6.5930/1

Hardware Architectures for Deep Learning

Cascade of Einsums and Fusion

February 25, 2026

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Cascade of Einsums

MIT

Cascade of Einsums

- Up to this point, we focused on the mapping of a single Einsum

Einsum Z: $Z_{m,n} = A_{k,m} \times B_{k,n}$

- In practice, there are multiple Einsums, often in a cascade

Einsum A: $\boldsymbol{A_{k,m}} = I_{q,k} \times W_{q,m}$

Einsum C: $C_{m,n} = \boldsymbol{A_{k,m}} \times B_{k,n}$

For this cascade of Einsums (*Einsum A* → *Einsum C*) , we refer to
Input tensors: $I_{q,k}$ $W_{q,m}$ and $B_{k,n}$
Output tensor: $C_{m,n}$
Intermediate tensors: $\boldsymbol{A_{k,m}}$

"Fusion" will address how to reduce data movement of intermediate tensors

Many Ways to Connect Einsums (Cascade)

Single path: Output of Einsum is input to one other Einsum

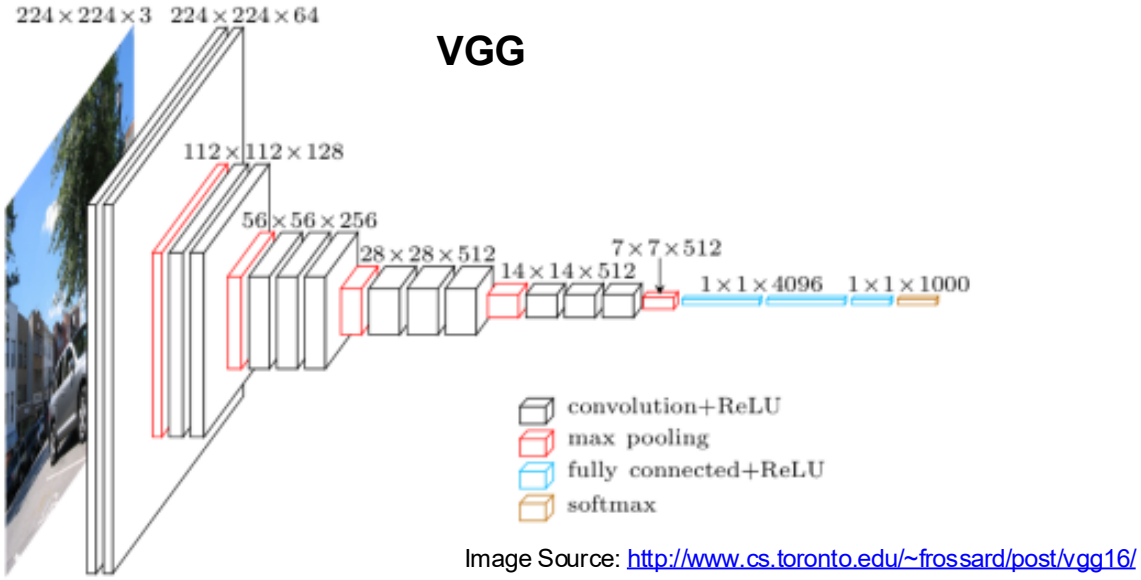


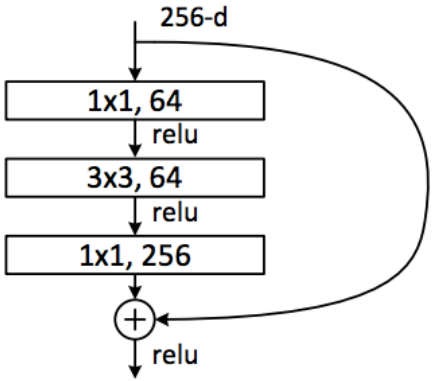
Image Source: http://www.cs.toronto.edu/~frossard/post/vgg16/

[**Simonyan**, *ICLR* 2015]

Many Ways to Connect Einsums (Cascade)

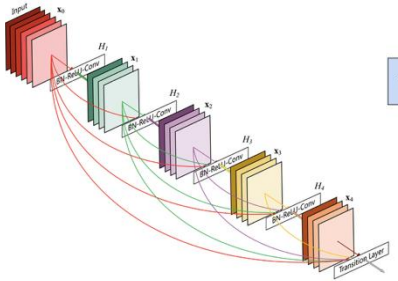
Multi-path: Output Einsum is input by multiple Einsums (branch out) and/or Multiple Einsums are input to single Einsum (merge)

ResNet (Short Cut block)



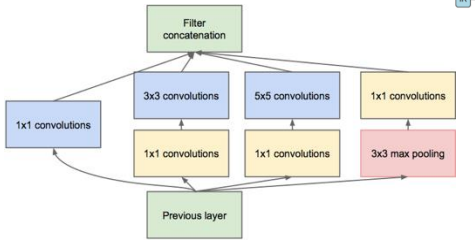
[**He**, *CVPR* 2016]

DenseNet



[**Huang**, *CVPR* 2017]

GoogleNet (Inception block)



[**Szegedy**, *CVPR* 2015]

Transformer (Attention)

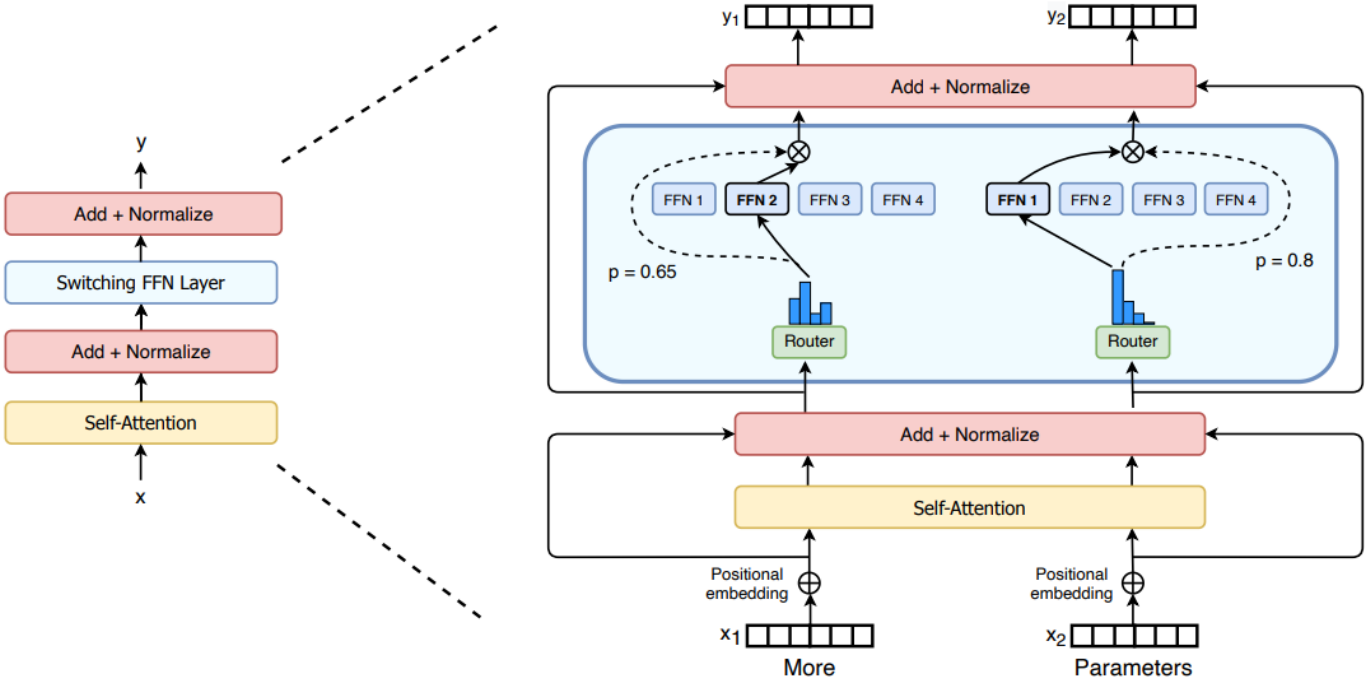
$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$
$$K_{m,e} = I_{m,d} \times WK_{d,e}$$
$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$
$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$
$$SN_{m,p} = exp(QK_{m,p})$$
$$SD_p = SN_{m,p}$$
$$A_{m,p} = SN_{m,p}/SD_p$$
$$V_{m,f} = I_{m,d} \times WV_{d,f}$$
$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$
$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

[**Vaswani**, *NeurIPS* 2017]

Vary in terms of connections between Einsums

Mixture of Expert (MoE)

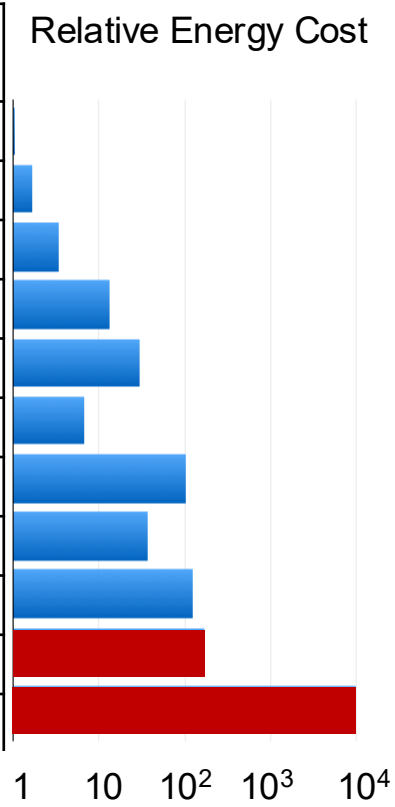
Cascade of Einsums where connections change dynamically



Source: Switch Transformer paper https://arxiv.org/abs/2101.03961

Energy Dominated by Data Movement

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640



Memory access is **orders of magnitude** higher energy than compute

[**Horowitz**, *ISSCC* 2014]

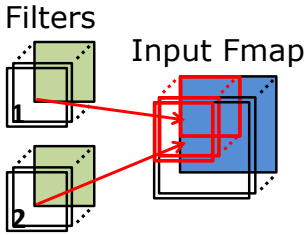
Example of Some Data Reuse Opportunities in DNNs

Input Reuse

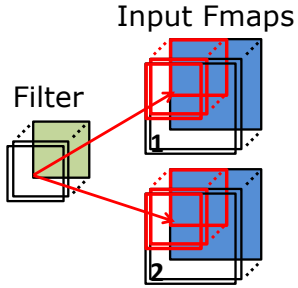
Weight Reuse

CNN

Input feature map reuse



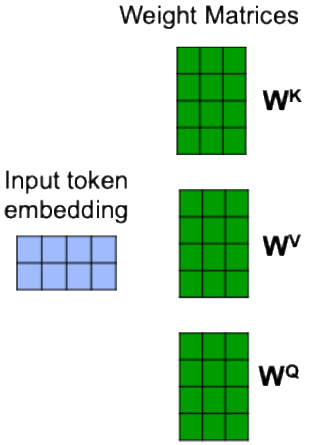
Filter reuse across batch



..and more

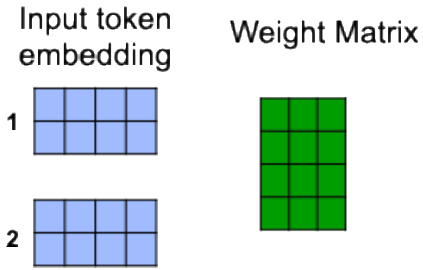
LLM

Input token embeddings reuse



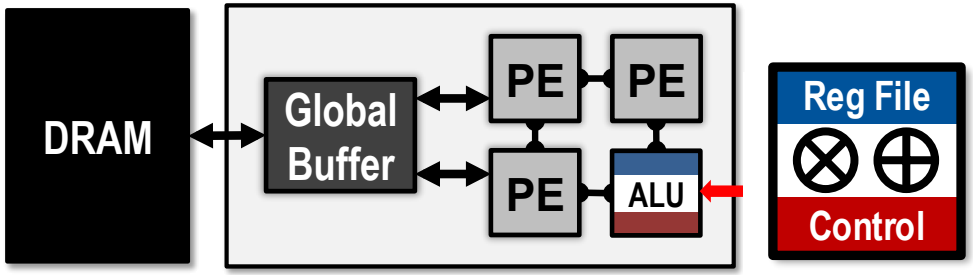
Weight reuse across sequence of token embeddings

Weight matrices reuse across batch

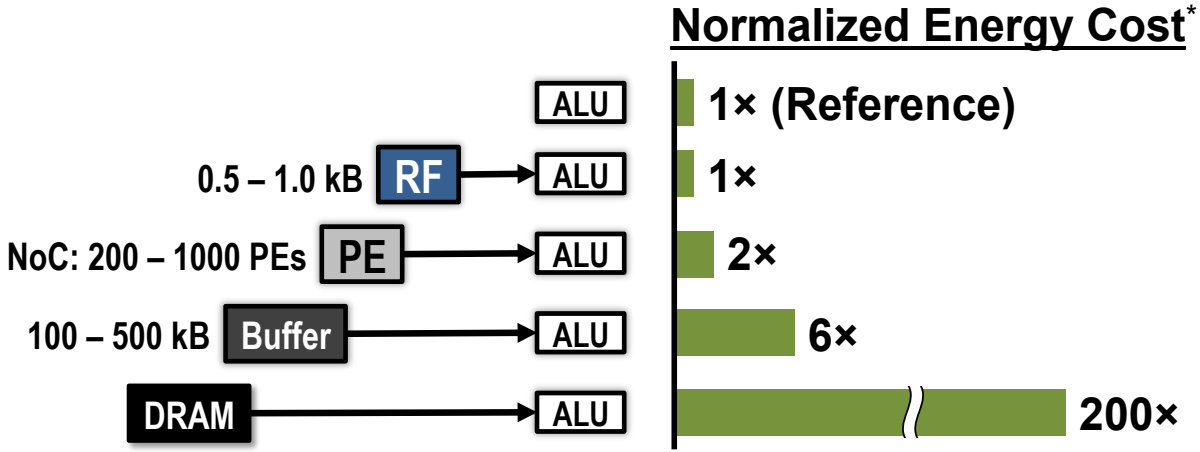


..and more

Exploit Data Reuse at Low-Cost Memories



Specialized hardware with small (< 1kB) low-cost memory near compute

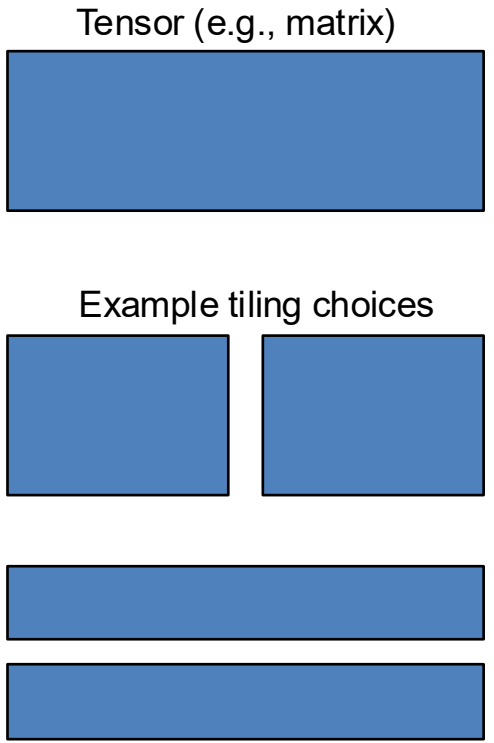


Farther and **larger** memories consume more power

* measured from a commercial 65nm process

Mapping Design Space

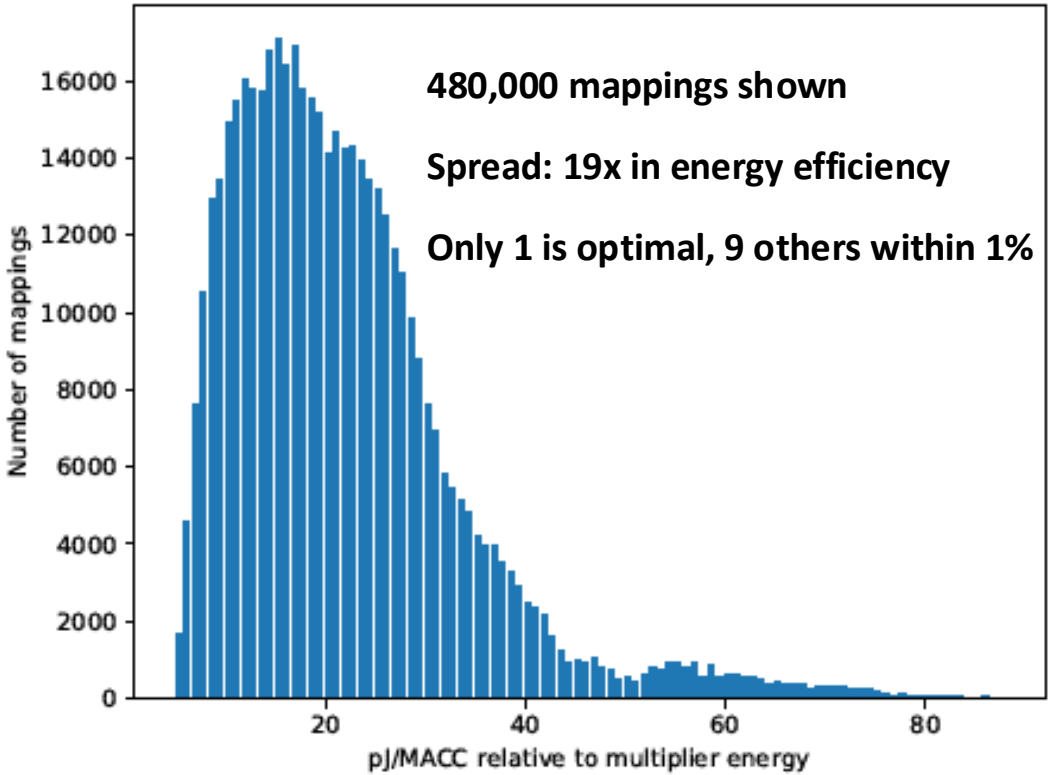
- Mapping design involves deciding
  - How to partition tensor into tiles (Tiling)
    - Ranks: Data is multi-dimension tensor – need to decide which rank to partitioning
    - Shape: (1) fit into memory (2) account for other data types (e.g., weights, inputs, outputs)
  - What is the processing order of the tiles (Dataflow)
    - Schedule across time and space (i.e., parallel PEs)
- Many combinations → Large Design Space!



Mapping Choice Impacts Energy Consumption

Mapping involves deciding
(1) how to partition tensors
(2) order of processing (dataflow)

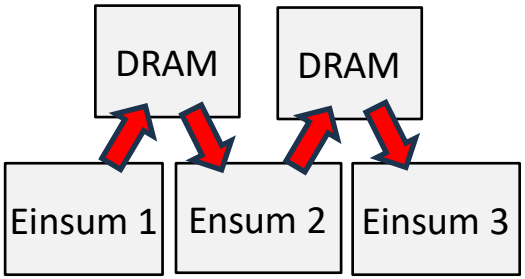
Mapping can significantly impact the energy efficiency and speed of a hardware platform



Source: Parashar, Timeloop

Another Opportunity for Data Reuse → Fusion

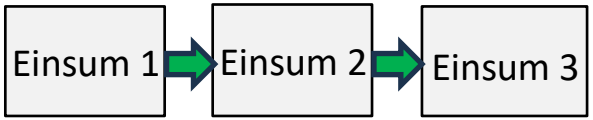
Off-chip transfers



Einsum-by-Einsum Processing

Fusion reduces off-chip transfers by retaining intermediate data on-chip (or by recomputing)

On-chip transfers



Fused Processing

Exploit data reuse data movement **across** multiple Einsums (e.g., layers of DNN)

Example of Fusion

Fully connected layer followed by ReLU activation function.
Assume entire O_m tensor too large to store on-chip.

Without Fusion

```
int i[CHW];          # Input activations
int f[M][CHW];       # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
          o[m] += i[chw]*f[CHW*m + chw]

for m in [0, M):
          o[m] = ReLU(o[m])
```

Number of off-chip (DRAM) accesses for o[m] ? 3M

write once for each m in first loop
+ read & write for each m in third loop

With Fusion

```
int i[CHW];          # Input activations
int f[M][CHW];       # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
          o[m] += i[chw]*f[CHW*m + chw]

  o[m] = ReLU(o[m])
```

Number of off-chip (DRAM) accesses for o[m] ? M

write once for each m in loop

Fusion Expands the Mapping Design Space

- Without Fusion
  - Select the mapping for each Einsum independently
    - M mapping options for N Einsum results in M × N mappings
- With Fusion
  - Choose to fuse or not to fuse
  - If choose to fuse, decide how many Einsums and which Einsums → 2^{N-1} combinations
  - For each combination, the mapping for each Einsum is dependent → M^N combinations
- Typically need to partition intermediate data since too large to fit on chip
  - Need to choose size of partition (partition is part of the mapping options)
  - This is referred to as ***tiled fusion***

Summary

- Data movement often dominates energy consumption in DNN accelerators
- Reduce amount of data movement with mappings (scheduling) that exploit data reuse at low-cost memories
- Previous lectures focus on reducing data movement within an Einsum
- Fusion involves reducing data movement across Einsums
- Fusion results in a much larger design space!

Explore Fusion in Lab 3