DSC-204A: Scalable Data Systems, Fall 2025

LLM, attention

Lecturer: Hao Zhang *Scribe: Jiayue Yuan, Manikya Bardhan, Zach Liu*

1 ML System View & LLM Focus

- ML system: data + model composed of math primitives (mainly matmuls) arranged as a data-flow graph.
- Autodiff generates backward pass; plus optimizer updates ⇒ huge computation graph.
- Systems goal: run this graph efficiently on large accelerator clusters (GPUs, etc.).

2 Language Modeling as Next-Token Prediction

- Autoregressive LM models a token sequence $x_{1:T}$ via

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

- Each factor is a next-token prediction: given a prefix, output a distribution over the vocabulary.
- Intuition: for "San Diego has very nice _" the model should put high mass on "weather" and very low on "snow".
- Training objective: maximize log-likelihood (sum of log probabilities of the true next tokens).
- Conceptually: generic sequence model f_θ takes $x_{1:t}$ and outputs the distribution for x_{t+1}.

3 Attention and Self-Attention

3.1 Basic attention

- Given hidden states $x_1, \ldots, x_T$, attention computes a weighted combination:

$$h = \sum_{i=1}^{T} s_i x_i,$$

  where s_i are attention scores / weights.

- s_i measures how relevant position i is to the current output.

3.2 Self-attention

- For self-attention, we derive queries, keys, values from the same input:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

- Core operation (single layer):

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V.$$

- Row-wise softmax over $QK^\top$ gives attention weights; multiplication with V aggregates values.

3.3 Multi-head attention

- Multiple heads compute attention in parallel with different projections.
- Each head:

$$H^{(j)} = \mathrm{Attn}(Q^{(j)}, K^{(j)}, V^{(j)}).$$

- Heads are concatenated and linearly projected back to model dimension.
- Variants like GQA share keys/values across groups of heads to reduce cost.

3.4 Causal masking

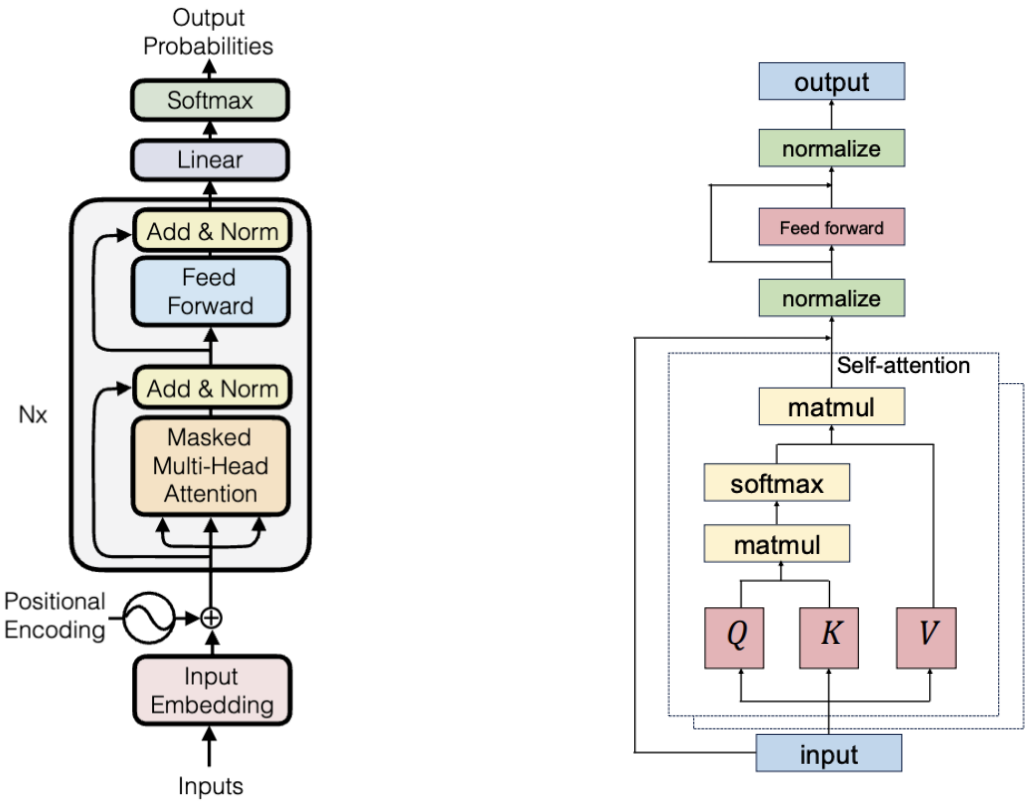
- For autoregressive LMs, token t must not see future tokens ($> t$).
- Use an upper-triangular mask M with $-\infty$ for disallowed locations:

$$\mathrm{MaskedAttn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}} + M\right) V.$$

- After softmax, masked entries have weight 0, enforcing causality.

4 Transformer Decoder Block

- Input: hidden states X from previous layer.
- Steps in a decoder block:
  1. Project to Q, K, V; apply **masked multi-head self-attention**.
  2. Add residual and normalize (LayerNorm or RMSNorm).
  3. Apply position-wise MLP (two linear layers + nonlinearity).
  4. Add residual and normalize again.
- Stack many decoder blocks to form a full LLM.



(a) Transformer architecture.

(b) Transformer decoder block.

Figure 1: High-level Transformer model and a single decoder block.

5 Full LLM Architecture

- **Embeddings:** token embeddings + positional / rotary embeddings.
- **Core:** N Transformer decoder layers (masked self-attention + MLP + residual + norm).
- **Output head:** final linear layer and softmax over vocabulary.
- **Training:** cross-entropy loss on next-token predictions over the entire sequence.
- **Inference:** given a prompt, repeatedly:
  1. run through the stack,
  2. read out next-token distribution,
  3. sample / take argmax,
  4. append token and continue.

6 Compute View: Where the Cost Is

- **Heavy ops (GPU-dominant):**

  - Self-attention: matmuls for Q, K, V, $QK^\top$, and attention-weighted V.
  - MLP: large dense matrix multiplications.
- **Light ops:**
  - Residual adds, layer norm / RMSNorm, nonlinearities (ReLU/SiLU).
  - Embedding lookups and final softmax / loss.
- Systems work focuses on optimizing attention and MLP (kernels, parallelism, memory).

7 Original Transformer vs. Modern LLMs

Component	Original Transformer	Modern LLMs (e.g., LLaMA)
Norm position	Post-norm	Pre-norm
Norm type	LayerNorm	RMSNorm
Nonlinearity	ReLU	SiLU / other newer variants
Positional encoding	Sinusoidal	Rotary embeddings (RoPE)

Table 1: Architectural differences between the original Transformer and modern decoder-only LLMs.

These choices affect stability and scalability but not the core data-flow (decoder blocks with attention + MLP).

7.1 Why Decoder-Only Won

In the original "Attention Is All You Need" paper, two variants were proposed: encoder and decoder. Over time, the decoder architecture has become dominant for language models:

- **Encoder models (e.g., BERT):** Used to be very successful for tasks like classification and understanding. However, encoders are harder to scale up.
- **Decoder models:** Easier to scale and better suited for generative tasks. Modern LLMs (GPT, LLaMA, etc.) all use decoder-only architectures.

The decoder's autoregressive nature aligns naturally with next-token prediction, and its simpler training objective (predicting the next token) scales more effectively with compute and data.

8 Training vs. Inference in LLMs

It was highlighted that a critical distinction in understanding LLM systems is the difference between training and inference, as they have fundamentally different computational profiles.

8.1 Training LLMs

- **Sequences are known a priori:** During training, complete sequences from the training data are available.
- **Parallel loss computation:** For each position t, the model looks at tokens $[1, 2, \ldots, t-1]$ to predict token t, then calculates the loss at position t.
- **Masking enables parallelism:** Because all tokens are known, computation can be parallelized across all token positions in one forward pass by applying causal masking.

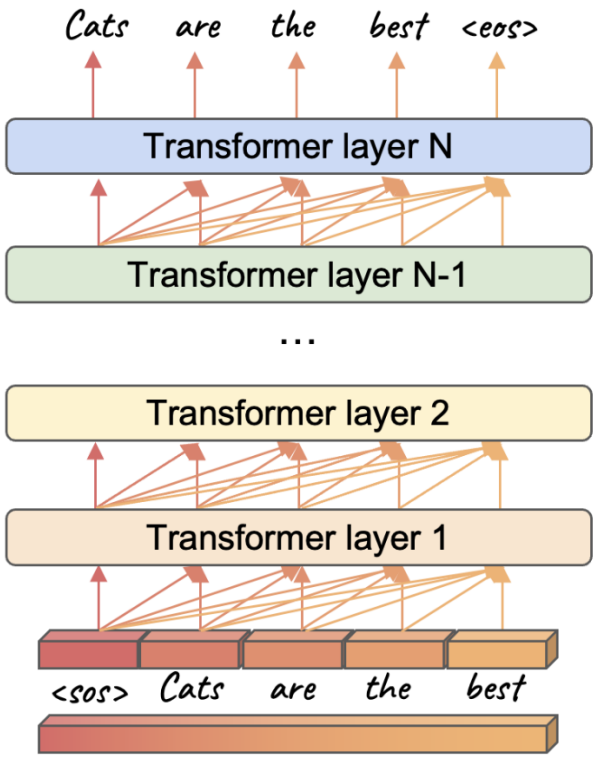


Figure 2: Training LLMs: sequences are known a priori, allowing parallel computation across all token positions with causal masking. Each output position attends only to current and previous positions.

8.2 Autoregressive Inference

- **Future tokens are unknown:** At inference time, given a prompt, the model must predict what comes next.
- **Sequential generation:** Multiple future tokens cannot be predicted in parallel:
  1. First, predict token $k+1$ given tokens 1 through k.
  2. Once token $k+1$ is obtained, feed it back into the model.
  3. Then predict token $k+2$ given tokens 1 through $k+1$.
  4. Continue until reaching a stop condition.
- **Key difference:** It was emphasized that training can be fully parallelized across sequence positions (with masking), while inference is inherently sequential for token generation.

9 Compute, Memory, and Communication in LLMs

Three key characteristics for analyzing LLM systems from a systems perspective were introduced:

- **Compute:** How many floating-point operations (FLOPs) are needed?

- **Memory:** How many parameters does the model have? How much activation memory is needed?
- **Communication:** When the model is spread across many GPUs, how much data must be exchanged?

10 Parameter Counting for Transformer LLMs

Understanding parameter counts is essential for practical LLM work. When model releases mention "LLaMA 7B" or "GPT-5 is a 1 trillion parameter model," where do these numbers come from?

Systems perspective: The instructor noted that this material is typically not covered in machine learning courses, which focus on learning theory. This lecture examines LLMs through the lens of compute, which is essential for systems engineers working on efficient training and inference.

Understanding where parameters come from is essential for:

- **Memory estimation:** A 7B model with FP16 precision needs ≈ 14 GB just for model weights ($7 \times 10^9 \times 2$ bytes).
- **Inference planning:** Larger models require distributed inference strategies.
- **Architecture design:** Parameter distribution affects compute characteristics.

10.1 Notation

- v: vocabulary size (typically $\approx$ 30k–32k for LLaMA-style tokenizers, ≈ 100k for GPT-style)
- h: hidden size (model dimension). This is the key parameter carried across all components.
- N: number of transformer decoder layers
- i: intermediate dimension of the feed-forward/SwiGLU block

Hidden size ranges:

- Smaller models ($\sim$7B): $h \approx 4096$ (around 4k)
- Larger models ($\sim$70B+): h can grow to 8k–16k or even higher

10.2 Full Architecture with Math Equations

Figure 3 shows the complete LLM architecture with corresponding mathematical equations for each component.

10.3 Parameter Breakdown by Component

Embedding Layer:

- $W_{\text{embedding}} \in \mathbb{R}^{v \times h}$
- Parameters: $v \cdot h$

Layer Normalization (RMSNorm):

- Scaling factors: $\gamma \in \mathbb{R}^h$

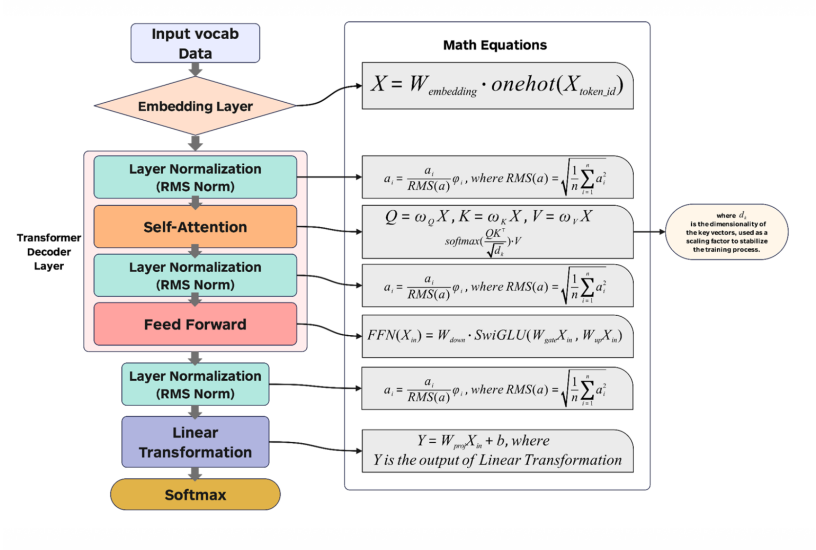


Figure 3: Complete LLM architecture showing each component and its corresponding mathematical equation.

- Parameters per norm layer: h (negligible compared to h^2 terms)

Self-Attention:

- $W_Q, W_K, W_V, W_O \in \mathbb{R}^{h \times h}$
- Parameters per layer: $4h^2$
- **Key insight:** The attention computation itself, $\text{softmax}(QK^\top/\sqrt{d}) \cdot V$, is *parameter-free*. Once Q, K, V are obtained, the attention mechanism involves no trainable parameters. This characteristic makes attention easier to optimize in terms of compute kernels.

Feed-Forward (SwiGLU):

- $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{h \times i}$, $W_{\text{down}} \in \mathbb{R}^{i \times h}$
- Parameters per layer: $3hi$
- The intermediate dimension i is typically $i \approx 2.67h$ (LLaMA) or $i = 4h$ (GPT-3)
- **Important:** This is the largest contributor to model parameters!

Scaling Laws and the Intermediate Dimension: Why does LLaMA use $i \approx 2.67h$ while GPT-3 used $4h$? This comes from **scaling law experiments**, an empirical research methodology where teams train many models with different architecture hyperparameters (scaling factors from 2.0 to 4.0, hidden sizes, etc.) and observe which configuration gives the best performance for a fixed compute budget. LLaMA's 2.67 factor was determined to be optimal through such experiments. This type of research requires enormous compute resources, which is why frontier labs (Google, Meta, OpenAI) lead this work.

Output Layer:

- $W_{\text{output}} \in \mathbb{R}^{h \times v}$
- Parameters: $h \cdot v$

10.4 Total Parameter Count

$$\#\text{params} \approx \underbrace{v \cdot h}_{\text{embedding}} + N \times \left(\underbrace{4h^2}_{\text{attention}} + \underbrace{3hi}_{\text{FFN}} \right) + \underbrace{v \cdot h}_{\text{output}}$$

11 Feed-Forward SwiGLU

Modern LLMs like LLaMA use SwiGLU instead of the simple ReLU-based FFN from the original Transformer.

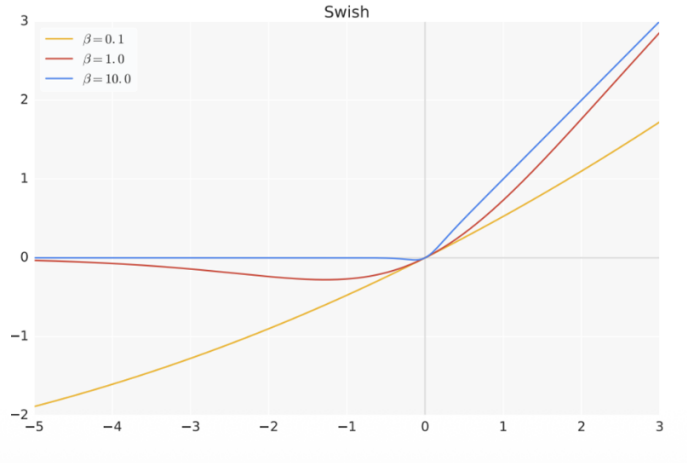


Figure 4: Feed Forward SwiGLU with Swish activation function.

SwiGLU Formula:

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

Swish Activation:

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

where σ is the sigmoid function.

Why SwiGLU?

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

12 Summary: Parameter Shapes

13 Mixture of Experts (MoE)

The parameter counting above describes **dense models**. However, modern frontier models achieve trillion-parameter scales using **Mixture of Experts (MoE)**.

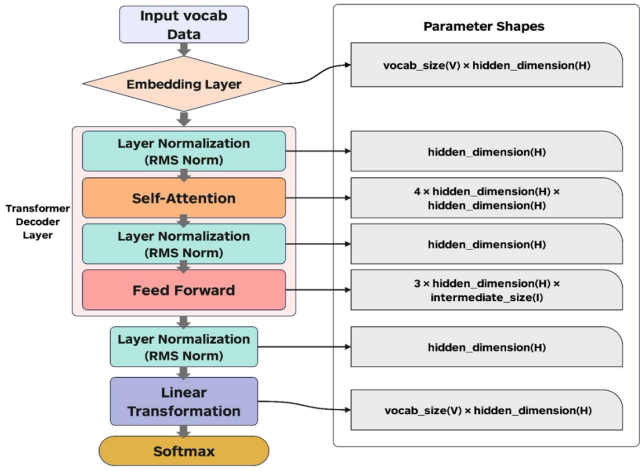


Figure 5: Summary of parameter shapes for each component in a LLaMA-style architecture.

13.1 Dense vs. MoE Architecture

In dense models, every token passes through the same feed-forward network. In MoE models:

- The feed-forward block is replicated E times, where E is the number of experts.
- A **router** selects which expert(s) to use for each token.
- During computation, typically only 1–2 experts are activated per token.

13.2 Parameter Inflation Without Compute Inflation

This is the key insight behind MoE:

$$\text{MoE Parameters} = \text{Dense Parameters} + (E-1) \times \underbrace{3hi \times N}_{\text{extra FFN copies}}$$

The philosophy: According to scaling laws, models with more parameters tend to be more powerful. MoE allows inflating the parameter count dramatically while keeping the actual computation per token roughly constant (since only one expert branch is activated).

13.3 Example: How Gemini Reaches 3T+ Parameters

- Dense 7B model: $\sim$7 billion parameters
- With $E = 1000$ experts: parameters can reach into the trillions
- Gemini 3 is reported to be a 5T+ parameter model

13.4 MoE Challenges: Memory for Inference

While MoE keeps *compute* manageable, the *memory* requirement still scales with total parameters:

- A 120B MoE model (FP16): ≈ 240 GB just for weights
- This exceeds single-GPU memory, requiring distributed inference
- Frontier labs often train large MoE models, then **distill** them into smaller dense models for practical deployment

Practical tip: When inspecting model configurations, check if it's an MoE model. If so, look for the number of experts and which experts are activated per token.

14 Compute characteristics of LLM

- How to estimate the compute
  - FLOPs of matrix multiplication with shape $m \times n$ and $n \times h$ has the formula **FLOPs** $= m \times h \times (2n - 1)$
  - m and h come from the outer dimension of the calucation, where $2n$ comes from operations of multiplying and adding from the inner dimension n
- Example calcuation using Llama 7B forward pass in training
  - Given parameters batch size: b, sequence length: s, # of attention heads n, hidden state size of one head: d, hidden state size: $h = n \times d$, SwiGLU proj dim: i, vocab size: v
  - **Long context problem**: large part of FLOPs comes from quadratic growth of varying size of sequence length $\Rightarrow$ computation explodes when input is very long
  - **Total FLOPs calculation**:

$$\begin{aligned}\text{Total FLOPs} &= \text{\# of layers} \times (\text{Attention Block + SwiGLU Block}) + \text{Prediction head} \\ &= \text{\# layers} \times (6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2) + \text{\# layers} \times 6bshi + 2bshv\end{aligned}$$

- Real life training:
  - Calculate FLOPs with small sequence length, then scale to the ratio
  - FLOPs also grows quadratically with number of parameters $\Rightarrow$ 2x in model size means 4x in FLOPs
  - Around 90% of the FLOPs are contributed by attention and MLPs
- Scaling up:
  - Context length will completely dominate the calculation in extremely long context situations like agents etc.
  - Scaling up model size means growth in h, which has much slower slope compared to context length s

Component	**Parameter Shape**
Embedding Layer	$v \times h$
Per Transformer Decoder Layer ($\times N$)	
Layer Normalization (RMS Norm)	h
Self-Attention (W_Q, W_K, W_V, W_O)	$4 \times h \times h$
Layer Normalization (RMS Norm)	h
Feed Forward ($W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}$)	$3 \times h \times i$
Output	
Layer Normalization (RMS Norm)	h
Linear Transformation	$v \times h$

Table 2: Summary of parameter shapes in a LLaMA-style architecture.

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

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