WashU

FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs

--A brief review

Howard Hua

Table of contents

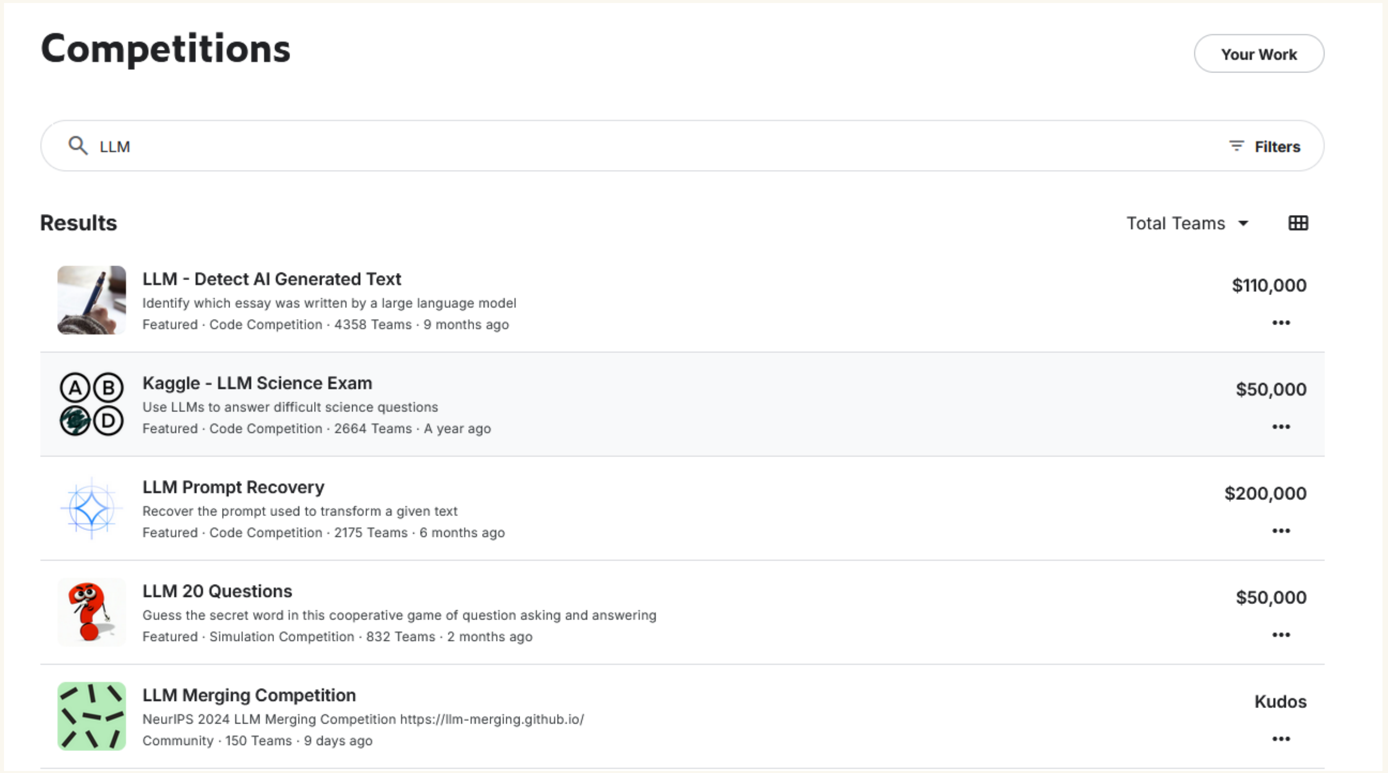
Table of contents

Background: LLM

OpenAI

GPT3 ~660TOPS

LLaMA by Meta


Competitions

Your Work

LLM

Filters

Results

Total Teams

LLM - Detect AI Generated Text
Identify which essay was written by a large language model
Featured · Code Competition · 4358 Teams · 9 months ago
$110,000

Kaggle - LLM Science Exam
Use LLMs to answer difficult science questions
Featured · Code Competition · 2664 Teams · A year ago
$50,000

LLM Prompt Recovery
Recover the prompt used to transform a given text
Featured · Code Competition · 2175 Teams · 6 months ago
$200,000

LLM 20 Questions
Guess the secret word in this cooperative game of question asking and answering
Featured · Simulation Competition · 832 Teams · 2 months ago
$50,000

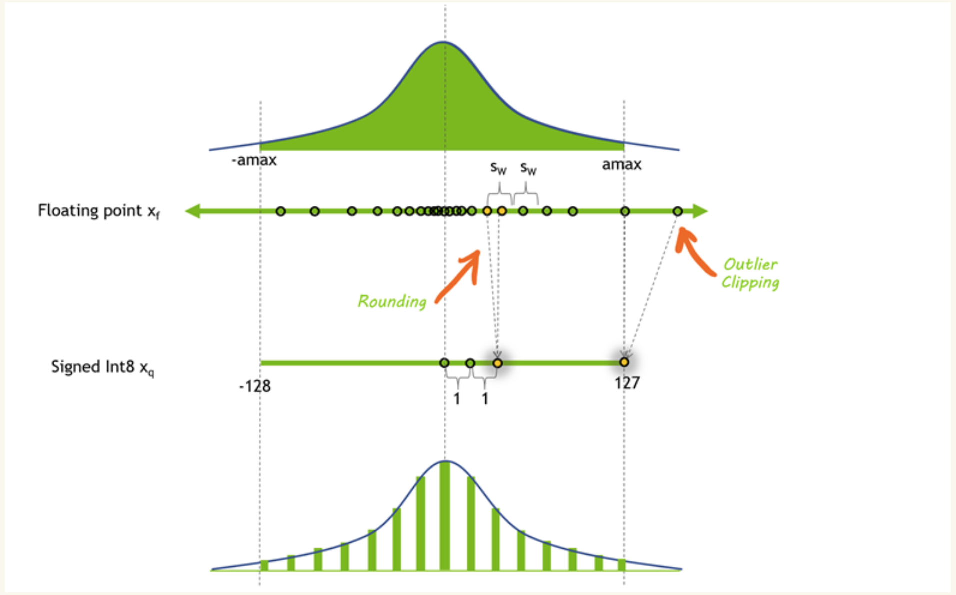
LLM Merging Competition
NeurIPS 2024 LLM Merging Competition https://llm-merging.github.io/
Community · 150 Teams · 9 days ago
Kudos

[1] OpenAI. (n.d.). *OpenAI*. Retrieved from https://openai.com/
[2] Kaggle. (n.d.). *Competitions*. Retrieved from https://www.kaggle.com/competitions
[3] Meta AI. (2024). *LLaMA 3: Advancements in Large Language Models*. Retrieved from https://ai.meta.com/blog/meta-llama-3-1/

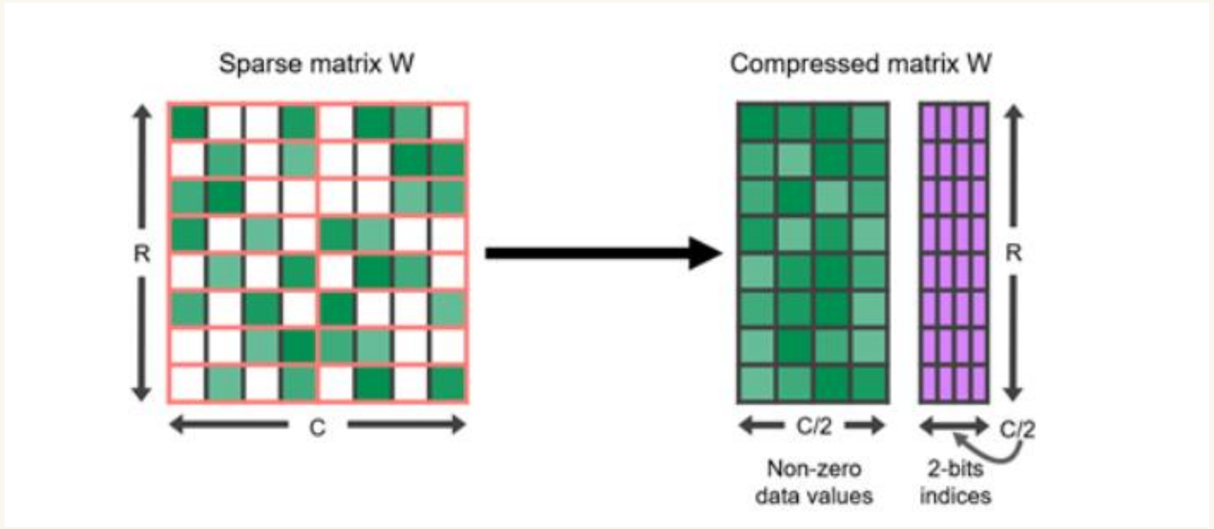
WashU

Background: principle of LLM

- Prefill phase or processing the input
- Decode phase or generating the output



The distribution of values before and after one possible method of quantization [4]



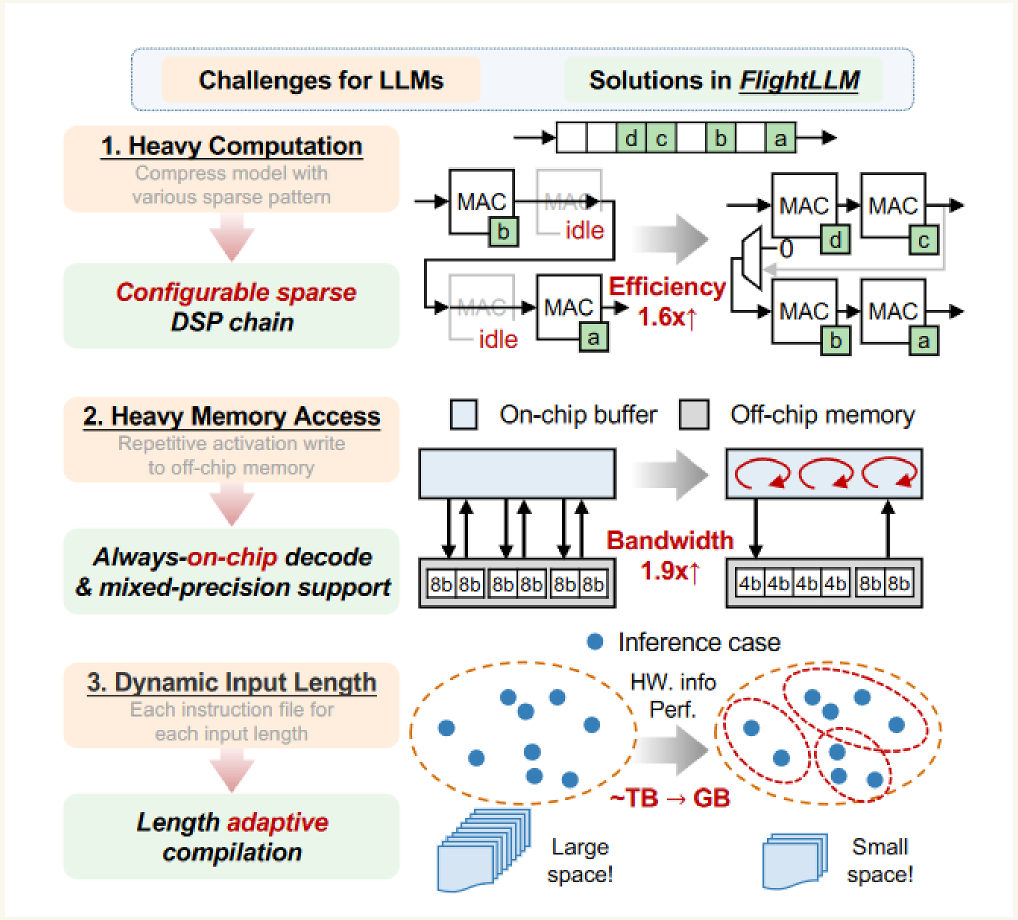
A sparse matrix represented in a compressed format [4]

Table of contents

Literature Review

In order to implement LLM on FPGAs...

1. Heavy computation
2. Heavy memory access
3. Dynamic input length

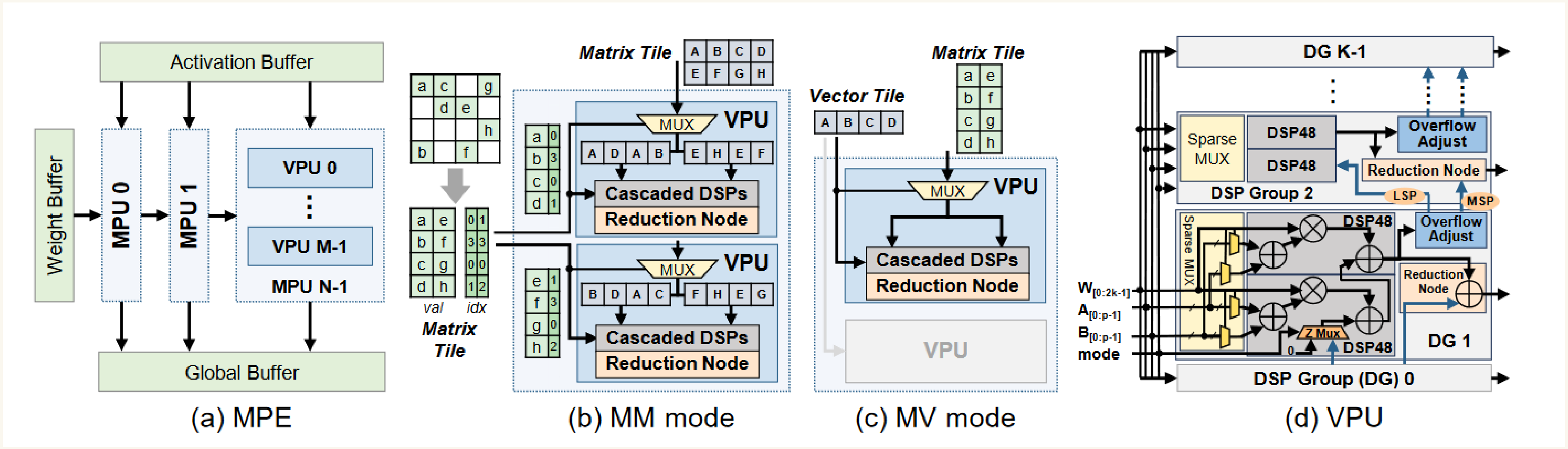

Challenges for LLMs
Solutions in FlightLLM
1. Heavy Computation
Compress model with various sparse pattern
Configurable sparse DSP chain
MAC
idle
Efficiency 1.6x↑
2. Heavy Memory Access
Repetitive activation write to off-chip memory
Always-on-chip decode & mixed-precision support
On-chip buffer
Off-chip memory
Bandwidth 1.9x↑
3. Dynamic Input Length
Each instruction file for each input length
Length adaptive compilation
Inference case
HW. info
Perf.
~TB → GB
Large space!
Small space!

Three challenges of LLM inference on FPGAs, and the corresponding solutions in FlightLLM [5]

Table of contents

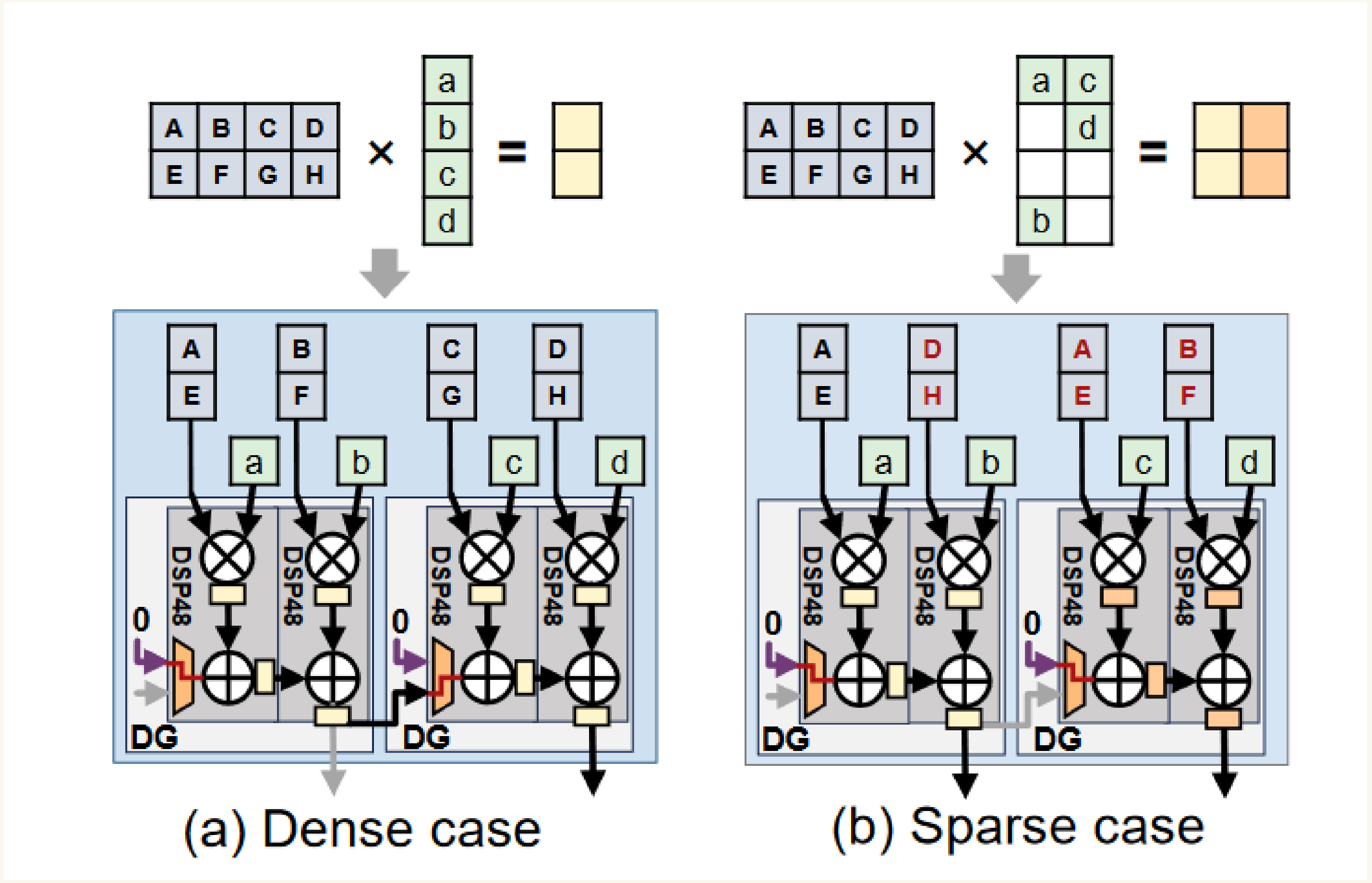
WashU

Computing Architecture: Matrix Processor



(a) The unified Matrix Processing Engine (MPE)
(b) matrix-matrix multiplication (MM) mode
(c) matrix-vector multiplication (MV) mode
(d) Vector Processing Unit (VPU) [5]

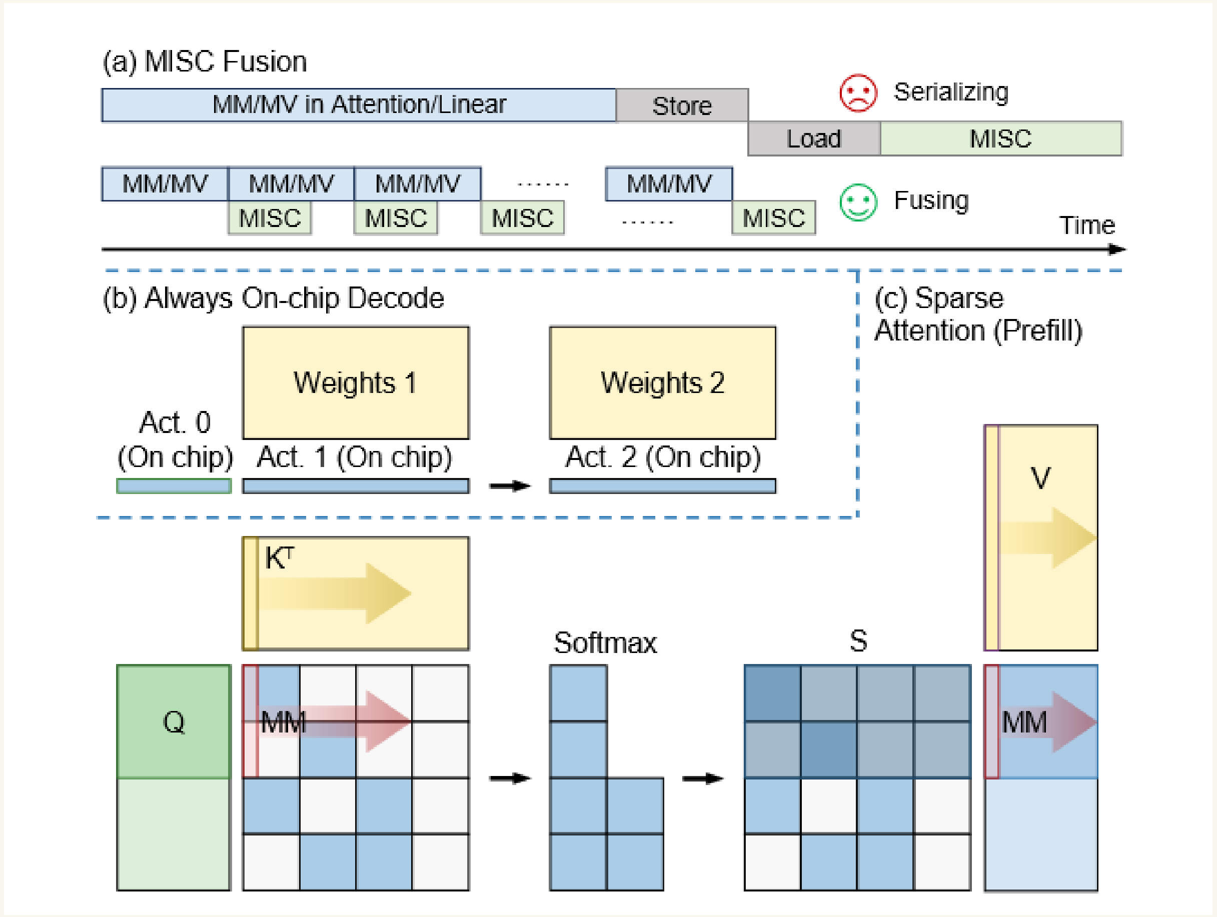
Computing Architecture: Matrix Processor



(a) Dense case
(b) Sparse case

By configuring the VPU, the MPE can support both (a) dense and (b) sparse cases [5]

Computing Architecture: Always on-chip decode

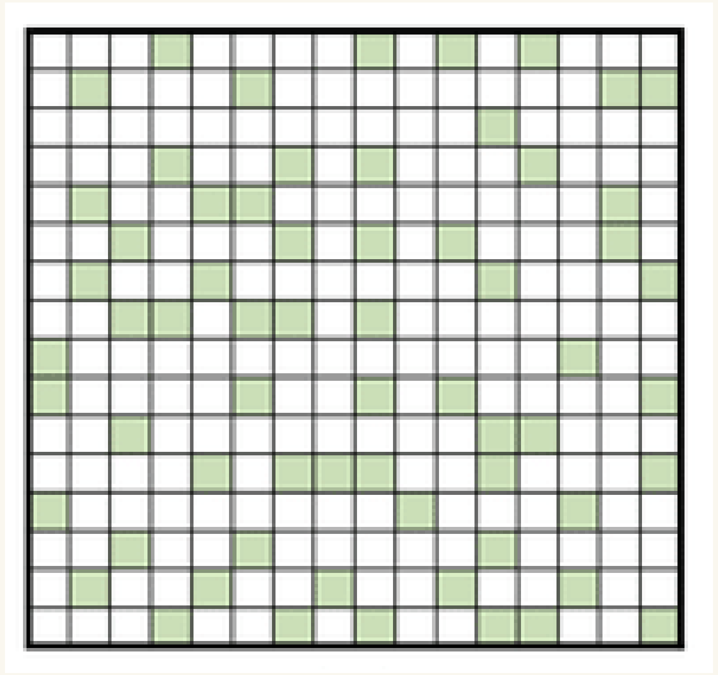


(a) miscellaneous (MISC) fusion with attention or linear operation. And an example of always on-chip decode approach in the (b) decode and (c) prefill stage [5]

Table of contents

Software Design: Reduce Instruction memory

- Instruction files: Designed to adapt tokens of different size
- Imagine this is a 64 by 64 sparse matrix:



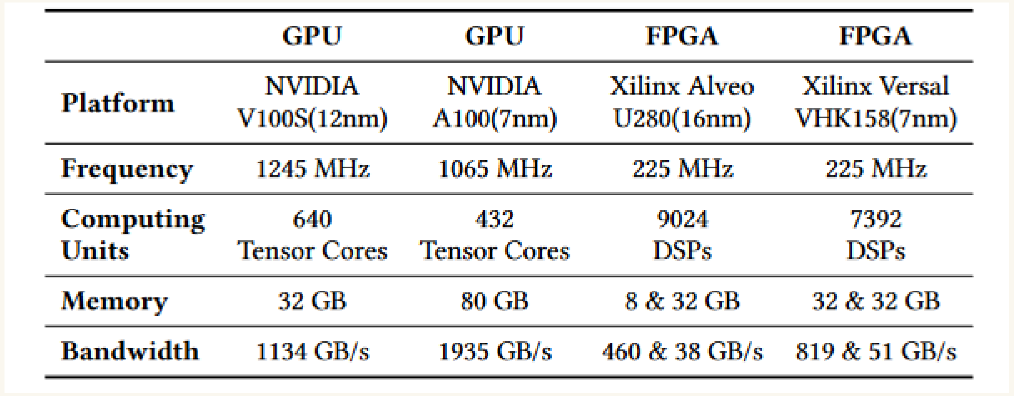
An illustrative demo of 64 by 64 matrix [4]

Table of contents

WashU

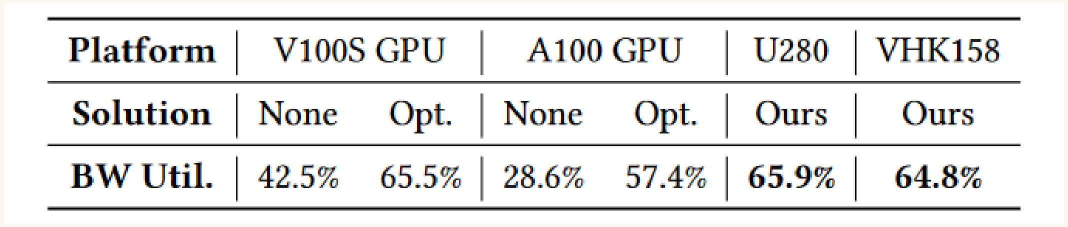
Evaluation: latency and throughput

Hardware used during evaluation of FlightLLM [5]



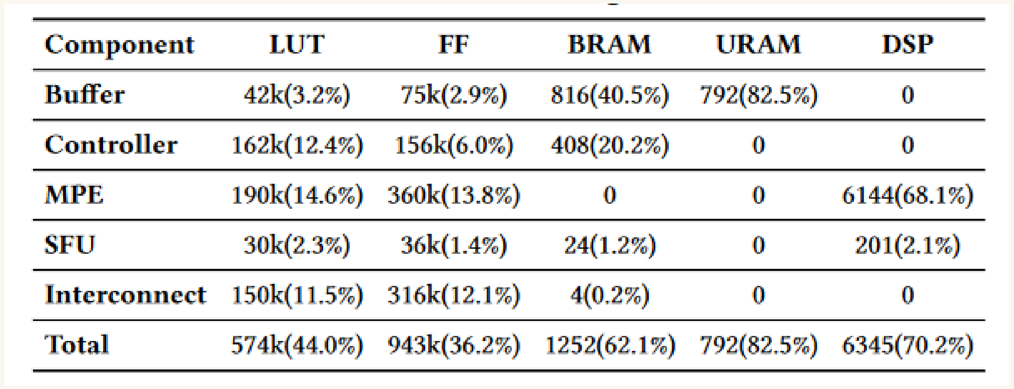
	GPU	GPU	FPGA	FPGA
Platform	NVIDIA V100S(12nm)	NVIDIA A100(7nm)	Xilinx Alveo U280(16nm)	Xilinx Versal VHK158(7nm)
Frequency	1245 MHz	1065 MHz	225 MHz	225 MHz
Computing Units	640 Tensor Cores	432 Tensor Cores	9024 DSPs	7392 DSPs
Memory	32 GB	80 GB	8 & 32 GB	32 & 32 GB
Bandwidth	1134 GB/s	1935 GB/s	460 & 38 GB/s	819 & 51 GB/s

Memory bandwidth utilization of FlightLLM across devices [5]



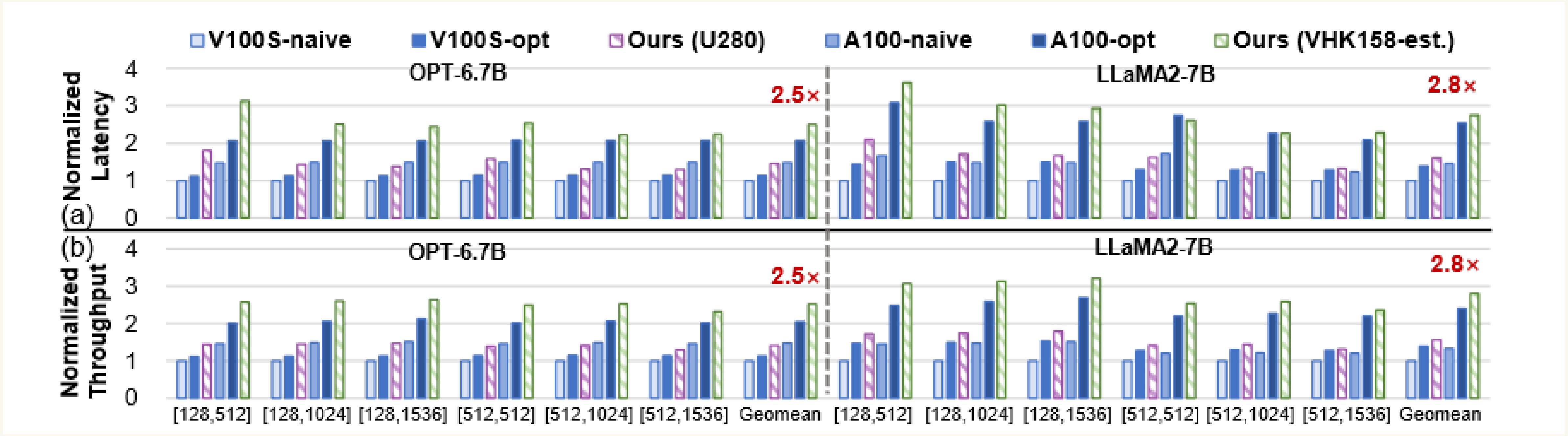
Platform	V100S GPU		A100 GPU		U280	VHK158
Solution	None	Opt.	None	Opt.	Ours	Ours
BW Util.	42.5%	65.5%	28.6%	57.4%	**65.9%**	**64.8%**

Hardware utilization of FlightLLM on Alveo U280 [5]

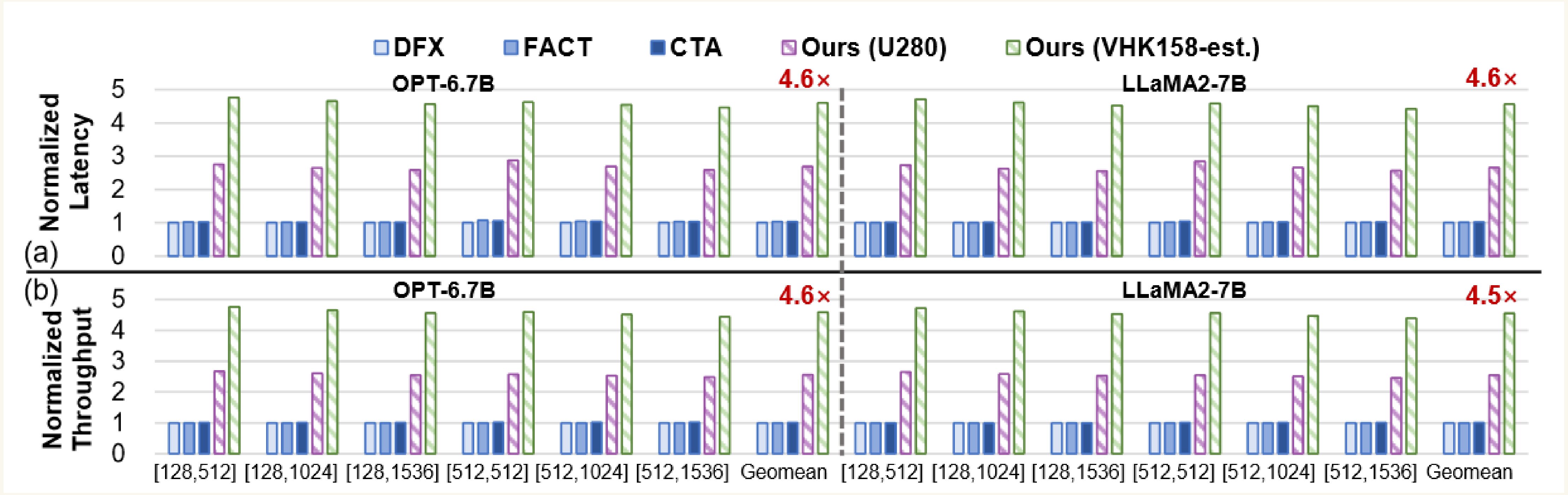


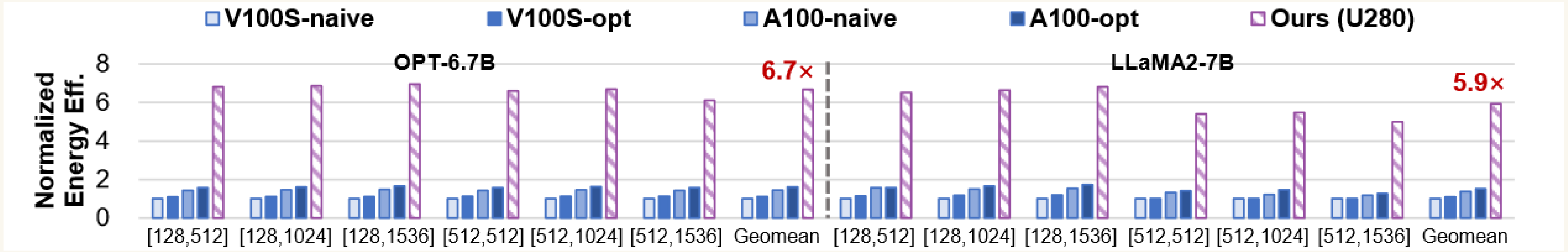
Component	LUT	FF	BRAM	URAM	DSP
Buffer	42k(3.2%)	75k(2.9%)	816(40.5%)	792(82.5%)	0
Controller	162k(12.4%)	156k(6.0%)	408(20.2%)	0	0
MPE	190k(14.6%)	360k(13.8%)	0	0	6144(68.1%)
SFU	30k(2.3%)	36k(1.4%)	24(1.2%)	0	201(2.1%)
Interconnect	150k(11.5%)	316k(12.1%)	4(0.2%)	0	0
Total	574k(44.0%)	943k(36.2%)	1252(62.1%)	792(82.5%)	6345(70.2%)

Evaluation: bandwidth and speed



Latency and throughput of FlightLLM and V100S/A100 GPU. The horizontal axis represents [prefill size, decode size] [5]





Top: Performance of FlightLLM, DFX, CTA, and FACT. The horizontal axis represents [prefill size, decode size]
Bottom: Energy efficiency of FlightLLM, NVIDIA V100S/A100 GPU. The horizontal axis represents [prefill size, decode size] [5]

WashU

Overall,

- This paper proposes FlightLLM, a novel implementation of recent LLM models on FPGAS that achieves:
    - 6.0× higher energy efficiency (against Nvidia V100)
    - 1.8× better cost efficiency (against Nvidia V100)
    - 1.2× higher throughput (against Nvidia A100)

Note: Xilinx Alveo U280 FPGA is used to obtain the first two datapoints, while the Xilinx Versal VHK158 FPGA is used for the final comparison against Nvidia A100.

WashU

Table of contents

Future Prospective

- Raw implementation not open source, but demo available on GitHub
- Emerging open source LLM models since 2023
- Further optimization on LLM inference

References

[1] OpenAI. (n.d.). *OpenAI*. Retrieved from https://openai.com/

[2] Kaggle. (n.d.). *Competitions*. Retrieved from https://www.kaggle.com/competitions

[3] Meta AI. (2024). *LLaMA 3: Advancements in Large Language Models*. Retrieved from https://ai.meta.com/blog/meta-llama-3-1/

[4] NVIDIA Technical Blog. (2024, January). Mastering LLM Techniques: Inference Optimization. Retrieved from https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

[5] Zeng, S., Liu, J., Dai, G., Yang, X., Fu, T., Wang, H., Ma, W., Sun, H., Li, S., Huang, Z., et al. (2024). FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 223–234).

Thank you for listening!

WashU

Q&A