of the activation functions  $\phi_i$ ; however, at inference time, they can be absorbed in the functions  $\phi_i$ . The second term of the equation, which is not represented in Fig. 1, serves as a form of bias. In practice, it can be considered an MLP layer with a fixed non-linear activation (e.g., SiLU or ReLU) applied before the dot product. While this paper focuses on the first term, the proposed accelerator is capable of executing conventional MLP workloads and therefore any KAN layer that follows (1).

As illustrated in Fig. 1, in an MLP layer, a matrix-vector multiplication between the inputs and the weights is followed by a non-linear activation using fixed functions such as ReLU. In the KAN layer, learnable non-linear functions are evaluated on the inputs, followed by the sum of all activations (see Fig. 1(b)).

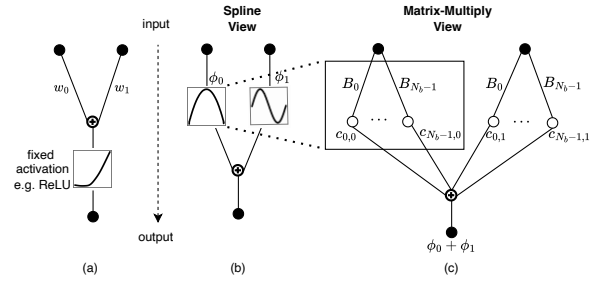


Fig. 1: A single layer of dimensions [2, 1] for MLP (left), KAN Spline View (middle) and KAN Matrix-Multiply View (right).

These activation functions are learned through parameterization in some function basis ( $\phi(x) = \sum_i c_i B_i(x)$ ). That is, the learnable parameters of the KAN layer are the coefficients  $c_i$  for the linear combination in this basis. Since the parameterization is a linear combination, we can view the KAN layer as a matrix multiply, as shown in Fig. 1(c). First, all the basis functions are evaluated at the inputs to obtain an intermediate matrix  $B$  of dimensions  $(M, N_b \times K)$ , with  $(M, K)$  being the dimensions of the input matrix, and  $N_b$  the size of the function basis. The matrix multiplication is then performed with the coefficient matrix of dimensions  $(N_b \times K, N)$ , with  $N$  being the output dimension of the layer (in Fig. 1,  $N = 1, K = 2$ ).

Therefore, the second part of the KAN layer can be efficiently accelerated by conventional hardware platforms, such as GPUs and SAs. However, the first part, i.e., the evaluation of the basis functions, poses a significant computational challenge. Indeed, the basis first proposed by [1] and also adopted by many KAN applications [2], [4], [15], [16], is the B-spline basis. This basis is defined by the following Cox-de Boor [17] recursion formula:

$$B_{i,P}(x) = \frac{x - t_i}{t_{i+P} - t_i} B_{i,P-1}(x) + \frac{t_{i+P+1} - x}{t_{i+P+1} - t_{i+1}} B_{i+1,P-1}(x) \quad (2)$$

with

$$B_{i,0}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1} \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

The points  $t_i$  are referred to as the knot sequence of the B-spline,  $P$  is the spline degree, and  $G$  is the grid size used for discretizing the input domain of the layer. These are hyperparameters of the KAN layer. Importantly, the recursive formulation of Eq. 3 introduces dependencies between computational stages, making it not efficiently parallelizable on GPUs.

Fig. 2 illustrates the case of a uniform grid ( $t_{k+1} - t_k = \Delta$ ) with  $G = 3$  and  $P = 3$ . As shown in the figure, the grid must be extended to account for the contributions of B-splines that are non-zero within the input domain,  $P$  extra intervals are added both before and after the input domain, leading to  $G + 2P$  total intervals, and  $N_b = G + P$  B-spline functions.

### B. Related Work

Regarding the hardware acceleration of KANs, a few approaches have been proposed in the literature. As for the ac-

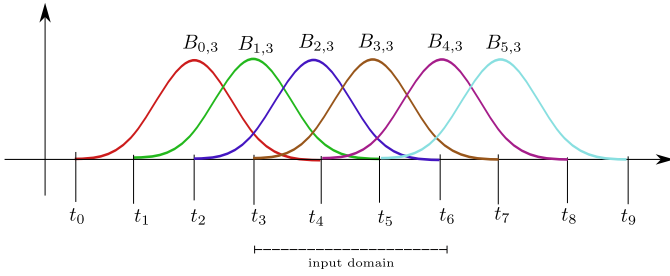


Fig. 2: B-spline basis functions of degree  $P = 3$  on a uniform grid of size  $G = 3$  for the input domain  $[t_3, t_6]$  that is extended on both ends to account for the contributions of B-spline functions that are non-zero within this input domain.

celeration of B-spline evaluation, ArKANE [13] has proposed a dataflow acceleration strategy, based on an efficient unroll of the recursive Cox-de Boor formula from Eq. 2. While this can help during training, for inference it incurs significant latency overhead (i.e., several cycles for each B-spline function and input), as it still relies on the recursive computing of B-splines. Moreover, ArKANE approach requires floating-point MAC units and is tightly coupled to AMD-Xilinx FPGA boards, as it relies on Xilinx’s recent spatial computing architecture (the Adaptive Intelligent Engine, AIE), for deployment on AMD-Xilinx Versal ASoCs. Finally, ArKANE does not target the acceleration of the full KAN, but only of B-splines. For an inference accelerator, a tabulation-based strategy is much more attractive for its low resource cost and minimal latency. Moreover, it can be adapted for both floating-point and integer-only inference [18].

A tabulation-based B-spline implementation for a CIM chip using resistive random access memory (RRAM) was proposed in [12]. However, this work does not target systolic arrays and does not address the structured sparsity generated by B-splines. Additionally, it introduces constraints specific to the accelerated software application, such as restricting grid points to powers of two, which limits the acceleration of more general cases. In contrast, our approach aims to implement a more generic systolic-array-based accelerator, assuming only a uniform grid, which maintains the generality of the B-spline unit. As shown in [1], fine-graining the grid is possible without retraining, using least squares to compute new coefficients, thus enabling the approximation of non-uniform grids.

Another CIM approach based on CMOS technology was proposed by [11]. This strategy consists of approximating the learned non-linear KAN activations  $\phi_i$  of Eq. 1 through piecewise linear (PWL) functions. In contrast, our approach focuses on accelerating the evaluation of  $\phi_i(x) = \sum_i c_i B_i(x)$  on a systolic array architecture through efficient tabular evaluation of B-splines  $B_i(x)$  and efficient GEMM operations.

### III. GENERAL PRINCIPLE AND BASIS FUNCTION UNIT OF KAN-SAS

In this work, we target a weight-stationary systolic array for GEMM acceleration. In our design depicted in Fig. 3, the weights (i.e., the B-spline coefficients) are pre-loaded into the Processing Elements (PEs) and remain stationary while the input activations

are propagated horizontally to other PEs and the resulting partial sums flow vertically towards an accumulator memory. This minimizes data movement by reusing weights and activations across processing elements, thereby enabling high energy efficiency. The memory hierarchy and additional architectural components needed for a practical DNN accelerator are beyond the scope of this paper.

#### A. General Architecture of KAN-SAs

Following the formulation presented in Section II-A, the only requirement for enabling hardware acceleration of KANs through a GEMM systolic array is the generation of the intermediate matrix  $B$  of dimensions  $(BS, (G + P)K)$ , with  $BS$  the batch size.  $B$  represents the B-spline activations, as shown in Fig. 1(c). To minimize the data movement required for KAN inference, we aim to enable on-the-fly generation of this intermediate matrix  $B$  close to the systolic array. This can be implemented by adding a basis function unit (the BSpline block in Fig. 3) that directly streams its values  $B_i(x)$  into the systolic array. The rest of this section focuses on this basis function unit, whereas Section IV will detail the PEs of the systolic array.

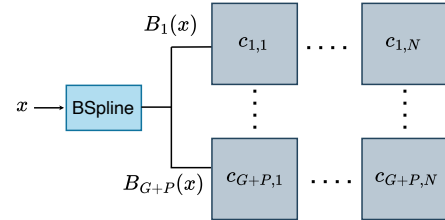


Fig. 3: GEMM weight stationary (WS) systolic array with an additional B-spline unit for on-the-fly B-spline activation computation

#### B. Basis Function Unit

Using the Cox-de Boor from Eq. 2 is not particularly efficient, as computing a single  $P = 3$  function would require 20 multipliers. Expressing the B-spline in the canonical basis  $1, x, \dots, x^3$  is still expensive, considering we need to evaluate the  $G + P$  functions. Moreover, as previously mentioned, the B-spline functions exhibit several properties that enable an efficient Lookup Table (LUT)-based implementation.

1) *Tabulation Strategy*: A B-spline is invariant under a translation and scaling transformation of its knot sequence [19]. i.e.,

$$B_{\mathbf{t}, P}(x) = B_{\alpha\mathbf{t} + \beta, P}(\alpha x + \beta), \quad \alpha, \beta \in \mathbb{R}, \quad \alpha \neq 0$$

where  $\mathbf{t} = (t_0, \dots, t_{G+2P+1})$ . If the grid is uniform, then  $t_{k+1} - t_k = \Delta$ . Applying the previous property with  $\alpha = \frac{1}{\Delta}$  and  $\beta = \frac{-t_0}{\Delta}$ , we can map our B-spline defined on an arbitrary knot sequence  $t_i$  to one defined on integer knots  $0, \dots, G + 2P + 1$ , also called cardinal B-spline, such as

$$B_{\mathbf{t}, P}(x) = B_{[0, \dots, G+2P+1], P}\left(\frac{x - t_0}{\Delta}\right).$$

Furthermore, with the translation-invariance applied to  $B_{k, P}$  we can further write

$$B_{\mathbf{t}_k, P}(x) = B_{k, P}\left(\frac{x - t_0}{\Delta}\right) = B_{0, P}\left(\frac{x - t_0}{\Delta} - k\right). \quad (4)$$

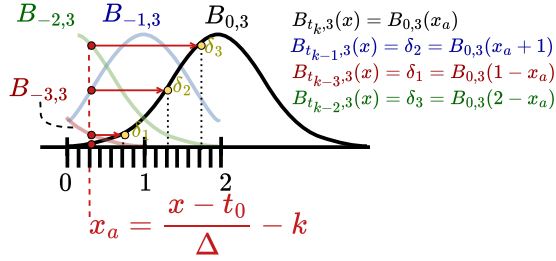


Fig. 4: Illustration of efficient tabular storing of half a cubic B-spline and computation of the others by aligning and shifting the input  $x$ . Only the values of  $B_{0,3}$  corresponding to the black sample points on the horizontal axis quantizing the interval  $[0, 2]$  are stored. For a given input  $x$ , all the B-spline values can be obtained by retrieving  $B_{0,3}$  stored values at the aligned input  $x_a$  and by shifting the evaluation point. E.g.,  $B_{t_{k-2},3}(x) = B_{0,3}(2 - x_a)$ .

This makes the B-spline independent of the KAN layer grid points  $t_i$ , since we only need to tabulate the function  $B_{0,P}$ , thereby enabling a ROM-based implementation of the B-spline unit. Of course, the unit must still perform the alignment required for  $x$  shown in Eq. 4. More importantly, the key property of B-splines for our purposes is their local support, which is given by the extreme knots used in their definition of Eq. 3,  $B_{t_k,P}(x) = 0$ ,  $x \notin [t_k, t_{k+P+1}]$ . This is also shown clearly in Fig. 2, and it implies that for any input  $x$ , there is at most  $P + 1$  non-zero B-spline activations.

To summarize, the B-spline unit must first handle the alignment required by Eq. 4, which enables the use of stored  $B_{0,P}$  values. Then, relying on translation-invariance, it must infer the value of all  $P + 1$  non-zero B-splines. Moreover, to efficiently tabulate  $B_{0,P}$ , we must leverage the fact that the cardinal B-spline is symmetric with respect to the midpoint of the support (for  $B_{0,P}$ , that is  $[0, \dots, P + 1]$ ),  $\frac{P+1}{2}$  [19]. Therefore, we only need to store half the B-spline corresponding to the interval  $[0, \frac{P+1}{2}]$ . For instance, as shown in Fig. 4, for  $P = 3$ , we only need to sample points from the interval  $[0, 2]$ ; for an input  $x$ , only  $P+1 = 4$  non-zero B-splines exist:  $B_{t_{k-3},3}(x)$ ,  $B_{t_{k-2},3}(x)$ ,  $B_{t_{k-1},3}(x)$ , and  $B_{t_k,3}(x)$ . The aligned  $x_a$  input shown in the figure ensures that these values correspond respectively to  $B_{-3,P}(x_a)$ ,  $B_{-2,3}(x_a)$ ,  $B_{-1,3}(x_a)$ , and  $B_{0,3}(x_a)$ .

2) *Implementation of the B-Spline Unit*: Fig. 5 shows the table storing the B-spline values of  $B_{0,P}$  required to infer all the non-zero B-spline activations. The *Align* unit implements the previously mentioned alignment in Eq. 4 and the *Compare* unit performs an interval search to compute  $k$ , necessary for both the alignment as well as the systolic array, as will be shown in Sec. IV. inputs  $x_q$  and  $t_q$ . Using affine integer quantization scheme with Eq. 4 we obtain the LUT address computed in *Align* unit as The LUT address for retrieving the stored value for  $x$  is the quantized aligned input  $x_a \in [0, 1]$  to the integer range  $[0, 255]$ . The unit must obtain  $x_{addr}$  using integer arithmetic based on its quantized

$$x_{addr} = \text{clip}((G + 2P)(x_q - t_{q0}) - 255 \times k, 0, 255). \quad (5)$$

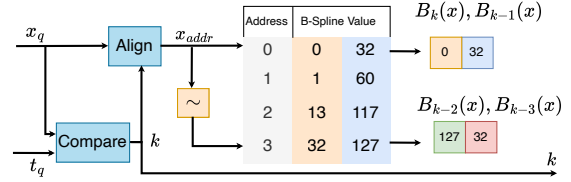


Fig. 5: Simplified version of the B-Spline unit. The address is inverted, and the corresponding values are reverse-packed to obtain the remaining non-zero activations. The example values 0, 32 at the output correspond to  $x_{addr} = 0$ . The unit also returns the values at  $\sim x_{addr} = 3$  in reverse. i.e. 127, 32.

Since we use an address based on  $x_a \in [0, 1]$  and we need to store values in  $[0, 2]$ . We simply store the value corresponding to  $x_a + 1$  in the same address as  $x_a$ . This value corresponds to  $B_{t_{k-1},3}(x)$  as shown in Fig. 4, hence the two values per row in Fig. 5. For the other values  $B_{t_{k-3},3}(x)$ ,  $B_{t_{k-2},3}(x)$ , which correspond to  $B_{0,3}(1 - x_a)$  and  $B_{0,3}(1 - x_a + 1)$ , they are stored in the address corresponding to  $1 - x_a$ , i.e.,  $255 - x_{addr}$ . This is implemented in the inversion unit  $\sim$  in Fig. 5.

#### IV. LEVERAGING B-SPLINES PROPERTIES IN KAN-SAS PROCESSING ELEMENTS

In this section, we present the analysis of the B-spline properties presented previously, and how they enable the efficient acceleration of KAN networks on SA-based accelerators. In particular, we show how to provide efficient handling of the  $N:M$  sparsity pattern generated by B-splines.

##### A. On the Inherent Sparsity of B-splines

As illustrated in Fig. 2 and explained in Section III-B, B-splines have local support, i.e., the B-spline  $B_{t_k,P}$  is only non-zero within  $[t_k, t_{k+P+1}]$ . Therefore, if  $x \in [t_k, t_{k+1}]$ , then only the  $P + 1$  functions  $B_{t_{k-P}}(x), \dots, B_{t_k}(x)$  are non-zero among all  $G + P$  functions. Moreover, when  $x$  is within the grid extension ( $k < P$  or  $k > G + P - 1$ ), there are even fewer non-zero B-splines. For example, with  $G = 10, P = 3$ , each input would at most contribute 4 non-zero B-spline activations among 13, which leads to at most  $\frac{4}{13} \approx 30\%$  PE utilization (i.e., computations involving non-zero B-spline activations) in the SA (as shown later in Fig. 8).

To address this inefficiency, the B-spline unit must output only the  $P + 1$  contiguous non-zero activations along with the integer index  $k$  specifying their positions among all the  $G + P$  B-splines. This nice property of B-splines enables the use of an  $N:M$  structured sparsity-aware PE (where  $N = P + 1$ ,  $M = G + P$ ) to avoid useless multiplications with zero. This means that while the classical scalar PE in Fig. 3 performs  $psum_i + c_i B_i(x)$ , a KAN-optimized PE has to be vectorized to perform  $psum_i + \sum_{i=0}^P c_{k-i} B_{k-i}(x)$ , i.e. only involving  $B_{k-i}(x)$ ,  $i \in [0, P]$  non-zero B-spline values. This  $N:M$  structured pattern, also known as density bound block (DBB), has been previously enforced in conventional DNNs static weights [20]–[22] through ad-hoc pruning. Conversely, in the KAN case, this  $N:M$  sparsity appears dynamically in the B-spline activations, and it is guaranteed by their local support property.

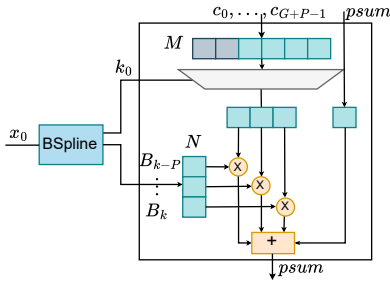


Fig. 6: Proposed KAN-SAs PE architecture showing how the B-spline unit interacts with the  $N:M$  vector PE. Different from the scalar PE SA in Fig. 3, each B-spline unit streams its output to a single row of  $N:M$  PEs.

### B. Designing the Processing Elements of KAN-SAs

A conventional systolic array uses a scalar PE to perform a multiply-accumulate. To exploit the intrinsic B-spline sparsity, we design an  $N : M$  sparsity-aware vector PE that performs a multiply-accumulate between the  $N$  non-zero B-spline values and their corresponding coefficients among the  $M$  coefficients. As shown in Fig. 6, each B-spline unit sends the  $P + 1$  non-zero activations to its corresponding row in the array, along with indices as control signals for the multiplexer selecting the  $N = P + 1$  coefficients among  $M = G + P$ . While the  $N$  multiplications occur in parallel, the additional  $M$ -to- $N$  multiplexer and multi-operand adder increase the critical path delay. Similar to the activations, the indices  $k$  are propagated and reused horizontally along the array.

## V. EVALUATION

In this section, we evaluate the proposed KAN-SAs architecture against the conventional SA setup. In the conventional SA, we assume B-spline units feeding a systolic array with scalar PEs, not able to handle the intrinsic  $N:M$  B-spline sparsity. We synthesized the conventional SA solution and the proposed one with Synopsys Design Compiler v2025.6, targeting the ST 28nm FD-SOI PDK [23]. Our integer-only hardware implementation follows the approach of [18] and was validated against an integer software baseline. The hardware implementation is bitwise-accurate to this baseline, which incurs a  $\sim 1\%$  accuracy drop relative to 32-bit floating point for KAN models due to the integer quantization.

### A. PE Comparison

In this section, we compare the hardware synthesis results for the scalar conventional PE and the  $N:M$  PE, as shown in Table I. Post-synthesis power estimation was performed using activity-based analysis at a frequency of 500MHz. We first examine the delay for different sparsity parameters  $N$  and  $M$ . As discussed in Section IV-B, the delay of the critical path increases due to both the  $M$ -to- $N$  multiplexer and the  $N + 1$ -operands adder. For example, increasing  $N$  from 2:6 to 4:6 in Table I results in increased delay due to the adder, as multiplexing occurs in parallel. Similarly, increasing  $M$  affects the multiplexer delay but not the adder. As shown in the table, the additional logic also increases the power of  $N:M$  PEs compared to 1:1 PEs. Switching activity was extracted from simulation traces for a

TABLE I: ST28nm FD-SOI post-synthesis delay and power for 8-bit inputs and 32-bit output PE at a target frequency of 500 MHz and estimated normalized energy. Columns refer to the sparsity pattern  $N:M$ ; 1:1 represents the scalar PE.

N:M	1:1	1:2	2:4	2:6	4:6	4:8
Delay (ns)	1.02	1.05	1.15	1.19	1.28	1.31
Power (mW)	0.35	0.40	0.62	0.77	0.98	1.12
Normalized Energy	1.00	0.57	0.44	0.37	0.47	0.40

typical KAN workload, where coefficients are loaded into the PE and reused for several cycles with different activations. The normalized energy in the table highlights the advantage of the  $N:M$  PE. It is estimated by multiplying the reported power by the number of cycles needed for a KAN workload to run on a scalar PE. With  $N = P + 1$ ,  $M = G + P$ , the 1:1 PE requires  $(G + P) \times$  more cycles than a PE handling  $N:M$  sparsity. For example, for  $P = G = 3$  (sparsity 4:6), the 1:1 PE requires  $6 \times$  more cycles than the 4:6 PE.

### B. B-spline Acceleration Comparison with Previous Work

As already mentioned, ArKANE [13] proposes an acceleration of the recursive implementation of B-splines. A direct comparison is not straightforward, as ArKANE B-spline implementation evaluates the B-splines using floating-point arithmetic across multiple cycles for the objective of KAN training acceleration. Moreover, the reported results correspond to a mapping on Xilinx FPGA over multiple AIE tiles. However, to propose a fair comparison, we estimate the possible area consumption for ArKANE proposal and compare it to KAN-SAs B-spline unit. ArKANE wavefront algorithm requires  $P + 1$  processing elements and has a latency of  $(P + 1) \times PE_{latency}$  cycles to evaluate a specific B-spline  $B_{i,P}$ . Then, thanks to pipelining, it requires  $G + P - 1$  cycles for all  $G + P$  activations. Hence, we can compute the ArKANE number of cycles for  $M$  inputs as  $(P + 1)PE_{latency} + G + P - 1 + M$ . We consider an existing single-precision FMA implementation, FPMax [24], as a reference for the ArKANE FP32 multiply-accumulate (FMA) operation in each PE. In FPMax, a single-precision FMA circuit has  $PE_{latency} = 4$  and is estimated to occupy approximately  $0.0081mm^2$  of standard cell area. By contrast, our tabulation-based B-spline unit occupies  $450\mu m^2$ , and requires at most a single cycle to retrieve the values of all  $G + P$  B-splines for a certain input. Therefore, in the same estimated area for ArKANE, i.e.,  $4 \times 0.0081mm^2$  we can fit 72 B-spline units to feed 72 rows of the systolic array. Tabulating the B-splines can offer a minimum of  $72 \times$  speedup for high values of  $M$  over the recursive floating-point implementation.

### C. KAN Applications Benchmark

We perform a design space exploration of different configurations of the proposed architecture on a representative set of KAN applications collected from prior work and reported in Table II. Each of the applications contributes a certain number of KAN workloads, i.e., matrix multiplications where the left matrix is B-spline activations  $B$  mentioned in Section II-A and the right matrix is the coefficients. For instance, the application Catch22-KAN relies on a single KAN layer [22,  $X$ ] where  $X$  is the

TABLE II: Collected KAN workloads from prior work.  $X$  in CF-KAN takes values from [2810, 34395, 6969]. In Catch22-KAN,  $X$  is the number of classes in UCR time series dataset, and we consider it smaller than 60 in our experiments.

Application	Layers	G	P
5G-STARBUST [2]	[168, 40, 40, 40, 24]	5	3
Catch22-KAN [26]	[22, X]	3	3
CF-KAN [3]	[X, 512, X]	2	3
U-KAN [4]	[512, 1024, 512], [512, 512]	5	3
GKAN [15]	[200, 16, 7], [100, 20, 7]	2,3	1,2,3
Prefetcher [27]	[5, 64, 128]	4	3
MNIST-KAN [28]	[784, 64, 10]	10	3
ResKAN18 [29]	20 ConvKAN layers	3	3

number of classes for one of the UCR datasets [25]. This layer implies a matrix  $B$  of dimensions  $(BS, 22 \times (G + P))$ . Some of these workloads also include the MLP bias term mentioned in Section II-A, which is included in the evaluation. Some studies explore different values of  $G$  and  $P$ . However, we limit our evaluation to workloads with  $P \leq 3$ . The applications MNIST-KAN and ResKAN18 are models we implemented and trained on MNIST [30] and CIFAR10 [31], respectively. MNIST-KAN is a two-layer KAN [784, 64, 10], and the ResKAN18 is a ResNet18 architecture, where the scalar filter weights in convolutions are replaced by learnable splines (referred to as ConvKAN or KAN-Conv in prior works [16], [32]). The average results reported in Figs. 7a and 7b consider different dimensions  $R$  rows and  $C$  columns for the array. In the graphs, for reference, we mark some points that correspond to square SAs (e.g.,  $2 \times 2$ ,  $4 \times 4$ , etc.). The other parameters are fixed as int8 multiplication and int32 accumulation,  $G = 5$  and  $P = 3$ . Hence, results are averaged over all collected workloads except MNIST-KAN, as it requires  $G = 10$ . Since the main objective of the study is to evaluate the proposed improvements, we focus solely on B-spline sparsity without considering other dynamic sources of sparsity, such as zero coefficients (i.e., weights) or activations. The workloads are tiled for running on the weight-stationary SA.

In Fig. 7a, we plot, for both conventional SA and KAN-SAs, the average PE utilization across all applications vs. the area obtained post-synthesis, for various sizes of the SA ( $R \times C$ ). The figure shows that the proposed solution improves the average

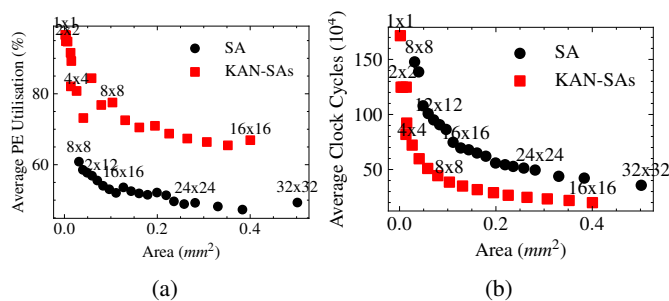


Fig. 7: Average PE utilization (a) and runtime in clock cycles (b) across all applications vs. the area obtained post-synthesis for both conventional SA and KAN-SAs, for various sizes of the SA ( $R \times C$ ).

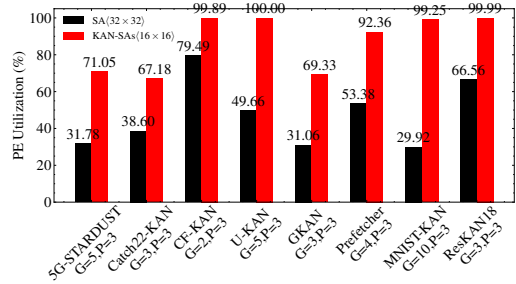


Fig. 8: PE Utilization(%) for KAN workloads.

PE utilization of the SA for various design parameters. Although KAN-SAs consistently provide PE utilization  $> 65\%$ , there is still some effect preventing 100%. PE utilization is affected by two main factors: (1) sparsity from the B-splines, addressed by KAN-SAs, and (2) imperfect tiling, which occurs when workload dimensions are not multiples of the SA's dimensions. Indeed, in Fig. 8, we show per-application average PE utilization for configurations with similar areas:  $0.47mm^2$  for KAN-SAs  $16 \times 16$  and  $0.50mm^2$  for the  $32 \times 32$  scalar PE SA. Since KANs are still in their early stages, most applications have small dimensions. This leads to a noticeable imperfect tiling effect in these applications. Additionally,  $G$  values affect utilization: (i) higher  $G$  values increase the number of parameters, reducing the impact of imperfect tiling, and (ii) they also increase sparsity, reducing utilization in conventional SA. For example, MNIST-KAN with  $G = 10$  shows low utilization (30%) on SA but high utilization (99.25%) on KAN-SAs. Larger applications like ResKAN18, CF-KAN, U-KAN, and Prefetcher (see Table II) suffer less from imperfect tiling, even at low  $G$  values, which also boosts PE utilization in conventional SA. On average, KAN-SAs improve utilization by 39.9%, with a maximum of 69.3% for MNIST-KAN.

Finally, Fig. 7b shows that, for both conventional SA and KAN-SAs, the average runtime (in clock cycles) across all applications decreases with area. KAN-SAs consistently reduce runtime by  $2 \times$  at a constant area, or area by  $2 \times$  at a constant runtime. This improvement is due to KAN-SAs loading  $(R \times M, C)$  tiles of B-spline activations or  $(R \times N, C)$  tiles of non-KAN workloads, leading to faster execution.

## VI. CONCLUSION

In this paper, we presented KAN-SAs, the first systolic-array-based efficient hardware accelerator for KANs. KAN-SAs efficiency is enabled by a thorough analysis of the inefficiencies that the B-spline functions inherently introduce, i.e., recursive nature and sparsity. KAN-SAs addresses them through an enhanced Processing Element including a non-recursive B-spline unit and efficient handling of B-spline sparsity. Results show that, compared to conventional SAs, KAN-SAs achieves 39.9% average PE utilization improvement, 50% average clock cycle reduction, and  $72 \times$  speedup in B-spline evaluation compared to the state-of-the-art approach.

## ACKNOWLEDGMENT

This work was supported by the French National Research Agency through the RADYAL project ANR-23-IAS3-0002.

## REFERENCES

- [1] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson *et al.*, “Kan: Kolmogorov-arnold networks,” 2024, arXiv:2404.19756 [cs.LG].
- [2] C. J. Vaca-Rubio, L. Blanco, R. Pereira, and M. Caus, “Kolmogorov-Arnold Networks (KANs) for Time Series Analysis,” Sep. 2024, arXiv:2405.08790 [eess.SP].
- [3] J.-D. Park, K.-M. Kim, and W.-Y. Shin, “Cf-kan: Kolmogorov-arnold network-based collaborative filtering to mitigate catastrophic forgetting in recommender systems,” 2024, arXiv:2409.05878 [cs.IR].
- [4] C. Li, X. Liu, W. Li, C. Wang, H. Liu *et al.*, “U-KAN Makes Strong Backbone for Medical Image Segmentation and Generation,” Aug. 2024, arXiv:2406.02918 [eess.IV].
- [5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal *et al.*, “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.
- [6] “Nvidia tesla v100 gpu architecture: The world’s most advanced data center gpu,” Whitepaper, <https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>, 2017.
- [7] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.
- [8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
- [9] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan *et al.*, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” 2017, arXiv:1708.0448 [cs.NE].
- [10] H. T. Kung, “Why systolic architectures?” *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.
- [11] C. Sudarshan, P. Manea, and J. P. Strachan, “A Kolmogorov–Arnold Compute-in-Memory (KA-CIM) Hardware Accelerator with High Energy Efficiency and Flexibility,” Jan. 2025, preprint 10.21203/rs.3.rs-5804189/v1.
- [12] W.-H. Huang, J. Jia, Y. Kong, F. Waqar, T.-H. Wen *et al.*, “Hardware Acceleration of Kolmogorov-Arnold Network (KAN) for Lightweight Edge Inference,” in *30th Asia and South Pacific Design Automation Conference*. ACM, Jan. 2025, pp. 693–699.
- [13] Y. Wu and M. T. Arafin, “Arkane: Accelerating kolmogorov-arnold networks on reconfigurable spatial architectures,” *IEEE Embedded Systems Letters*, pp. 1–1, 2025.
- [14] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park *et al.*, “Sparse-TPU: adapting systolic arrays for sparse matrices,” in *34th ACM International Conference on Supercomputing*, Jun. 2020, pp. 1–12.
- [15] M. Kiamari, M. Kiamari, and B. Krishnamachari, “GKAN: Graph Kolmogorov-Arnold Networks,” Jun. 2024, arXiv:2406.06470 [cs].
- [16] A. D. Bodner, A. S. Tepsich, J. N. Spolski, and S. Pourteau, “Convolutional Kolmogorov-Arnold Networks,” Mar. 2025, arXiv:2406.13155 [cs].
- [17] C. De Boor, “On calculating with b-splines,” *Journal of Approximation theory*, vol. 6, no. 1, pp. 50–62, 1972.
- [18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” 2017, arXiv:1712.05877 [cs.LG].
- [19] T. Lyche, C. Manni, and H. Speleers, *Foundations of Spline Theory: B-Splines, Spline Approximation, and Hierarchical Refinement*. Springer International Publishing, 2018, pp. 1–76.
- [20] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, “S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration,” 2022, arXiv:2107.07983 [cs.AR].
- [21] H.-J. Kang, “Accelerator-aware pruning for convolutional neural networks,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 2093–2103, 2020.
- [22] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, “Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus,” in *52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-52, 2019, p. 359–371.
- [23] A. Cathelin, “Fully Depleted Silicon on Insulator Devices CMOS: The 28-nm Node Is the Perfect Technology for Analog, RF, mmW, and Mixed-Signal System-on-Chip Integration,” *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 18–26, 2017.
- [24] J. Pu, S. Galal, X. Yang, O. Shacham, and M. Horowitz, “FPMax: a 106GFLOPS/W at 217GFLOPS/mm<sup>2</sup> Single-Precision FPU, and a 43.7GFLOPS/W at 74.6GFLOPS/mm<sup>2</sup> Double-Precision FPU, in 28nm UTBB FDSOI,” 2016, arXiv:1606.07852 [cs.AR].
- [25] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu *et al.*, “The ucr time series archive,” *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, 2019.
- [26] A. Ismail-Fawaz, M. Devanne, S. Berretti, J. Weber, and G. Forestier, “Feature-based time series classification with kolmogorov–arnold networks,” <https://github.com/MSD-IRIMAS/Simple-KAN-4-Time-Series>, 2024.
- [27] D. Kulkarni, B. Bhammar, H. Thaker, P. Dhobi, R. P. Gohil *et al.*, “A case for kolmogorov-arnold networks in prefetching: Towards low-latency, generalizable ml-based prefetchers,” 2025, arXiv:2504.09074 [cs.AR].
- [28] “An efficient implementation of kolmogorov-arnold network,” <https://github.com/Blealtan/efficient-kan>.
- [29] V. Starostin, “Convolutional kan layer,” <https://github.com/StarostinV/convkan>, 2024.
- [30] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [31] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.
- [32] I. Drokun, “Kolmogorov-arnold convolutions: Design principles and empirical studies,” 2024, arXiv:2407.01092 [cs.AR].