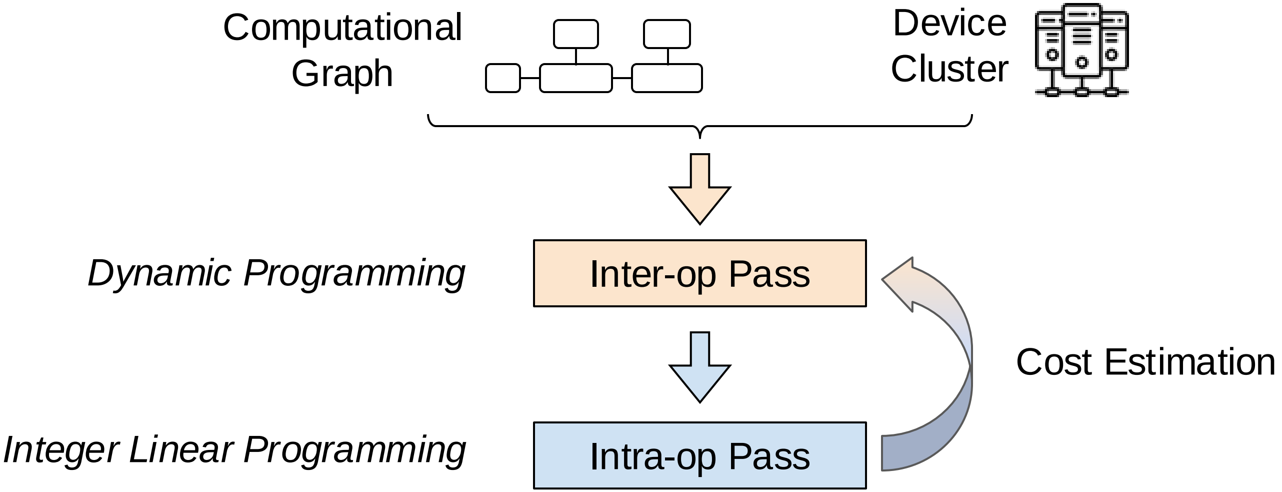
Logistics

- PA3 is posted.
  - Two programming assignments
    - One MoE
    - One LLM inference
  - One theoretical assignment (light programming)
    - Scaling law
  - You can collaborate on the above three
  - One essay
    - You shall finish independently

Where We Are

- Motivation
- History
- Parallelism Overview
- Data parallelism
- Model parallelism
    - Inter-op parallelism
    - Intra-op parallelism
- **Auto-parallelization**

Alpa Compiler: Hierarchical Optimization



Computational Graph

Device Cluster

Dynamic Programming

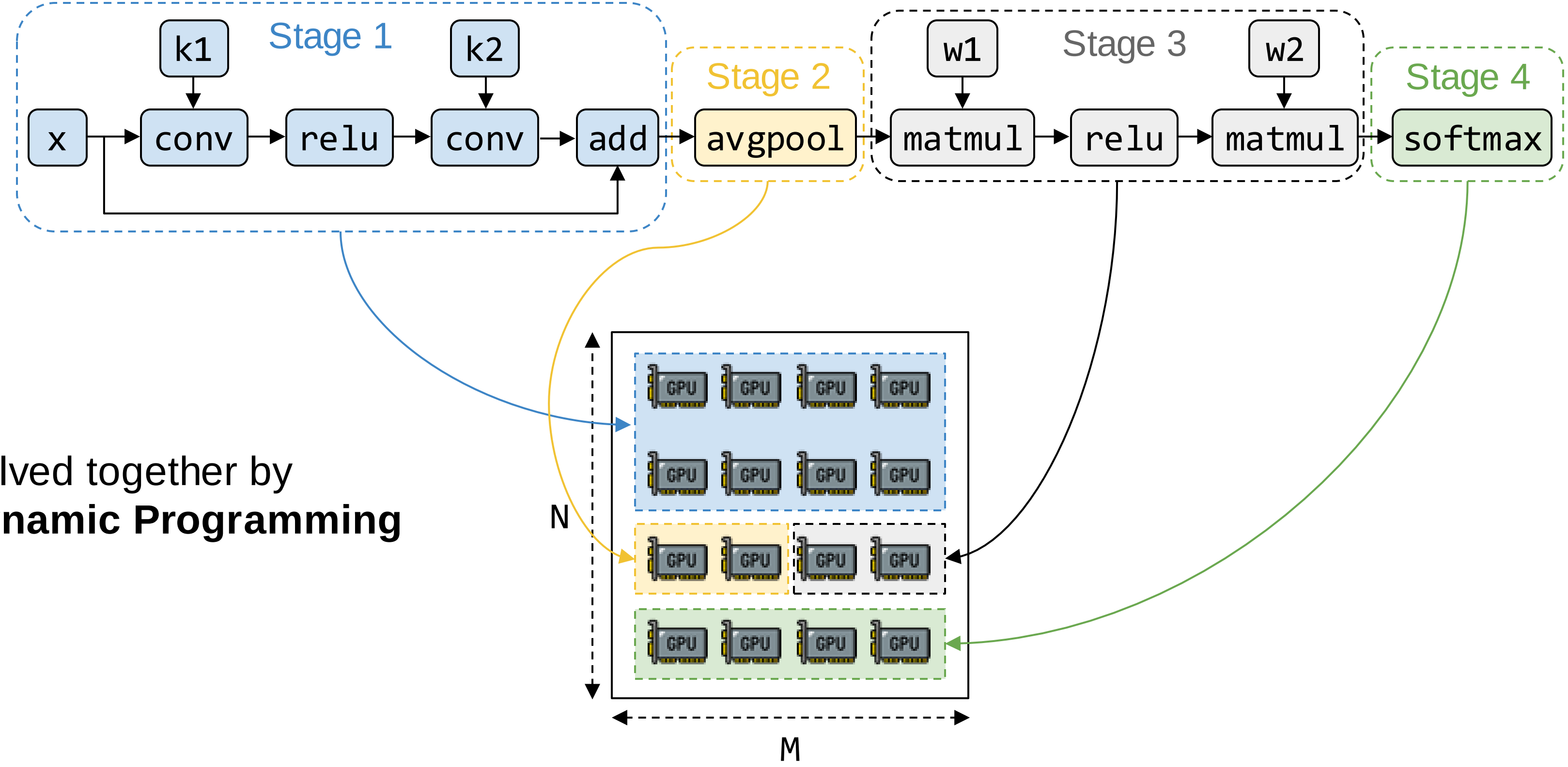
Inter-op Pass

Integer Linear Programming

Intra-op Pass

Cost Estimation

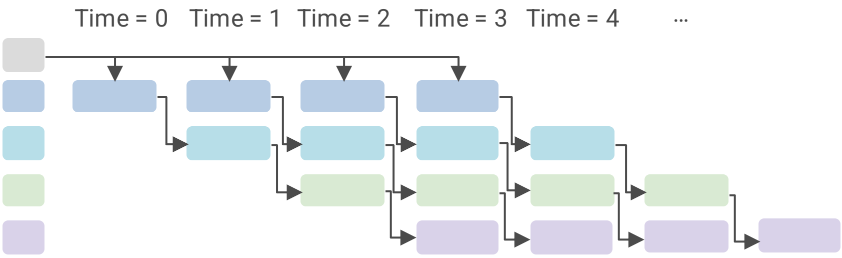
Inter-op Pass

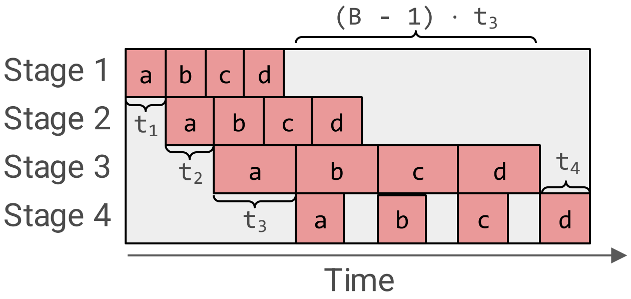


*S*olved together by
Dynamic Programming

Pipeline Execution Latency

Inter-op Pass





warmup phase

stable phase

$$T = \sum_{i}^{S} t_i + (B-1) \cdot \max_{1 \le j \le S} \{t_j\}$$

Inter-op Pass: Dynamic Programming

Optimization objective: Find the optimal $(\text{stage}, \text{mesh})$ pairs that minimize T.

$$T = \overset{\text{warmup phase}}{\sum_{i}^{S} t_i} + \overset{\text{stable phase}}{(B-1)\cdot \max_{1\le j\le S}\{t_j\}}$$

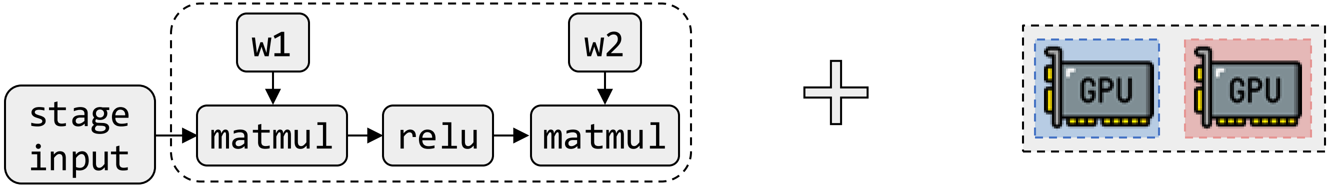
the **optimal** latency of executing stage i on its assigned mesh i :

$$t_i = t^{*}_{\text{intra}}(\text{stage}_i, \text{mesh}_i)$$

Solution:

Enumerate all possible $\max_{1\le j\le S}\{t_j\}$ (stable phase) and convert the first term $\sum_i^S t_i$ (warmup phase) into a *2-dimensional knapsack problem*.

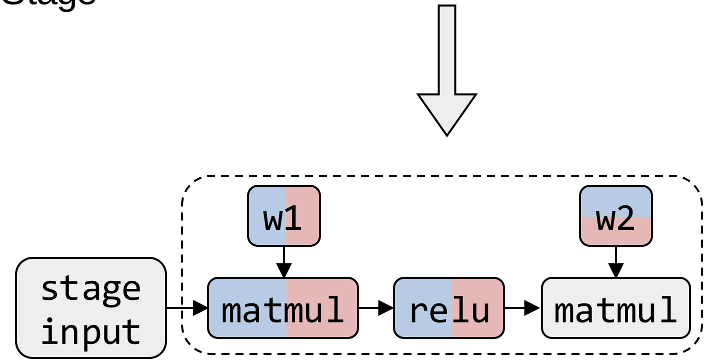
Intra-op Pass



Stage

Submesh

Solved by
Integer Linear Programming



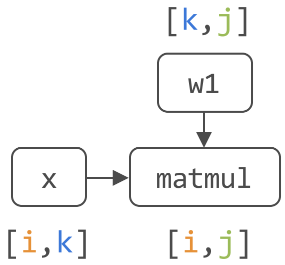
Stage with intra-operator parallelization

Intra-op Pass: Computation

🟦 Device 1

🟥 Device 2

⬒ Row-partitioned ◧ Column-partitioned ⬜ Replicated



	$\text{matmul}[i, j]$	$=$	$\Sigma_k\ x[i, k]$	$\times$	$w1[k, j]$	Cost
Algo#1: loop i	⬒	=	⬒	×	⬜	Cost1
Algo#2: loop j	◧	=	⬜	×	◧	Cost2
Algo#3: loop k	🟦 🟥	=	◧	×	⬒	Cost3
Algo#4: ...						

Intra-op Pass: Communication

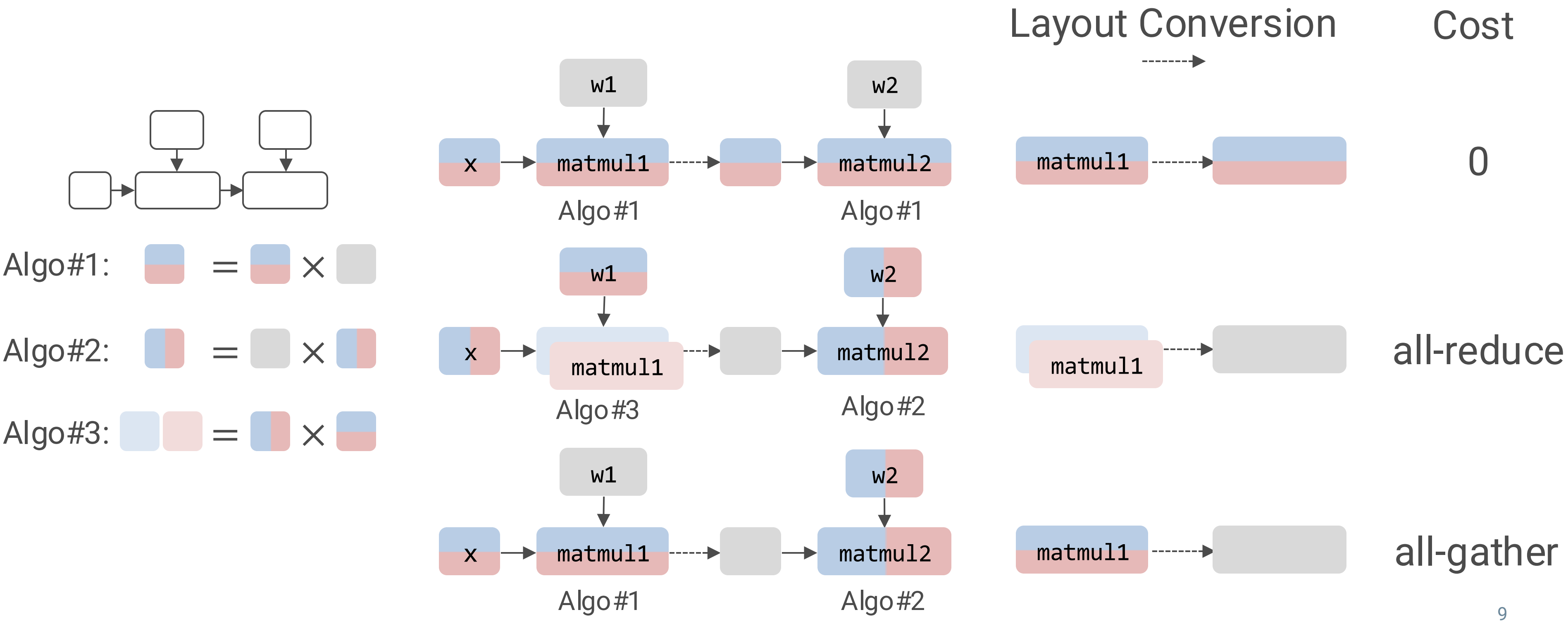
Device 1

Device 2

Row-partitioned

Column-partitioned

Replicated



Intra-op Pass: Layout Conversion

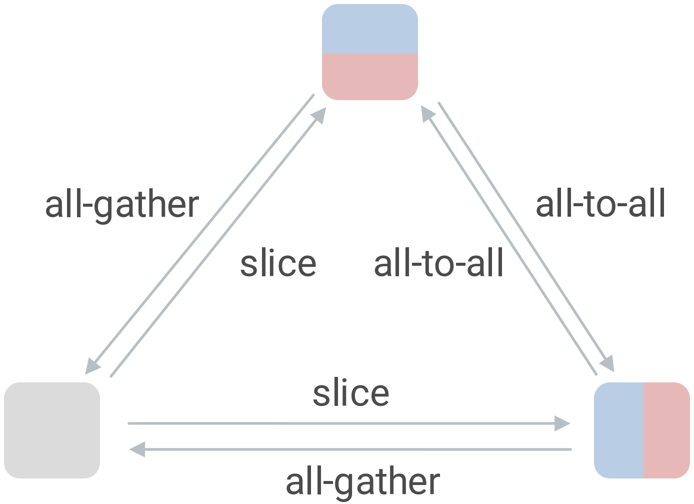
🟦 Device 1

🟥 Device 2

⬒ Row-partitioned

◧ Column-partitioned

⬜ Replicated

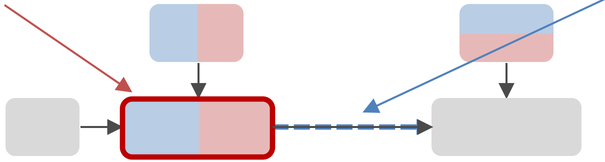


Intra-op Pass: ILP Formulation

Goal: Within each stage, "color" every node in the stage, so the execution latency of this stage on its assigned mesh is minimized.

For every node (op), enumerate all possible parallel algorithms

For every edge, infer the cost due to layout conversion

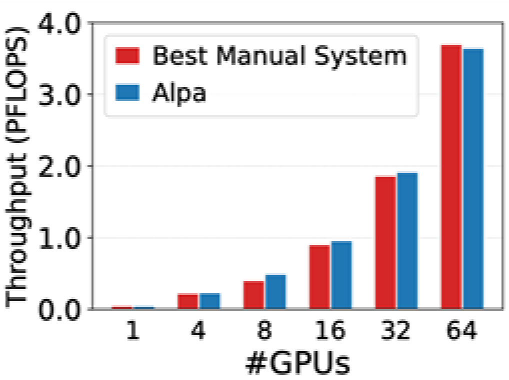


Minimize node-cost + edge-cost

s.t. peak memory usage < memory budget

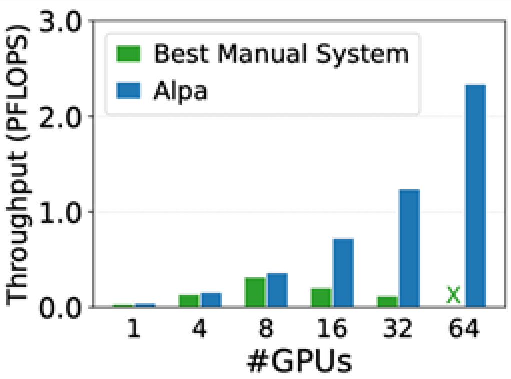
Evaluation: Comparing with Previous Works

GPT (up to 39B)



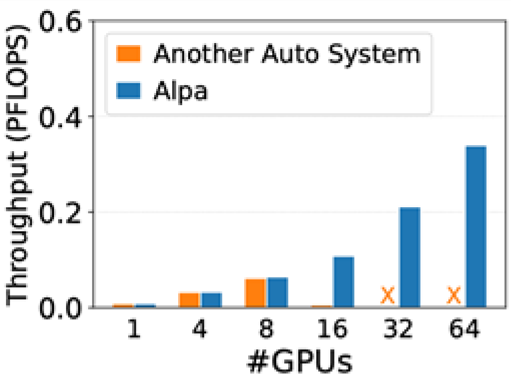
Match specialized manual systems.

GShard MoE (up to 70B)



Outperform the manual baseline by up to 8x.

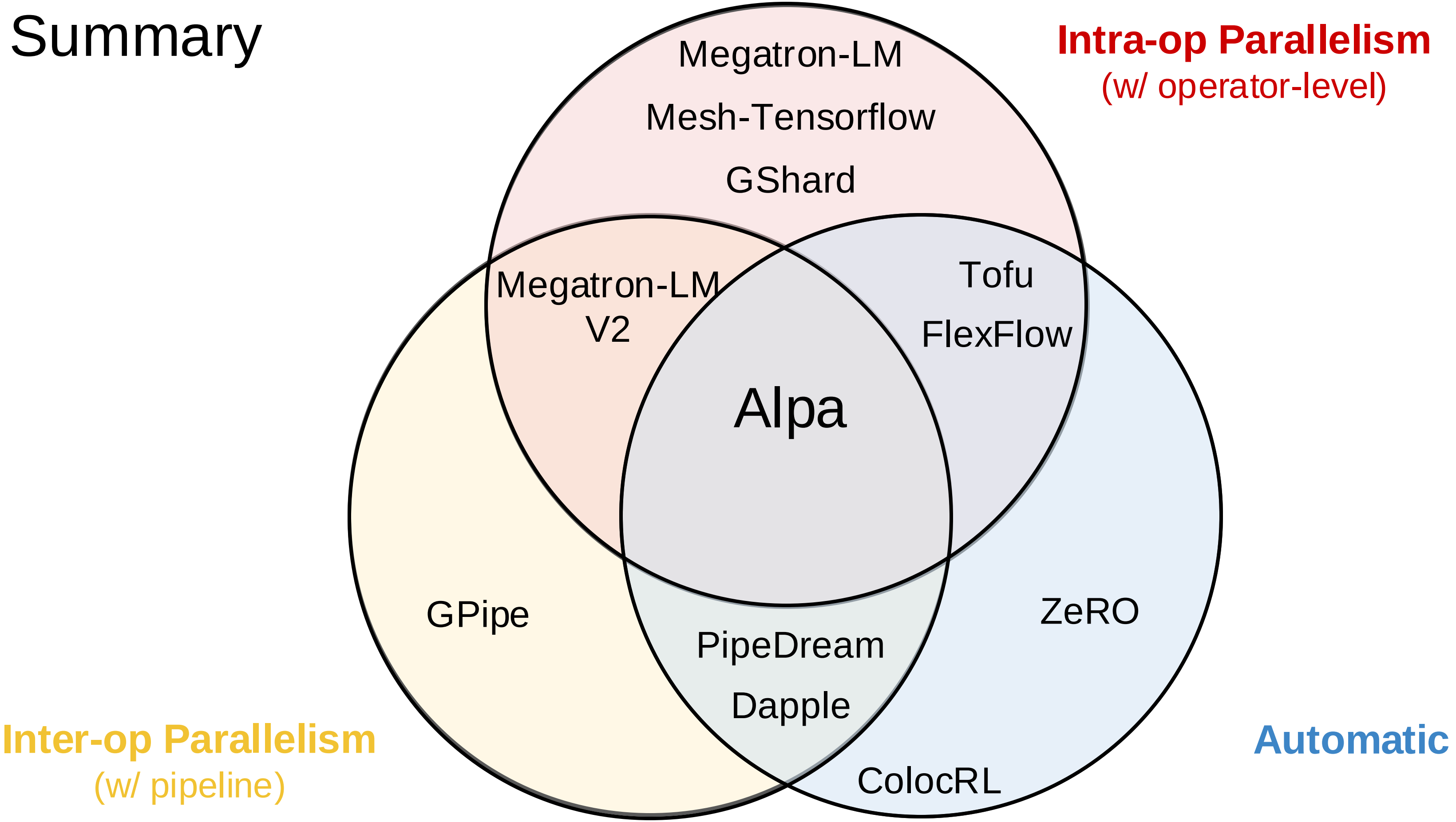
Wide-ResNet (up to 13B)



Generalize to models without manual plans.

Weak scaling results where the model size grow with #GPUs.
Evaluated on 8 AWS EC2 p3.16xlarge nodes with 8 16GB V100s each (64 GPUs in total).

Summary

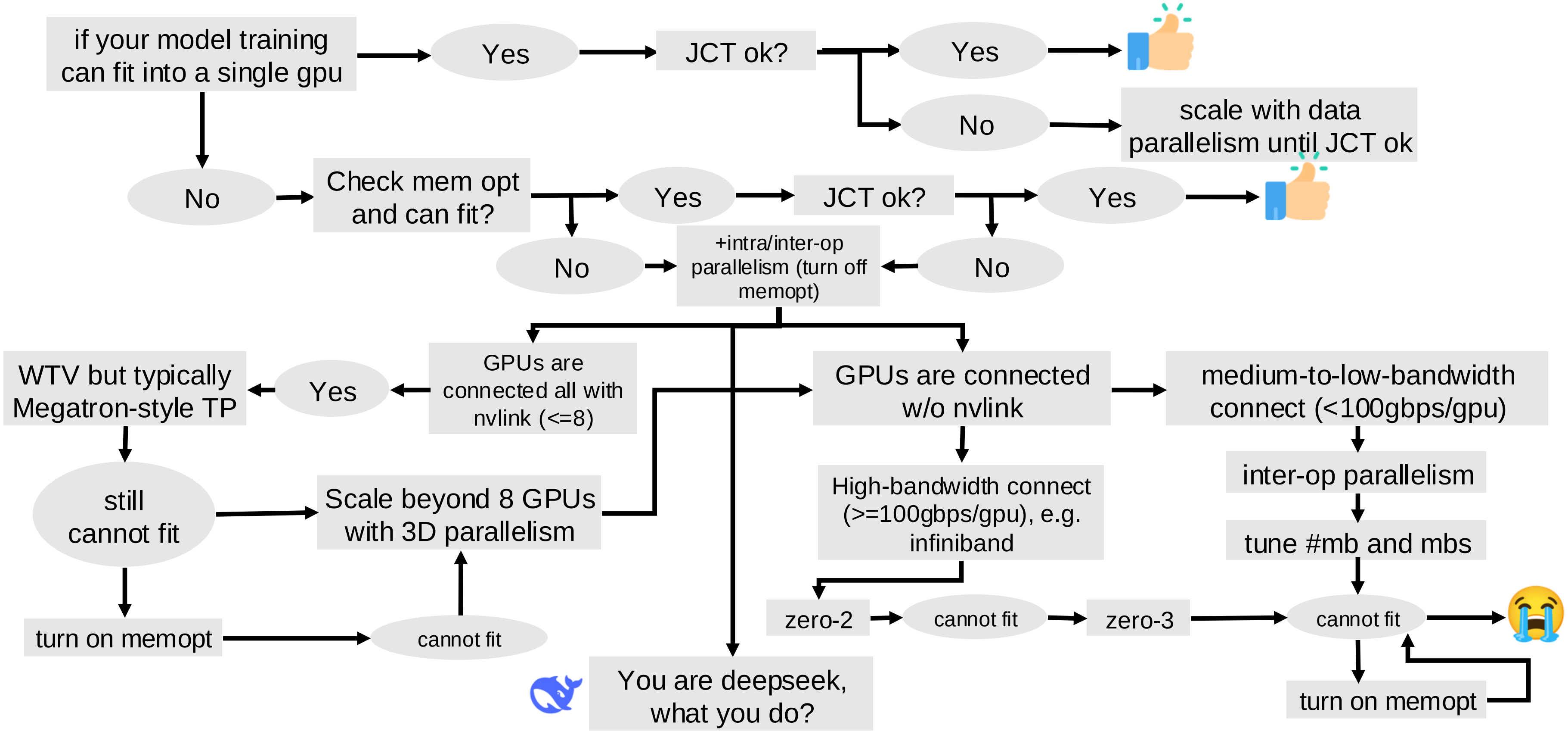

Intra-op Parallelism
(w/ operator-level)
Megatron-LM
Mesh-Tensorflow
GShard
Megatron-LM V2
Tofu
FlexFlow
Alpa
GPipe
PipeDream
Dapple
ZeRO
ColocRL
Inter-op Parallelism
(w/ pipeline)
Automatic

Summary: How to Choose Parallelism

1. Use automatic compiler if not transformer
2. Manual parallelism search for transformers:

- Factors to consider
  - #GPUs you have
  - Model size
  - JCT (Job completion time)
  - Communication bandwidth
  - etc.

Hao’s Ultimate Guide



if your model training can fit into a single gpu

Yes

JCT ok?

Yes

No

scale with data parallelism until JCT ok

No

Check mem opt and can fit?

Yes

JCT ok?

Yes

No

No

+intra/inter-op parallelism (turn off memopt)

GPUs are connected all with nvlink (<=8)

Yes

WTV but typically Megatron-style TP

still cannot fit

Scale beyond 8 GPUs with 3D parallelism

turn on memopt

cannot fit

You are deepseek, what you do?

GPUs are connected w/o nvlink

High-bandwidth connect (>=100gbps/gpu), e.g. infiniband

zero-2

cannot fit

zero-3

cannot fit

medium-to-low-bandwidth connect (<100gbps/gpu)

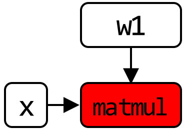
inter-op parallelism

tune #mb and mbs

cannot fit

turn on memopt

Big Picture



Dataflow Graph

Autodiff

Graph Optimization

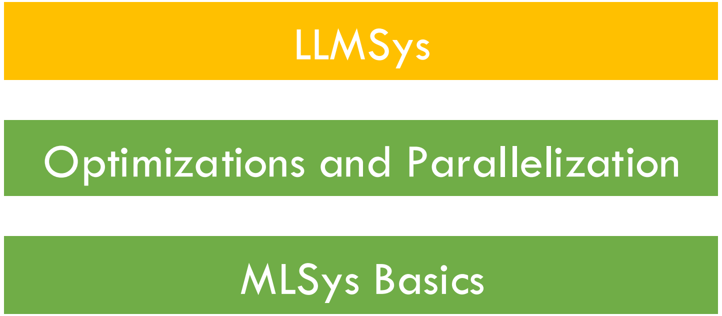
Parallelization

Runtime: schedule / memory

Operator optimization/compilation

GPU CPU

Next: Connecting the Dots



LLMSys

Optimizations and Parallelization

MLSys Basics

Large Language Models

- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Review Deepseek-v3
- Hot topics

Next Token Prediction

$$\mathrm{P}(next\,word \mid prefix)$$

San Diego has very nice _	surfing	0.4
	weather	0.5
	snow	0.01
San Francisco is a city of _	innovation	0.6
	homeless	0.3

Next Token Prediction

Probability(”San Diego has very nice weather”)
= P(“San Diego”) P(“has”|”San Diego”)P(“very”|”San Diego has”)P(“city”|…)…P(“weather”|…)

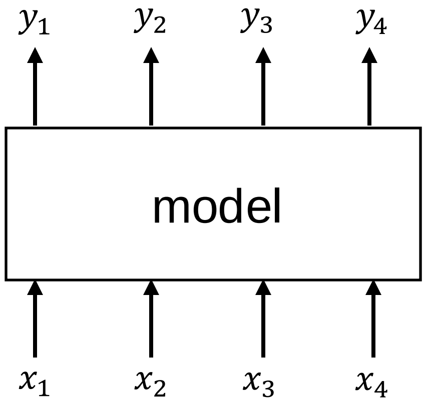
$$\text{Max}\,Prob(x_{1:T}) = \prod_{t=1}^{T} P(x_{t+1}|x_{1\ldots t})$$

MLE on observed data $x_{1:T}$,

This is next token prediction.
Predicting using seq2seq NNs.

Sequence Prediction

Take a set of input sequence, predict the output sequence



….

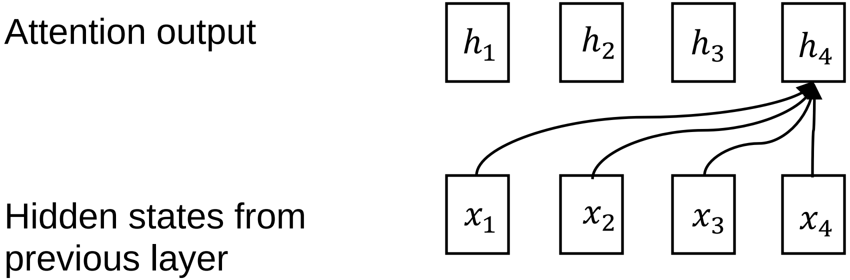
$$\prod_{t=1}^{T} P(x_{t+1}|x_{1...t})$$

Predict each output based on history $y_t = f_\theta (x_{1:t})$

There are many ways to build up the predictive model

"Attention" Mechanism

Generally refers to the approach that weighted combine individual states



$$h_t = \sum_{i=1}^{t} s_i x_t$$

Intuitively s_i is "attention score" that computes how relevant the position i's input is to this current hidden output

There are different methods to decide how attention score is being computed

Self-Attention Operation

Self attention refers to a particular form of attention mechanism.

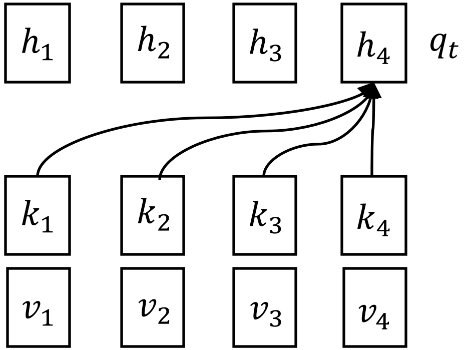
Given three inputs $Q, K, V \in \mathbb{R}^{T \times d}$ ("queries", "keys", "values")

Define the self-attention as:

$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right) V$$

A Closer Look at Self-Attention

Use q_t, k_t, v_t to refers to row t of the K matrix



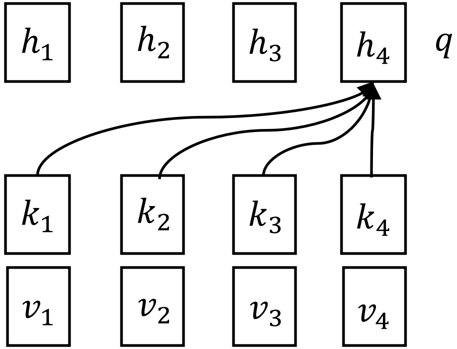
Ask the following question:

How to compute the output h_t, based on q_t, K, V one timestep t

To keep presentation simple, we will drop suffix t and just use q to refer to q_t in next few slide

A Closer Look at Self-Attention

Use q_t, k_t, v_t to refers to row t of the K matrix



Conceptually, we compute the output in the following two steps:

Pre-softmax "attention score"

$$s_i = \frac{1}{\sqrt{d}} q k_i^T$$

Weighed average via softmax

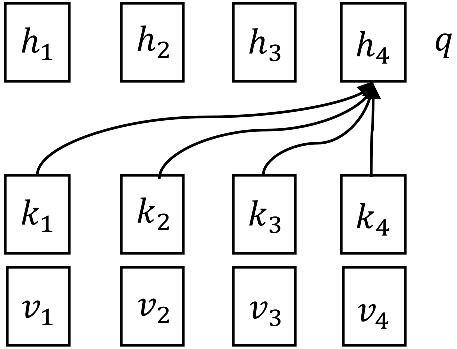
$$h = \sum_i \mathrm{softmax}(s)_i v_i = \frac{\sum_i \exp(s_i) v_i}{\sum_j \exp(s_j)}$$

Intuition: s_i computes the relevance of k_i to the query q,
then we do weighted sum of values proportional to their relevance

Comparing the Matrix Form and the Decomposed Form

Use q_t, k_t, v_t to refers to row t of the K matrix

$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$



Pre-softmax “attention score”

$$S_{ti} = \frac{1}{\sqrt{d}} q_t k_i^T$$

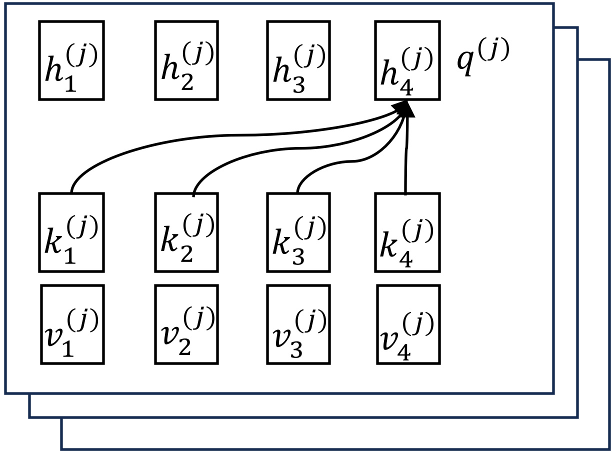
Weighed average via softmax

$$h_t = \sum_i \mathrm{softmax}(S_{t,:})_i v_i = \mathrm{softmax}(S_{t,:})V$$

Intuition: s_i computes the relevance of k_i to the query q, then we do weighted sum of values proportional to their relevance

Multi-Head Attention

Have multiple "attention heads" $Q^{(j)}, K^{(j)}, V^{(j)}$ denotes j-th attention head



Apply self-attention in each attention head

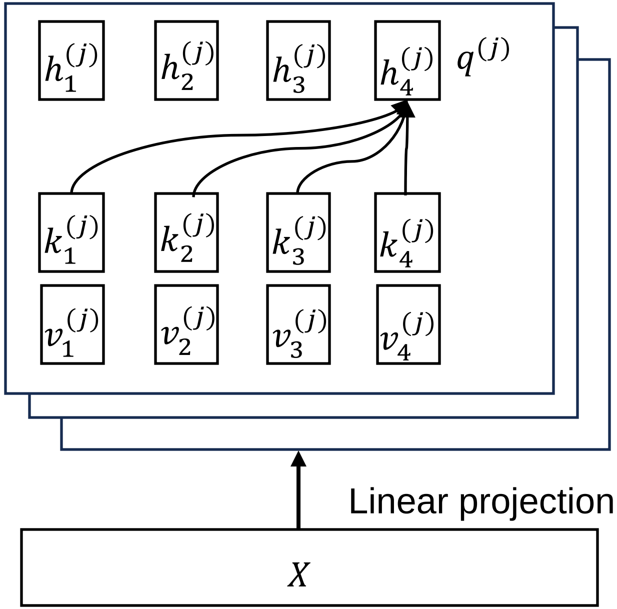
$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$

Concatenate all output heads together as output

Each head can correspond to different kind of information.
Sometimes we can share the heads: GQA(group query attention) all heads share K, V but have different Q

How to get Q K V?

Obtain Q, K, V from previous layer's hidden state X by linear projection



$$Q = XW_q$$
$$K = XW_K$$
$$V = XW_V$$

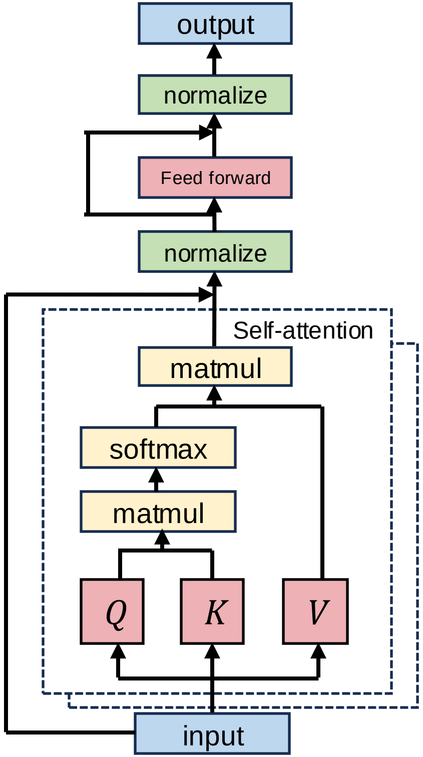
Can compute all heads and Q, K, V together then split/reshape out into individual Q, K, V with multiple heads

Transformer Block

A typical transformer block

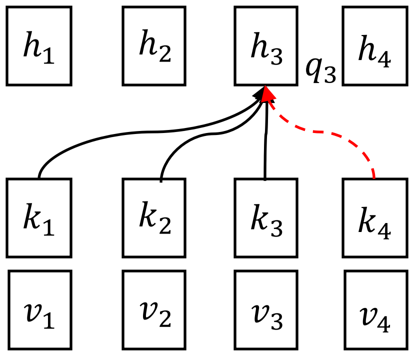
$$
\begin{aligned}
Z &= \mathrm{SelfAttention}\left(XW_K, XW_Q, XW_V\right) \\
Z &= \ \mathrm{LayerNorm}(X + Z) \\
H &= \mathrm{LayerNorm}(\mathrm{ReLU}(ZW_1)W_2 + Z)
\end{aligned}
$$

(multi-head) self-attention, followed by a linear layer and ReLU and some additional residual connections and normalization



Masked Self-Attention

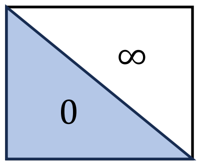
In the matrix form, we are computing weighted average over all inputs



In auto regressive models, usually it is good to maintain casual relation, and only attend to some of the inputs (e.g. skip the red dashed edge on the left). We can add “attention mask”

$$\text{MaskedSelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{d^{1/2}} - M\right)V$$

$$M_{ij} = \begin{cases} \infty, j > i \\ 0, j \leq i \end{cases}$$

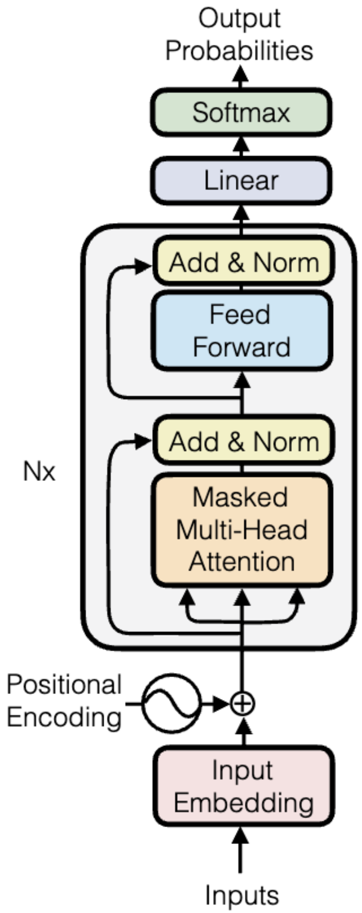


Only attend to previous inputs. Depending on input structure and model, attention mask can change.

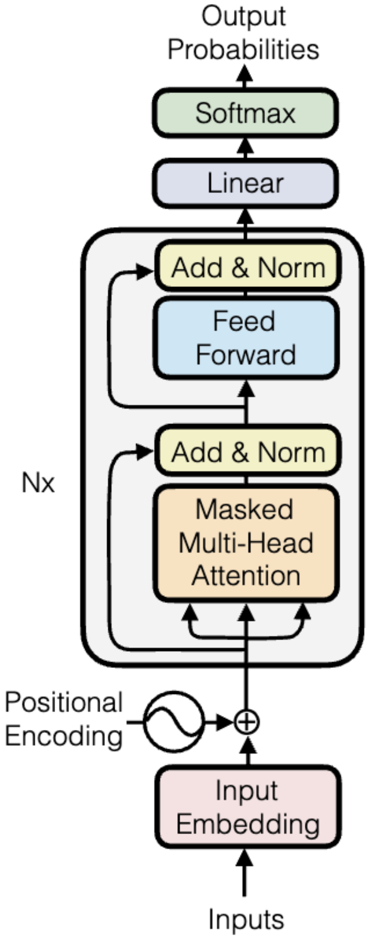
We can also simply skip the computation that are masked out if there is a special implementation to do so

Transformers

- Transformer decoders
  - Many of them
  - Really just: attentions + layernorm + MLPs dual
- Word embeddings
- Position embeddings
  - Rotary embedding
- Loss function: cross entropy loss over a seque

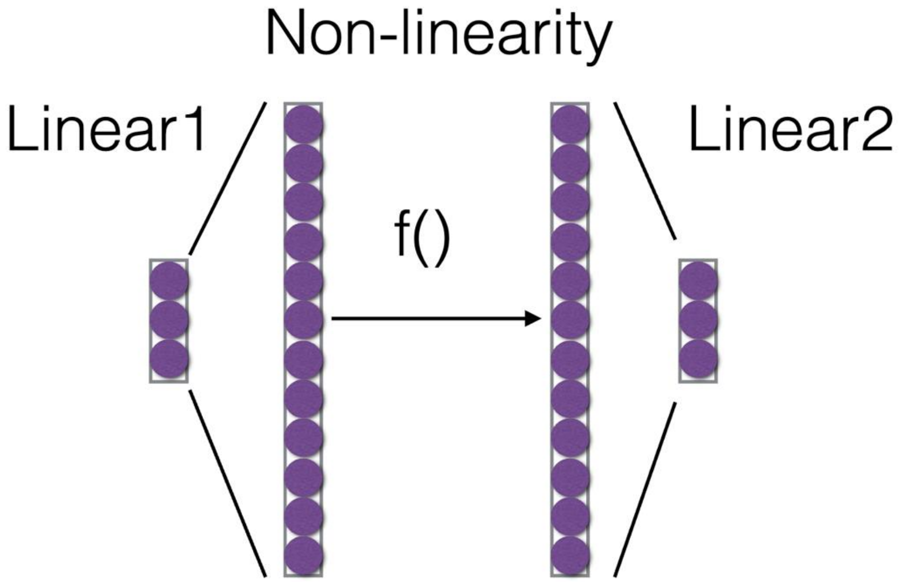


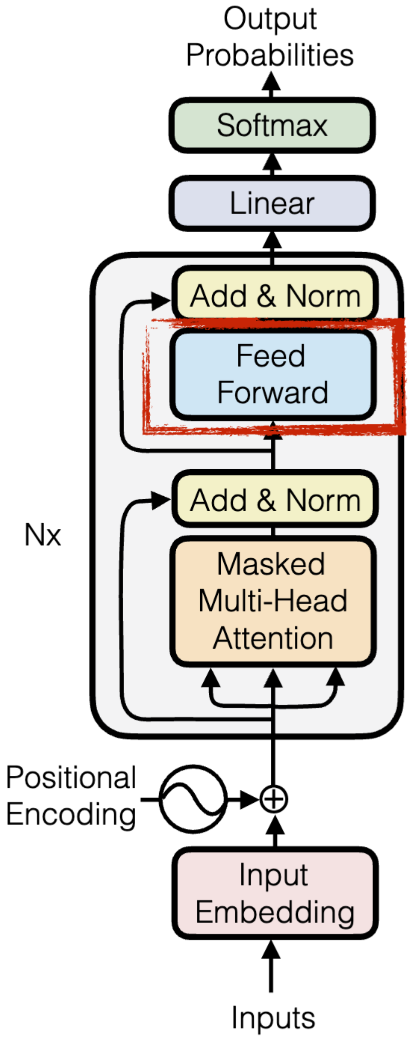
Transformers



Feedforward Layers

$$\mathrm{FFN}(x; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2) = f(\mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

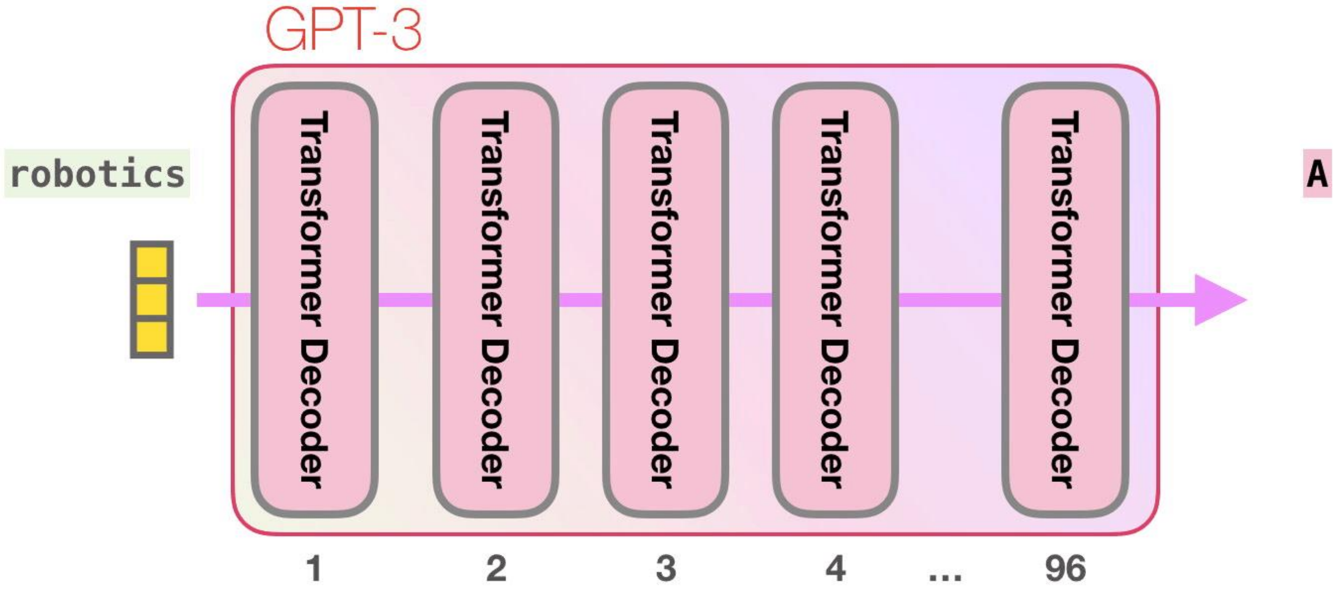




Computing Components in LLMs?

- Transformer decoders (many of them)
  - self-attentions (slow)
  - layernorm, residual (fast)
  - MLPs (slow)
  - Nonlinear (fast)
- Word embeddings (fast)
- Position embeddings (fast)
  - Absolute embedding vs. relative embedding
- Loss function: cross entropy loss over a sequence of words

LLMs

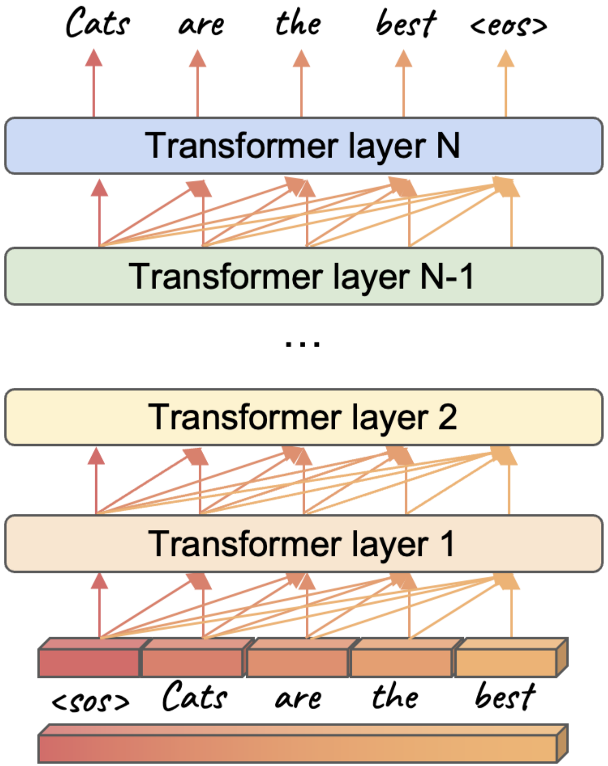

GPT-3
robotics
Transformer Decoder
Transformer Decoder
Transformer Decoder
Transformer Decoder
Transformer Decoder
1
2
3
4
...
96
A

Original Transformer vs. LLM today

	Vaswani et al.	LLaMA
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SiLU
Positional Encoding	Sinusoidal	RoPE

Training LLMs

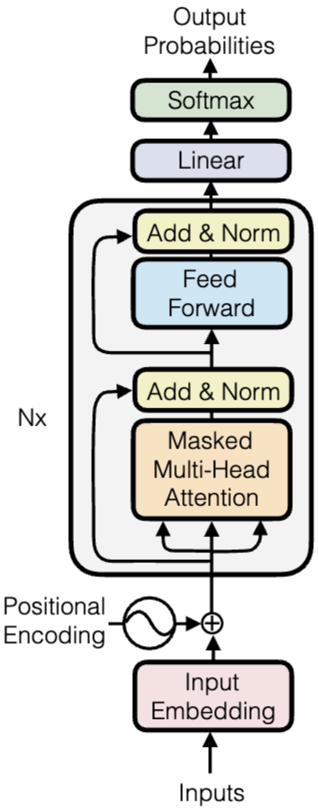
- Sequences are **known a priori**
- For each position, look at [1, 2, …, t-1] words to predict word t, and calculate the loss at t
- Parallelize the computation across all token positions, and then apply masking



Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

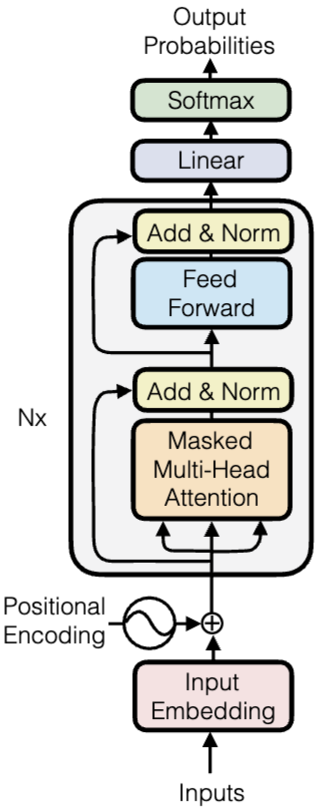
- calculate the number of parameters of an LLM?
  - memory, communication
- calculate the flops needed to train an LLM?
  - compute
- calculate the memory needed to train an LLM?
  - memory, communication

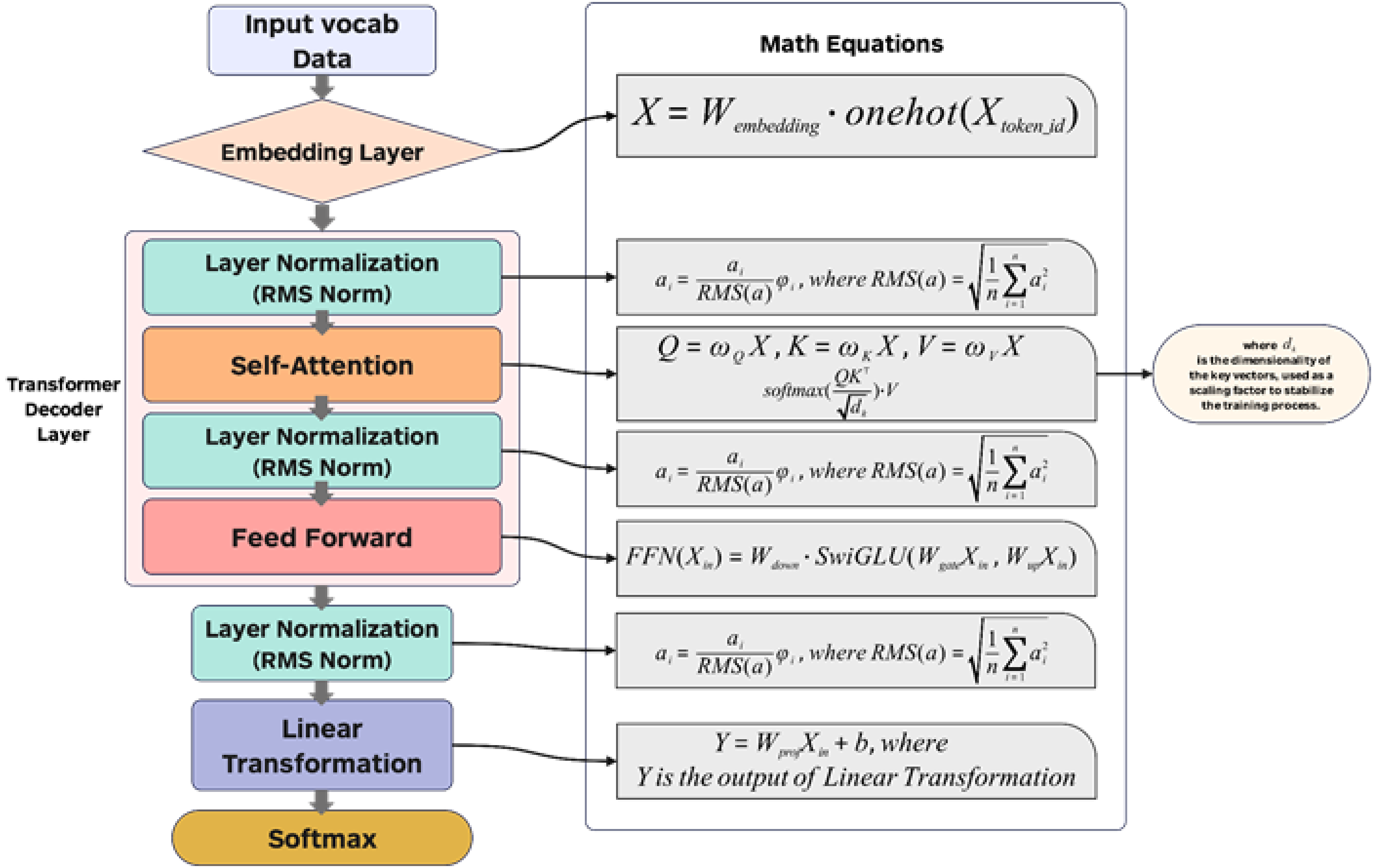


Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?





Input vocab Data

Embedding Layer

Transformer Decoder Layer

Layer Normalization (RMS Norm)

Self-Attention

Layer Normalization (RMS Norm)

Feed Forward

Layer Normalization (RMS Norm)

Linear Transformation

Softmax

Math Equations

$$X = W_{embedding} \cdot onehot(X_{token\_id})$$

$$a_i = \frac{a_i}{RMS(a)} \varphi_i \text{, where } RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$$Q = \omega_Q X \text{, } K = \omega_K X \text{, } V = \omega_V X$$

$$softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V$$

where d_k is the dimensionality of the key vectors, used as a scaling factor to stabilize the training process.

$$a_i = \frac{a_i}{RMS(a)} \varphi_i \text{, where } RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$$FFN(X_{in}) = W_{down} \cdot SwiGLU(W_{gate} X_{in} \text{, } W_{up} X_{in})$$

$$a_i = \frac{a_i}{RMS(a)} \varphi_i \text{, where } RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$Y = W_{proj} X_{in} + b$, where

Y is the output of Linear Transformation

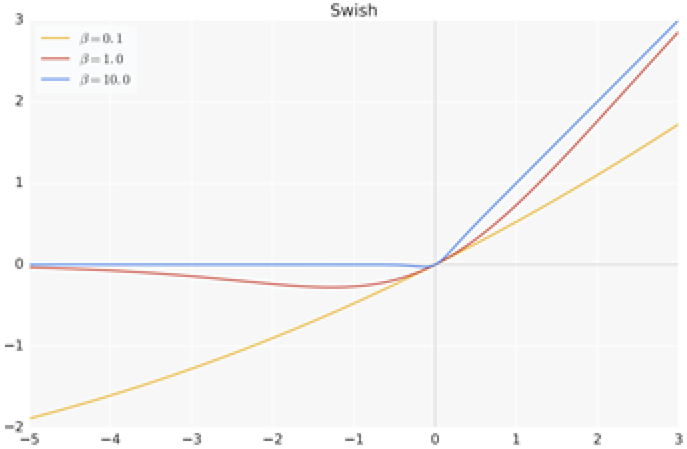
Feed Forward SwiGLU

The general formula for SwiGLU is:

$$\mathrm{SwiGLU}(x) = \mathrm{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

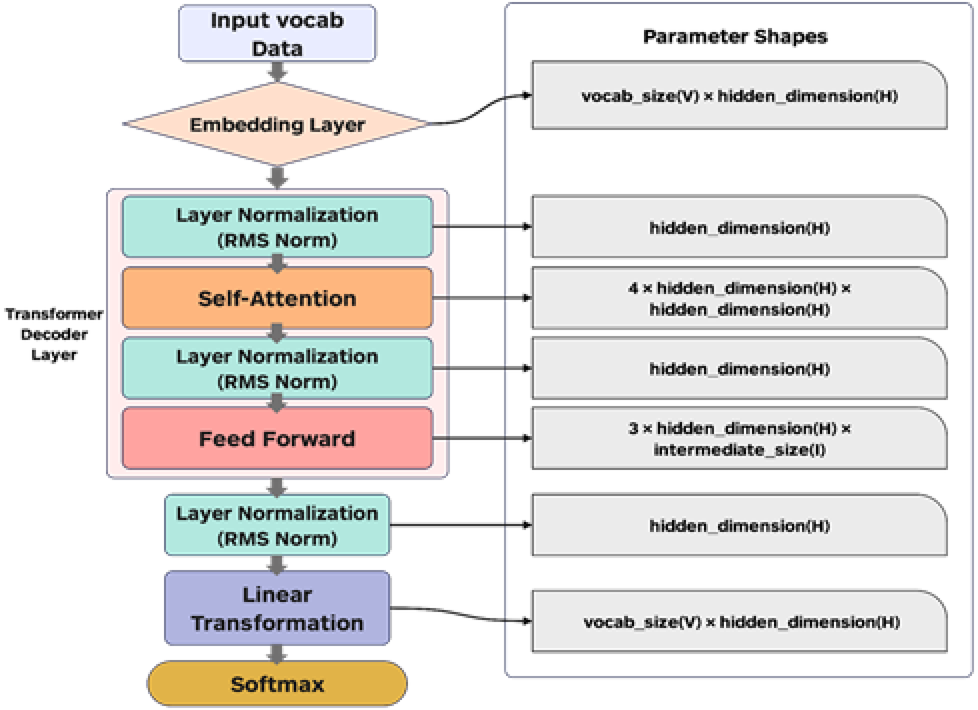
Swish is the activation function applied to one branch, defined as:

$$\mathrm{Swish}(z) = z \cdot \sigma(z)$$

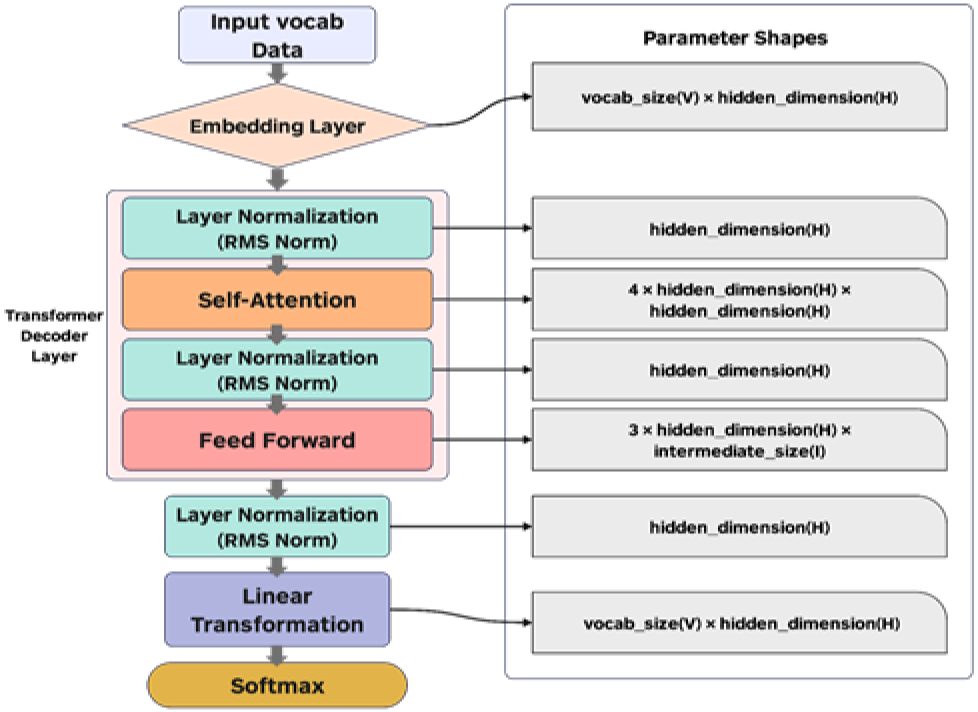


- SwiGLU helps the model capture more complex patterns by selectively gating information
- Swish is smoother than traditional activations ReLU

Summary


Input vocab Data
Embedding Layer
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Parameter Shapes
vocab_size(V) × hidden_dimension(H)
hidden_dimension(H)
4 × hidden_dimension(H) × hidden_dimension(H)
hidden_dimension(H)
3 × hidden_dimension(H) × intermediate_size(I)
hidden_dimension(H)
vocab_size(V) × hidden_dimension(H)

Scaling Up: Where is the Potential Bottleneck?



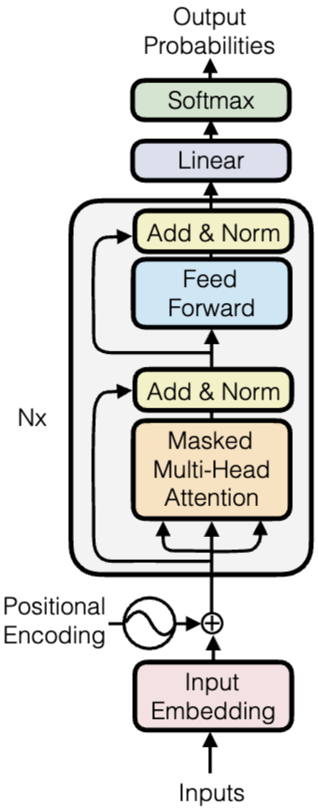
Input vocab Data
Embedding Layer
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Parameter Shapes
vocab_size(V) × hidden_dimension(H)
hidden_dimension(H)
4 × hidden_dimension(H) × hidden_dimension(H)
hidden_dimension(H)
3 × hidden_dimension(H) × intermediate_size(I)
hidden_dimension(H)
vocab_size(V) × hidden_dimension(H)

In PA3, you will implement this function😉

Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?

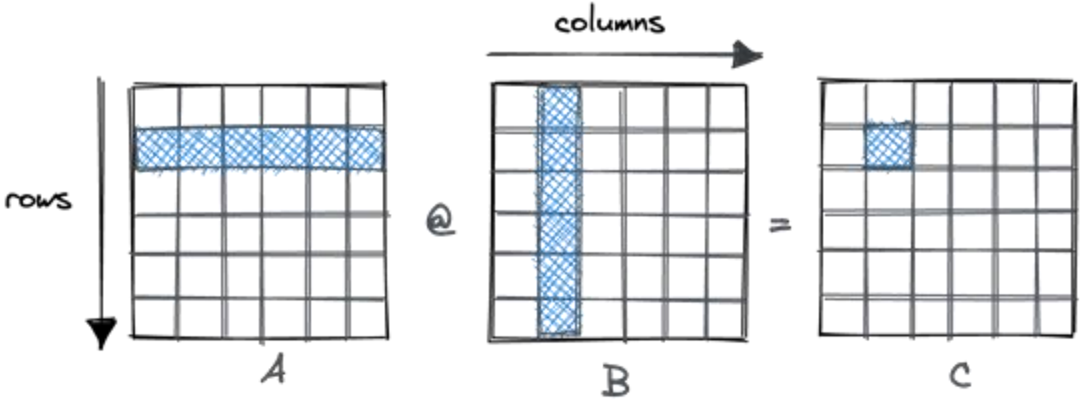


Estimate the Compute: FLOPs

The FLOPs for multiplying two matrices of dimensions m×n and n×h can be calculated as follows:

$$\text{FLOPs} = m \times h \times (2n - 1)$$

So the total number of FLOPs is roughly $\text{FLOPs} \approx 2m \times n \times h$



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b

Sequence length: s

The number of attention heads: n

Hidden state size of one head: d

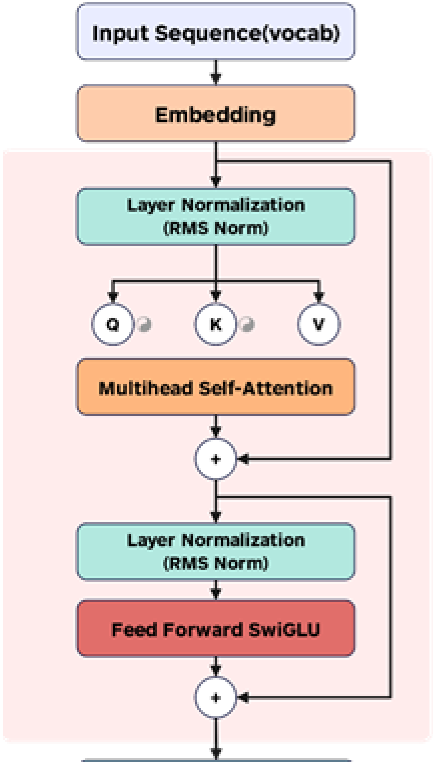
Hidden state size: h (h = n * d)

SwiGLU proj dim: i

Vocab size: v

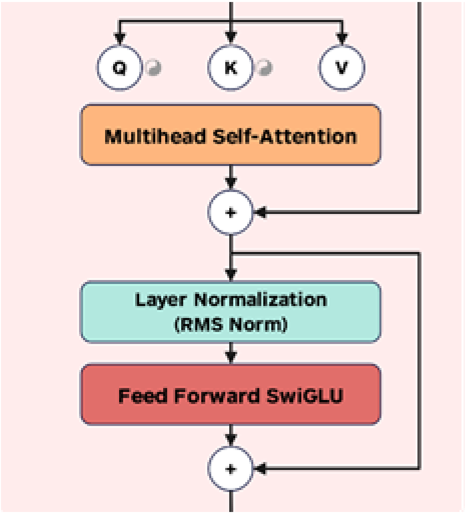
Batch size: b
Sequence length: s
# of attention heads: n
Hidden state dim of one head: d
Hidden state dim: h

Input:	Output Shape:	FLOPs
X	(b, s, h)	0
Self Attention:		
XW_Q, XW_K, XW_V	(b, s, h)	$3 * 2bsh^2$
RoPE	(b, n, s, d)	3bsnd
$P = \text{Softmax}(QK^T/\sqrt{d})$	(b, n, s, s)	$2bs^2nd + 3bs^2n$
PV	(b, n, s, d)	$2bs^2nd$
AW_O	(b, s, h)	$2bsh^2$
Residual Connection:	(b, s, h)	bsh



Batch size: b
Sequence length: s
Hidden state dim: h
SwiGLU proj dim: i

Output from Self Attn:	Output Shape:	FLOPs
X	(b, s, h)	0
Feed-Forward SwiGLU:		
XW_{gate}, XW_{up}	(b, s, i)	2 * 2bshi
Swish Activation	(b, s, i)	4bsi
Element-wise *	(b, s, i)	bsi
XW_{down}	(b, s, h)	2bshi
RMS Norm:		
	(b, s, h)	4bsh + 2bs



1. **Calculate Root Mean Square:**
   - $\mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$
2. **Normalize:**
   - $\mathrm{RMSNorm}(x) = \frac{x}{\mathrm{RMS}(x)+\epsilon} \cdot \gamma$

$$\mathrm{SwiGLU}(x) = \mathrm{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

LLama 2 7B Flops Forward (Training)

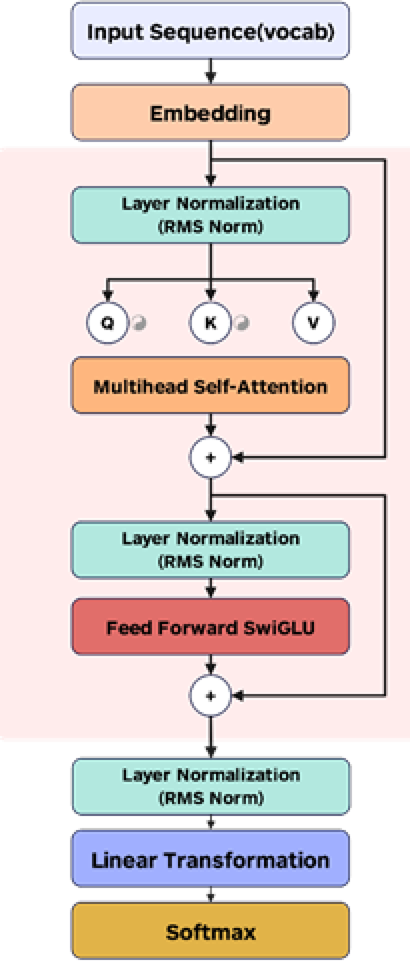
Total Flops ≈ #num_layers * (Attention block + SwiGLU block)

+ Prediction head

$= \#num\_layers * (6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2)$

+ #num_layers (6bshi)

+ 2 bshv



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b=1

Sequence length: s=4096

The number of attention heads: n=32

Hidden state size of one head: d=128

Hidden state size: h =4096

SwiGLU proj dim: i=11008

Vocab size: v=32000

The number of layers: N=32

Total Flops ≈ N * $(6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2)$

$+ N(6bshi)$

$+ 2\,bshv$

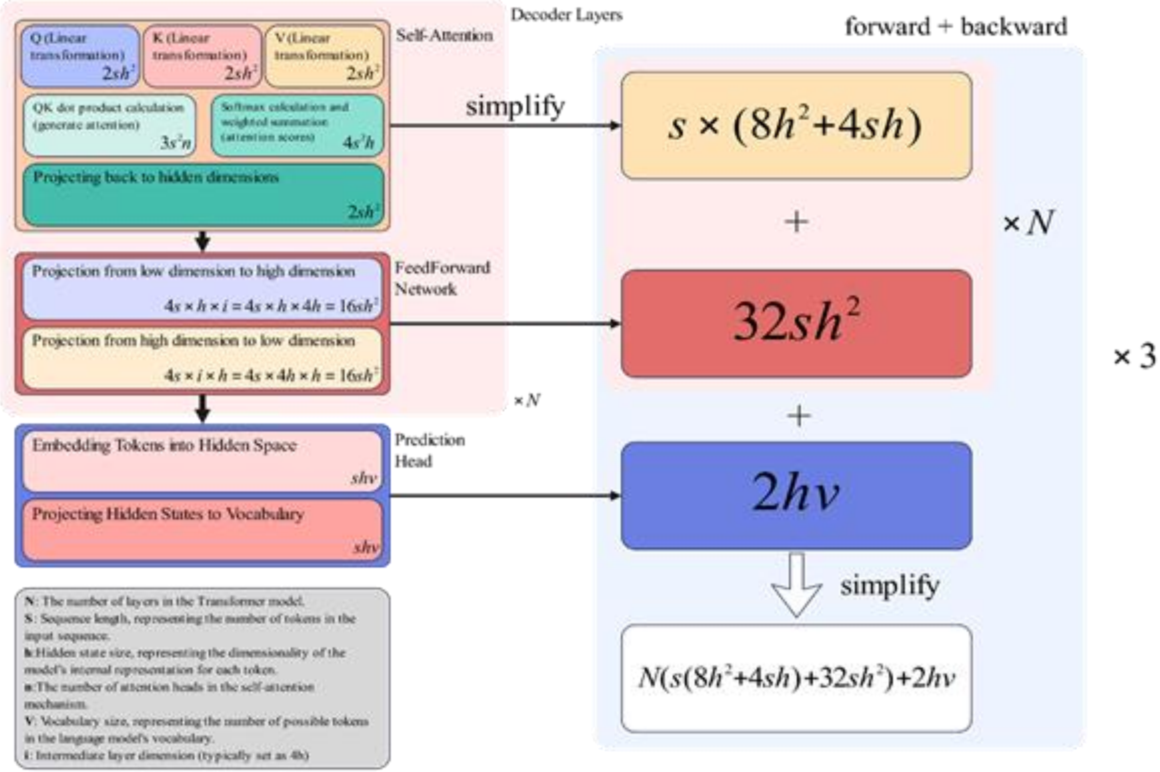
≈ 63 TFLOPs

Flops Distribution

Training Computational Costs Breakdown:

- **Total Training TeraFLOPs:** 192.17 TFLOPs
- **FLOP Distribution by Layer:**
  - **Embedding Layer:** 1.676%
  - **Normalization:** 0.007%
  - **Residual:** 0.003%
  - **Attention:** 41.276%
  - **MLP (Multi-Layer Perceptron):** 55.361%
  - **Linear:** 1.676%

Scaling Up: Where is the Potential Bottleneck?


Decoder Layers
Self-Attention
Q (Linear transformation) $2sh^2$
K (Linear transformation) $2sh^2$
V (Linear transformation) $2sh^2$
QK dot product calculation (generate attention) $3s^2n$
Softmax calculation and weighted summation (attention scores) $4s^2h$
Projecting back to hidden dimensions $2sh^2$
FeedForward Network
Projection from low dimension to high dimension $4s \times h \times i = 4s \times h \times 4h = 16sh^2$
Projection from high dimension to low dimension $4s \times i \times h = 4s \times 4h \times h = 16sh^2$
$\times N$
Prediction Head
Embedding Tokens into Hidden Space shv
Projecting Hidden States to Vocabulary shv
simplify
forward + backward
$s \times (8h^2+4sh)$
+
$32sh^2$
+
$2hv$
$\times N$
$\times 3$
simplify
$N(s(8h^2+4sh)+32sh^2)+2hv$

N: The number of layers in the Transformer model.

S: Sequence length, representing the number of tokens in the input sequence.

h:Hidden state size, representing the dimensionality of the model's internal representation for each token.

n:The number of attention heads in the self-attention mechanism.

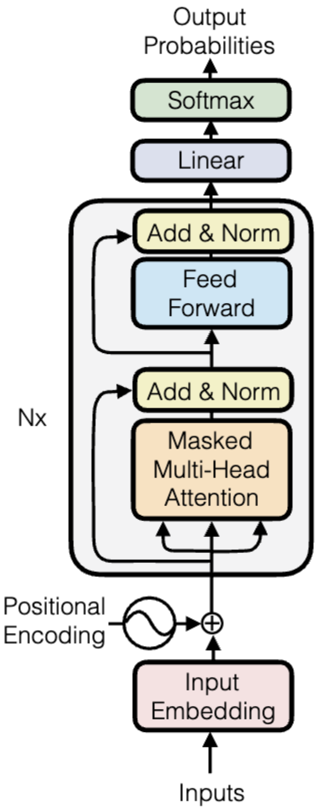
V: Vocabulary size, representing the number of possible tokens in the language model's vocabulary.

i: Intermediate layer dimension (typically set as 4h)

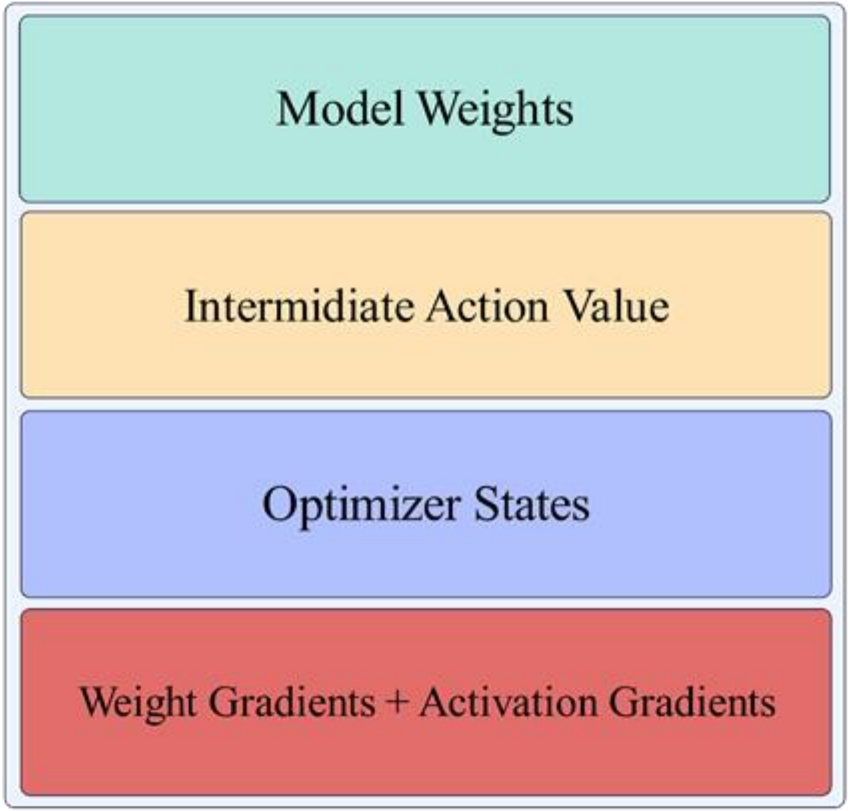
Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?



Composition of Memory Usage (Training)



Model Weights

Intermidiate Action Value

Optimizer States

Weight Gradients + Activation Gradients

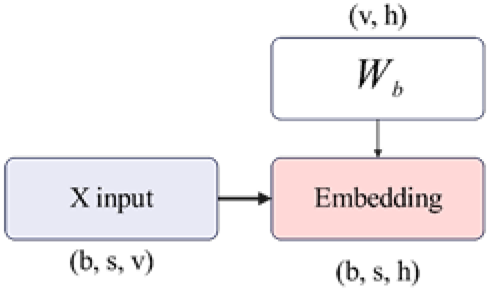
Llama2-7b Mix Precision(16bit-32bit)

X input

(b, s, v)

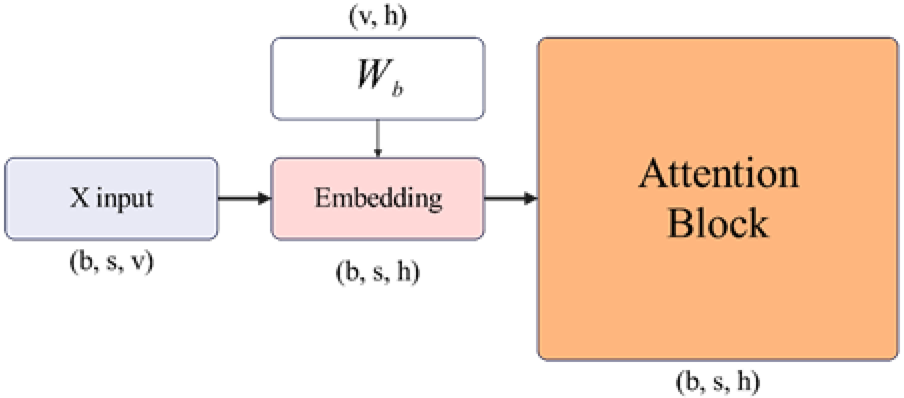
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



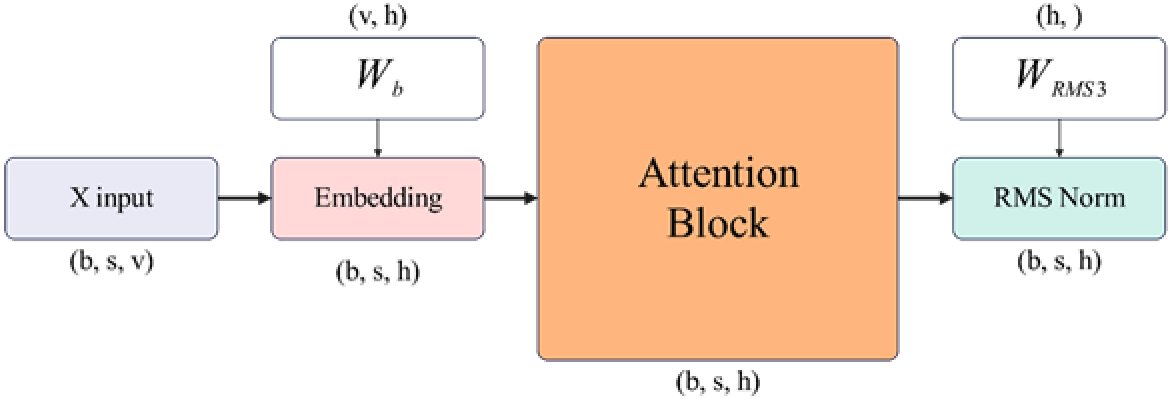
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



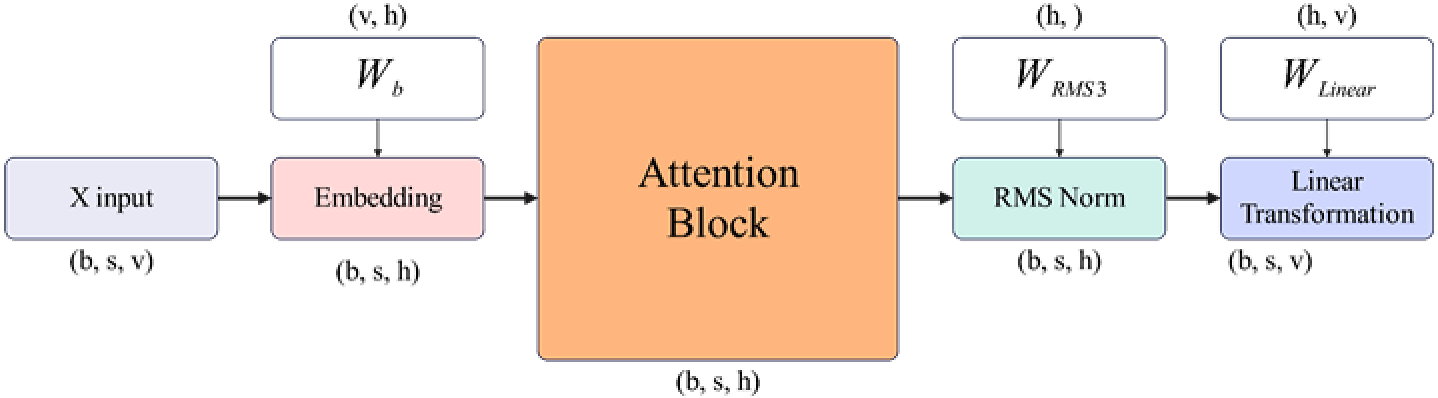
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



(v, h)

W_b

X input

(b, s, v)

Embedding

(b, s, h)

Attention Block

(b, s, h)

(h,)

$W_{RMS\,3}$

RMS Norm

(b, s, h)

(h, v)

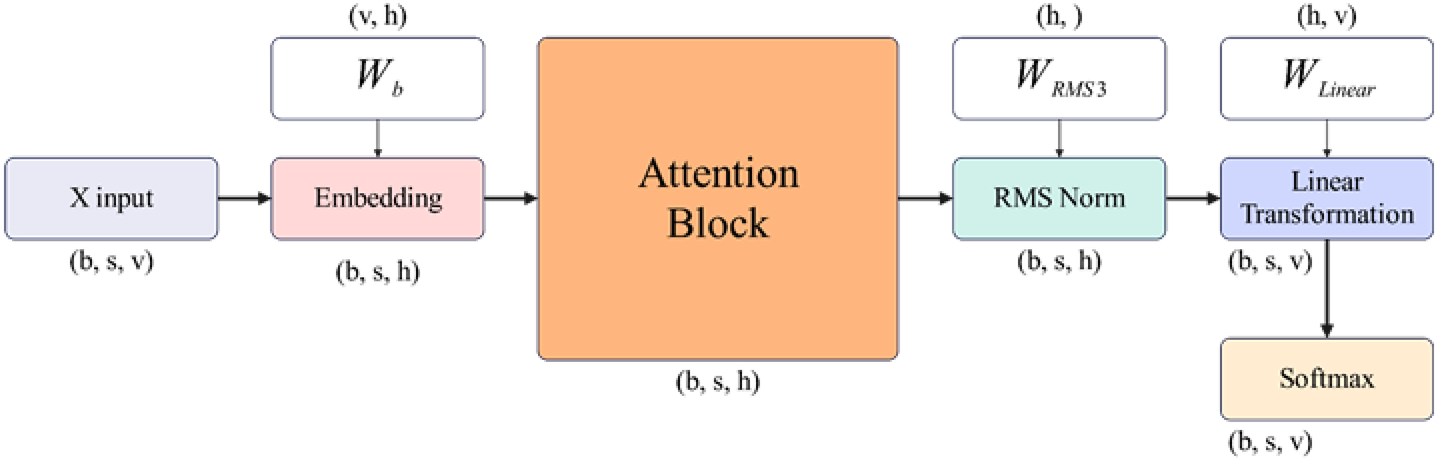
W_{Linear}

Linear Transformation

(b, s, v)

b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



(v, h)

W_b

(h,)

W_{RMS3}

(h, v)

W_{Linear}

X input

(b, s, v)

Embedding

(b, s, h)

Attention Block

(b, s, h)

RMS Norm

(b, s, h)

Linear Transformation

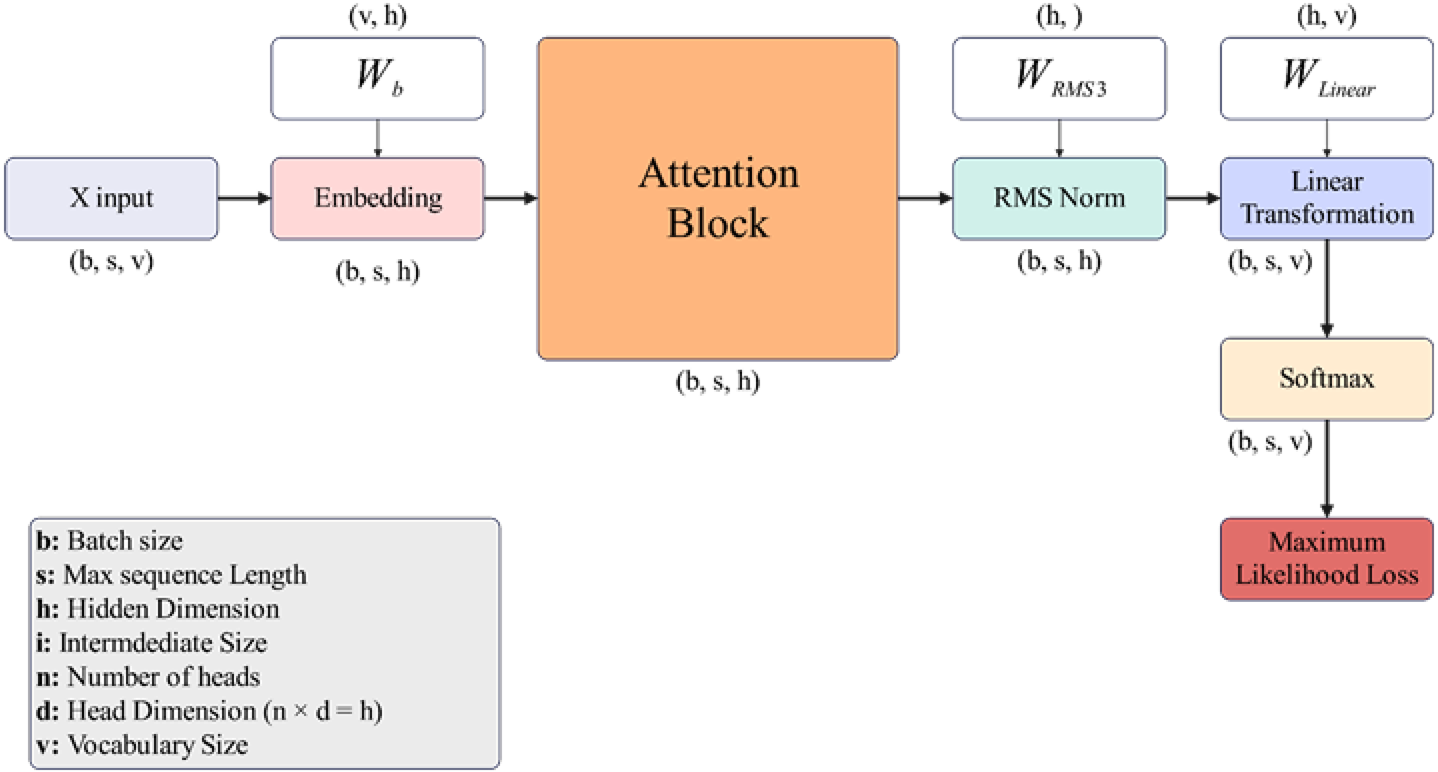
(b, s, v)

Softmax

(b, s, v)

b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)

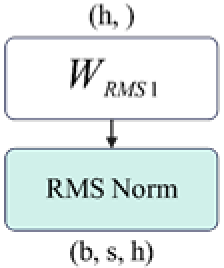


b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)

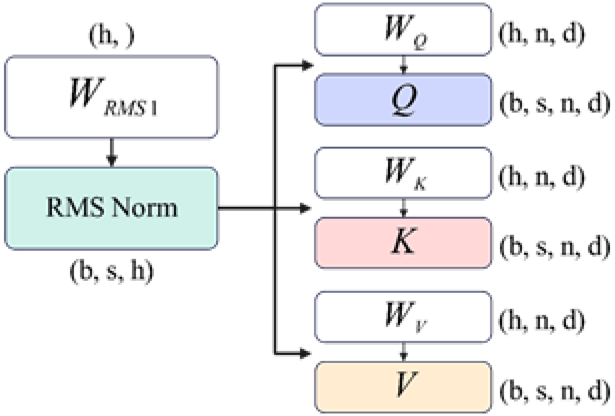
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



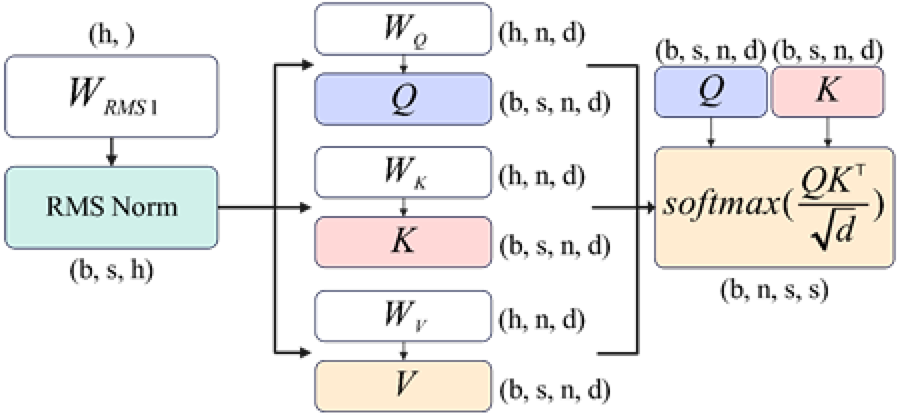
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



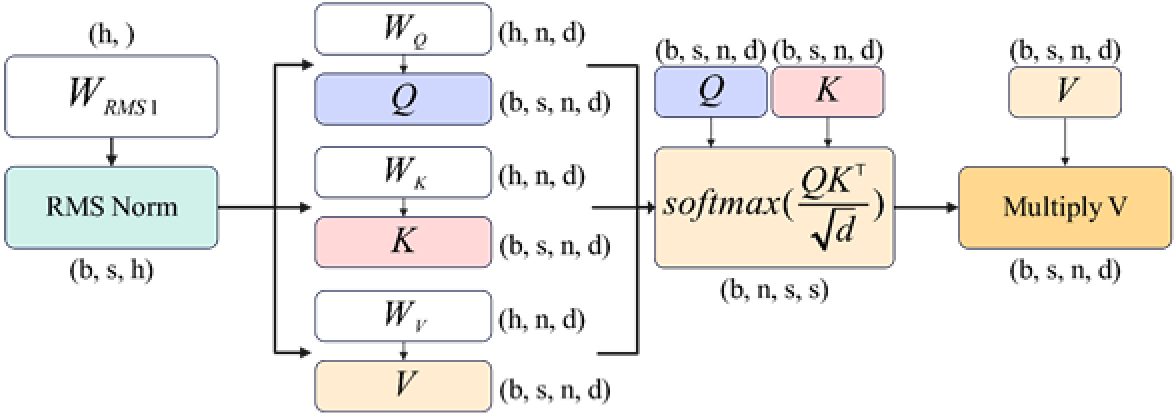
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



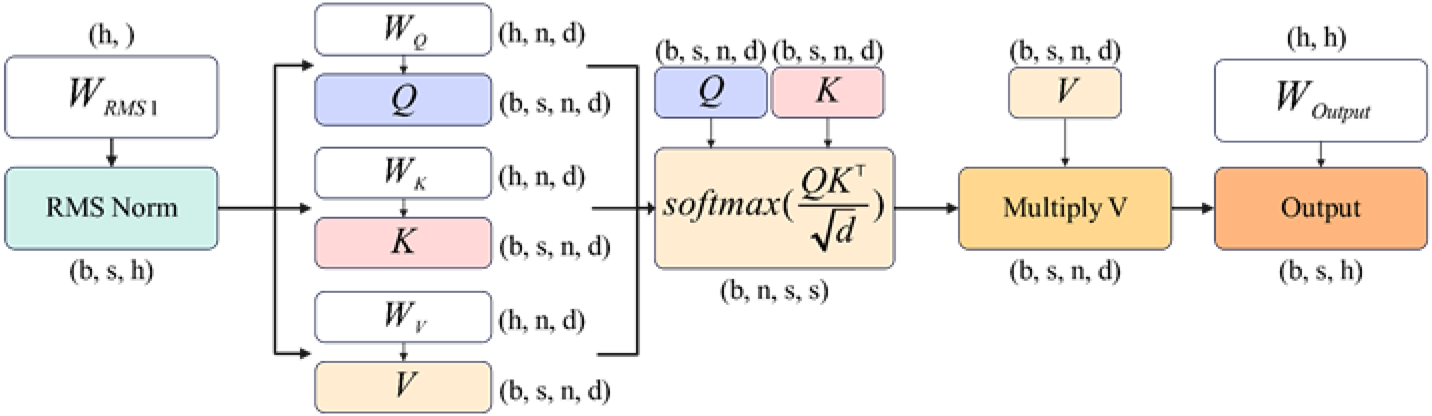
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)

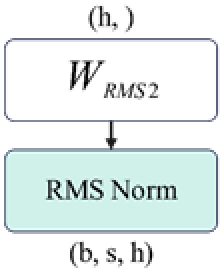


b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)

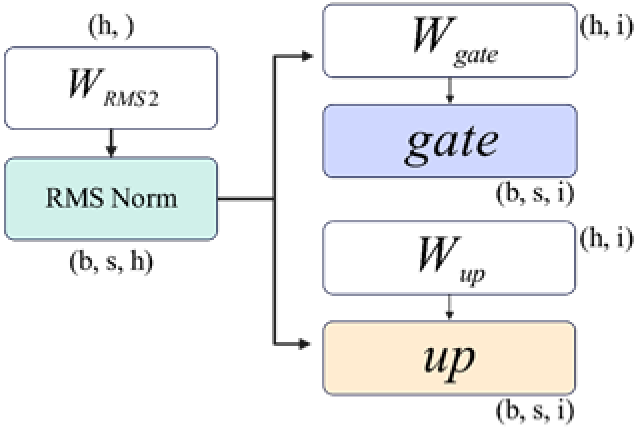
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



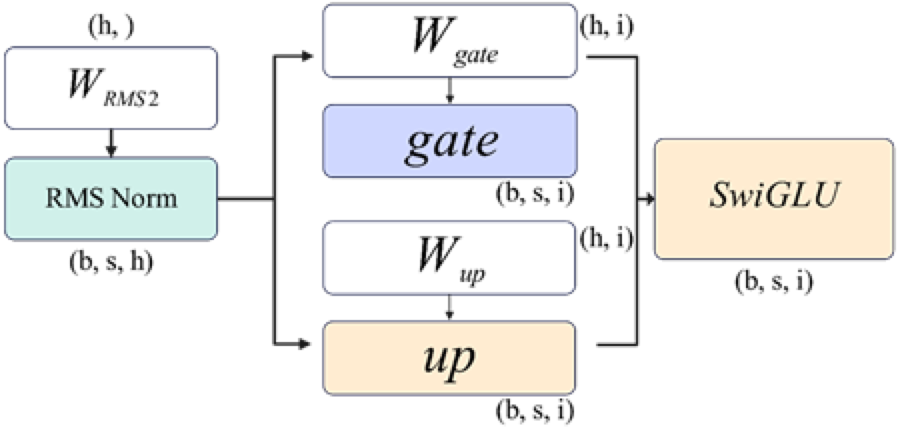
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



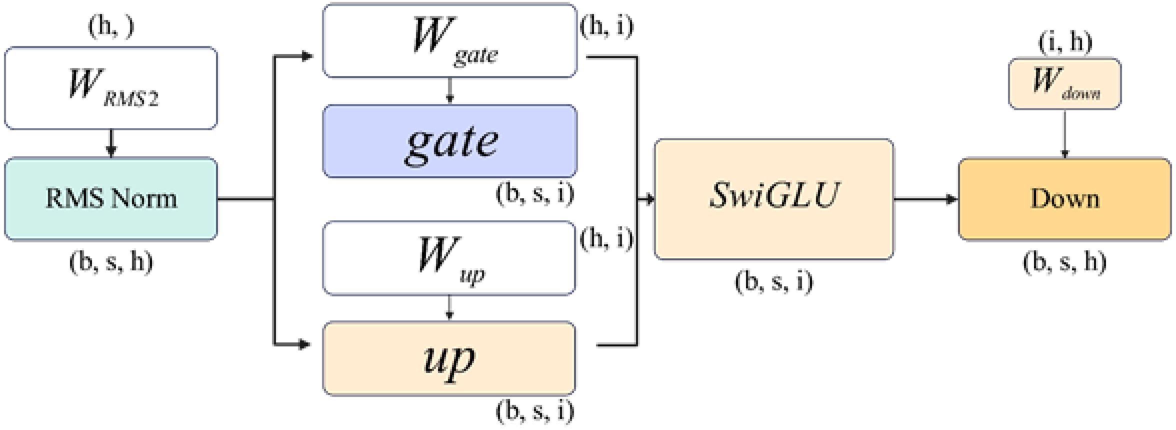
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



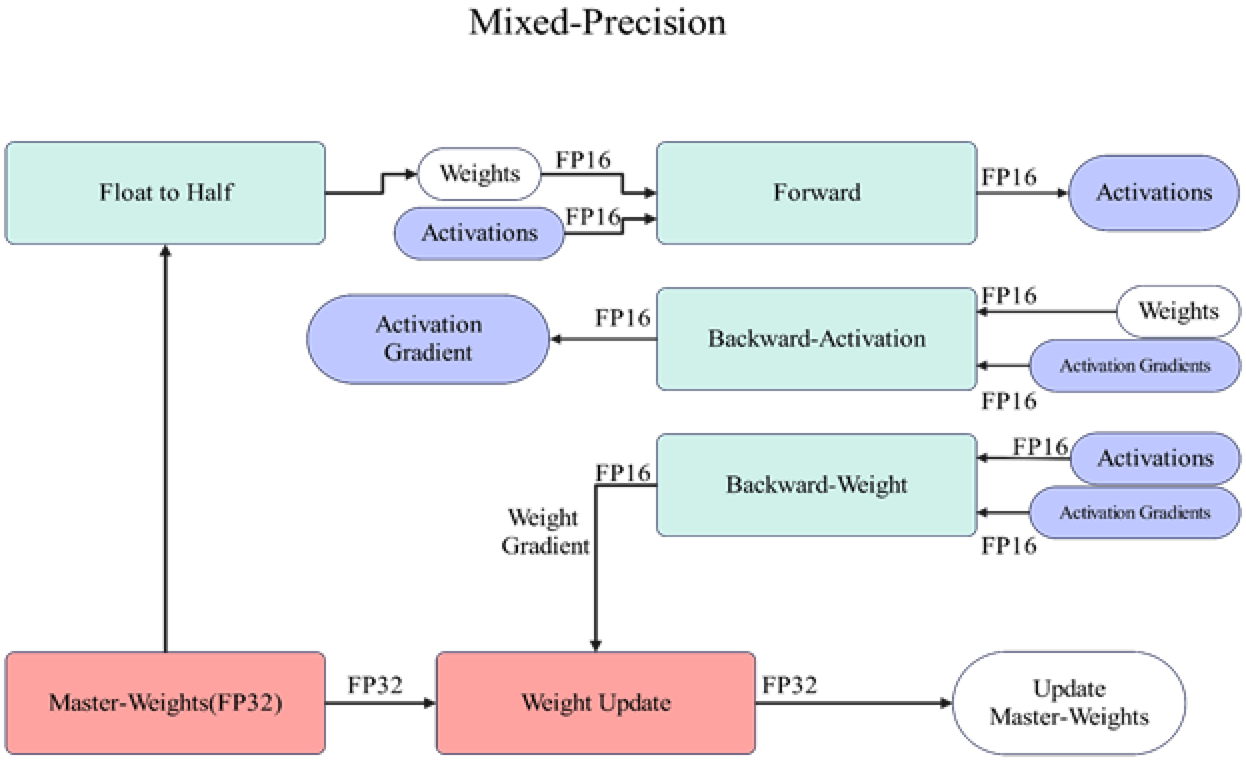
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Optimizer States: 16M



Large Language Models

- Transformers, Attentions
- **Scaling Law**
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention
  - Long context, parallelism
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

Some Observations

- compute is a function of: h, i, b
- #parameter is a function of: h, i
- Hence: compute correlates with #parameters
  - more parameters, more compute
  - more data, more compute (of course)
- Problem: we have limited compute ($)
- how should we allocate our limited resources:
  - Train models longer vs train bigger models?
  - Collect more data vs get more GPUs?

Motivation of Scaling Laws

- We want to know:
  - how large a model should we train…
  - How many data should we use…
  - To achieve a given performance…
  - Subject to a compute budget ($)?

How do we do that in traditional ML: data scaling law

Input: $x_1 \ \ldots x_n \ \sim N(\mu, \sigma^2)$

Task: estimate the average as $\hat{\mu} = \frac{\sum_i x_i}{n}$

What's the error? By standard arguments..

$$\mathrm{E}[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$$

This is a scaling law!!

$$\log(Error) = -\log n + 2 \log \sigma$$

More generally, any polynomial rate $1/n^{\alpha}$ is a scaling law

- Can we do this for transformers LLMs?
- **Unfortunately NO**

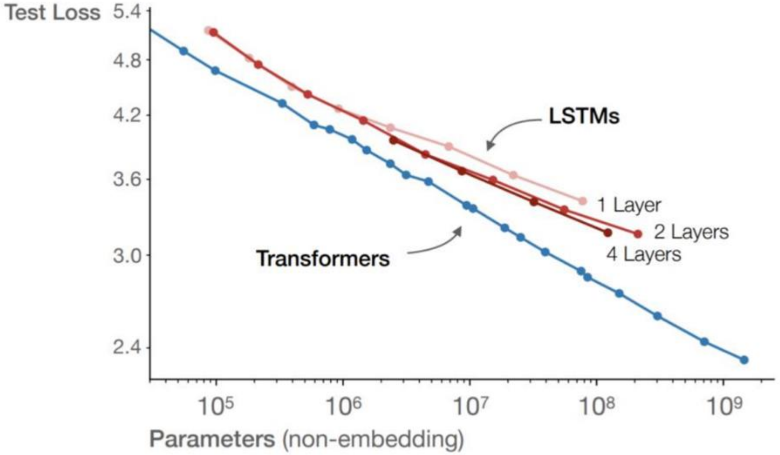
Think in this way

Mathematics vs.

Physics

Transformers vs LSTMs

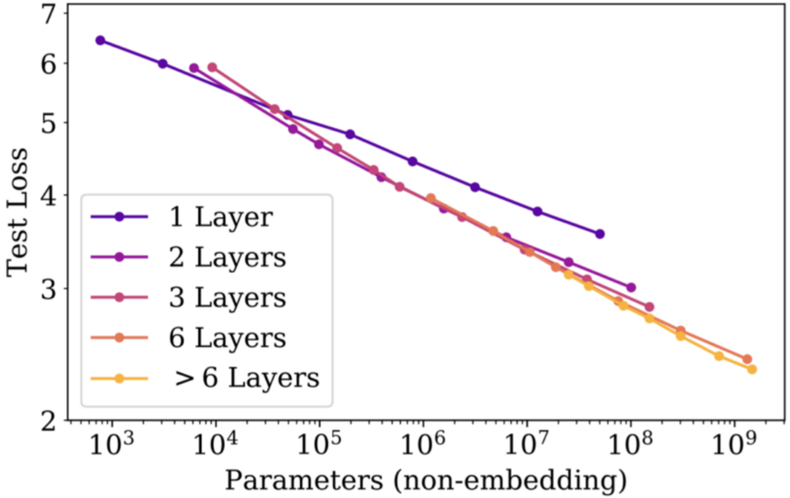
- Q: Are transformers better than LSTMs?
  - Brute force way: spend tens of millions to train a LSTM GPT-3
- Scaling law way:



[Kaplan+ 2021]

Number of Layers

- Does depth or width make a huge difference?
  - 1 vs 2 layers makes a huge difference.
  - More layers have diminishing returns below 107 params



The Scaling law way: Physics Way

- Approach:
  - Train a few smaller models
  - Establish a scaling law (LSTM vs. transformers)
  - Select optimal hyperparam based on the scaling law prediction.

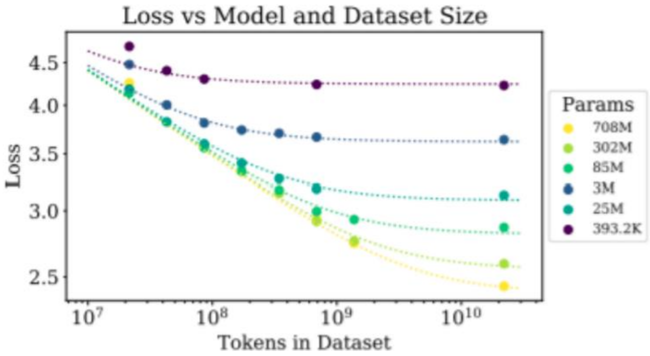
- Rationale
  - The effect of hyperparameters on big LMs can be predicted before training!
    - Optimizer choice
    - Model Depth
    - Arechitecture choice

Back to our problem:

- how large a model should we train…
- How many data should we use…
- To achieve a given performance…
- Subject to a compute budget?

- Approach: estimate a law between model size data joint scaling

Model size data joint scaling



- Do we need more data or bigger model
  - Clearly, lots of data is wasted on small models
- Joint data-model scaling laws describe ho the two relate

From Rosenfeld+ 2020,

$$Error = n^{-\alpha} + m^{-\beta} + C$$

From Kaplan+ 2021

$$Error = [m^{-\alpha} + n^{-1}]^{\beta}$$

Provides surprisingly good fits to model-data joint error.

Compute Trade-offs

- Q: what about other resources? Compute vs. performance?

- For a fixed compute budget…
  - Big models that's undertrained vs small model that's well trained?
  - Solving the following optimization?

$$N_{opt}(C), D_{opt}(C) = \underset{N,D \text{ s.t. } \text{FLOPs}(N,D)=C}{\text{argmin}} L(N, D).$$

Approach: empirical scaling law

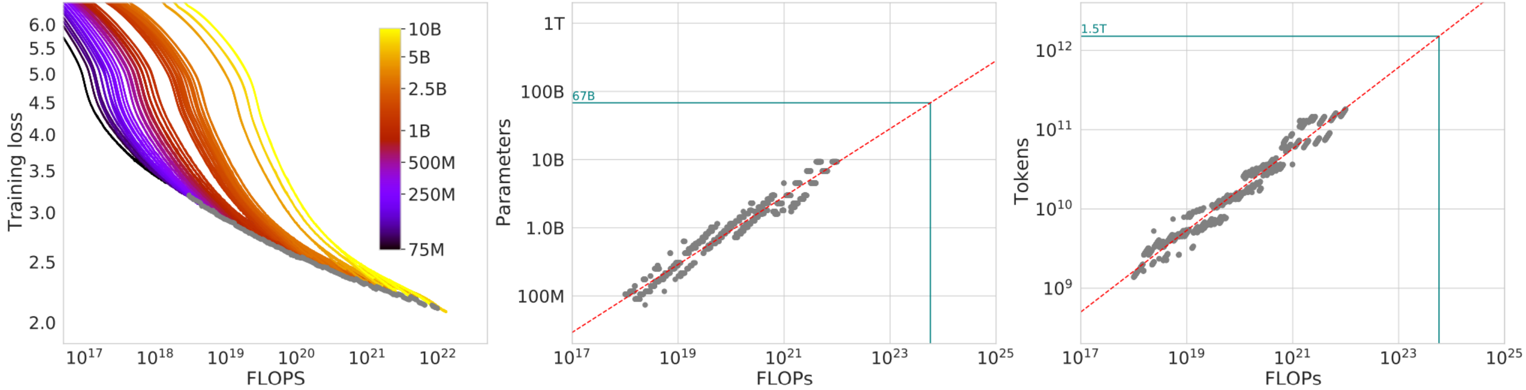


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Today's SoTA Law

$$L(N, D) = \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.29}} + 1.69$$

Summary

- Scaling law: the physics of ML
- Scaling law marks a new era of ML research:
  - Rigorous theoretical analysis -> empirical laws
  - Exploration of different model architectures -> Scaling transformers
  - Due to scaling law: ML systems become essential

PA3: Q3

You already know:

- How to estimate the number of parameters of an LLM?
- How to estimate the flops needed to train an LLM?
- How to estimate the memory needed to train a transformer?

- We will give you a scaling law and compute budget
  - Task: design your optimal LLM

Next Lecture: What is MoE

- Superficially: experts
- Essentially: a model with a better scaling law.