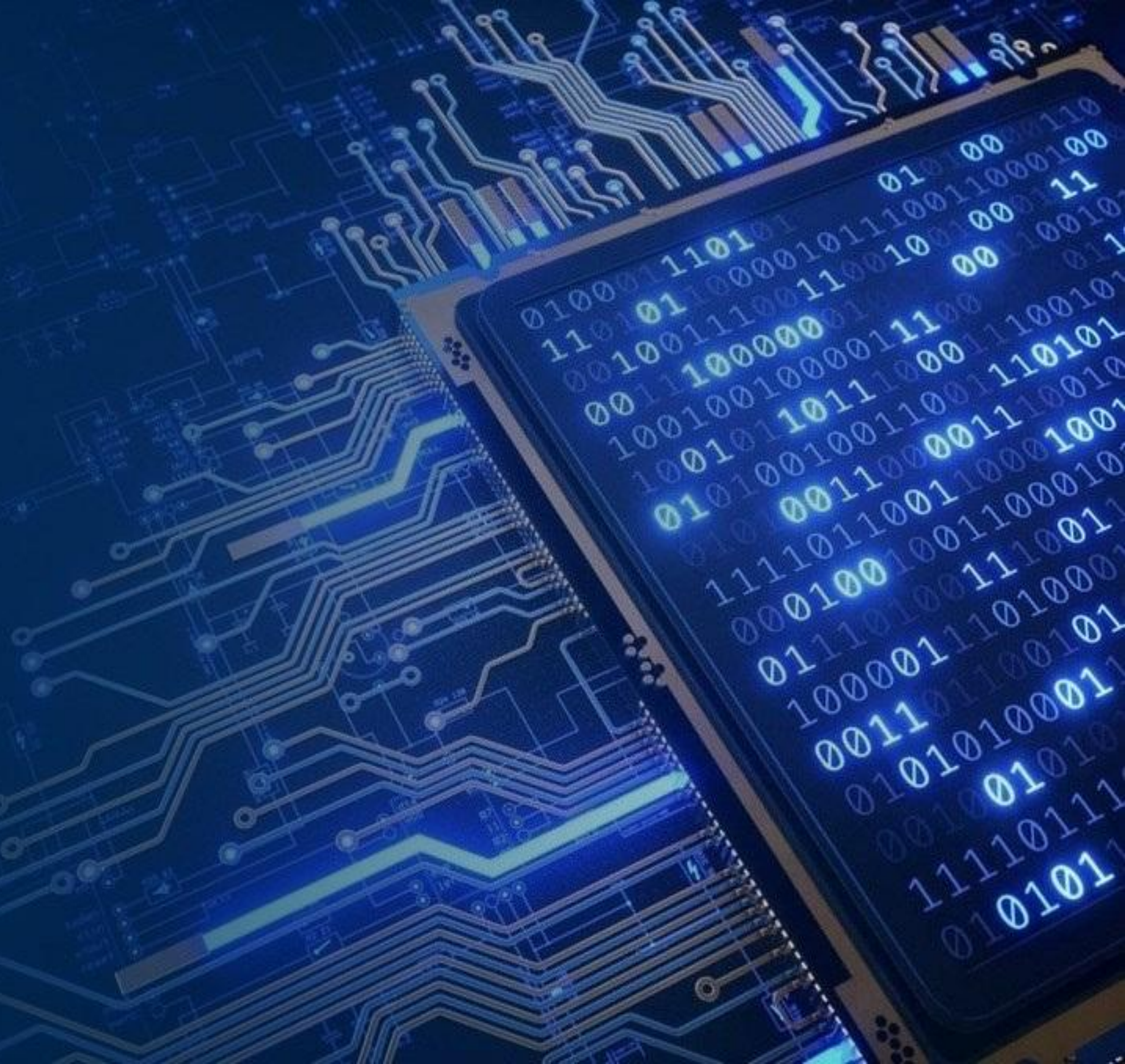
GHOST DAY

Applied Machine Learning Conference

Chasing Efficiency in the Era of Generative-AI and Large Language Models

Halima Bouzidi, PhD
Research Fellow in Trustworthy AI
The Queen’s University of Belfast, UK
h.bouzidi@qub.ac.uk

GHOST Day: AMLC 2024
05-06 April 2024



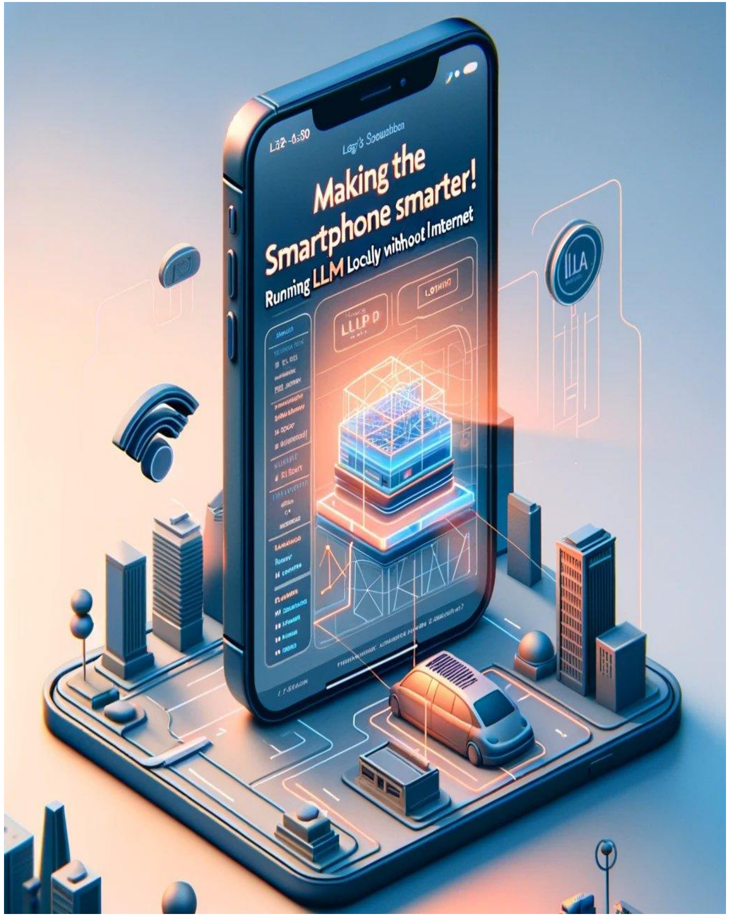
Agenda

- **Introduction: A Quest for Efficiency**
    - Understanding the LLM Landscape
    - LLM Inference Challenges

- **Software-Level Optimization**
    - Pay More Attention to Attention Layers!
    - To compress or not to compress?
    - Dynamic Model Scaling

- **Hardware-Level Optimization**
    - Parallel Computing
    - Domain-specific and IMC Accelerators

- **Conclusion**



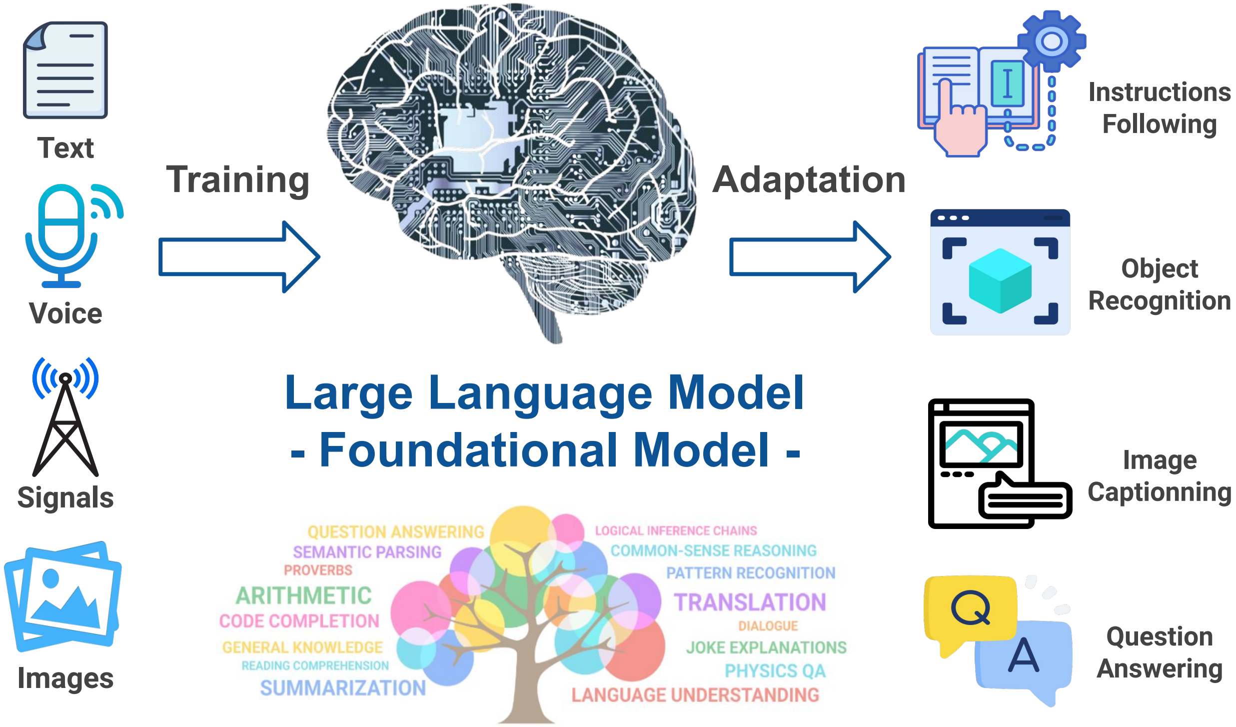
LLM is the New Fuel of Today's Digital Landscape

- The history of LLMs is built upon advancements in **NLP** and **ML**.
- Progress in **NN architectures**, availability of **datasets**, and **computational power** are the magic recipe of today's LLMs.

Effective LLMs need **more scaling** (data, computation, budget)

Larger == better performance

On-device LLM is the key to enable future generations of **smart IoT**



Encapsulating such interdisciplinary knowledge requires **Billions of Parameters**

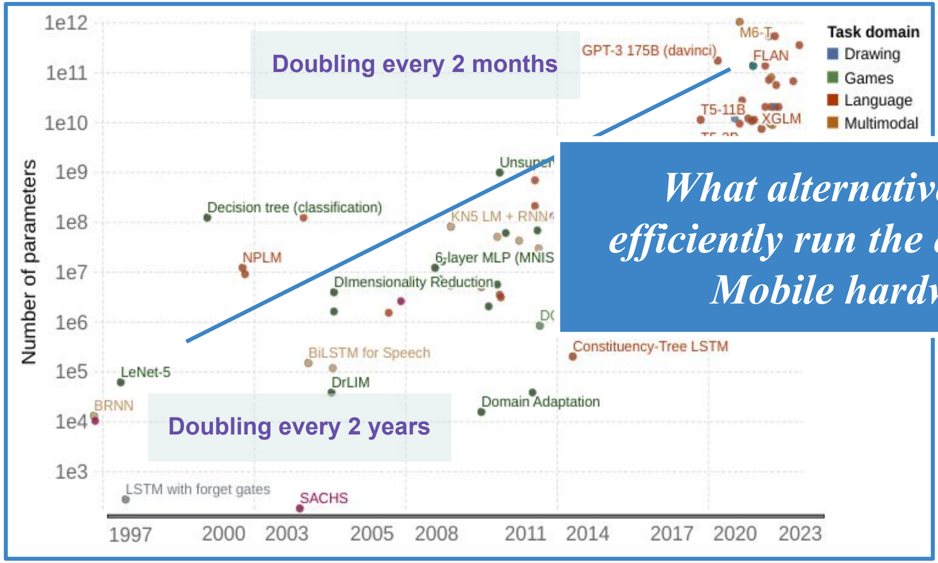
When GenAI outgrows Edge/Mobile Hardware Limits

"The exponential growth in LLM applications and complexity has outstripped the hardware scaling capabilities of Moore's law"

"Hardware Technology Scaling is all you Need"

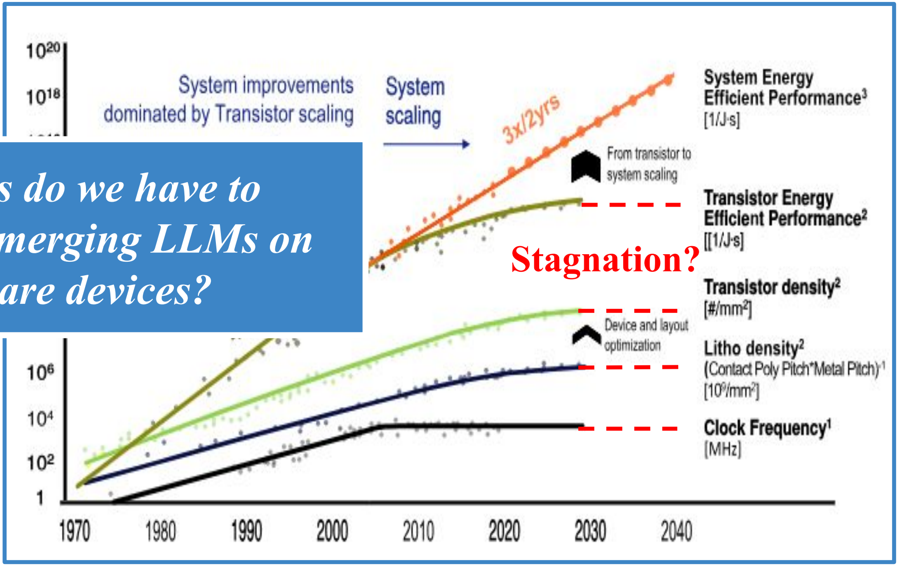
However

"Physical and Thermal Limits are your Constraints"



Data from OurWorldinData:
https://ourworldindata.org/grapher/artificial-intelligence-parameter-count

What alternatives do we have to efficiently run the emerging LLMs on Mobile hardware devices?



Data from AMSL's Investor Day Event:
https://leimao.github.io/downloads/blog/2023-04-10-Moore-Law/ASML-Investor-Day-2021.pdf

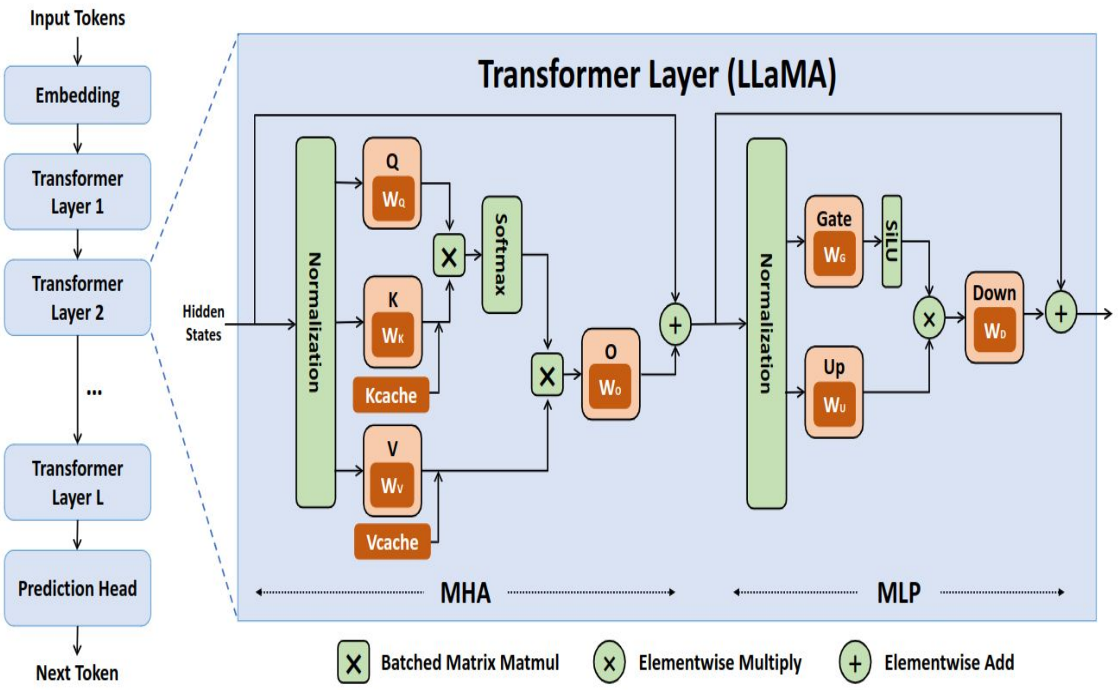
LLM Architecture and Computing Scaling

The compute has two separate components:

- **Linear terms:** Vector-matrix multiplication w/ the W matrix (fixed cost per time step)
- **Attention terms:** Matrix-vector multiplication w/ the K key matrix (scales with # of tokens)

An Example: Llama-2 7B

- **Linear terms:** 7B MACs per token (1-MAC per parameter)
- **Attention and linear** costs are approximately equal after ≅ 400 tokens.
- Mobile Performance: on Snapdragon 8 Gen 3, Llama-2 7B can be run with 20 token/second



Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).
Llama-2 7B: https://huggingface.co/meta-llama/Llama-2-7b

LLM Architecture and Computing Scaling

The compute has two separate components:

- Linear terms: Vector-matrix multiplication w/ the W matrix (fixed cost per time step)

- Attention terms: Matrix-vector multiplication w/ the K key ma

An Example: Ll

- Linear terms:

per paramete

- Attention and linear costs are approximately equal after ≅ 400 tokens.

- Mobile Performance: on Snapdragon 8 Gen 3, Llama-2 7B can be run with 20 token/second

However, for real-time mobile LLM applications we require ~ 2.5 words / second >= 3.3 tokens / second

More Optimization is needed!!

Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).
Llama-2 7B: https://huggingface.co/meta-llama/Llama-2-7b

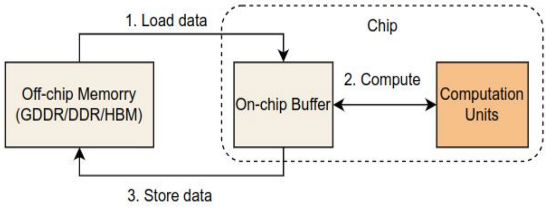
The Compute-Memory Dilemma in LLMs

Considerations before LLMs deployment:

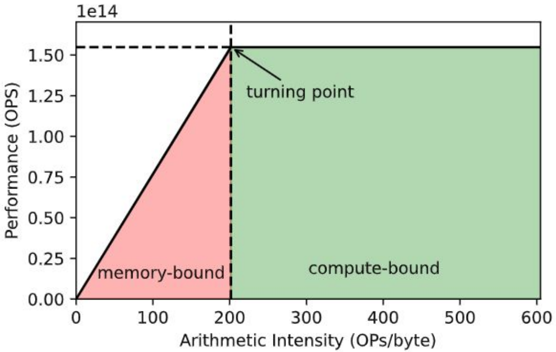
- **Computation Capability:** Support LLM op, mixed precision, and high computational power.
- **Memory size/bandwidth:** Scaling KV cache (text), multi-tasking (batch size), data movement.
- **Harmony:** computation and memory capacities should be well aligned w.r.t. LLM requirements.

- Most layers are compute bound in the **Prefill stage,** where the same layers become memory-bound in the decode stage.
- All layers in the **Decode stage** are memory bound as they need extensive read and write operation from memory.

Execution of an operation (linear/non-linear) on hardware.



Roofline Model on A6000 GPU



Analysis of Llama-2 7B layers on the A6000 GPU from NVIDIA

Layer Name	OPs	Memory Access	Arithmetic Intensity	Max Performance	Bound
			Prefill		
q_proj	69G	67M	1024	155T	compute
k_proj	69G	67M	1024	155T	compute
v_proj	69G	67M	1024	155T	compute
o_proj	69G	67M	1024	155T	compute
gate_proj	185G	152M	1215	155T	compute
up_proj	185G	152M	1215	155T	compute
down_proj	185G	152M	1215	155T	compute
qk_matmul	34G	302M	114	87T	memory
sv_matmul	34G	302M	114	87T	memory
softmax	671M	537M	1.25	960G	memory
norm	59M	34M	1.75	1T	memory
add	8M	34M	0.25	192G	memory
			Decode		
q_proj	34M	34M	1	768G	memory
k_proj	34M	34M	1	768G	memory
v_proj	34M	34M	1	768G	memory
o_proj	34M	34M	1	768G	memory
gate_proj	90M	90M	1	768G	memory
up_proj	90M	90M	1	768G	memory
down_proj	90M	90M	1	768G	memory
qk_matmul	17M	17M	0.99	762G	memory
sv_matmul	17M	17M	0.99	762G	memory
softmax	328K	262K	1.25	960G	memory
norm	29K	16K	1.75	1T	memory
add	4K	16K	0.25	192G	memory

Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288 (2023).
Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).

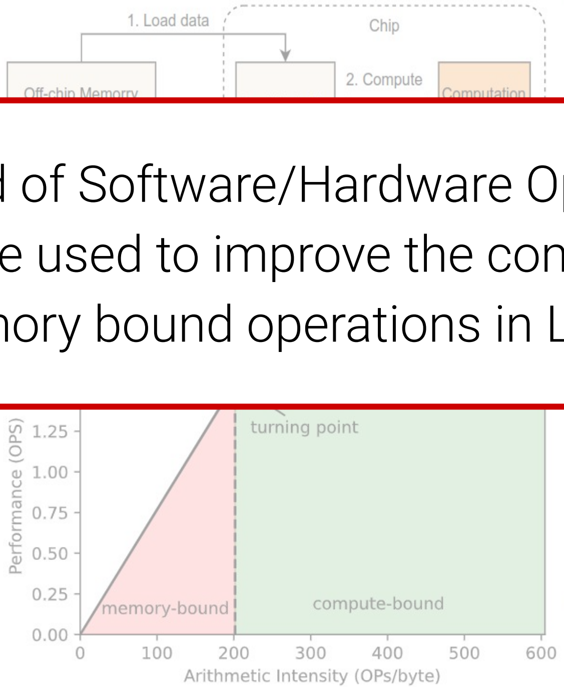
The Compute-Memory Dilemma in LLMs

Considerations before LLMs deployment:

- **Computation Capability:** Support LLM op, mixed precision, and high compute power.
- **Memory size/bandwidth:** Scaling KV cache (text) multi-tasking (batch size), data
- **Harmony:** computation and m should be well aligned w.r.t. LL

- Most layers are compute bo **stage,** where the same layers become memory-bound in the decode stage.
- All layers in the **Decode stage** are memory bound as they need extensive read and write operation from memory.

Execution of an operation (linear/non-linear) on hardware.



What Kind of Software/Hardware Optimization should be used to improve the compute and memory bound operations in LLMs?

Analysis of Llama-2 7B layers on the A6000 GPU from NVIDIA

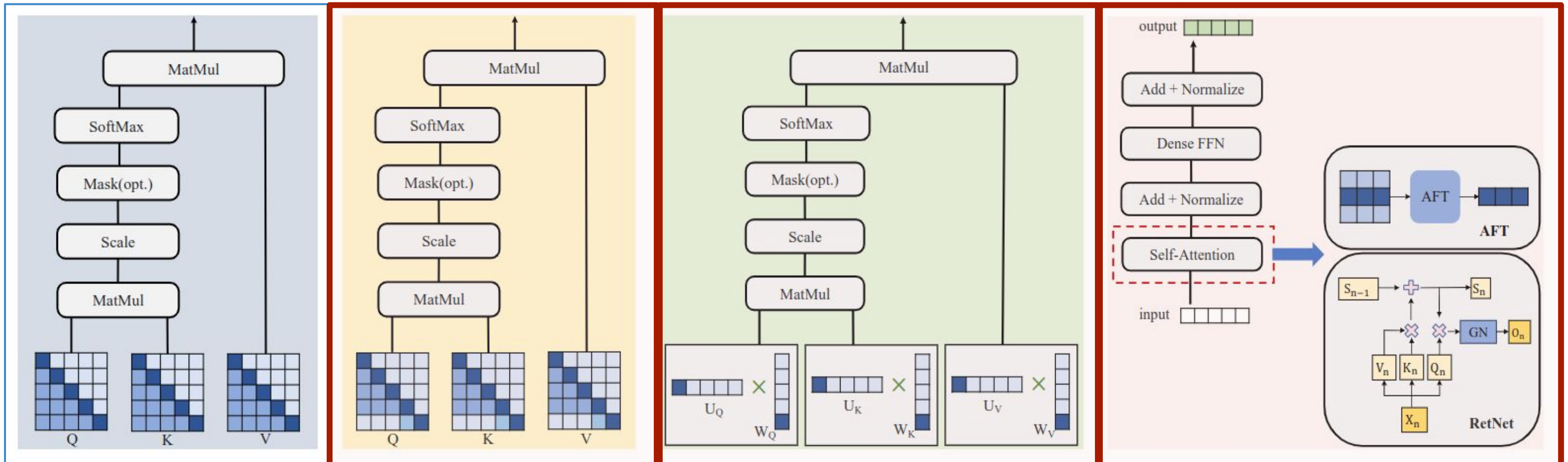
Layer Name	OPs	Memory Access	Arithmetic Intensity	Max Performance	Bound
Prefill					
q_proj	69G	67M	1024	155T	compute
			1024	155T	compute
			1024	155T	compute
			1024	155T	compute
			1215	155T	compute
			1215	155T	compute
			1215	155T	compute
			114	87T	memory
			114	87T	memory
			1.25	960G	memory
			1.75	1T	memory
			0.25	192G	memory
Decode					
q_proj	34M	34M	1	768G	memory
k_proj	34M	34M	1	768G	memory
v_proj	34M	34M	1	768G	memory
o_proj	34M	34M	1	768G	memory
gate_proj	90M	90M	1	768G	memory
up_proj	90M	90M	1	768G	memory
down_proj	90M	90M	1	768G	memory
qk_matmul	17M	17M	0.99	762G	memory
sv_matmul	17M	17M	0.99	762G	memory
softmax	328K	262K	1.25	960G	memory
norm	29K	16K	1.75	1T	memory
add	4K	16K	0.25	192G	memory

Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288 (2023).
Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).

Software Optimization – Pay More Attention to Attention Layers!

- Attention layers in LLMs are associated with **quadratic** computing complexity.
- Many optimized variants of attention have been proposed by exploiting: **sparsity**, **approximation**, **or replacement** with attention-free operations.

Model	Time	Space
Transformer	$O(T^2 d)$	$O(T^2 + Td)$
AFT	$O(T^2 d)$	$O(Td)$
Reformer	$O(T \log Td)$	$O(T \log T + Td)$
Hyena	$O(T \log Td)$	$O(Td)$
SSM	$O(T \log Td)$	$O(Td)$
Linear Transformers	$O(Td^2)$	$O(Td + d^2)$
RetNet	$O(Td)$	$O(Td)$
RWKV	$O(Td)$	$O(d)$



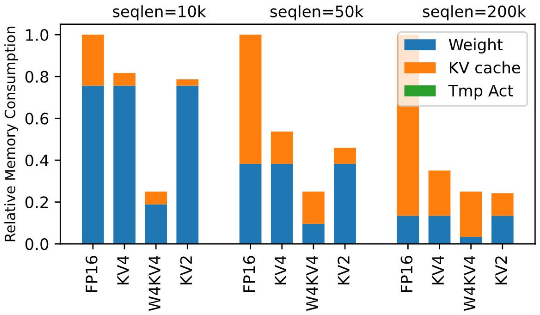
(a) Self-Attention. (b) Sparse Attention. (c) Approximate attention. (d) Attention-free.

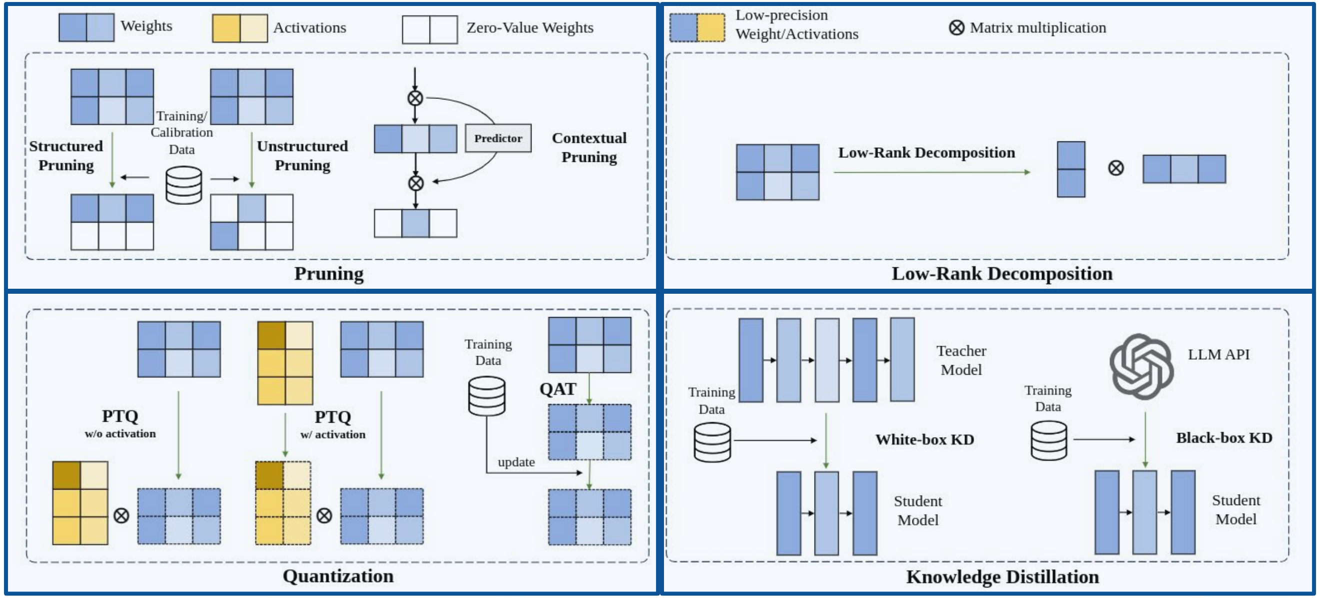
Xu, Mengwei, et al. "A survey of resource-efficient llm and multimodal foundation models." arXiv preprint arXiv:2401.08092 (2024).
Tornede, Alexander, et al. "Automl in the age of large language models: Current challenges, future opportunities and risks." arXiv preprint arXiv:2306.08107 (2023).

Software Optimization – To Compress or Not to compress

Quantization is the most used technique to deploy LLMs on Mobile Devices.
Quantization reduces both the **memory** footprint and inference **time**.
Quantization is **parametric** and finding the **right recipe** is key to **optimality**.

What and **When** to quantize?
Less Memory **>>>** More Efficiency





Evolution of Quantization Techniques for LLM

Xu, Mengwei, et al. "A survey of resource-efficient llm and multimodal foundation models." arXiv preprint arXiv:2401.08092 (2024).
Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).

Software Optimization – Dynamic Model Scaling

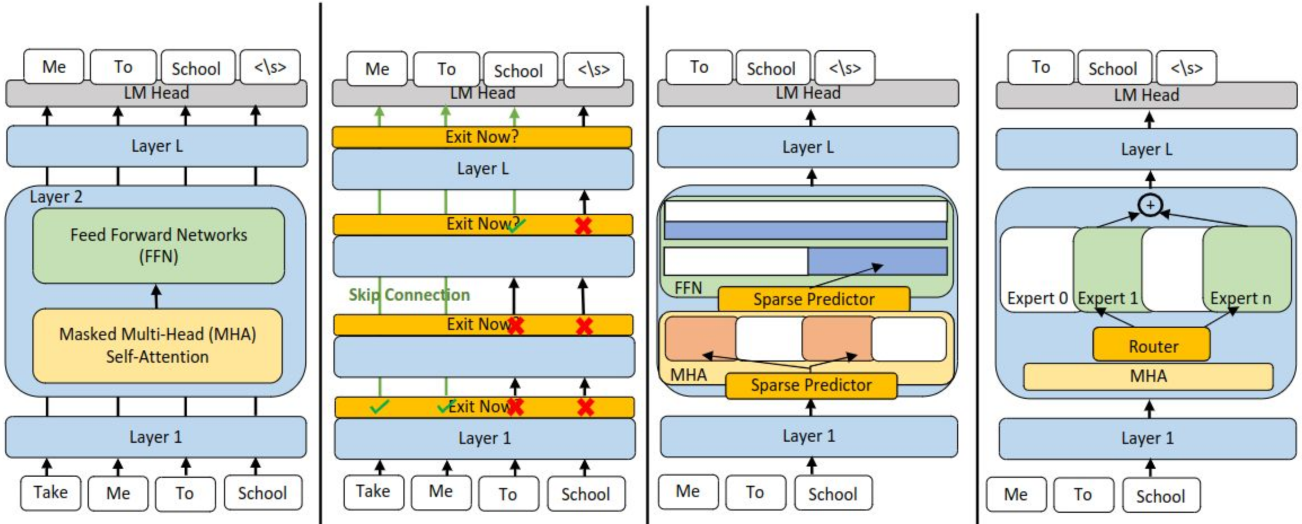
- **Dynamic Scaling** is an inference strategy that operates depending on the input prompt **context** & **difficulty**
- Use **less** computation for **easy** prompts and **more** computation for **difficult** prompts >> Dynamic Scaling

Three techniques used:

→ Depth-wise early-exit

→ Width-wise early-exit

→ Mixture-of-experts



(a) Normal Transformer Decoding | (b) Early Exiting (Depth-wise) | (c) Contextual Sparsity (Width-wise) | (d) Mixture of Expert

Full Execution
Equal computing loads for easy and difficult prompt

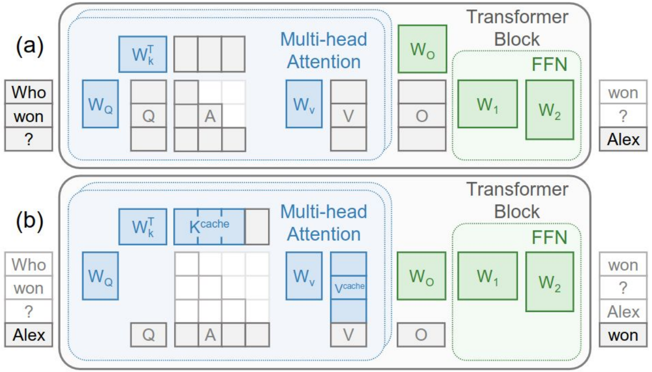
Partial Execution
Depending on prompt difficulty, exit on the layer with high confidence

Partial Execution
Depending on prompt difficulty, use less heads and neurons

Partial Execution
Depending on prompt difficulty, select less experts

Han, Yizeng, et al. "Dynamic neural networks: A survey." IEEE Transactions on Pattern Analysis and Machine Intelligence 44.11 (2021): 7436-7456.
Yuan, Zhihang, et al. "LLM Inference Unveiled: Survey and Roofline Model Insights." arXiv preprint arXiv:2402.16363 (2024).

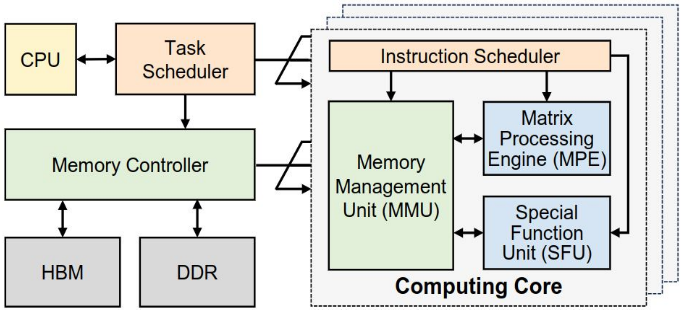
Hardware Optimization – Computation Parallelism



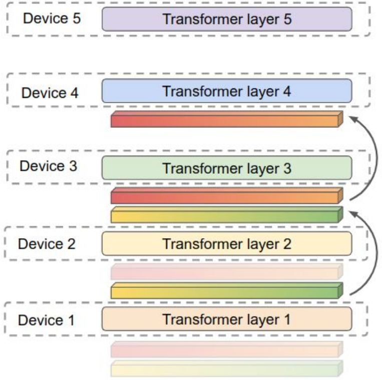
(a) Prefill stage in LLMs | (b) Decode stage in LLMs

How to map the LLM layers (Attention & FFN) on the computing devices of Hardware devices?

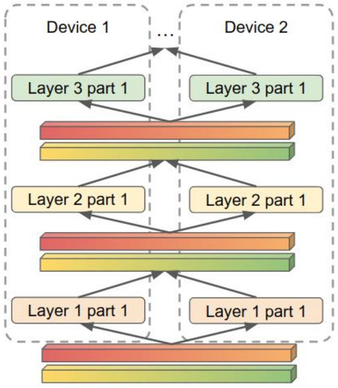
LLM Deployment



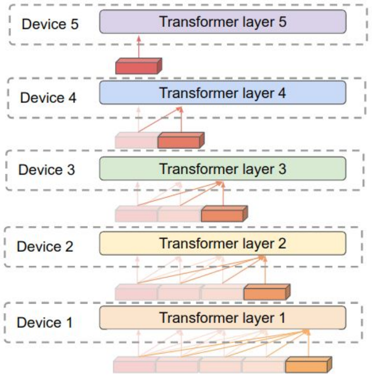
Layer parallelism
One-layer per device



Operator parallelism
Sub-layer per device



Token parallelism
One-token per device

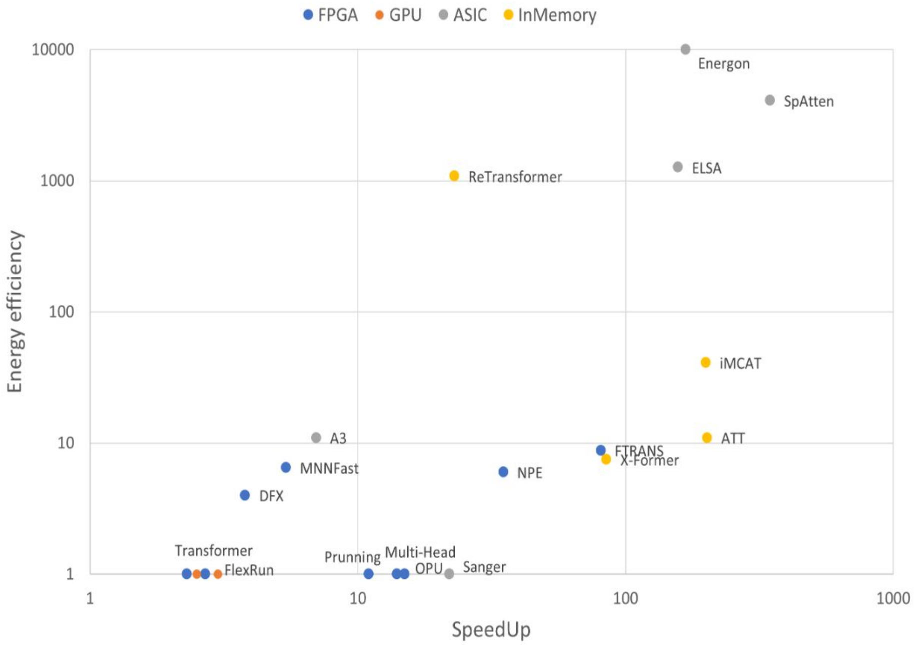


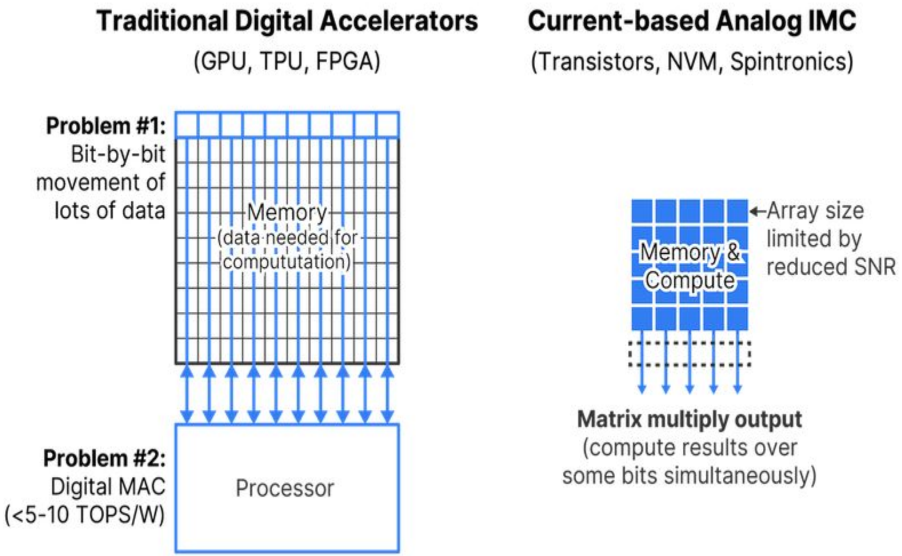
Zeng, Shulin, et al. "FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGA." arXiv preprint arXiv:2401.03868 (2024).
Li, Zhuohan, et al. "Terapipe: Token-level pipeline parallelism for training large-scale language models." International Conference on Machine Learning. PMLR, 2021.

Hardware Optimization – Domain-specific and IMC Accelerators

- Unlike **general-purpose** accelerators (CPU & GPU), **domain-specific** accelerators hold a lot of potential for LLM → Attention acceleration

- LLM operators are memory-bounds → **high overhead from data movement**
- **In-Memory-Computing** is a promising as memory and computation can be in the same physical arrays → **Avoid data movements overhad**





Kachris, Christoforos. "A Survey on Hardware Accelerators for Large Language Models." arXiv preprint arXiv:2401.09890 (2024).
https://www.enchargeai.com/technology

Conclusion – Future Directions