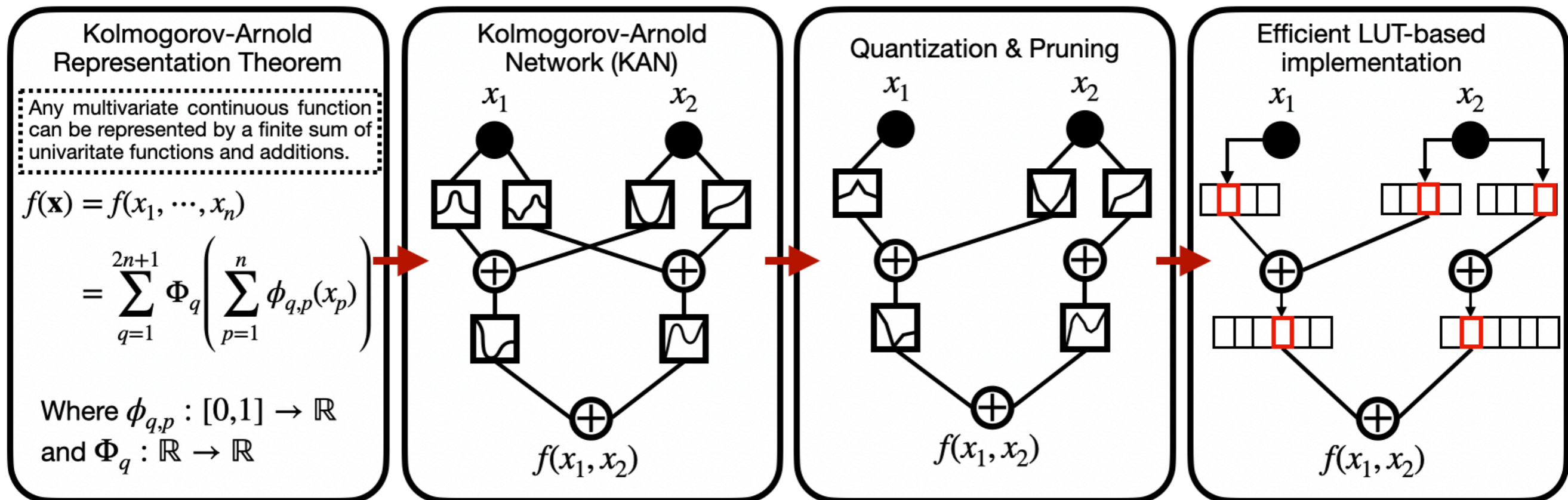
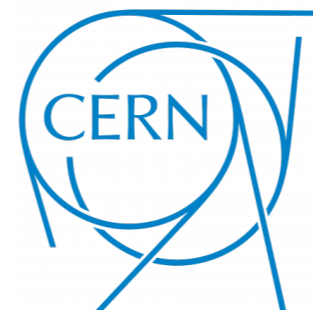


# FPGA-optimized Kolmogorov-Arnold Networks (KANs) via LUT-based design



Duc Hoang, Aarush Gupta, Philip Harris - MIT  
Vladimir Loncar - CERN



# Kolmogorov Arnold Representation Theorem

[Wikipedia]

ArXiv: 2404.19756

If  $f$  is a multivariate continuous function, then  $f$  can be written as a continuous function of a single variable and the binary operation of addition:

$$f(\mathbf{x}) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

Where  $\phi_{q,p} : [0,1] \rightarrow \mathbb{R}$  and  $\Phi_q : \mathbb{R} \rightarrow \mathbb{R}$

In a sense, they showed that the **only true multivariate function is the sum**, so any function can be written using **univariate function and summing**.

# Pictorial understanding

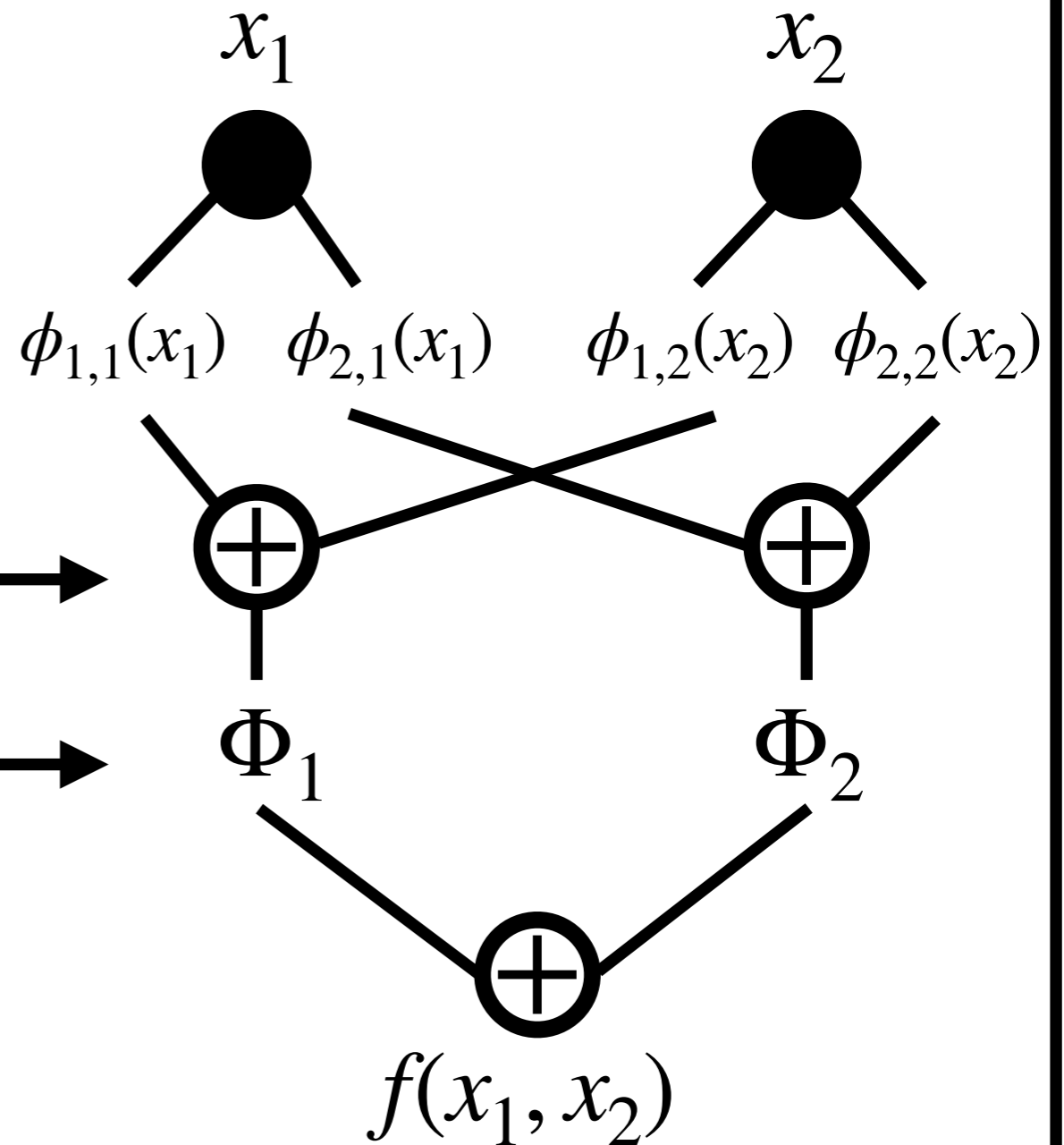
ArXiv: 2404.19756

$$f(\mathbf{x}) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

1D function on **edges**

Sum on **nodes**

1D function on **edges**



# Pictorial understanding

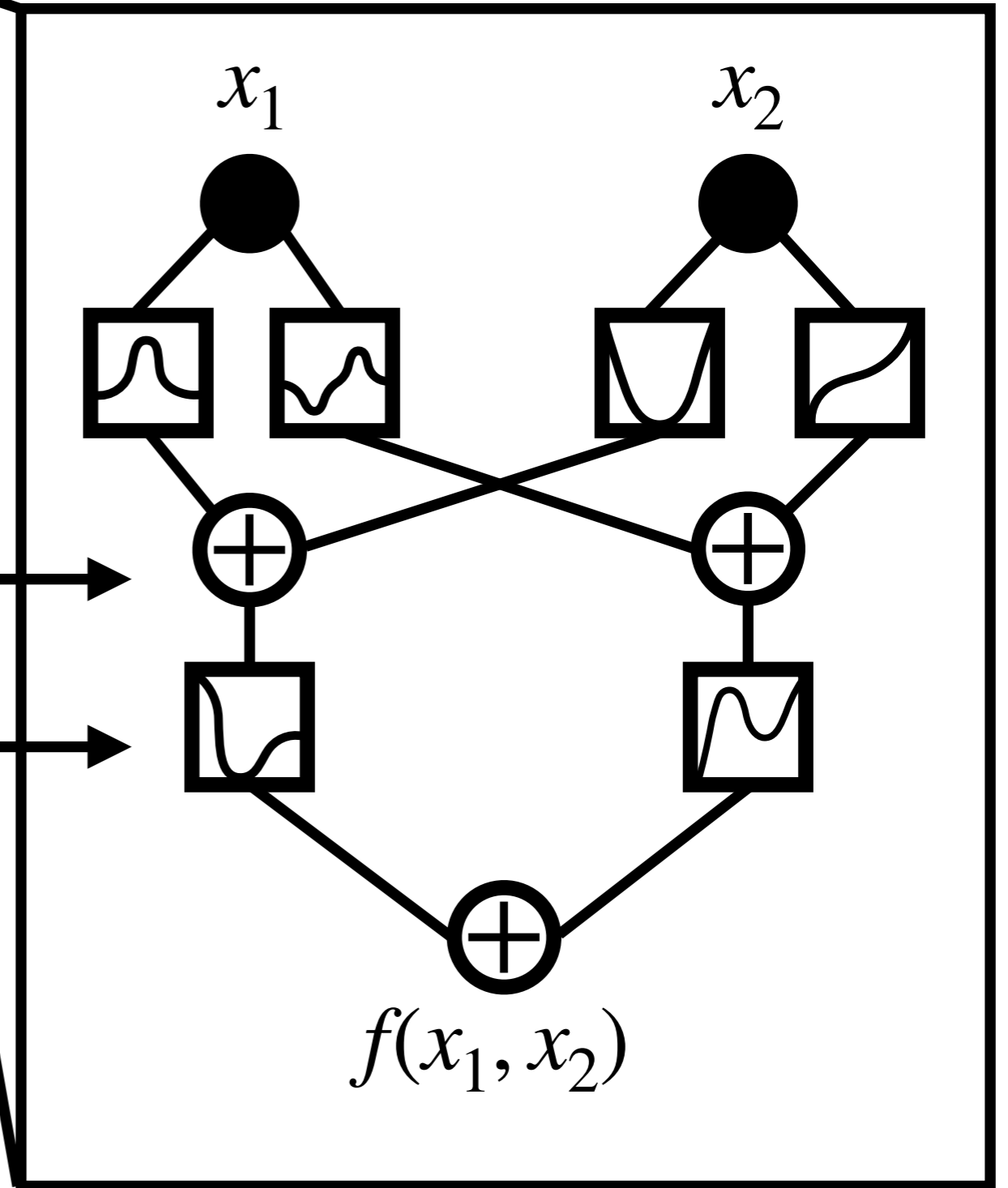
ArXiv: 2404.19756

$$f(\mathbf{x}) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

1D function on *edges*

Sum on *nodes*

1D function on *edges*



# From KART to **KAN**

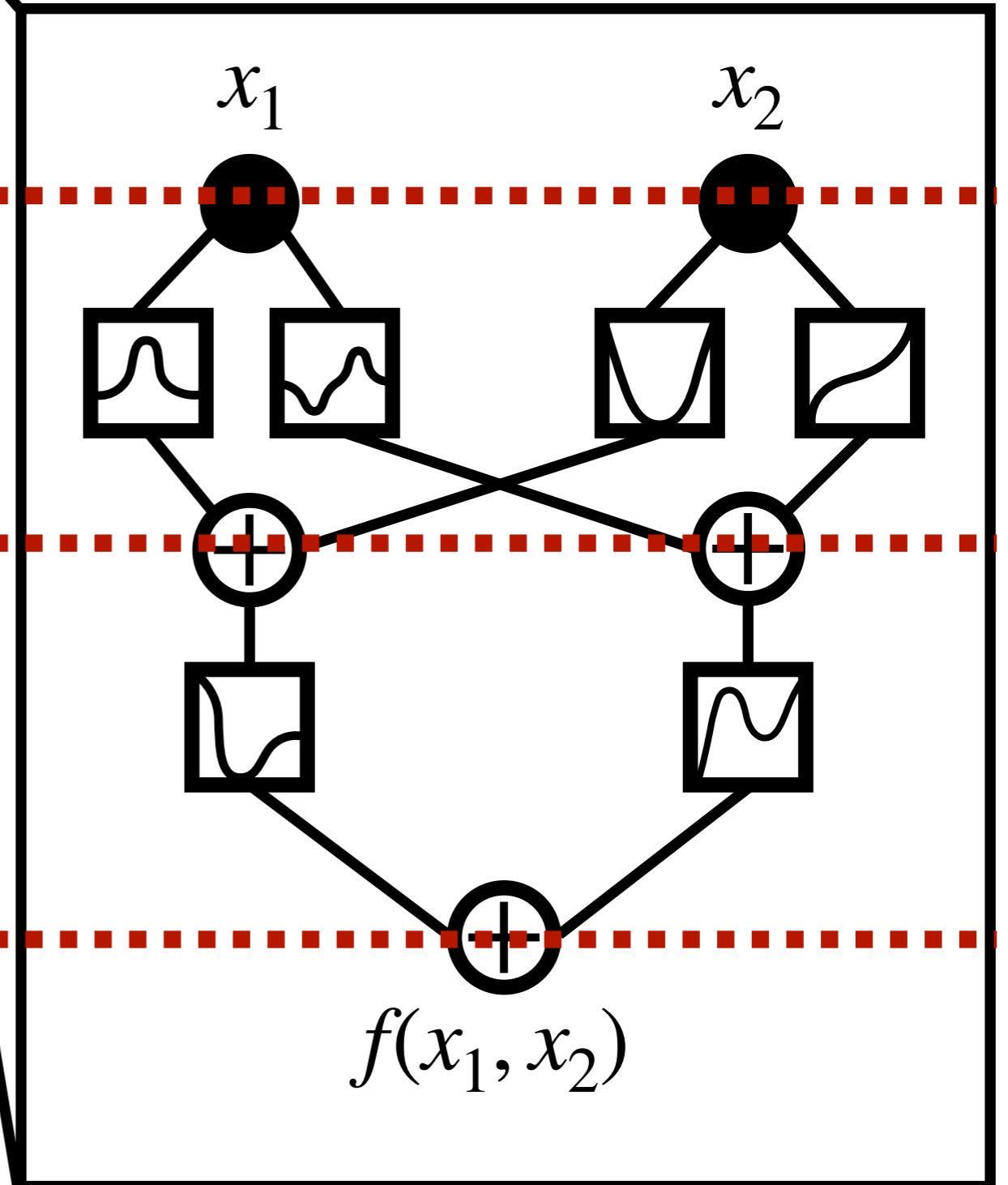
Liu et al. ArXiv: 2404.19756

$$f(\mathbf{x}) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

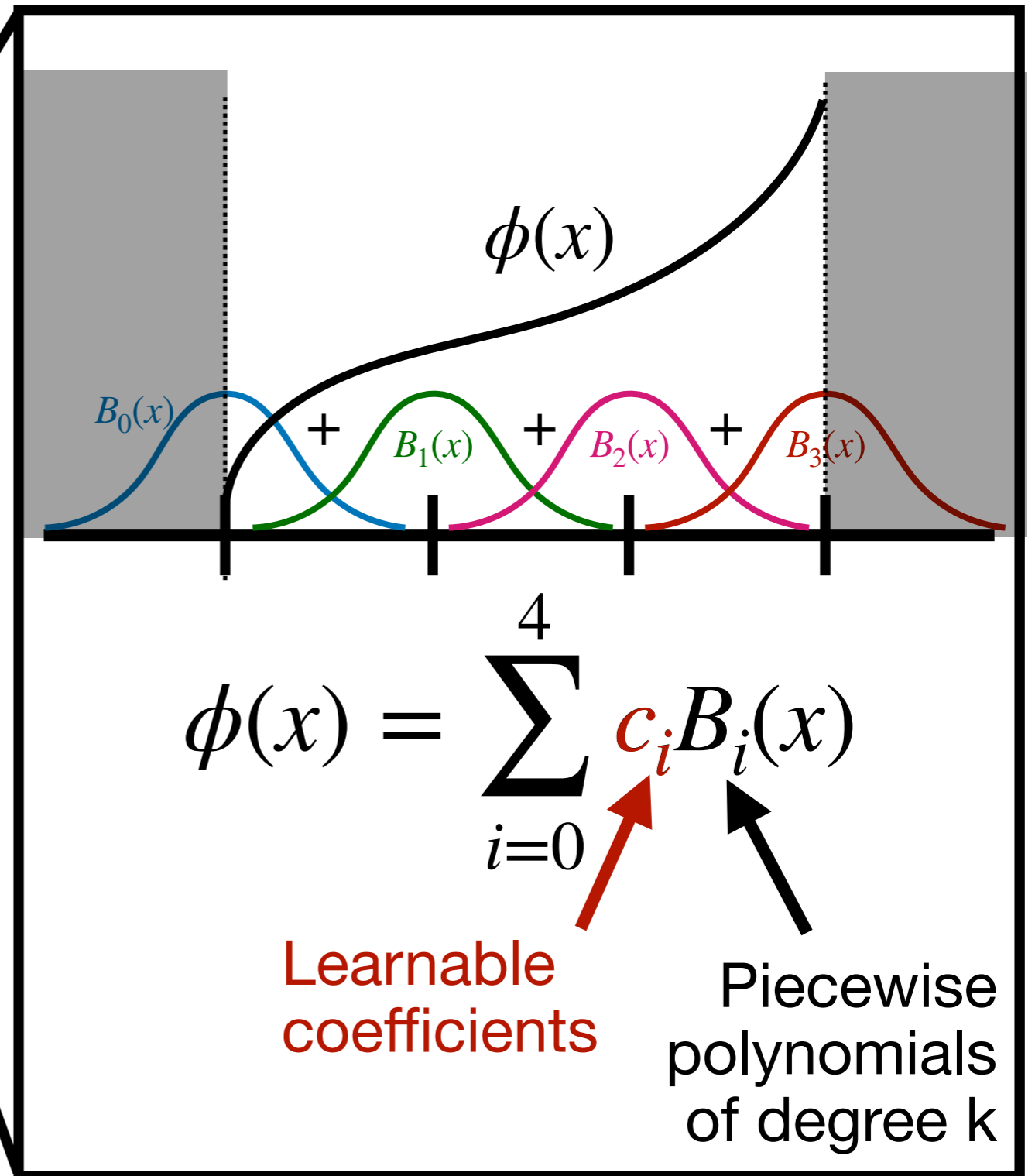
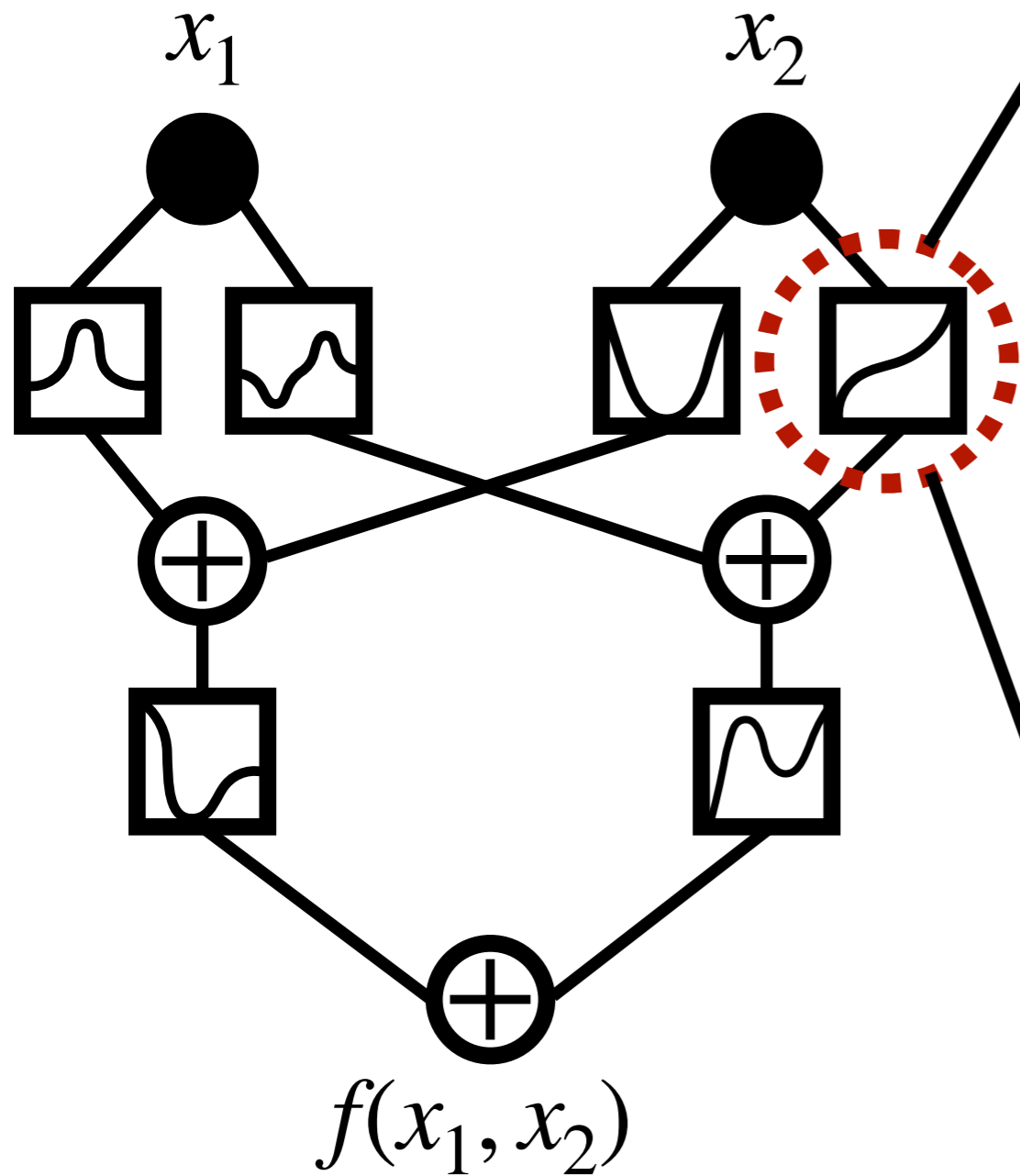
$\Phi_1$ :  $2 \times 2$  KAN Layer

$\Phi_1$ :  $1 \times 2$  KAN Layer

**No linear weight matrices at all**, each weight parameter is replaced by a learnable 1D function parametrized as a spline.



# B(asis)-splines



# If you stop right here ...

## And try to put in on hardware

[ArXiv: 2407.17790](https://arxiv.org/abs/2407.17790)

### Exploring the Limitations of Kolmogorov-Arnold Networks in Classification: Insights to Software Training and Hardware Implementation

Van Duy Tran<sup>1</sup>, Tran Xuan Hieu Le<sup>2</sup>, Thi Diem Tran<sup>2</sup>, Hoai Luan Pham<sup>1</sup>, Vu Trung Duong Le<sup>1</sup>, Tuan Hai Vu<sup>1</sup>, Van Tinh Nguyen<sup>3</sup>, and Yasuhiko Nakashima<sup>1</sup>

<sup>1</sup> Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma, Nara, 630-0192 Japan.

<sup>2</sup> University of Information Technology - VNUHCM, Vietnam.

<sup>3</sup> Le Quy Don technical University, Vietnam.

**Abstract**—Kolmogorov-Arnold Networks (KANs), a novel type of neural network, have recently gained popularity and attention due to the ability to substitute multi-layer perceptrons (MLPs) in artificial intelligence (AI) with higher accuracy and interoperability. However, KAN assessment is still limited and cannot provide an in-depth analysis of a specific domain. Furthermore, no study has been conducted on the implementation of KANs in hardware design, which would directly demonstrate whether KANs are truly superior to MLPs in practical applications. As a result, in this paper, we focus on verifying KANs for classification issues, which are a common but significant topic in AI using four different types of datasets. Furthermore, the corresponding hardware implementation is considered using the Vitis high-level synthesis (HLS) tool. To the best of our knowledge, this is the first article to implement hardware for KAN. The results indicate that KANs cannot achieve more accuracy than MLPs in high complex datasets while utilizing substantially higher hardware resources. Therefore, MLP remains an effective approach for achieving accuracy and efficiency in software and hardware implementation.

**Index Terms**—Kolmogorov-Arnold Networks, KAN, Neural Networks, Classification, FPGA, Hardware

#### I. INTRODUCTION

Nowadays, Artificial Intelligence (AI) increasingly shows the advancement to solve problems in human life [1]. In AI models, multi-layer perceptrons (MLPs) or fully connected neural networks provide the learnable ability and work as the most important component to approximate nonlinear functions [2]. However, Kolmogorov-Arnold Networks (KANs) [3], a new rise neural network architecture for AI and science, has recently been developed based on the Kolmogorov-Arnold representation theorem [4], [5] to replace MLPs due to its advantages in terms of accuracy and interpretability. Indeed, KANs in [3] have demonstrated their power by applying to issues such as partial differential equations (PDEs) [6], continual learning [7], Knot theory [8], Anderson localization [9], etc.

Although KAN could be applied in many applications, its endorsement is still not sufficient due to the limited number and complexity of possible scenarios. Hence, there is a need to implement more evaluations to fairly assess KANs' advantages and disadvantages compared to MLPs. Besides, there is no

evaluation of KANs on hardware design which is essential to ensure KANs are acceptable for applying in practical applications efficiently and cost-effectively. Therefore, it is necessary to analyze KANs in terms of hardware design to clarify any potential concerns regarding the trade-off between accuracy and utilization of hardware resources.

Though KANs have just recently been developed and published, there have been many works applying KANs to a wide range of applications [10]. Bozorgasl and Chen have developed Wavelet Kolmogorov-Arnold Networks to improve the original KANs with Wavelet transform and achieved better accuracy and shorter training time compared to MLPs. In addition, the authors in [11] demonstrate the use of KANs in analyzing time series data for a satellite traffic forecasting task. The findings indicate that KANs outperform MLPs in terms of lower error metrics, higher accuracy, and a reduced number of learnable parameters. Furthermore, KANs are also used to solve PDEs and gain outstanding accuracy in comparison with MLPs in [12] except for the complex geometry issue. Moreover, the authors in [13] used KANs in quantum architecture search (QAS) and obtained notable results although the execution time of KANs-based systems is higher than MLPs-based systems. Additionally, the works in [14], [15] used KANs for the classification problems which are a common but important problem in MLPs-based models and achieve good results compared to MLPs. However, the experimental results are still limited to a small range of classification problems and cannot ensure the applicability of KANs in other classification issues. Besides, all the above previous works only implement their ideas on software platforms to obtain accuracy and there is no evaluation of hardware design about the trade-off between accuracy and hardware resource usage.

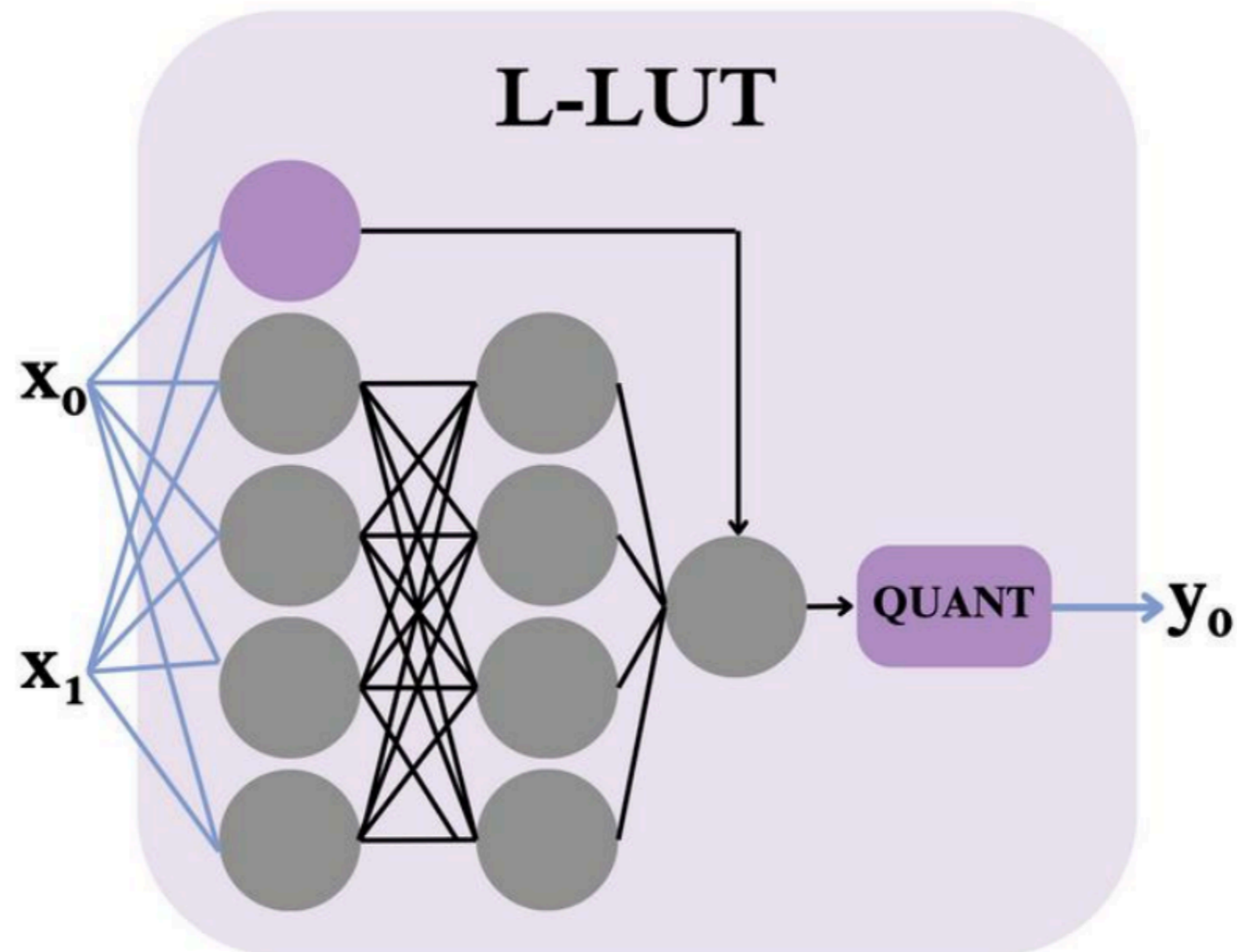
Therefore, in this paper, we introduce an analysis of four kinds of different classification problems to evaluate the effectiveness of KANs on classification problems. These classification problems consist of moons binary classification [16], three-label wine classification [17], seven-label dry bean classification [18], and mushroom binary classification [19]. In addition, we also execute the hardware designs of the MLPs and KANs from the four datasets given above to assess the

synthesis (HLS) tool. To the best of our knowledge, this is the first article to implement hardware for KAN. The results indicate that KANs cannot achieve more accuracy than MLPs in high complex datasets while utilizing substantially higher hardware resources. Therefore, MLP remains an effective approach for achieving accuracy and efficiency in software and hardware implementation.

We argue that they are not looking at KAN inference from the right perspective.

# I was inspired from a lot of LUT-based NNs done in the FastML community

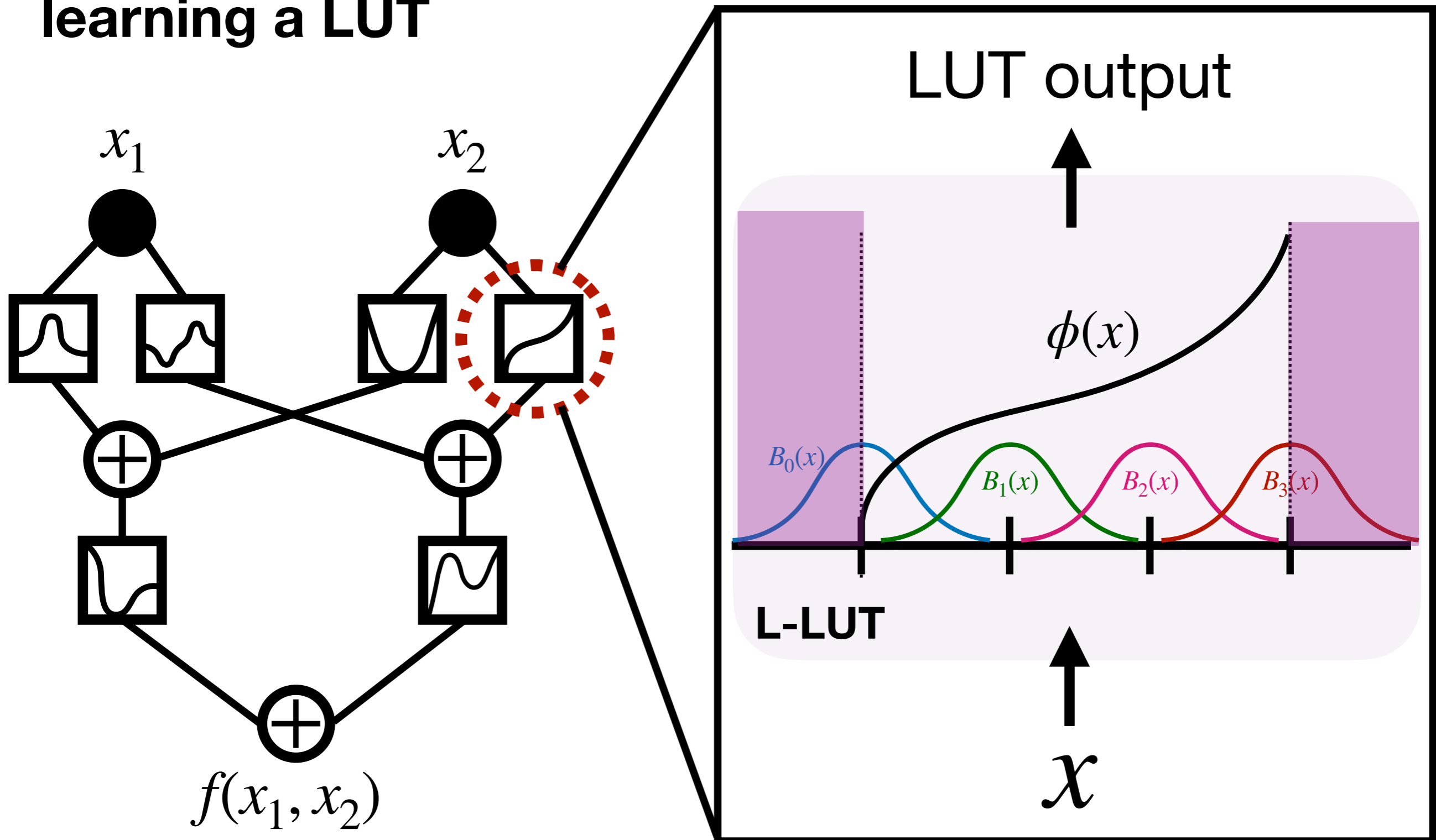
PolyLUT, NeuralLUT, etc.



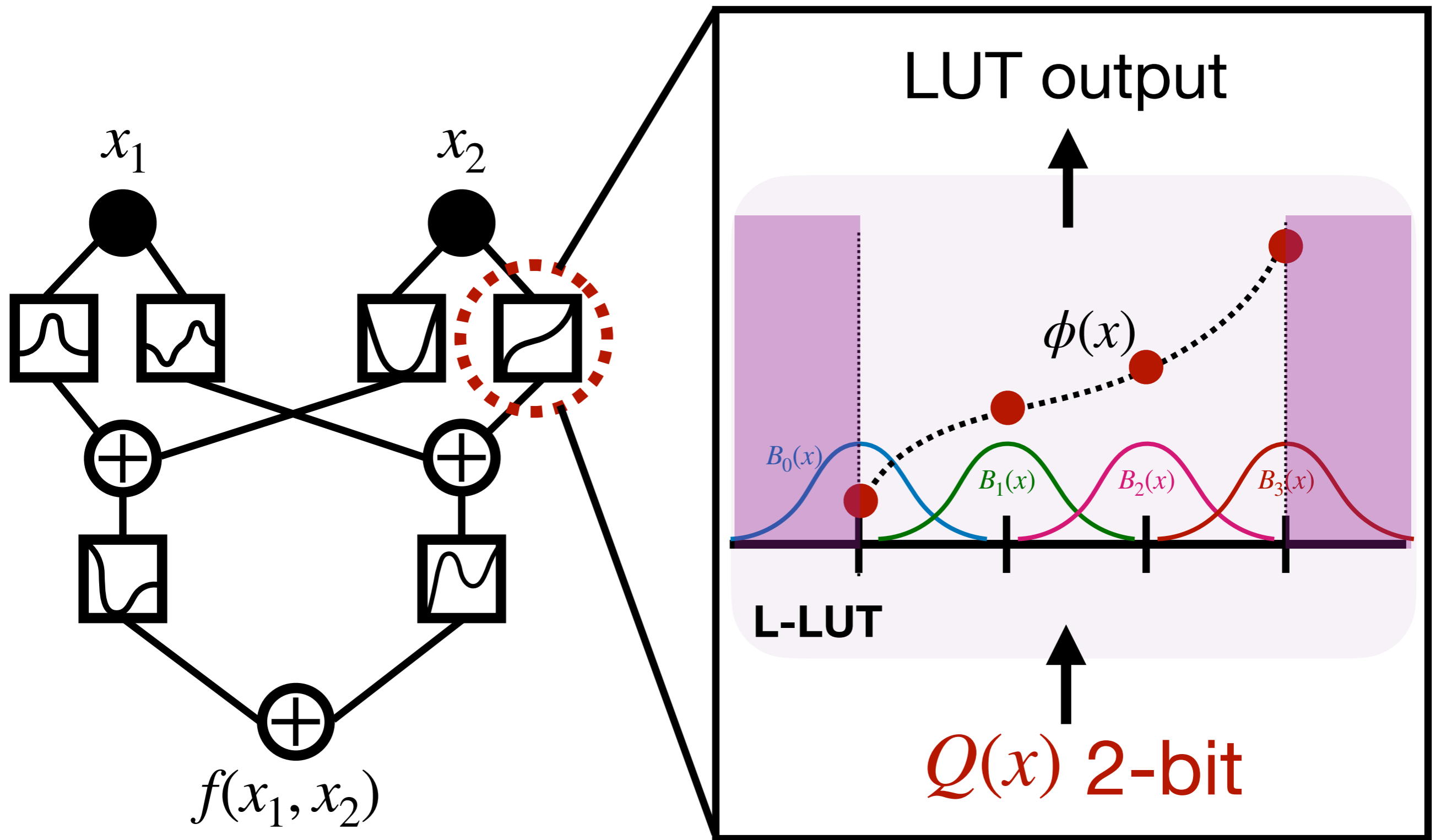
[Excellent tutorial on Monday!](#)

# Looking from this perspective

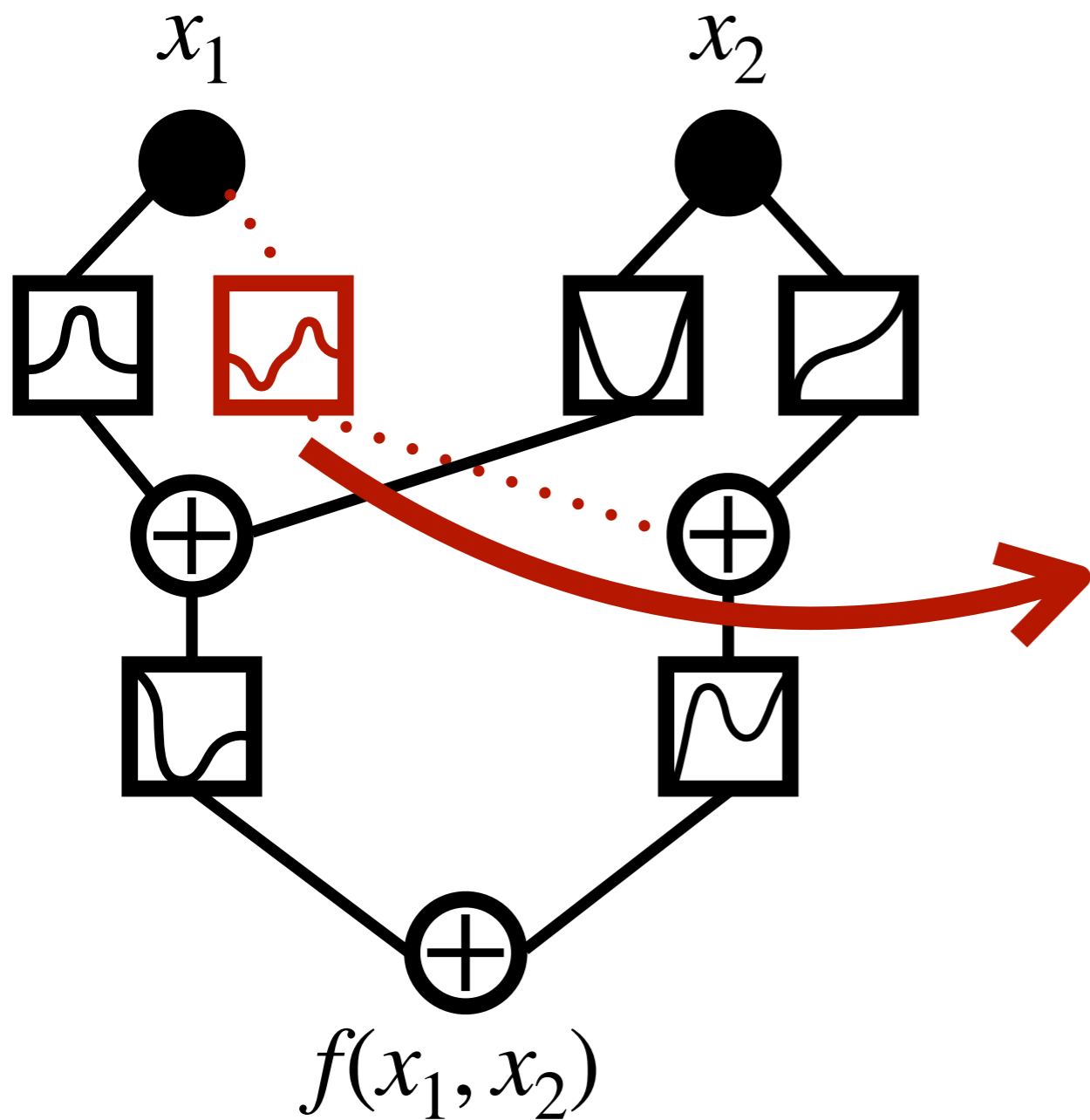
You can think of learning the B-splines as directly learning a LUT



# Then you can quantize it

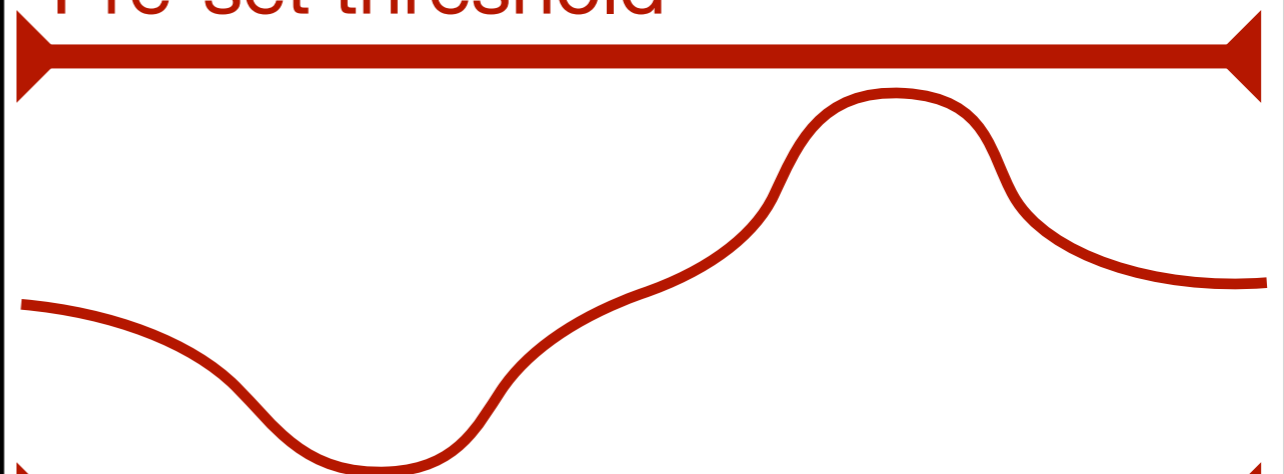


# And then prune KAN based on the magnitude of the output



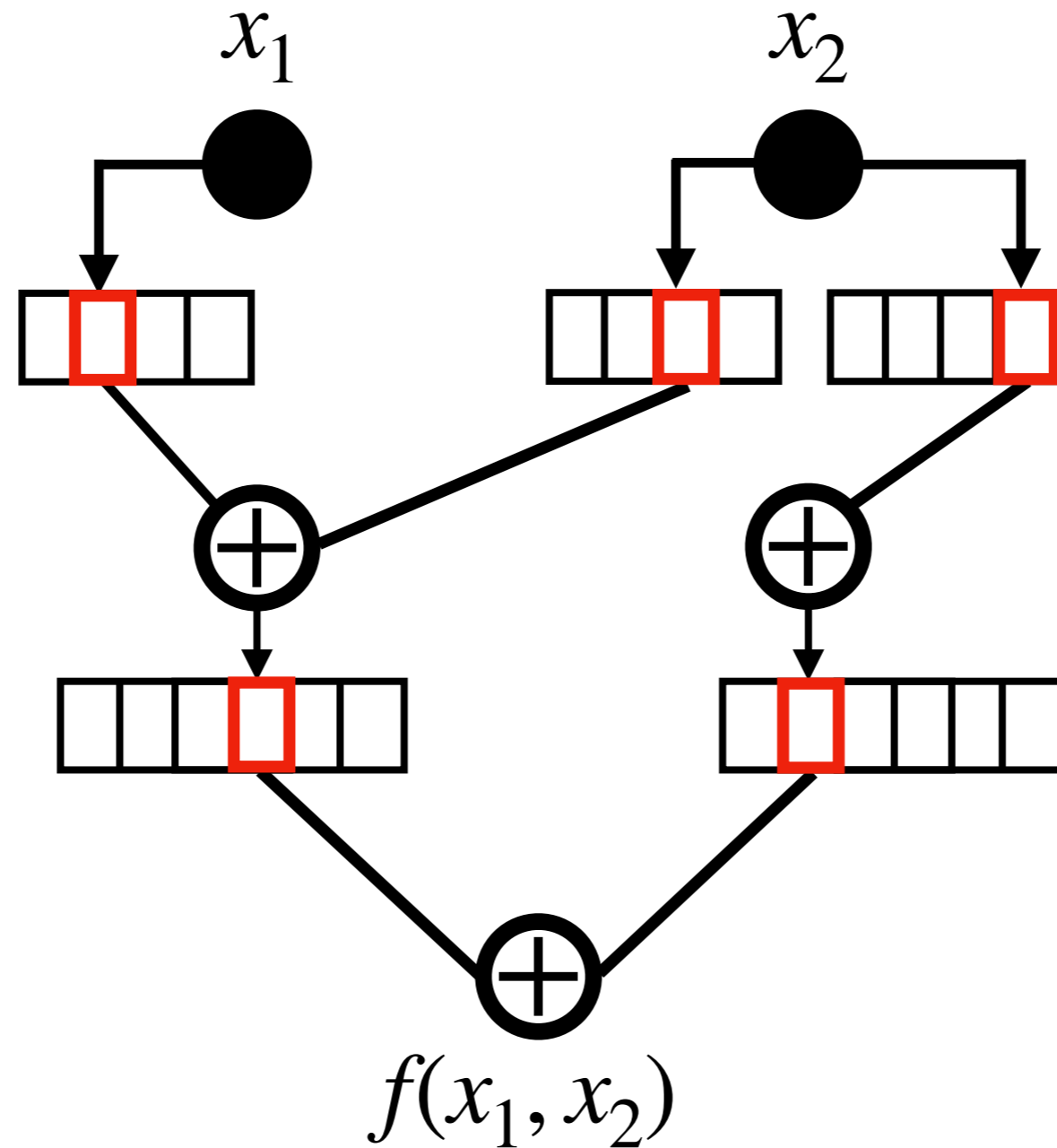
## Magnitude-based pruning

Pre-set threshold



If within prune it

# In the end, you get something like this on firmware



# Benchmarks

## Previous KAN FPGA Benchmarks

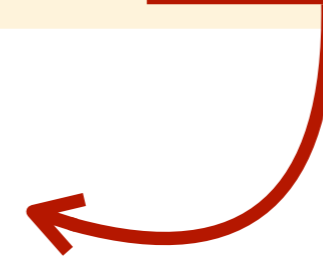
Datasets	Model Dimension	Accuracy (%)		
		MLP FP	KAN FP	KAN Quantized & Pruned
<b>KAN FPGA Benchmarks [28]</b>				
Moons	[2, 2, 1]	87.2	<b>97.7</b>	97.3
Wine	[13, 4, 3]	96.3	<b>98.1</b>	96.3
Dry Bean	[16, 2, 7]	90.9	<b>92.2</b>	91.9
<b>LUT-based neural network benchmarks</b>				
MNIST	[784, 62, 10]	96.7	<b>97.9</b>	96.1
JSC CERNBox	[16, 8, 5]	73.0	<b>75</b>	73.3
JSC OpenML	[16, 8, 5]	76.5	<b>76.5</b>	74.8

## LUT-based NNs benchmarks

# Benchmarks

Datasets	Model Dimension	Accuracy (%)		
		MLP FP	KAN FP	KAN Quantized & Pruned
<b>KAN FPGA Benchmarks [28]</b>				
Moons	[2, 2, 1]	87.2	<b>97.7</b>	97.3
Wine	[13, 4, 3]	96.3	<b>98.1</b>	96.3
Dry Bean	[16, 2, 7]	90.9	<b>92.2</b>	91.9
<b>LUT-based neural network benchmarks</b>				
MNIST	[784, 62, 10]	96.7	<b>97.9</b>	96.1
JSC CERNBox	[16, 8, 5]	73.0	<b>75</b>	73.3
JSC OpenML	[16, 8, 5]	76.5	<b>76.5</b>	74.8

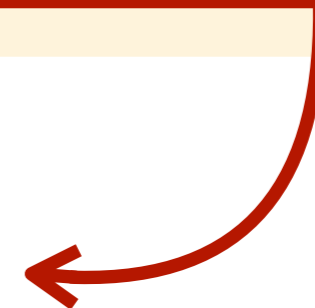
Theoretically, if you convert an MLP of the same dimension in floating point precision to LUTs, it will not reach the same accuracy as a KAN FP LUT.



# Benchmarks

Datasets	Model Dimension	Accuracy (%)		
		MLP FP	KAN FP	KAN Quantized & Pruned
<b>KAN FPGA Benchmarks [28]</b>				
Moons	[2, 2, 1]	87.2	<b>97.7</b>	97.3
Wine	[13, 4, 3]	96.3	<b>98.1</b>	96.3
Dry Bean	[16, 2, 7]	90.9	<b>92.2</b>	91.9
<b>LUT-based neural network benchmarks</b>				
MNIST	[784, 62, 10]	96.7	<b>97.9</b>	96.1
JSC CERNBox	[16, 8, 5]	73.0	<b>75</b>	73.3
JSC OpenML	[16, 8, 5]	76.5	<b>76.5</b>	74.8

Quantization and Pruning scheme results in small loss of accuracy, however can still be improved.



# Benchmarks

Datasets	Model Dimension	Accuracy (%)		
		MLP FP	KAN FP	KAN Quantized & Pruned
<b>KAN FPGA Benchmarks [28]</b>				
Moons	[2, 2, 1]	87.2	97.7	97.3
Wine	[13, 4, 3]	96.3	98.1	96.3
Dry Bean	[16, 2, 7]	90.9	92.2	91.9
<b>LUT-based neural network benchmarks</b>				
MNIST	[784, 62, 10]	96.7	97.9	96.1
JSC CERNBox	[16, 8, 5]	73.0	75	73.3
JSC OpenML	[16, 8, 5]	76.5	76.5	74.8

Quantization and Pruning scheme results in small loss of accuracy, however can still be improved.



My head clears up after talking to Marta yesterday and I was able to implement another version with very minimal loss! Thank you!

# Coming back to the earlier claims

synthesis (HLS) tool. To the best of our knowledge, this is the first article to implement hardware for KAN. The results indicate that KANs cannot achieve more accuracy than MLPs in high complex datasets while utilizing substantially higher hardware resources. Therefore, MLP remains an effective approach for achieving accuracy and efficiency in software and hardware implementation.

Dataset	Model	Accuracy (%)	$F_{\max}$ (MHz)	BRAM	DSP	FF	LUT	Cycles	Latency (ns)	Area×Delay (LUT × ns)
Moons	<b>KAN (ours)</b>	<b>97.3</b>	<b>1109</b>	0	0	<b>30</b>	<b>39</b>	<b>3</b>	<b>2.7</b>	<b><math>1 \times 10^2</math></b>
	KAN (Tran et al) [28]	96.8	100	10	120	8622	17877	128	1280	$2.3 \times 10^7$
Wine	<b>KAN (ours)</b>	96.3	<b>316</b>	0	0	<b>260</b>	<b>590</b>	<b>3</b>	<b>9.5</b>	<b><math>5.6 \times 10^3</math></b>
	KAN (Tran et al) [28]	<b>97.3</b>	100	132	950	74741	146843	688	6880	$1 \times 10^9$
Dry Bean	<b>KAN (ours)</b>	91.9	<b>299</b>	0	0	<b>240</b>	<b>584</b>	<b>3</b>	<b>10</b>	<b><math>5.4 \times 10^3</math></b>
	KAN (Tran et al) [28]	91.9	100	781	9111	734544	1677558	1896	18960	$3.2 \times 10^{10}$

Up to 1000x reduction

Up to  $10^6$   
reduction

**So KAN inference “KAN” be very efficient.**

# Competitive with other LUT-based designs

## Inherently a bit lower latency due to structure

Dataset	Model	Accuracy (%)	LUT	FF	DSP	BRAM	$F_{\max}$ (MHz)	Latency (ns)	Area×Delay (LUT×ns)
MNIST	<b>KAN-Quantized-Pruned</b>	96.1	<b>1323</b>	<b>546</b>	0	0	316	9.6	$1.27 \times 10^4$
	NeuraLUT-Assemble [4]	<b>97.9</b>	5070	725	0	0	863	<b>2.1</b>	<b><math>1.06 \times 10^4</math></b>
	TreeLUT [16]	96.6	4478	597	0	0	791	2.5	$1.12 \times 10^4$
	DWN [5]	97.8	2092	1757	0	0	873	9.2	$1.92 \times 10^4$
	PolyLUT-Add [20]	96.0	14810	2609	0	0	625	10	$1.48 \times 10^5$
	AmigoLUT-NeuraLUT [34]	95.5	16081	13292	0	0	<b>925</b>	7.6	$1.22 \times 10^5$
	NeuraLUT [3]	96.0	54798	3757	0	0	431	12	$6.58 \times 10^5$
	PolyLUT [2]	97.5	75131	4668	0	0	353	17	$1.38 \times 10^6$
	FINN [30]	96.0	91131	—	0	5	200	310	$2.82 \times 10^7$
	hls4ml (Ngadiuba et al.) [23]	95.0	260092	165513	0	345	200	190	$4.94 \times 10^7$
JSC CERNBox	<b>KAN-Quantized-Pruned</b>	73.3	<b>1302</b>	<b>612</b>	0	0	338	8.9	<b><math>1.15 \times 10^4</math></b>
	NeuraLUT-Assemble [4]	75.0	8539	1332	0	0	352	<b>5.7</b>	$4.87 \times 10^4$
	AmigoLUT-NeuraLUT [34]	74.4	42742	4717	0	0	<b>520</b>	9.6	$4.10 \times 10^5$
	PolyLUT-Add [20]	75.0	36484	1209	0	0	315	16	$5.84 \times 10^5$
	NeuraLUT [3]	75.0	92357	4885	0	0	368	14	$1.29 \times 10^6$
	PolyLUT [2]	<b>75.1</b>	246071	12384	0	0	203	25	$6.15 \times 10^6$
	LogicNets [29]	72.0	37931	810	0	0	427	13	$4.93 \times 10^5$
JSC OpenML	<b>KAN-Quantized-Pruned</b>	74.8	<b>1235</b>	623	0	0	315	9.5	$1.17 \times 10^4$
	NeuraLUT-Assemble [4]	76.0	1780	540	0	0	<b>941</b>	<b>2.1</b>	<b><math>3.92 \times 10^3</math></b>
	TreeLUT [16]	75.6	2234	<b>347</b>	0	0	735	2.7	$6.03 \times 10^3$
	DWN [5]	<b>76.3</b>	6302	4128	0	0	695	14.4	$9.07 \times 10^4$
	hls4ml (Fahim et al.) [9]	76.2	63251	4394	38	0	200	45	$2.85 \times 10^6$

Table adapted directly from [NeuraLUT-Assemble](#)

# Summary & Outlook

Credit: Ziming Liu

- KAN is a promising alternative to MLP.
- LUT-based KAN can be much more efficient than previously thought.
- B-splines learning provide a natural and scalable way to directly learn look-up tables
- We hope that this work benefit both the KAN and LUT-based NNs community.

