

# Adaptable Butterfly Accelerator for Attention-based NNs via Hardware and Algorithm Co-design

Proceedings of MICRO'22

Fan, Hongxiang, et al.

Deep Learning Compiler Study

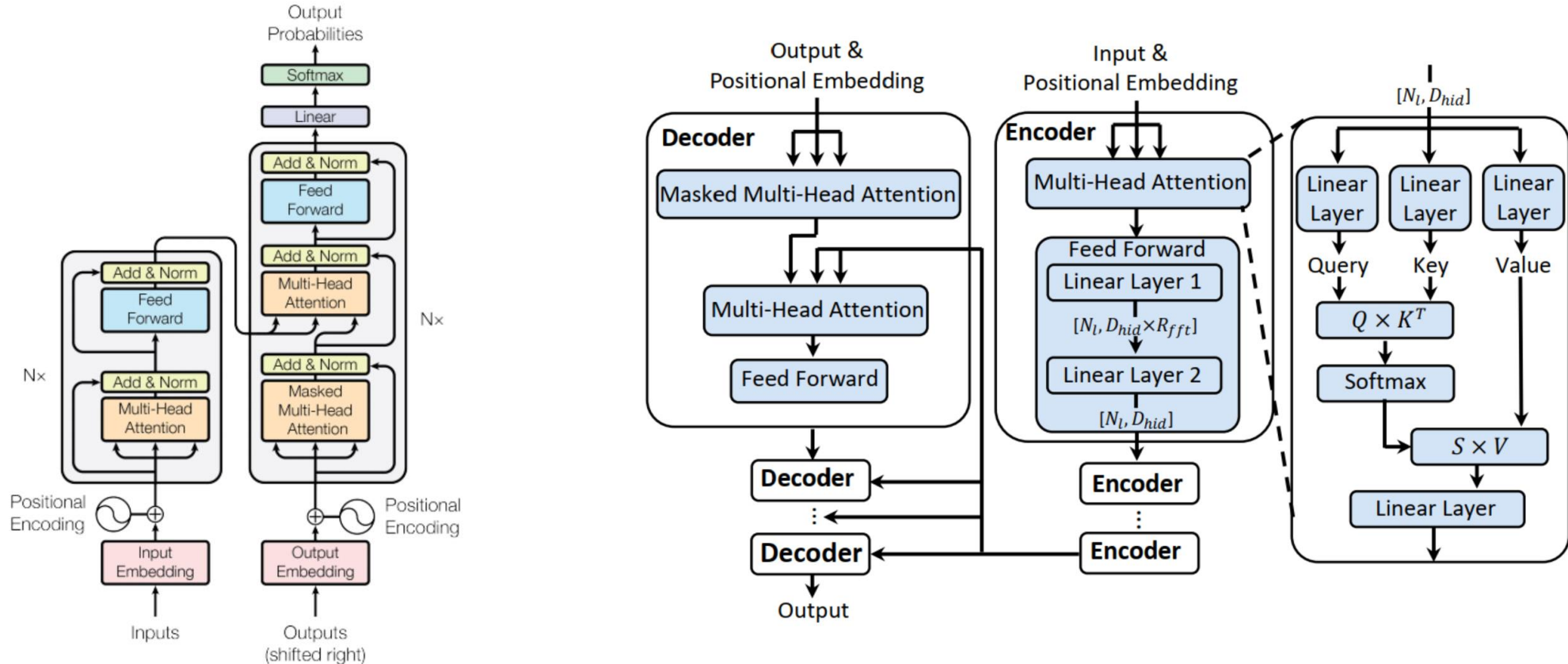
Nov 11, 2022

Presenter: Hyunwoo Cho  
(hyunwoocho@sogang.ac.kr)

# Introduction

➤ Recently, attention-based neural networks (NNs) have shown great success in many AI tasks.

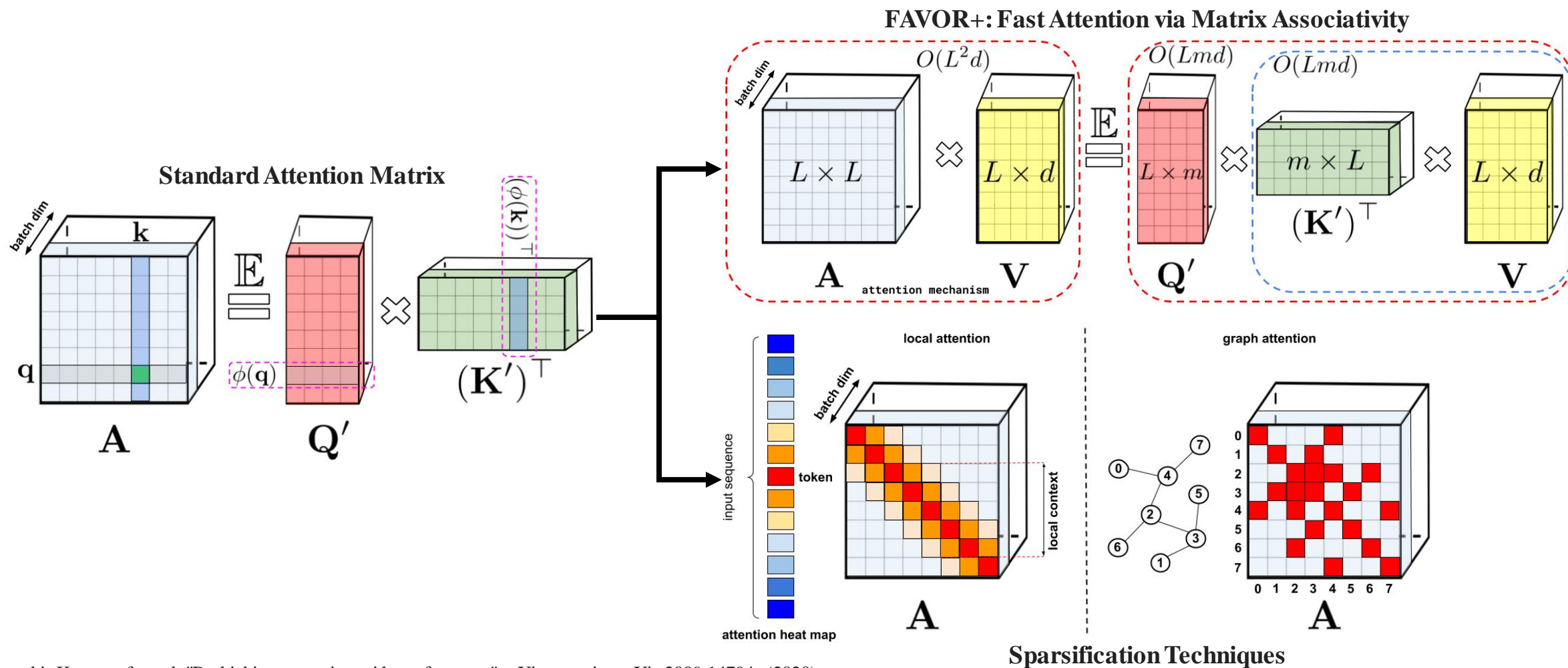
- 1) Encoder-decoder Structured: *Transformer*
- 2) Encoder-only Structured: *BERT, XLM, ViT, etc.*
- 3) Decoder-only Structured: *GPT, Autoregressive Models, etc.*



# Introduction

➤ Despite the successive performance, attention-based models are computationally expensive.

- E.g., *Transformer*: quadratically scales in both computation and memory (as a function of input sequence length).

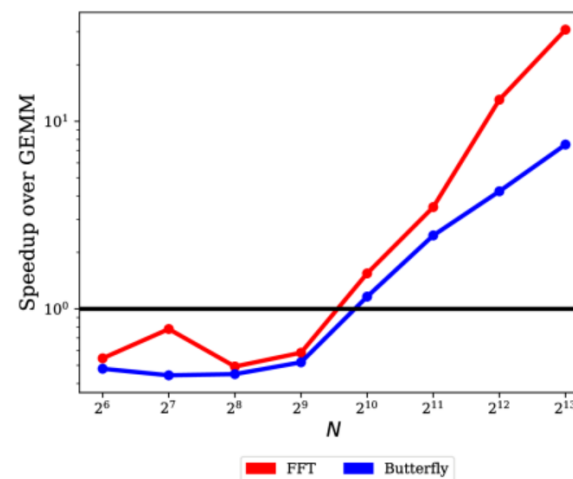
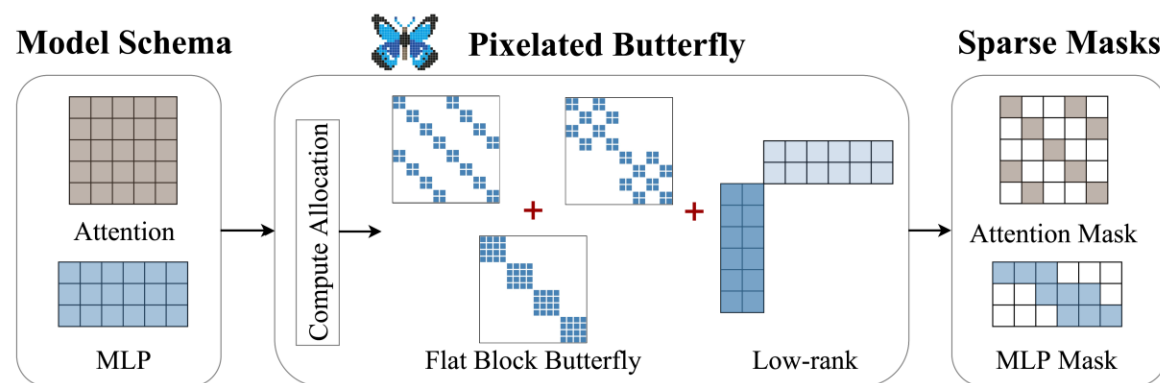


# Introduction

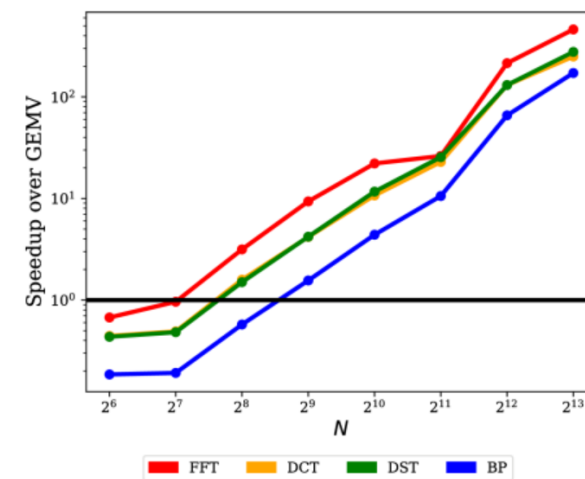
➤ Choosing an appropriate structure for each linear mapping, however, is **application-dependent**, often requiring domain expertise and entailing an arduous process of hand-picking solutions as different structures have different trade-offs in accuracy and speed.

- Butterfly matrices → universal representations of structured matrices that have a simple recursive structure.
- Successfully compressed attention and weight matrices, improving the accuracy and efficiency of attention-based NNs.

$$\mathbf{W}_{\text{Bfly}} = \left( \mathbf{W}'_N \begin{bmatrix} \mathbf{W}'_{N/2} & 0 \\ 0 & \mathbf{W}'_{N/2} \end{bmatrix} \cdots \begin{bmatrix} \mathbf{W}'_2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{W}'_2 \end{bmatrix} \right)$$



(a) Training

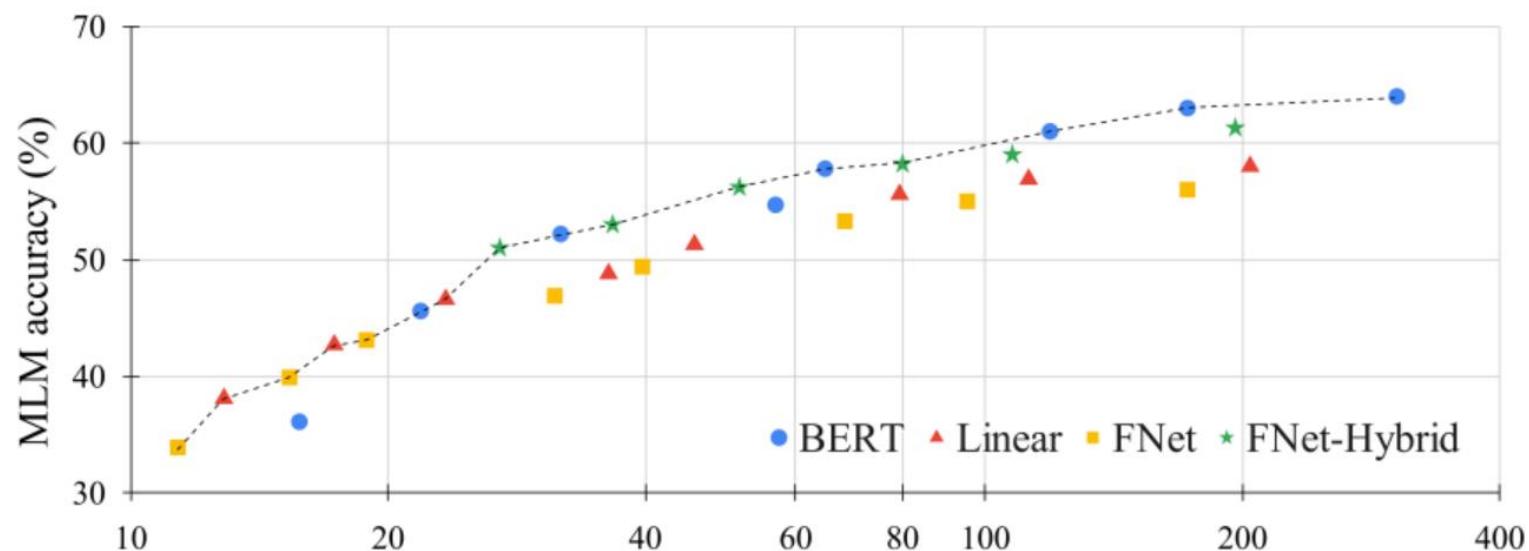
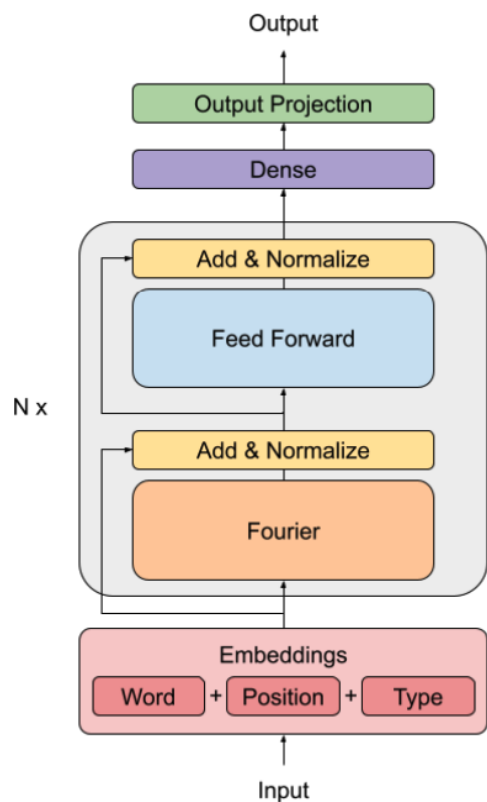


(b) Inference

# Introduction

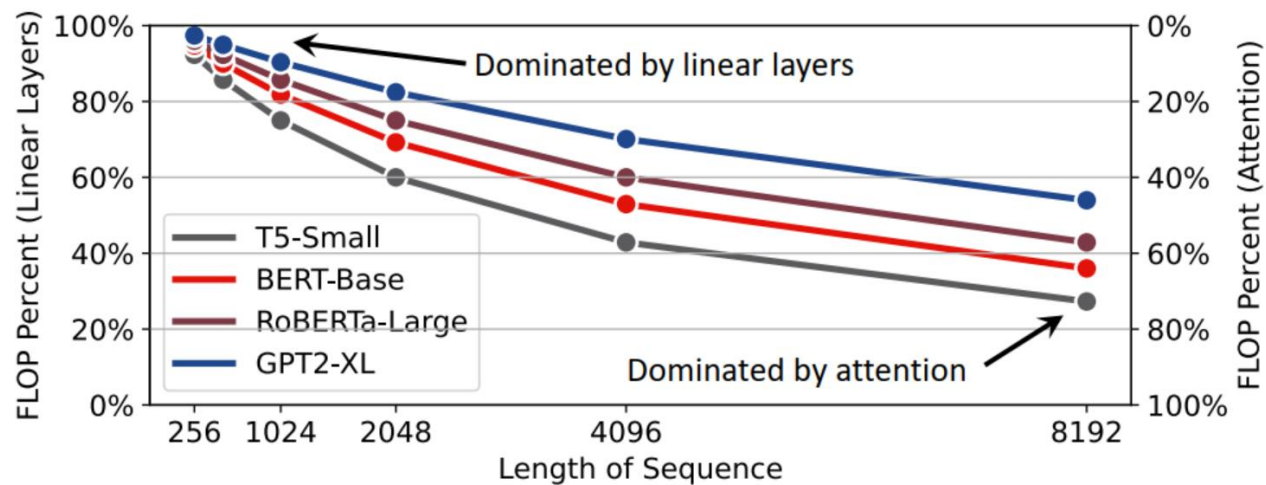
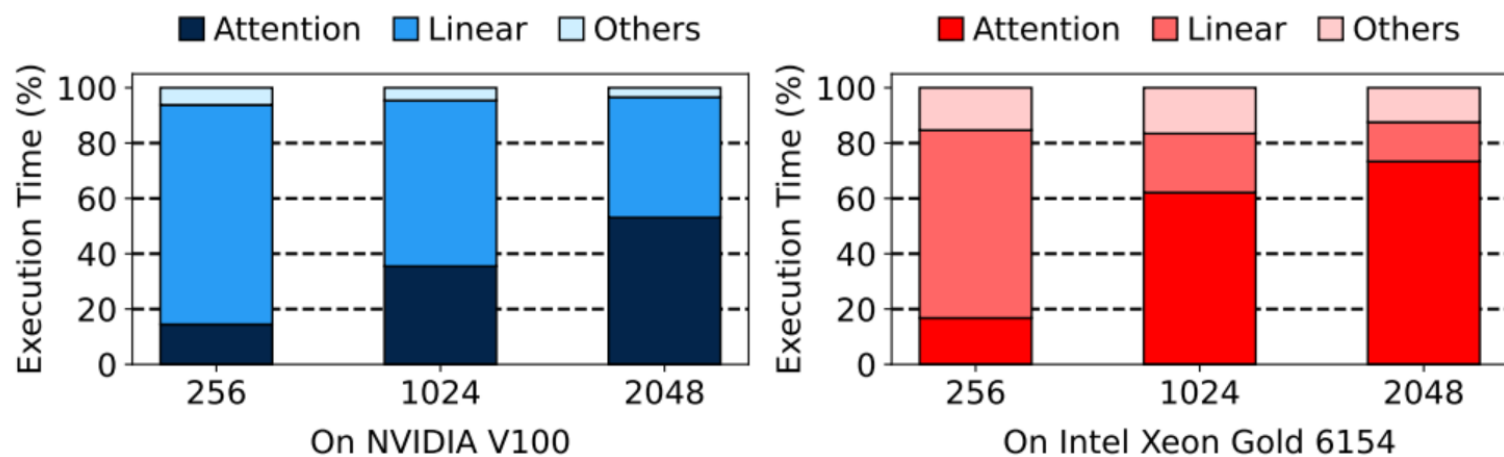
➤ Besides attention and weight matrices, some designs have explored replacing the entire attention mechanism with more efficient counterparts.

- E.g., FNet → self-attention modules are replaced by 2D DFT (FFT) operations.



# Introduction

- To achieve high **hardware** performance across different input lengths, optimizing both **attention** and **linear** layers is necessary.



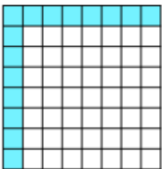
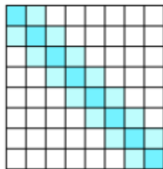
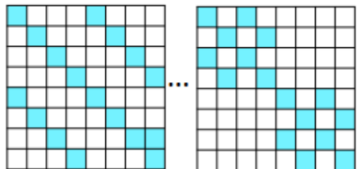
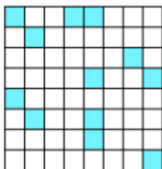
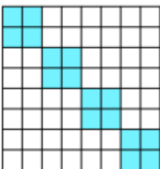
# Introduction

## ➤ Contributions of this study

- 1) A hardware-friendly attention-based model, FABNet is proposed.
  - → Adopts butterfly sparsity pattern on both attention and linear layers: end-to-end acceleration.
- 2) A novel adaptable butterfly accelerator configurable at runtime is proposed.
  - → Accelerate different layers using a single unified computing engine: run-time configurable.
- 3) Hardware optimizations to jointly optimize both algorithmic and hardware parameters.

# Methods

## ➤ Algorithm Optimization: FABNet

Sparsity	Low Rank	Sliding Window	Butterfly	Random	Block-wise
Patten					
Data Access	Sequential row & column read	Regular stride read	Regular stride read	Random read	Regular stride read
HW Eff.	No	Yes	Yes	No	Yes
Info.	Global	Local	Global & Local	Global&Local	Local

### → Objective

1. Structured data access pattern to memory design simplification
2. Captures both global and local information with unified sparsity
3. Applicable to both attention and FFNs: retain performance on varying sequence

### → Drawbacks of Butterfly sparsity

1. Complicated controller (hardware resource consumption)\
2. Not optimized on various sparsity (i.e., different datasets or tasks)

Model	Sparsity pattern	Att.	FFN	Unified Sparsity	Co-Design
Performer [7] Linformer [8]	Low-Rank (Extra kernels)	✓	✗	✗	✗
Reformer [9]	Block-wise (Extra kernels)	✓	✗	✗	✗
Sparse Sinkhorn [10]	Block-wise + Random	✓	✗	✗	✗
Longformer [11]	Sliding-Window + Low-Rank	✓	✗	✗	✗
BigBird [12]	Random + Sliding-Window + Low-Rank	✓	✗	✗	✗
FNet [34]	Butterfly	✓	✗	✗	✗
Kaleidoscope [32]	Butterfly	✗	✓	✓	✗
Sparse Trans. [13]	Low-Rank + Butterfly + Sliding-Window	✓	✗	✗	✗
Pixelfly [31] Monarch [37]	Butterfly + Block-Wise + Low-Rank	✓	✓	✗	✗
<b>Our work</b>	Butterfly	✓	✓	✓	✓

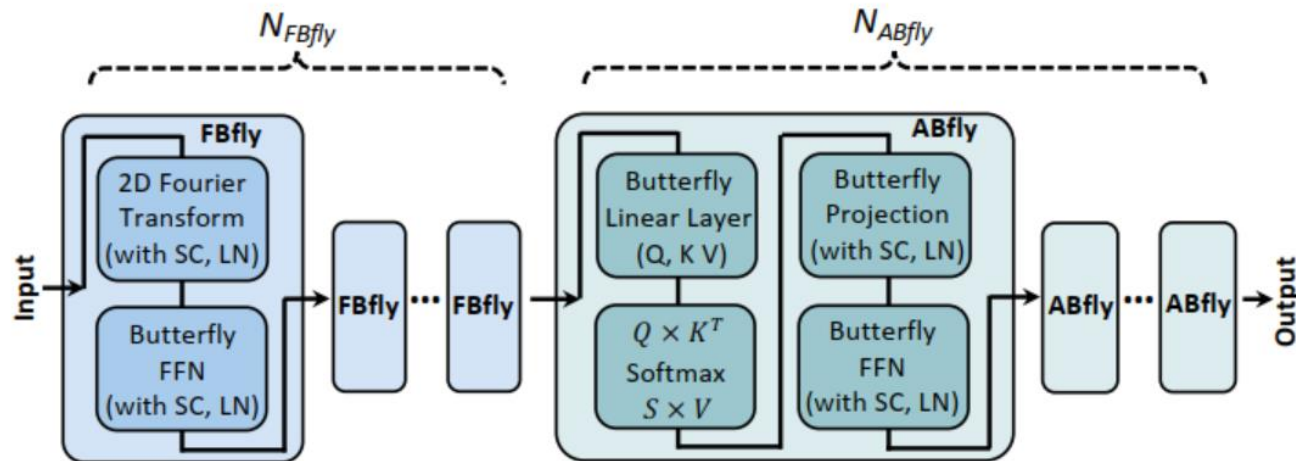


# Methods

## ➤ Algorithm Optimization: FABNet

### → Unified Butterfly Pattern for Attention and Linear Layers

- Attention Butterfly (ABfly)
    - 1) Starts with three butterfly linear layer generating Q, K, and V matrices: **Butterfly Linear layer**
    - 2) Q, K, V fed into multi-head attention and butterfly linear layer: **Butterfly projection layer**
    - 3) Two butterfly linear layers for additional processing: **Butterfly FFN**
  - Fourier Butterfly (FBfly)
    - 1) Replace the attention module with a 2D Fourier transform
    - 2) Used the aforementioned butterfly FFN layer
- 1) FABNet → **hybrid of ABfly and FBfly**: Numbers are optimized in the co-design process.

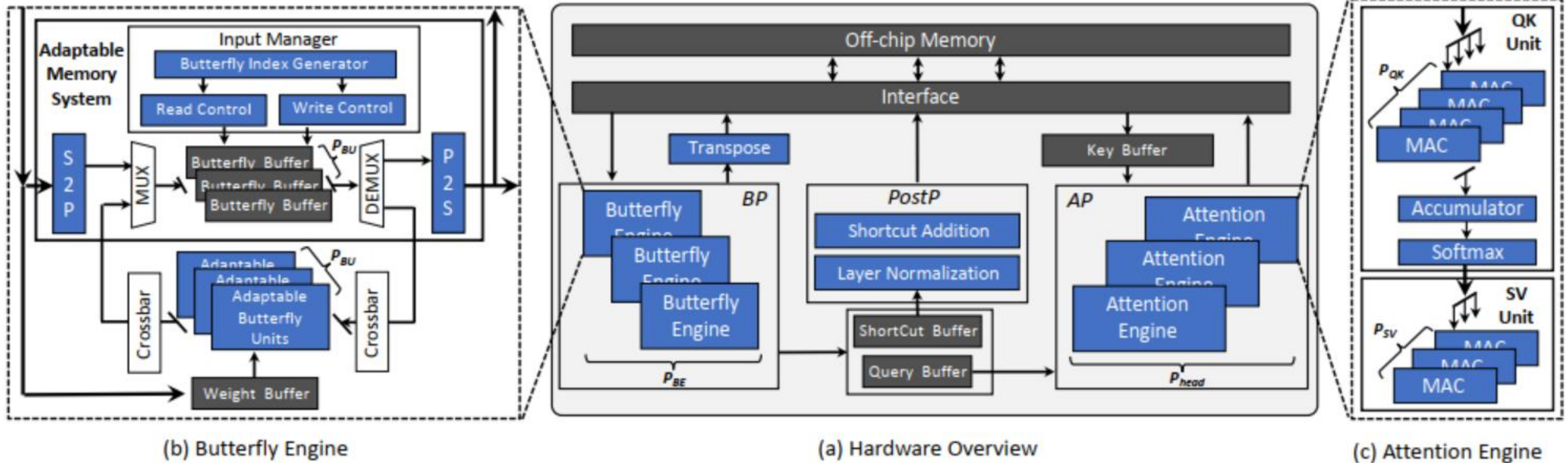


# Methods

## ➤ Hardware Accelerator

### → Overall Architecture

- Butterfly Processor (BP): consists of Butterfly Engines (BE) → to accelerate butterfly pattern operations
- Attention Processor (AP): consists of Attention Engines (AE) → QK units and SV units
- Post-processing Processor (PostP) → layer normalization, shortcut addition, miscellaneous.

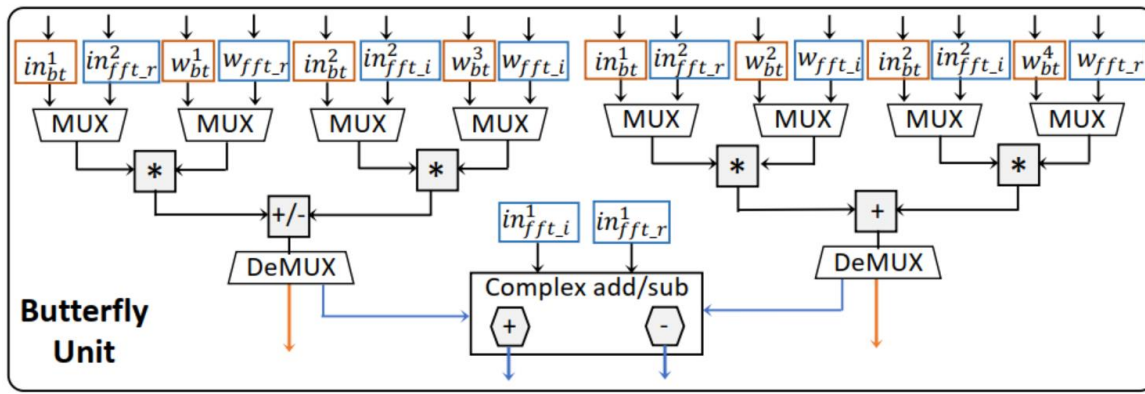


# Methods

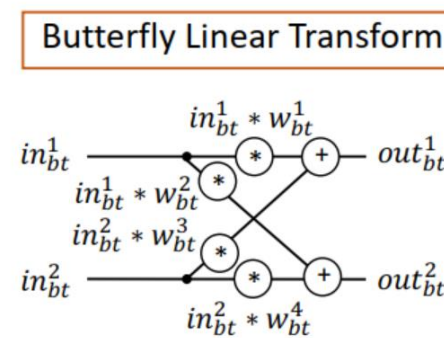
## ➤ Hardware Accelerator

### ➔ Adaptable Butterfly Units

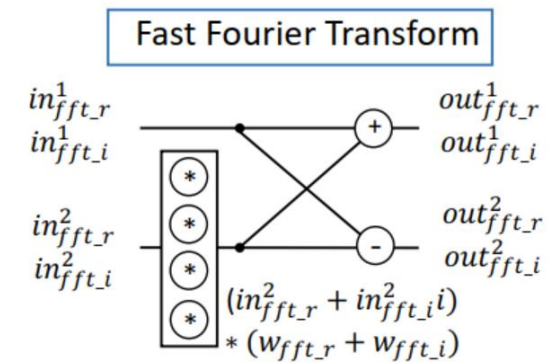
- Each BE is mainly composed of a butterfly memory system and adaptable Butterfly Units (BUs).
- **Adaptability** → programmable multiplexers and de-multiplexers are utilized
  - Configuration to either execute an FFT or butterfly linear transformation on runtime
- Each BU consists of four real-number multipliers, two real-number adders, and two complex-number adders.
- Butterfly linear transform →  $out_{bt}^1 = in_{bt}^1 \cdot w_{bt}^1 + in_{bt}^2 \cdot w_{bt}^3$ ,  $out_{bt}^2 = in_{bt}^1 \cdot w_{bt}^2 + in_{bt}^2 \cdot w_{bt}^4$
- FFT → Gisselquist, Dan. "[An Open Source Pipelined FFT Generator.](#)"



(a) Hardware architecture of adaptable butterfly unit



(b) Dataflow of both Butterfly linear transform



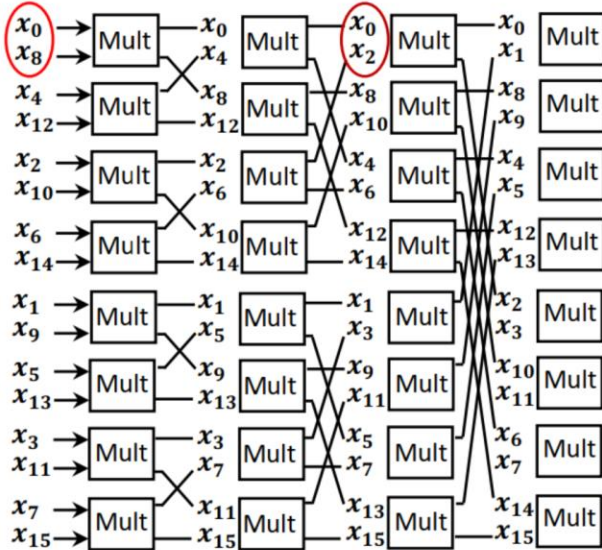
(c) Dataflow of both FFT

# Methods

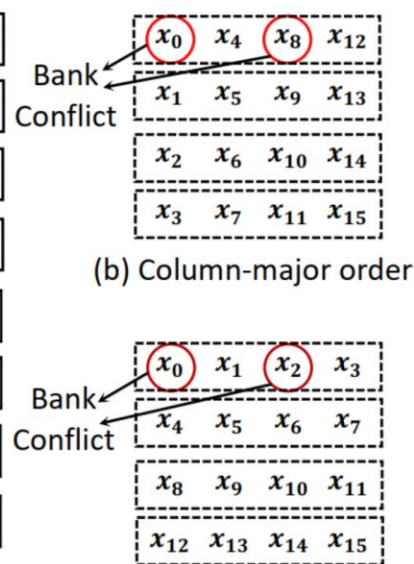
## ➤ Hardware Accelerator

### ➔ Butterfly Memory System

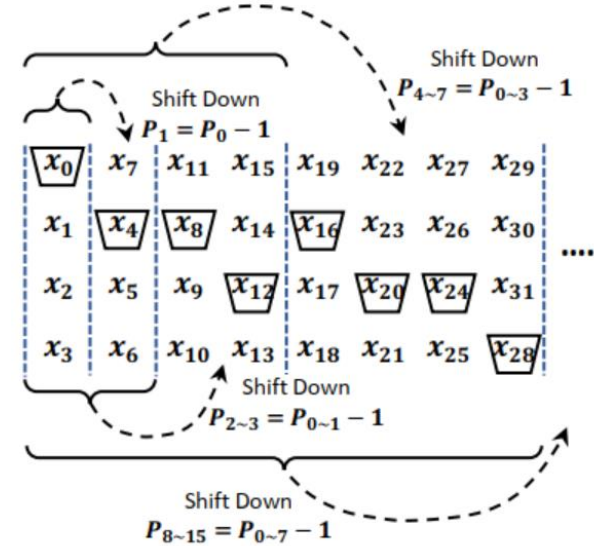
- Input managers: to avoid bank conflict (different access pattern at each stage)
  - serial-to-parallel (S2P), parallel-to-serial (P2S), butterfly buffers
- Permute each column  $i$  using the starting position  $P_i$
- Recovered in the P2S module when data is written back.



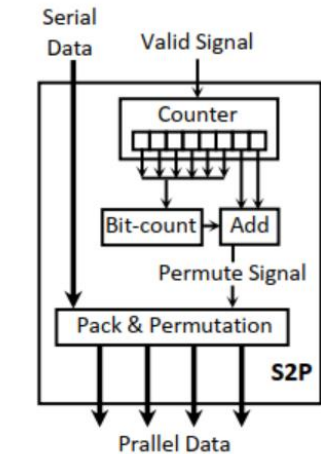
(a) Data access in each stage of Butterfly



(c) Row-major order



(a) Data storage of input data in Butterfly Buffers



(b) Hardware design of S2P module

$$P_0 = 0, \quad P_{2^{n-1} \sim (2^n - 1)} = P_{0 \sim (2^{n-1} - 1)} - 1 \text{ for } 1 \leq n \leq N$$



# Methods

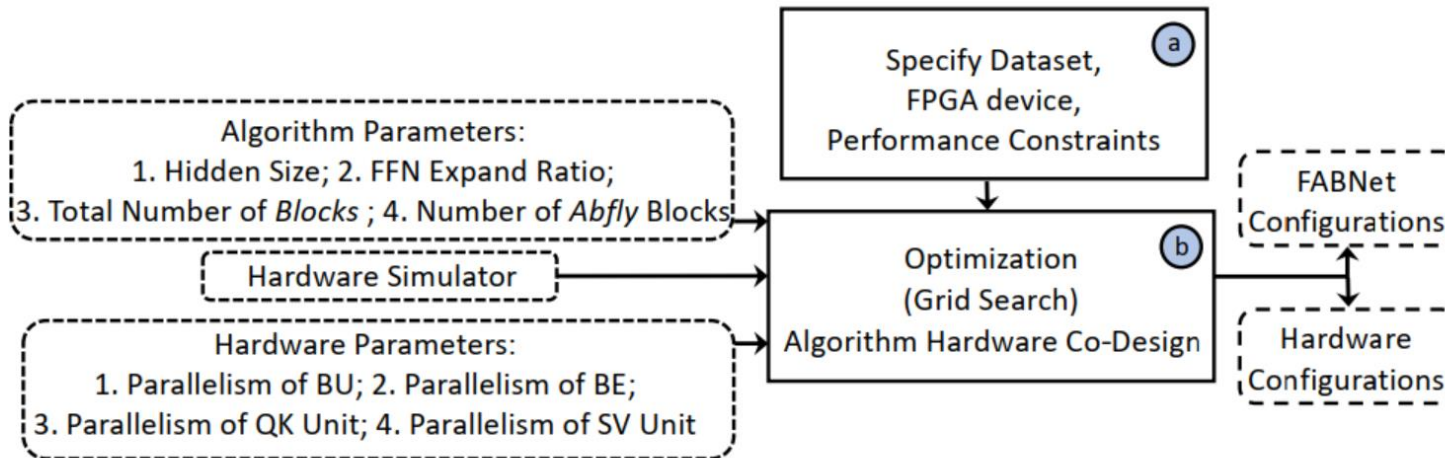
## ➤ Algorithm and Hardware Co-Design

### → Design Parameters

- Algorithm parameters
  - hidden size  $D_{hid}$ , expand ratio of FFN  $R_{ffn}$ , total number of blocks  $N_{total}$ , number of ABfly blocks  $N_{ABfly}$
- Hardware parameters
  - parallelism of BU, BE, QK, SV:  $P_{bu}, P_{be}, P_{qk}, P_{sv}$

### → Searching Strategy

- Algorithmic performance: train and evaluation FABNet
- Hardware performance: custom simulator for **latency**, analytic model for resource **consumption**



$$DSP_{usage} = P_{be} \times P_{bu} \times 4 + P_{head} \times (P_{qk} + P_{sv})$$

$$BRAM_{usage} = (BRAM_{bfly} + BRAM_{weight}) \times P_{be} + BRAM_{key} + BRAM_{sc} + BRAM_{query}$$

# Results

## ➤ Algorithm Performance

- Evaluated with LRA-Text, LRA-Image dataset
- Compared with the vanilla Transformer, around **10 ~ 60×** reduction in FLOPs and **2 ~ 22×** reductions in model size.
- Furthermore, compared with FNet, reduces FLOPs by **2 ~ 10×** and model size by **2 ~ 32×**.

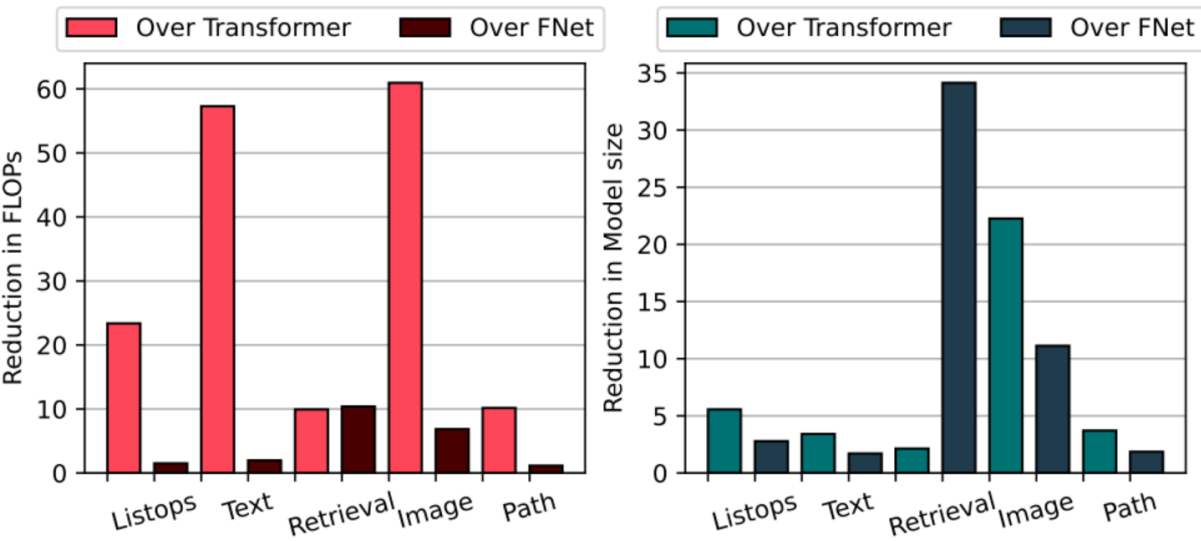


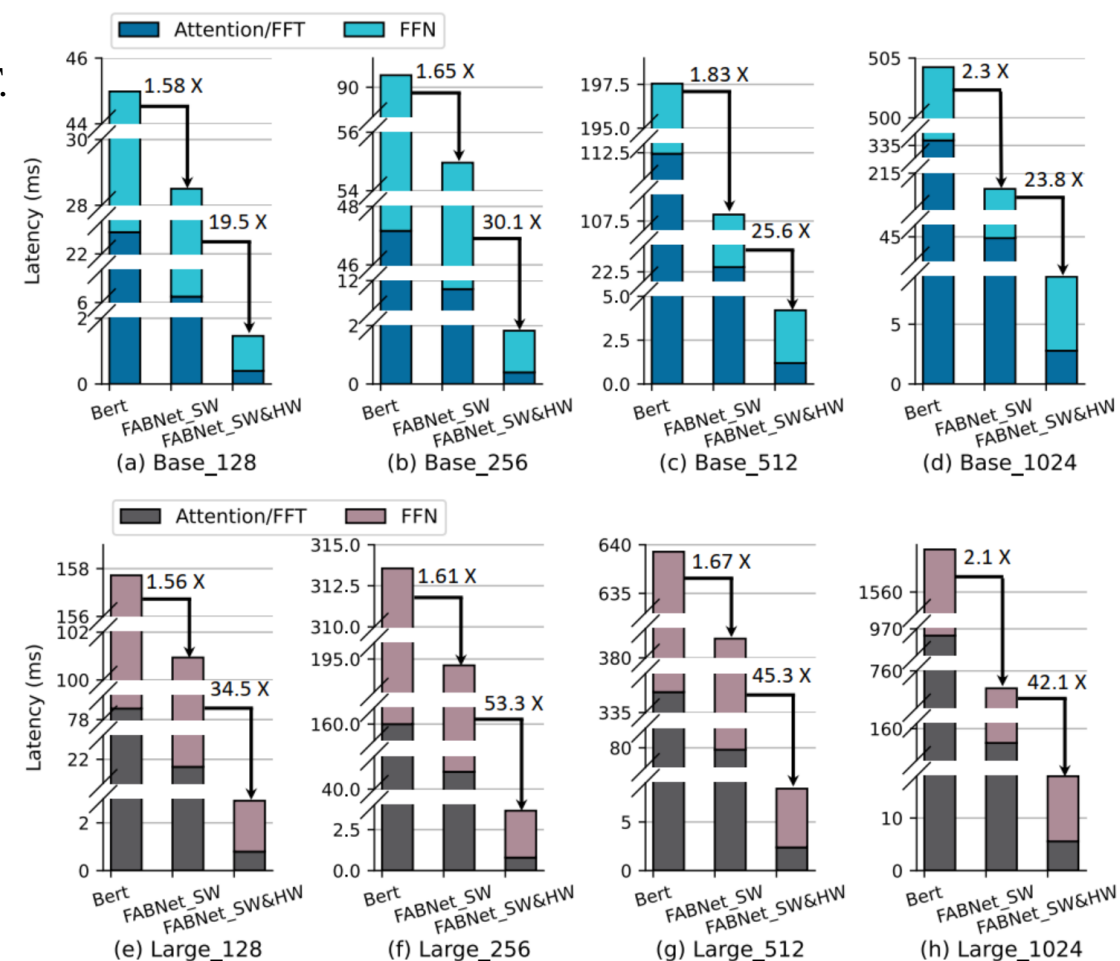
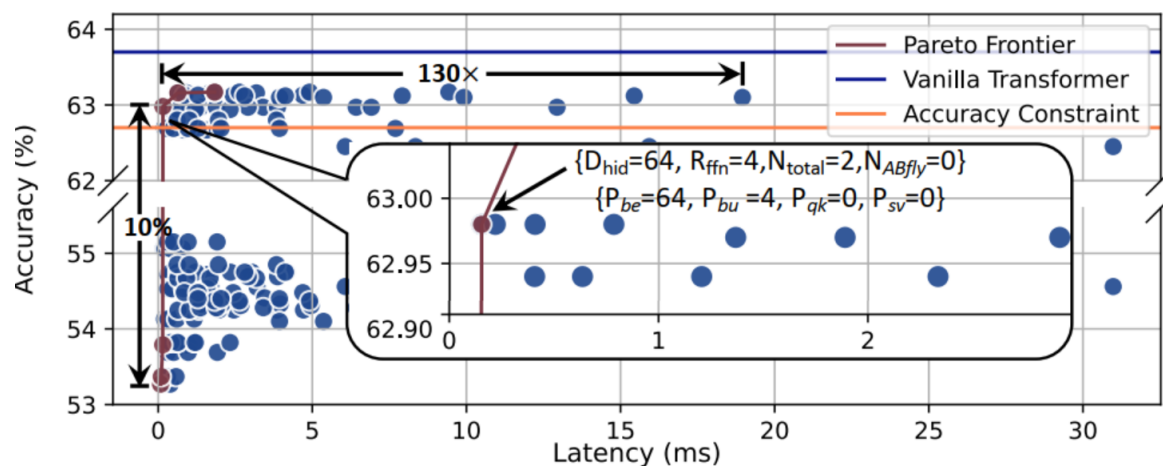
Table III  
ACCURACY OF DIFFERENT MODELS ON LRA.

	ListOps	Text	Retrieval	Image	Pathfinder	Avg.
Vanilla Transformer	0.373	<b>0.637</b>	0.783	0.379	<b>0.709</b>	<b>0.576</b>
Vanilla FNet	0.365	0.630	0.779	0.288	0.66	0.544
FABNet	<b>0.374</b>	0.626	<b>0.801</b>	<b>0.398</b>	0.679	<b>0.576</b>

# Results

## ➤ Co-design and Evaluation on Baseline Design

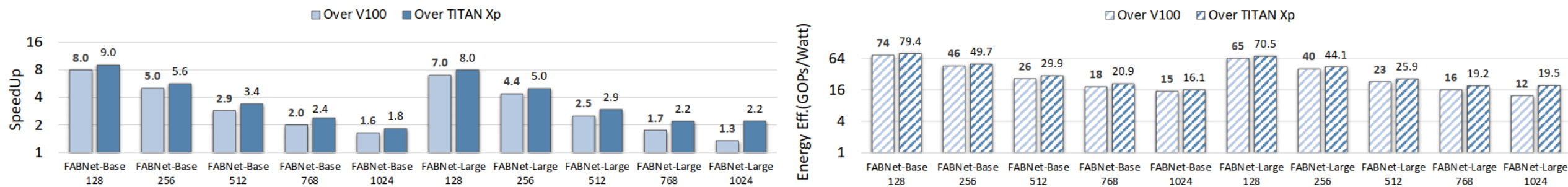
- Evaluated with MAC units to accelerate linear transform and QKV matrix multiplication.
- Evaluated on VCU128 FPGA with 2048 multipliers. (with HBM memory and 200MHz clock)
- FFT layers: multiplying linear layer with DFT matrices.
- Algorithm comparison: **1.6 ~ 2.3×** speedup compared to BERT.
- With butterfly accelerator: **19.5 ~ 53.3×** speedups.
- Optimizations: **30.8 ~ 87.3×** speedups.



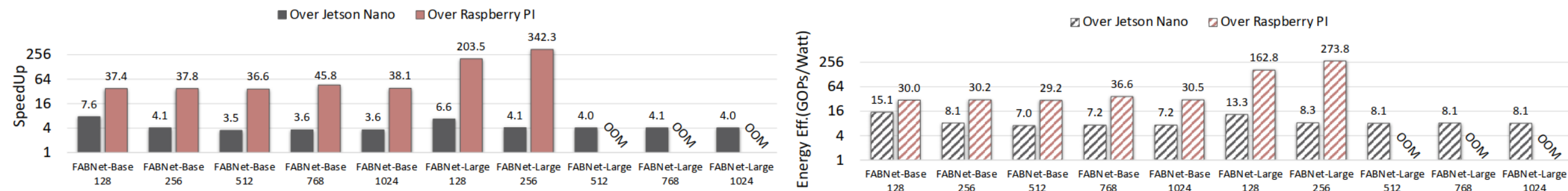
# Results

## ➤ Comparison with CPU and GPU

- Edge scenario: Xilinx Zynq 7045 FPGA, Nvidia Jetson Nano, Raspberry Pi4 (GPU, CPU respectively)
  - Proposed design on Zynq 7045 FPGA  $\rightarrow$  **3.5 ~ 8 $\times$**  speedup over GPU and **36.6 ~ 342.3 $\times$**  over CPU.
  - For energy efficiency, **7.0 ~ 15.1 $\times$**  and **162.8 ~ 273.8 $\times$**  higher than Jetson Nano and Raspberry Pi4, respectively
- Server scenario: Xilinx VCU128 FPGA, Nvidia V100, TITAN Xp.
  - 8.0 and 9.0 $\times$**  faster, **4.0 and 79.4 $\times$**  more energy-efficient.



(a) VCU128 FPGA compared with high-end GPUs in terms of speedup and energy efficiency

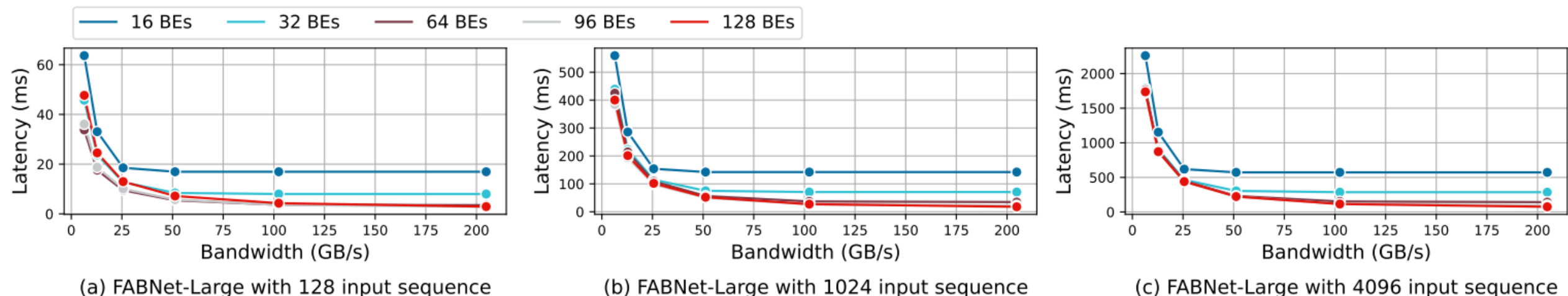


(b) Zynq 7045 FPGA compared with edge GPU and CPU in terms of speedup and energy efficiency



# Results

## ➤ Comparison with Other Accelerators and Off-chip Bandwidth Analysis



Accelerators	$A^3$ [14] (HPCA'20)	SpAtten [6] (HPCA'21)	Sanger [15] (MICRO'21)	Energion [16] (TCAD'21)	ELSA [17] (ISCA'21)	DOTA [18] (ASPLOS'22)	FTRANS [19] (ISLPED'20)	Our work
Technology	ASIC (40nm)	ASIC (40nm)	ASIC (55nm)	ASIC (45nm)	ASIC (40nm)	ASIC (22nm)	FPGA (16nm)	FPGA (16nm)
Frequency	1 GHz						170 MHz	200 MHz
# of Multipliers	128						6531	640
Latency (ms)	56.0	48.8	45.2	44.2	34.7	34.1	61.6	<b>2.4</b>
Throughput (Pred./s)	17.86	20.49	22.12	22.62	28.82	29.32	16.23	<b>416.66</b>
Power (W)	1.217	1.060	0.801	2.633	0.976	0.858	25.130	11.355
Energy Eff. (Pred./J)	14.67	19.33	27.62	8.59	29.52	34.18	0.65	<b>36.69</b>

Q & A