

Architecture, Training, and Scaling Laws

Lecture 2

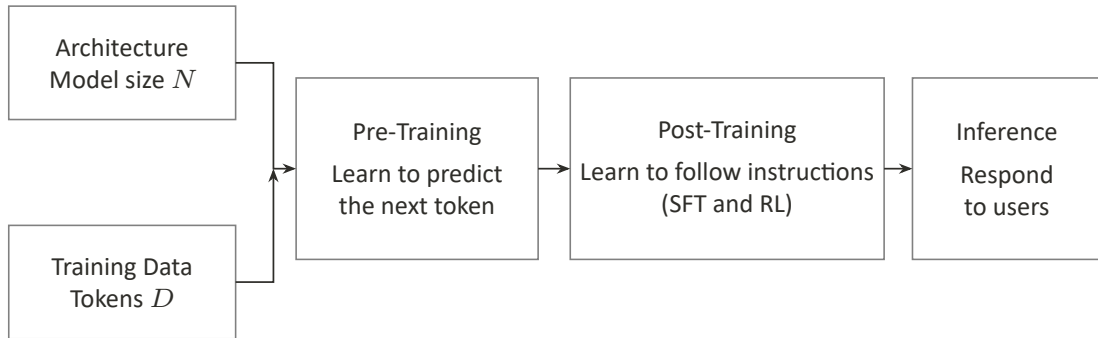
Anay Mehrotra

MS&E 319

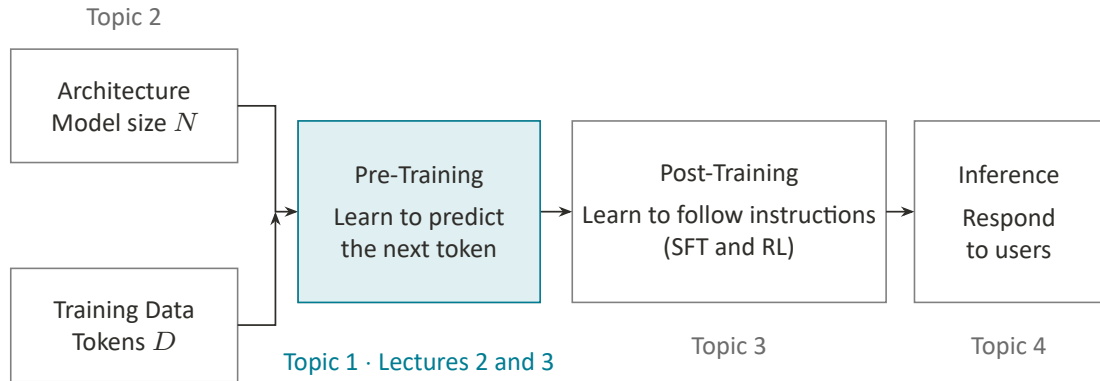
Autumn 2026



The LLM Pipeline



The LLM Pipeline



Topic 1: Innovations in Pre-Training

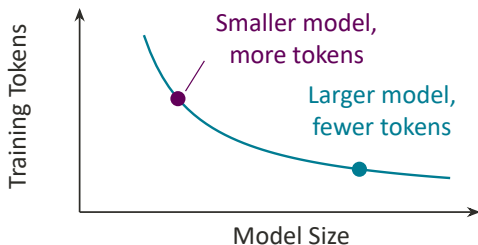
Fix training compute. How do we get the best model from it?



Topic 1: Innovations in Pre-Training

Fix training compute. How do we get the best model from it?

Lecture 2 (Today)



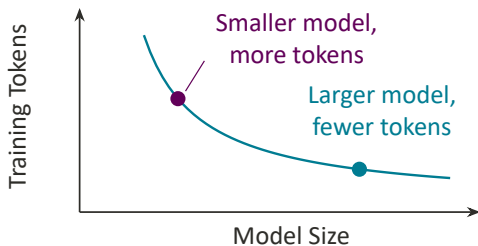
How should we split compute between model size and data?



Topic 1: Innovations in Pre-Training

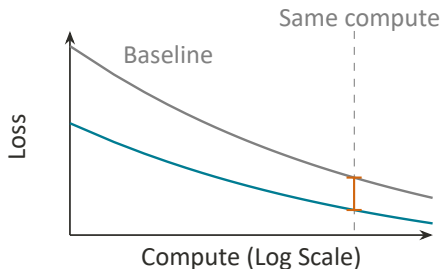
Fix training compute. How do we get the best model from it?

Lecture 2 (Today)



How should we split compute between model size and data?

Lecture 3



Which optimizer and objective make compute go further?



Today's Plan

1

LLM Architecture

What does a modern LLM with N parameters look like?

2

Training

Given D training tokens, how do we train the model?

3

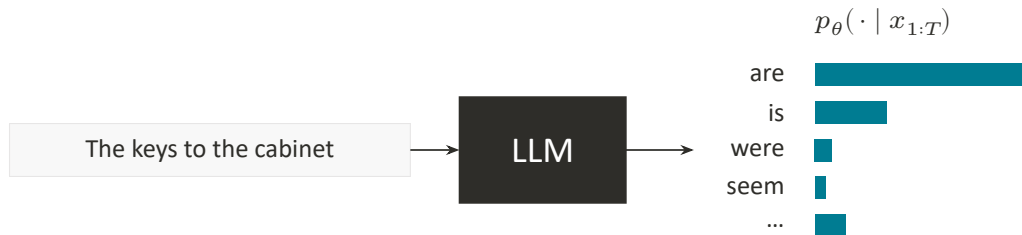
Scaling Laws

How should we split resources between N and D ?



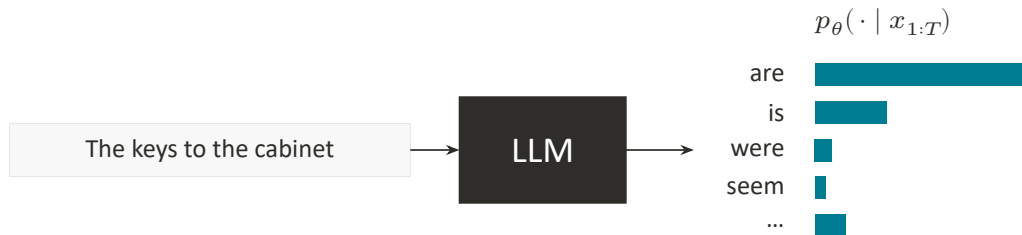
A Language Model as a Black Box

A prefix determines a distribution over the next token.



A Language Model as a Black Box

A prefix determines a distribution over the next token.

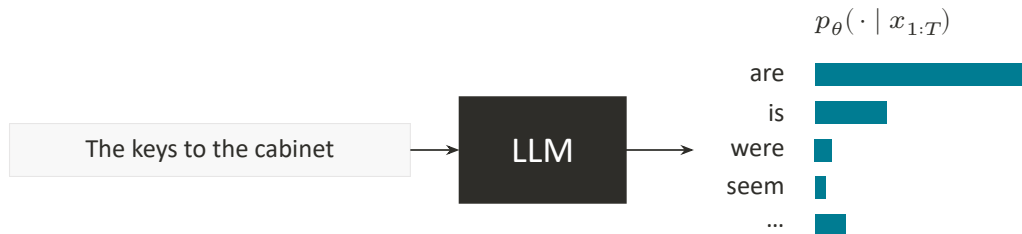


Choose a token, append it to the prefix, and repeat.



A Language Model as a Black Box

A prefix determines a distribution over the next token.



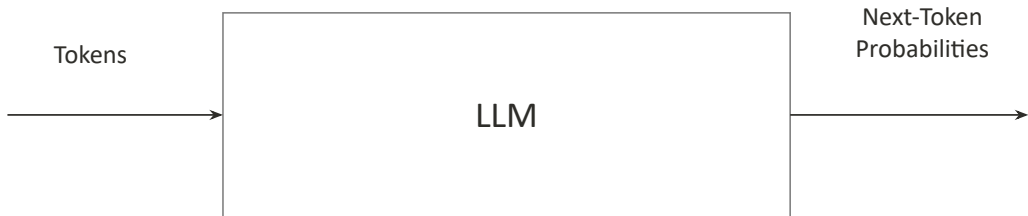
Choose a token, append it to the prefix, and repeat.

The same parameters θ are used at every step.

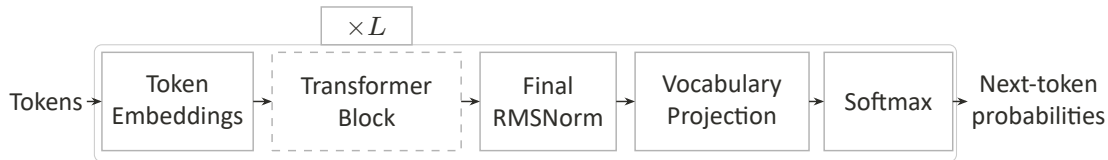


A Modern Decoder-Only Transformer

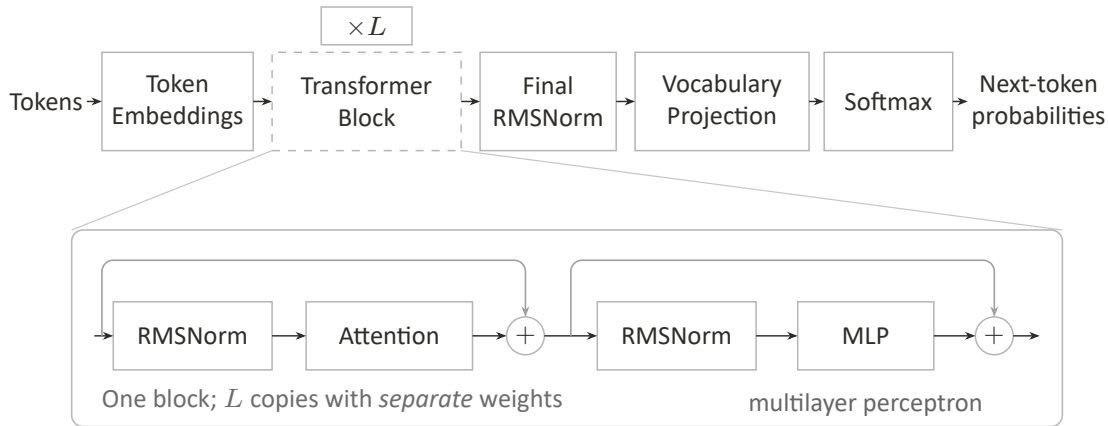
LLMs have been Transformer-based since 2017. We will review one modern decoder-only design.



A Modern Decoder-Only Transformer



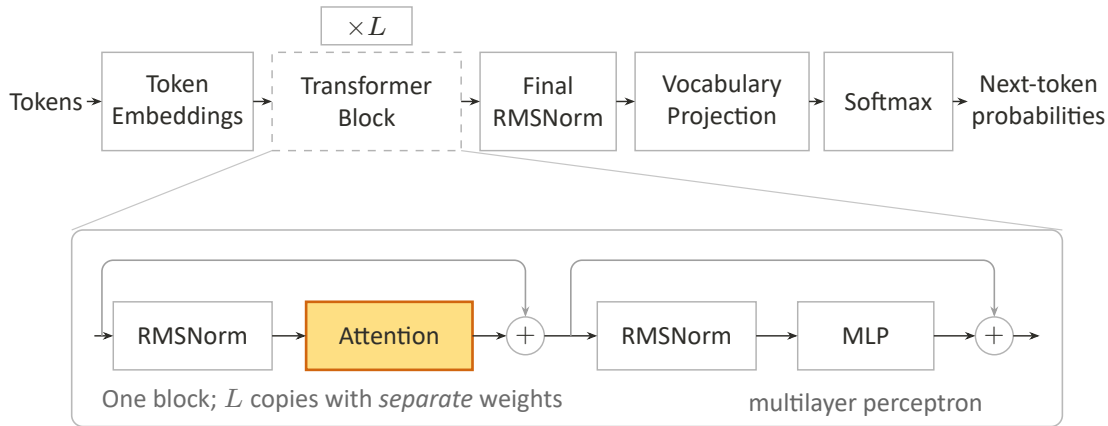
A Modern Decoder-Only Transformer



Each block combines attention, an MLP, norms, and residuals.



A Modern Decoder-Only Transformer

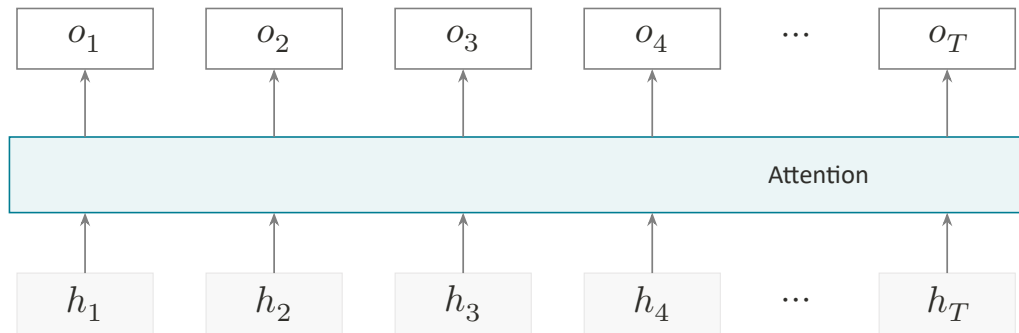


First: Attention



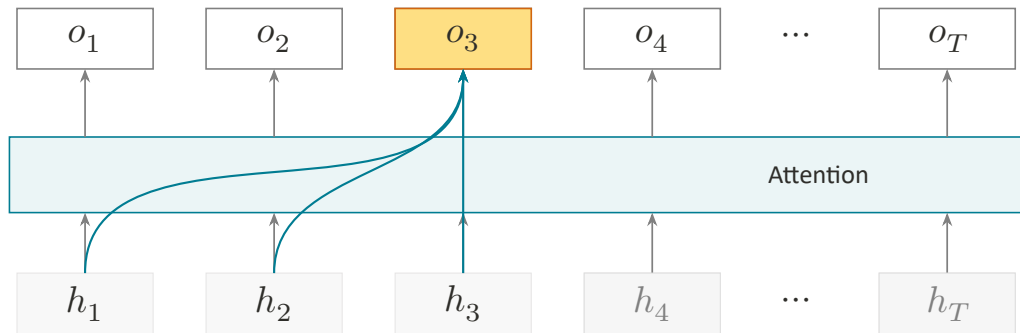
Attention: Inputs and Outputs

Attention takes T vectors and returns T vectors.



Attention: Inputs and Outputs

Attention takes T vectors and returns T vectors.

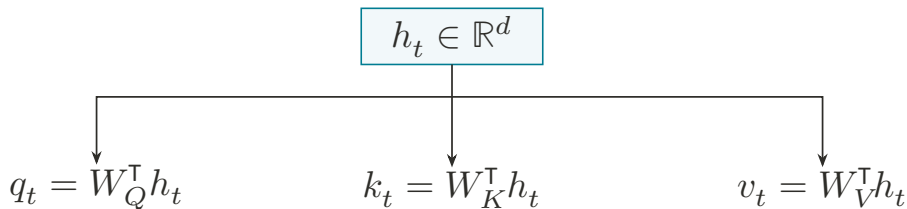


o_t can use inputs $h_1, \dots, h_t$; later inputs are “masked”.



Queries, Keys, and Values

Each hidden state h_t produces three learned projections.



“What It Looks For”

“What It Is About”

“What It Passes On”



Attention Mechanism

For each query position t :

Logits $s_{tj} = \frac{q_t^\top k_j}{\sqrt{d_k}}$



Attention Mechanism

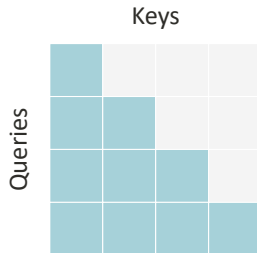
For each query position t :

Logits

$$s_{tj} = \frac{q_t^\top k_j}{\sqrt{d_k}}$$

Weights

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_{j' \leq t} e^{s_{tj'}}} \quad (j \leq t)$$



Causal mask:
 $a_{tj} = 0$ for $j > t$



Attention Mechanism

For each query position t :

Logits

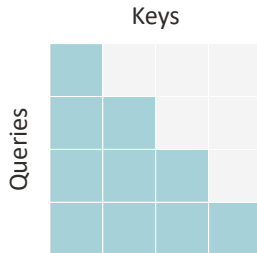
$$s_{tj} = \frac{q_t^\top k_j}{\sqrt{d_k}}$$

Weights

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_{j' \leq t} e^{s_{tj'}}} \quad (j \leq t)$$

Output

$$o_t = \sum_{j \leq t} a_{tj} v_j$$

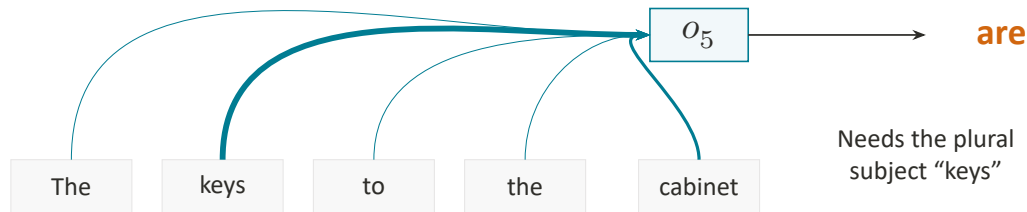


Causal mask:
 $a_{tj} = 0$ for $j > t$



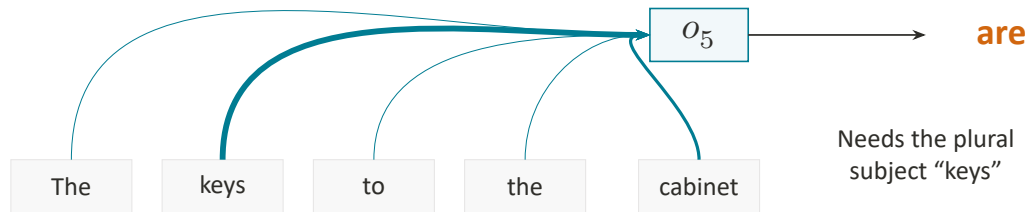
What Attention Does

Each position gathers “information” from the tokens so far and combines it into an updated vector.



What Attention Does

Each position gathers “information” from the tokens so far and combines it into an updated vector.

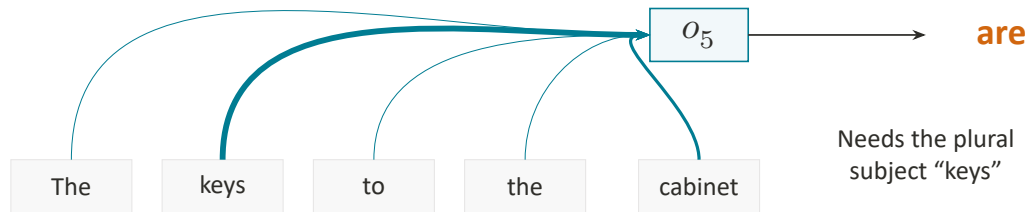


Attention is the only step where tokens exchange information.



What Attention Does

Each position gathers “information” from the tokens so far and combines it into an updated vector.

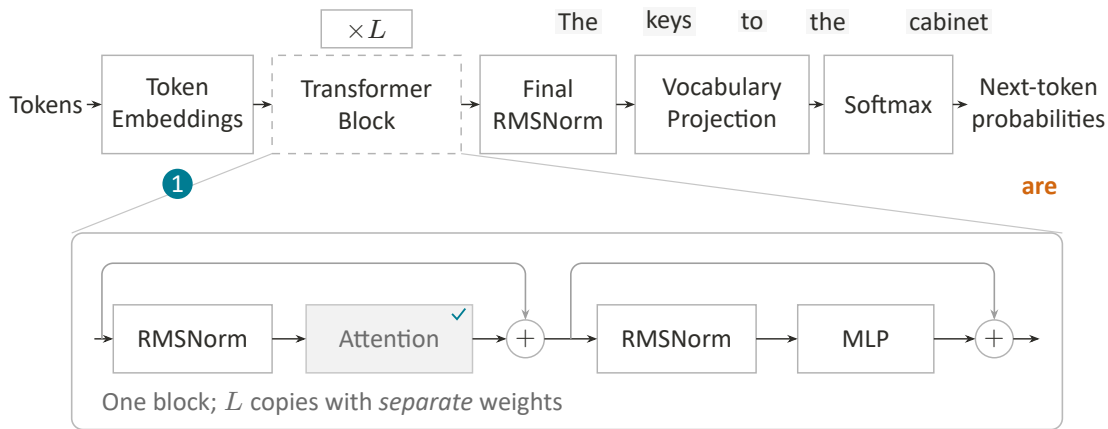


Attention is the only step where tokens exchange information.

After L blocks, the last token's vector alone predicts the next token.



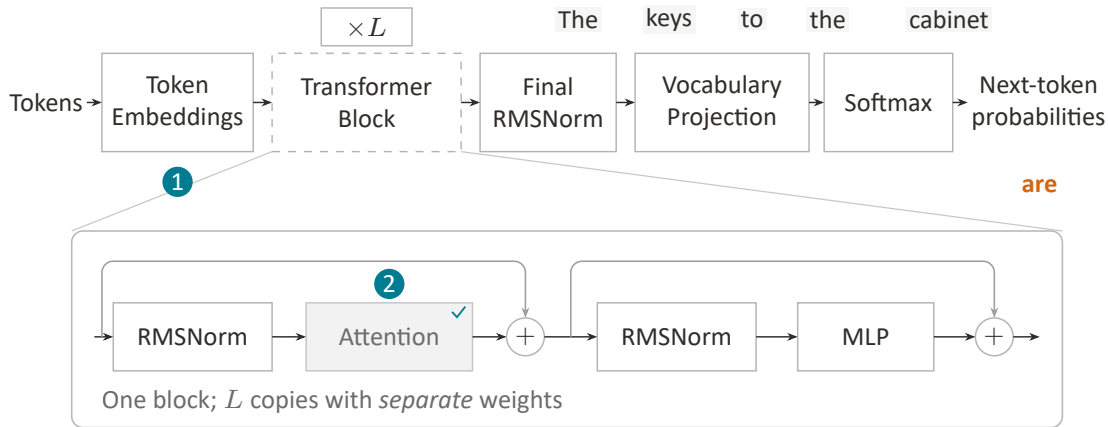
Following One Prediction



Each token becomes a vector in $\mathbb{R}^d$.



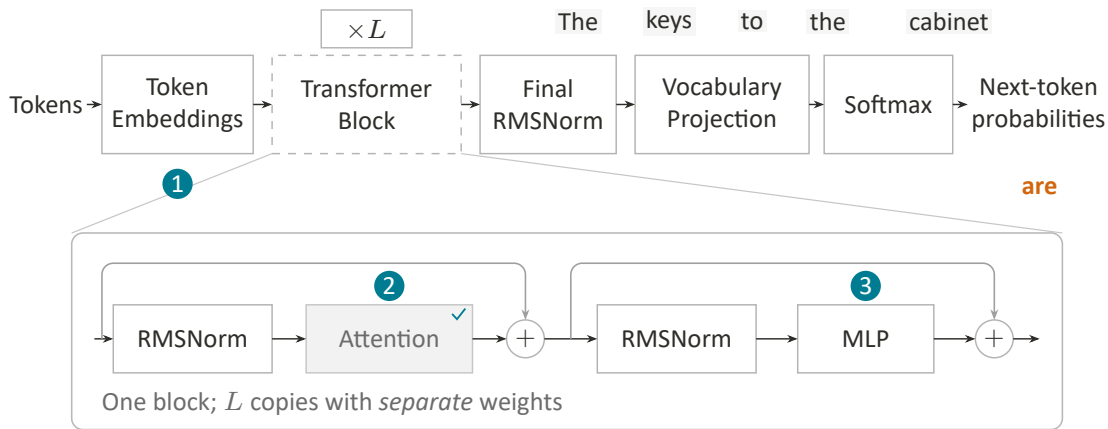
Following One Prediction



Each position reads information from its prefix.



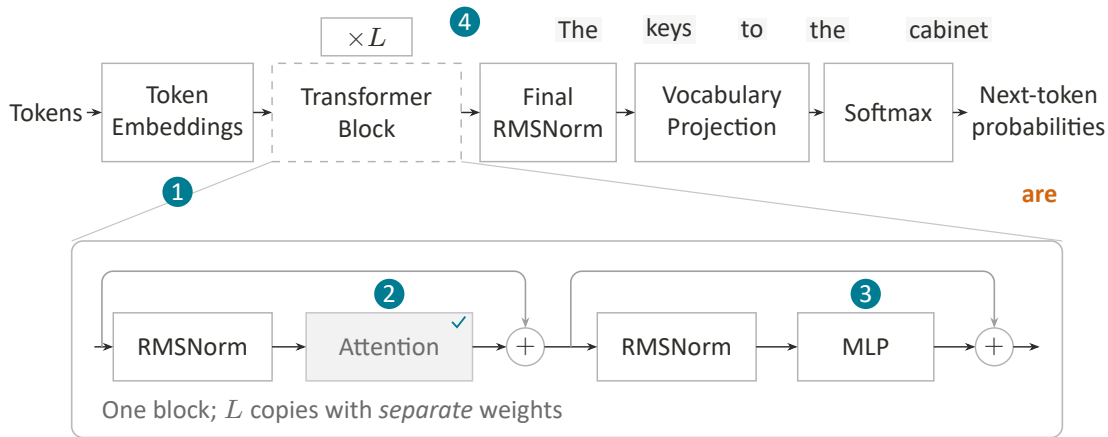
Following One Prediction



The MLP works on each token separately.



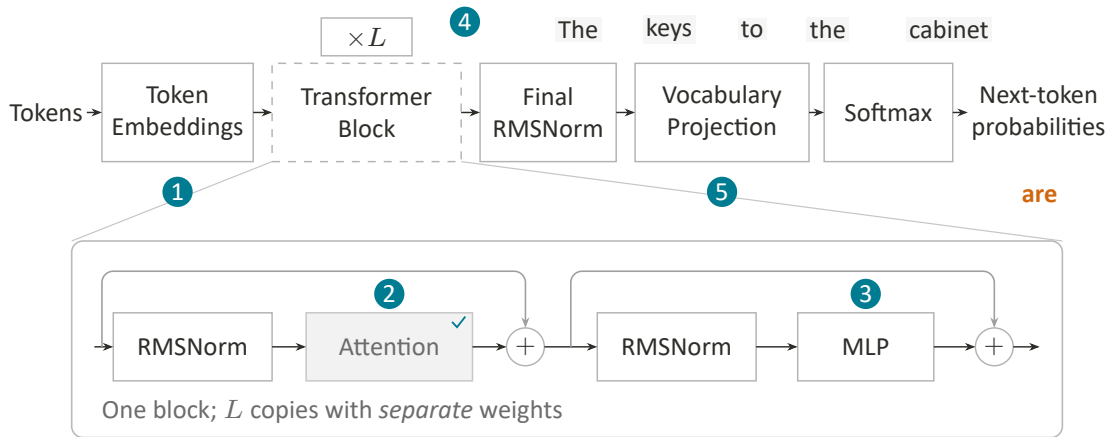
Following One Prediction



Repeat L times; every block has its own weights.



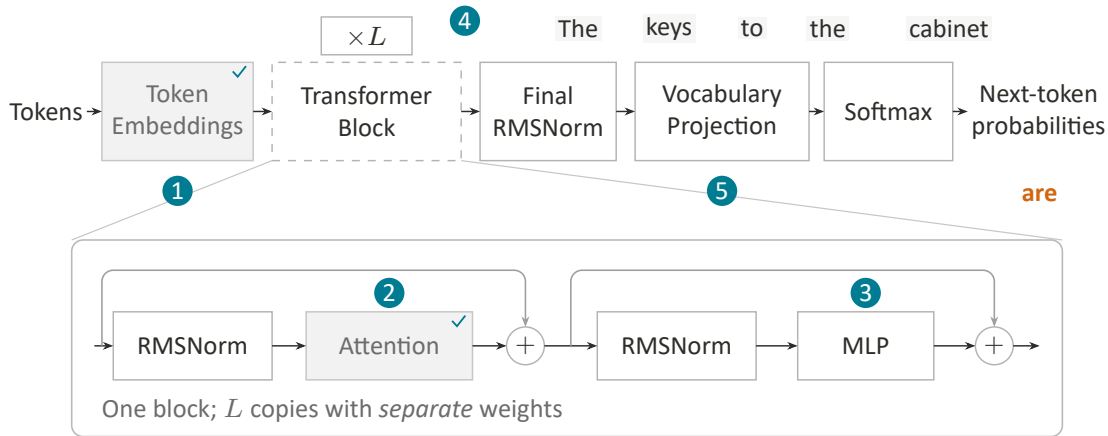
Following One Prediction



The last vector gives next-token probabilities: "are".



Following One Prediction



Back to attention: why does it look the way it does?



Attention Mechanism

Score how relevant key k_j is to query q_t . Why a dot product?

Logits

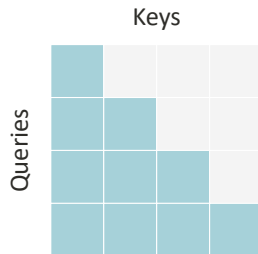
$$s_{tj} = \frac{q_t^\top k_j}{\sqrt{d_k}}$$

Weights

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_{j' \leq t} e^{s_{tj'}}} \quad (j \leq t)$$

Output

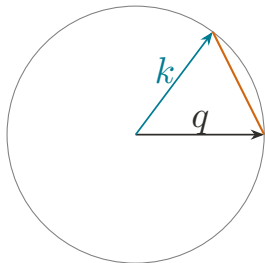
$$o_t = \sum_{j \leq t} a_{tj} v_j$$



Causal mask:
 $a_{tj} = 0$ for $j > t$



Why Dot Products?



Dot products are a standard measure of similarity.

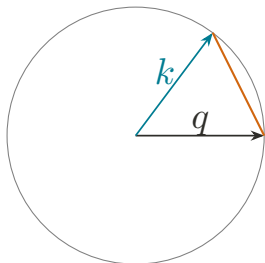
For unit vectors q, k :

$$\|q - k\|_2^2 = 2 - 2q^T k.$$

Larger dot product, smaller squared distance.



Why Dot Products?



Dot products are a standard measure of similarity.

For unit vectors q, k :

$$\|q - k\|_2^2 = 2 - 2q^\top k.$$

Larger dot product, smaller squared distance.

We also divide by $\sqrt{d_k}$. Why?

If $q_t, k_j \sim \mathcal{N}(0, I_{d_k})$ are independent, $\text{Var}(q_t^\top k_j) = d_k$; dividing by $\sqrt{d_k}$ keeps $\text{Var}(s_{tj}) = 1$.



Attention Mechanism

These weights are the **softmax** of the scores. **Why softmax?**

Logits

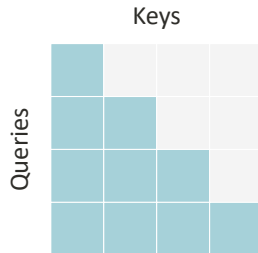
$$s_{tj} = \frac{q_t^\top k_j}{\sqrt{d_k}}$$

Weights

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_{j' \leq t} e^{s_{tj'}}} \quad (j \leq t)$$

Output

$$o_t = \sum_{j \leq t} a_{tj} v_j$$



Causal mask:
 $a_{tj} = 0$ for $j > t$



Why Softmax?

For finite scores and a “temperature” $\tau > 0$:

$$P_i(s) := \frac{e^{s_i/\tau}}{\sum_j e^{s_j/\tau}}$$

Claim: only softmax, for some $\tau > 0$, has these properties.

Monotonicity

Higher score,
higher probability

Symmetry

Permuting scores
permutes
probabilities

Continuity

Small score changes,
small probability
changes

Positivity

Every option gets
positive probability

The odds of i over j depend only on $s_i - s_j$.

For some function $g > 0$:

$$\frac{P_i(s)}{P_j(s)} = g(s_i - s_j)$$


Deriving Softmax from Axioms

Pairwise odds:

$$\frac{P_i(s)}{P_j(s)} = g(s_i - s_j)$$



Deriving Softmax from Axioms

Pairwise odds:

$$\frac{P_i(s)}{P_j(s)} = g(s_i - s_j)$$

Take three scores: $s_A = x + y$, $s_B = y$, $s_C = 0$.

$$\frac{P_A}{P_C} = \frac{P_A}{P_B} \frac{P_B}{P_C}$$



Deriving Softmax from Axioms

Pairwise odds: $\frac{P_i(s)}{P_j(s)} = g(s_i - s_j)$

Take three scores: $s_A = x + y$, $s_B = y$, $s_C = 0$.

$$g(x + y) = \frac{P_A}{P_C} = \frac{P_A}{P_B} \frac{P_B}{P_C} = g(x)g(y)$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$

Rational inputs:

$$f(m/n) = (m/n)c, \quad c = f(1)$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$

Rational inputs:

$$f(m/n) = (m/n)c, \quad c = f(1)$$

Continuity:

$$f(x) = cx \quad (x \in \mathbb{R}) \Rightarrow g(x) = e^{cx}$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$

Rational inputs:

$$f(m/n) = (m/n)c, \quad c = f(1)$$

Continuity:

$$f(x) = cx \quad (x \in \mathbb{R}) \Rightarrow g(x) = e^{cx}$$

Monotonicity:

$$g(1) > 1 \Rightarrow c > 0, \quad \tau = 1/c$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$

Rational inputs:

$$f(m/n) = (m/n)c, \quad c = f(1)$$

Continuity:

$$f(x) = cx \quad (x \in \mathbb{R}) \Rightarrow g(x) = e^{cx}$$

Monotonicity:

$$g(1) > 1 \Rightarrow c > 0, \quad \tau = 1/c$$

Normalize:

$$P_i(s) = \frac{e^{s_i/\tau}}{\sum_j e^{s_j/\tau}}$$



From the Product Rule to Softmax

$$g(x + y) = g(x)g(y)$$

Take logs:

$$f = \log g \Rightarrow f(x + y) = f(x) + f(y)$$

Rational inputs:

$$f(m/n) = (m/n)c, \quad c = f(1)$$

Continuity:

$$f(x) = cx \quad (x \in \mathbb{R}) \Rightarrow g(x) = e^{cx}$$

Monotonicity:

$$g(1) > 1 \Rightarrow c > 0, \quad \tau = 1/c$$

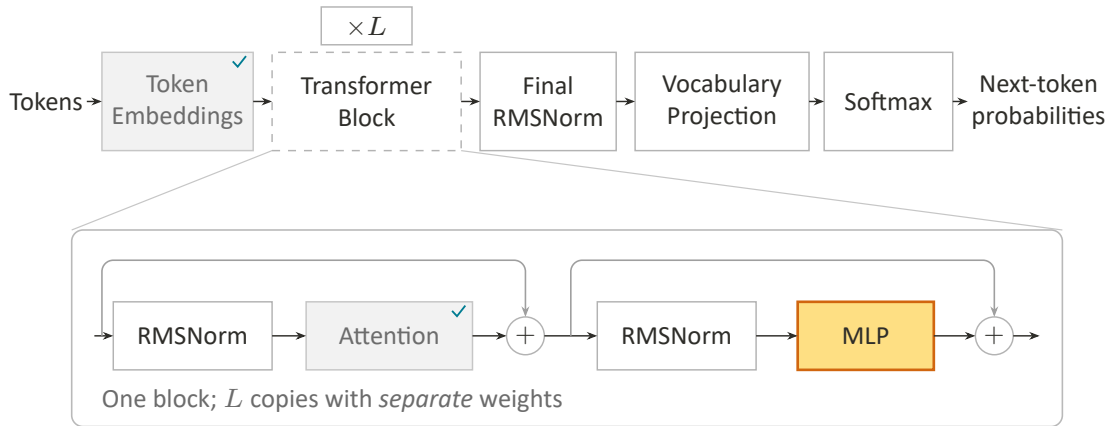
Normalize:

$$P_i(s) = \frac{e^{s_i/\tau}}{\sum_j e^{s_j/\tau}}$$

These properties leave only the temperature τ free.



Back to the Model: The MLP



Next: the MLP, which works on each token separately.



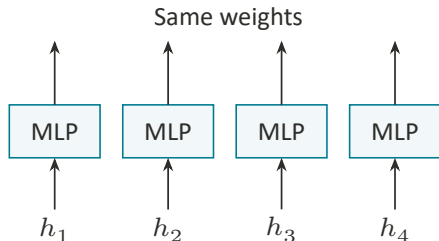
The MLP

The same MLP acts separately on each position's vector.

$$\text{MLP}(h) = W_2 \varphi(W_1 h)$$

$$W_1 \in \mathbb{R}^{d_{\text{ff}} \times d}, \quad W_2 \in \mathbb{R}^{d \times d_{\text{ff}}}$$

φ is a nonlinearity.



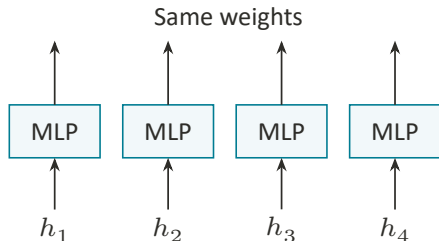
The MLP

The same MLP acts separately on each position's vector.

$$\text{MLP}(h) = W_2 \varphi(W_1 h)$$

$$W_1 \in \mathbb{R}^{d_{\text{ff}} \times d}, \quad W_2 \in \mathbb{R}^{d \times d_{\text{ff}}}$$

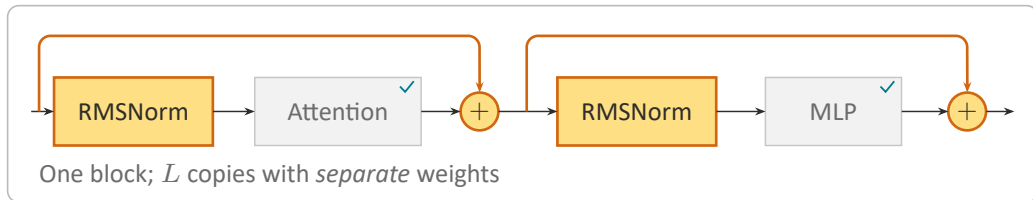
φ is a nonlinearity.



Attention moves information between tokens; the MLP then works on each token separately.



One Transformer Block

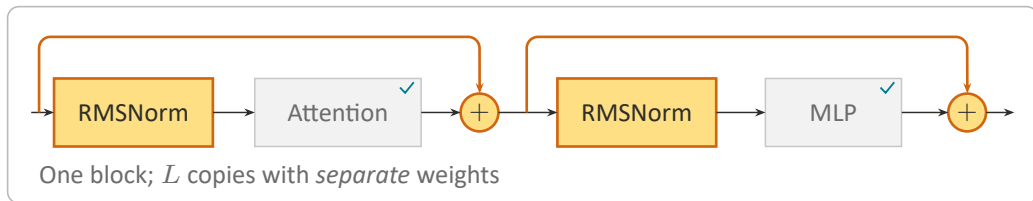


$$H' = H + \text{Attn}(\text{RMSNorm}(H))$$

$$H'' = H' + \text{MLP}(\text{RMSNorm}(H'))$$



One Transformer Block



$$H' = H + \text{Attn}(\text{RMSNorm}(H))$$

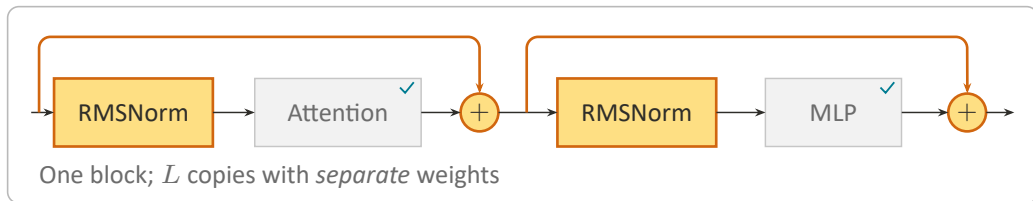
$$H'' = H' + \text{MLP}(\text{RMSNorm}(H'))$$

Residuals: each sublayer adds an update to the state.

RMSNorm first: each sublayer reads a normalized copy of the state.



One Transformer Block



$$H' = H + \text{Attn}(\text{RMSNorm}(H))$$

$$H'' = H' + \text{MLP}(\text{RMSNorm}(H'))$$

Residuals: each sublayer adds an update to the state.

RMSNorm first: each sublayer reads a normalized copy of the state.

Residual paths let information and gradients bypass each sublayer.



Root-Mean-Square Normalization

Updates accumulate across layers, so the state's scale can drift. Each sublayer needs an input of controlled scale.

$$\text{RMSNorm}(h) = \frac{g \odot h}{\sqrt{d^{-1} \|h\|_2^2 + \varepsilon}}$$

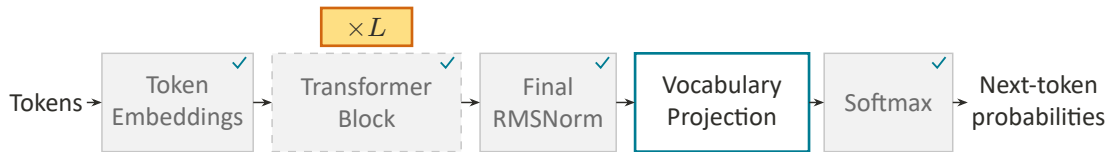
g is learned; $\varepsilon > 0$ stabilizes the denominator.

Source: [Zhang and Sennrich \(2019\)](#).

MS&E 319 Lecture 2



Back to the Model: Stacking Blocks



Model

Blocks L

Width d

Llama 3 8B

32

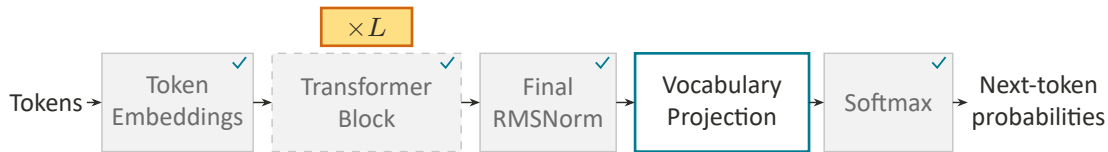
4,096

Sources: [Llama Team \(2024\)](#), Table 3; configs: [DS Flash](#), [DS Pro](#), [GLM](#), [Kimi](#) (23 Sep. 2026).

MS&E 319 Lecture 2



Back to the Model: Stacking Blocks



Model	Blocks L	Width d
Llama 3 8B	32	4,096
DeepSeek V4.1 Flash	40	5,120
DeepSeek V4 Pro	61	7,168
GLM-5.2 (Z.ai)	78	6,144
Kimi K3 (Moonshot)	93	7,168

Recent models stack more and wider blocks.

Sources: [Llama Team \(2024\)](#), Table 3; configs: [DS Flash](#), [DS Pro](#), [GLM](#), [Kimi](#) (23 Sep. 2026).



Vocabulary Projection

We use only the last position's vector to predict the next token.



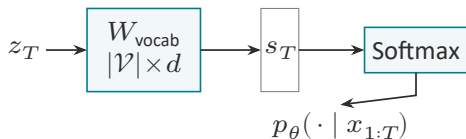
Vocabulary Projection

We use only the last position's vector to predict the next token.

$$z_T = \text{RMSNorm}(h_T^{(L)})$$

$$s_T = W_{\text{vocab}} z_T$$

$$p_{\theta}(\cdot \mid x_{1:T}) = \text{softmax}(s_T)$$



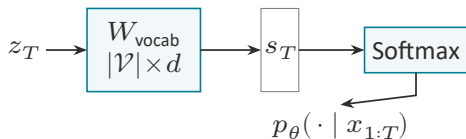
Vocabulary Projection

We use only the last position's vector to predict the next token.

$$z_T = \text{RMSNorm}(h_T^{(L)})$$

$$s_T = W_{\text{vocab}} z_T$$

$$p_{\theta}(\cdot \mid x_{1:T}) = \text{softmax}(s_T)$$



Model	$ \mathcal{V} \times d$	Output-head weights
Llama 3 8B	$128,256 \times 4,096$	$\approx 525\text{M}$

Sources: [Llama Team \(2024\), Table 3](#); configs: [DS Flash](#), [DS Pro](#), [GLM](#), [Kimi](#) (23 Sep. 2026).

MS&E 319 Lecture 2



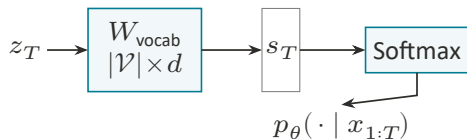
Vocabulary Projection

We use only the last position's vector to predict the next token.

$$z_T = \text{RMSNorm}(h_T^{(L)})$$

$$s_T = W_{\text{vocab}} z_T$$

$$p_{\theta}(\cdot \mid x_{1:T}) = \text{softmax}(s_T)$$



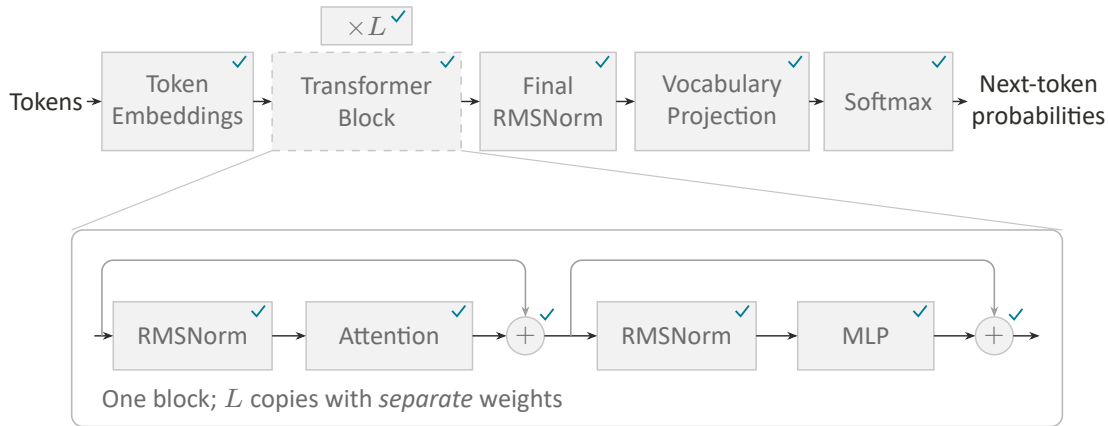
Model	$ \mathcal{V} \times d$	Output-head weights
Llama 3 8B	$128,256 \times 4,096$	$\approx 525\text{M}$
DeepSeek V4.1 Flash	$129,280 \times 5,120$	$\approx 662\text{M}$
DeepSeek V4 Pro	$129,280 \times 7,168$	$\approx 927\text{M}$
GLM-5.2	$154,880 \times 6,144$	$\approx 952\text{M}$
Kimi K3	$163,840 \times 7,168$	$\approx 1.17\text{B}$

Embedding: token $\rightarrow$ vector. Projection: vector $\rightarrow$ scores.

Sources: [Llama Team \(2024\)](#), Table 3; configs: [DS Flash](#), [DS Pro](#), [GLM](#), [Kimi](#) (23 Sep. 2026).



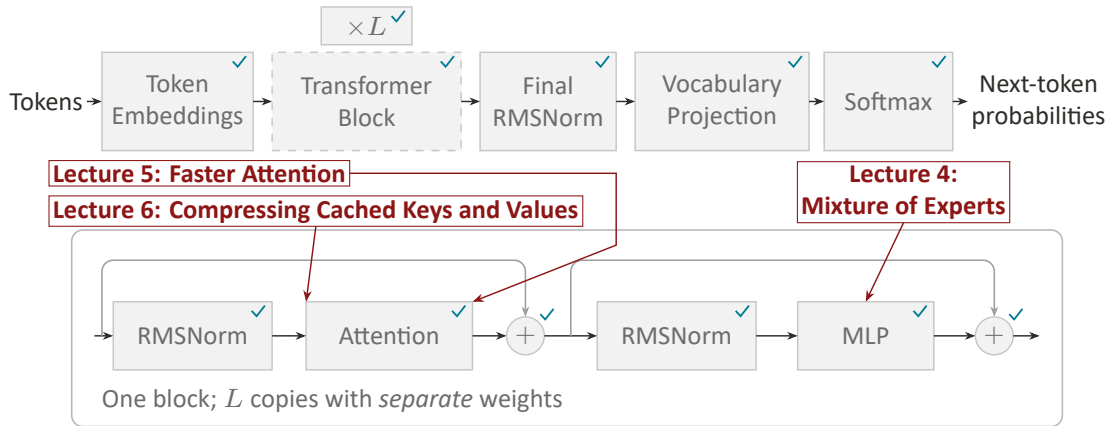
The Complete Model



Tokens $\rightarrow$ vectors $\rightarrow L$ blocks $\rightarrow$ scores $\rightarrow$ probabilities.



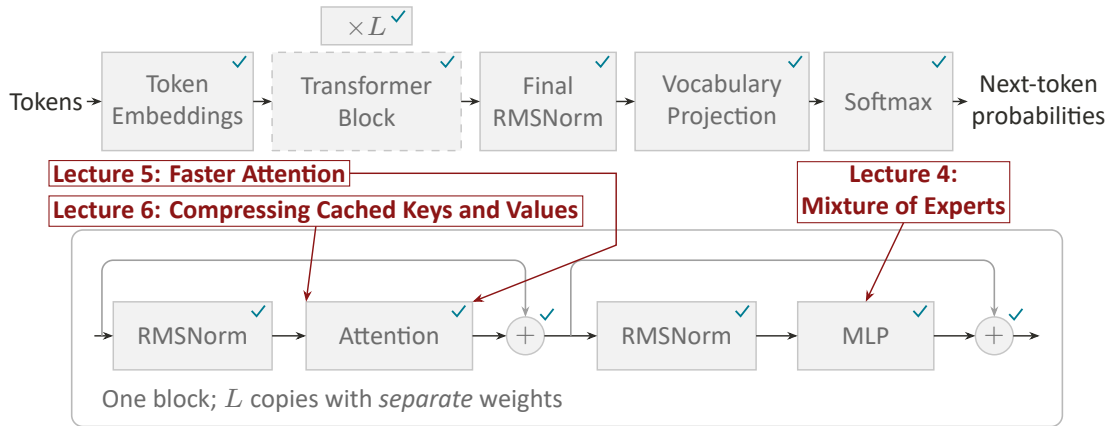
The Complete Model



Tokens → vectors → L blocks → scores → probabilities.



The Complete Model



We have the model. Next: how do we learn its weights?



Today's Plan

1



LLM Architecture

What does a modern LLM with N parameters look like?

2

Training

Given D training tokens, how do we train the model?

3

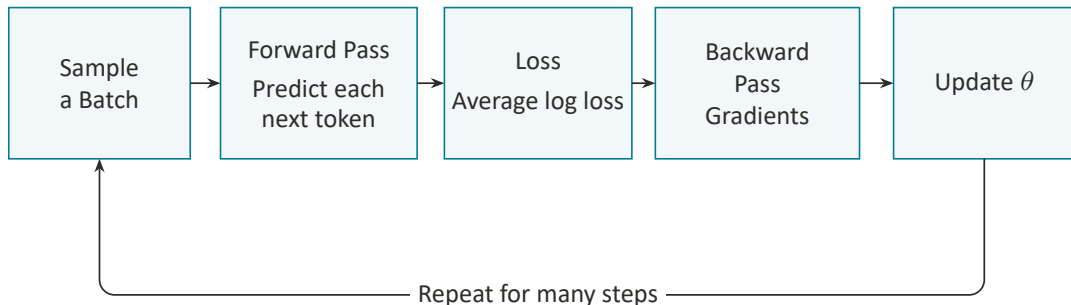
Scaling Laws

How should we split resources between N and D ?



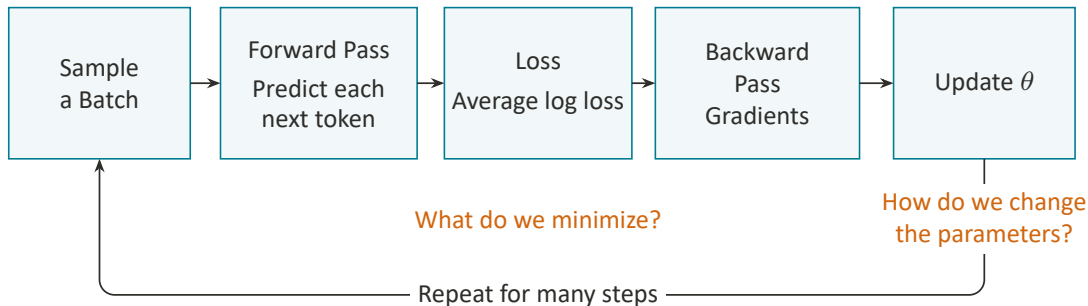
How Training Works

Adjust θ so the model assigns high probability to real text.

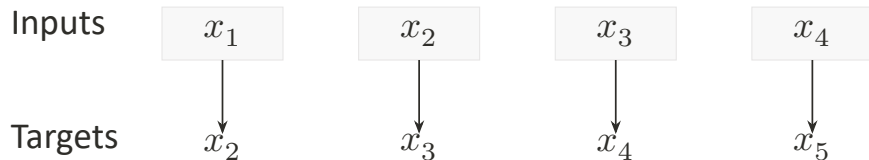


How Training Works

Adjust θ so the model assigns high probability to real text.



Training at Every Position



$$\widehat{\mathcal{L}}(\theta) = -\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(x_{t+1} \mid x_{1:t})$$

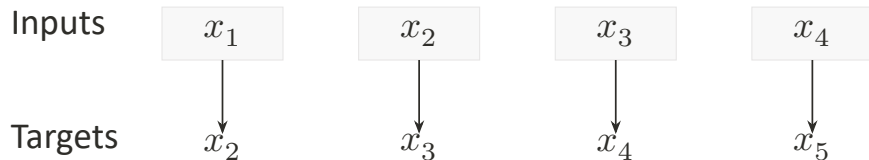
Targets are deduced from the text, without any human labels.

One pass gives T predictions.

The causal mask hides target x_{t+1} from position t .



Training at Every Position



Next-token log loss: $\widehat{\mathcal{L}}(\theta) = -\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(x_{t+1} \mid x_{1:t})$

Targets are deduced from the text, without any human labels.

One pass gives T predictions.

The causal mask hides target x_{t+1} from position t .



Why Next-Token Log Loss?

Minimizing this loss maximizes the probability of the observed text.

$$p_{\theta}(x_2, \dots, x_{T+1} \mid x_1) = \prod_{t=1}^T p_{\theta}(x_{t+1} \mid x_{1:t})$$
$$\widehat{\mathcal{L}}(\theta) = -\frac{1}{T} \log p_{\theta}(x_2, \dots, x_{T+1} \mid x_1)$$



Why Next-Token Log Loss?

Minimizing this loss maximizes the probability of the observed text.

$$p_{\theta}(x_2, \dots, x_{T+1} \mid x_1) = \prod_{t=1}^T p_{\theta}(x_{t+1} \mid x_{1:t})$$
$$\widehat{\mathcal{L}}(\theta) = -\frac{1}{T} \log p_{\theta}(x_2, \dots, x_{T+1} \mid x_1)$$

Let P be the true next-token distribution and Q the model's:

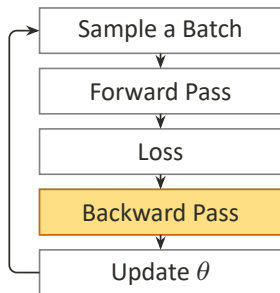
$$\mathbb{E}_{X \sim P}[-\log Q(X)] = \underbrace{H(P)}_{\text{independent of } Q} + \underbrace{D_{\text{KL}}(P \parallel Q)}_{\geq 0, =0 \text{ iff } Q=P}$$

So the unique minimizer of the expected loss is $Q = P$.



From Gradients to Updates

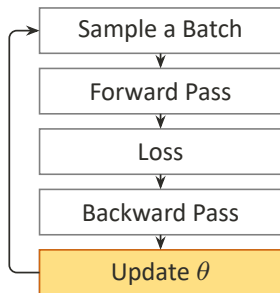
Gradients of the loss with respect to parameters are computed by backpropagation.



From Gradients to Updates

Gradients of the loss with respect to parameters are computed by backpropagation.

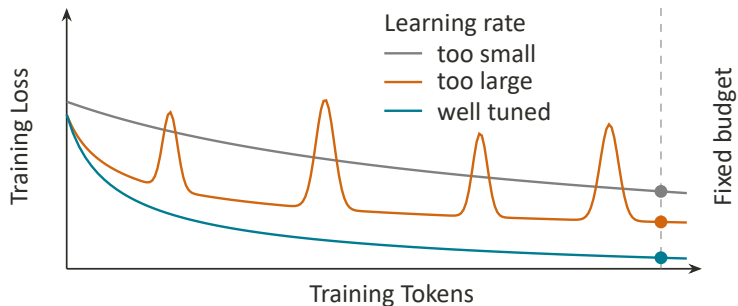
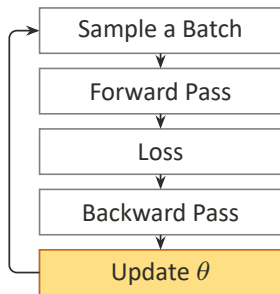
Given gradients, optimizers (typically Adam) update the parameters.



From Gradients to Updates

Gradients of the loss with respect to parameters are computed by backpropagation.

Given gradients, optimizers (typically Adam) update the parameters.
Same compute, three learning rates



Optimizer choice can substantially change loss at fixed “compute.”



Today's Plan

1



LLM Architecture

What does a modern LLM with N parameters look like?

2



Training

Given D training tokens, how do we train the model?

3

Scaling Laws

How should we split resources between N and D ?



Planning the Next Training Run

We have a fixed training budget of about 6×10^{23} FLOPs.

A: Larger Model
280B parameters
300B tokens

B: Smaller Model
70B parameters
1.4T tokens

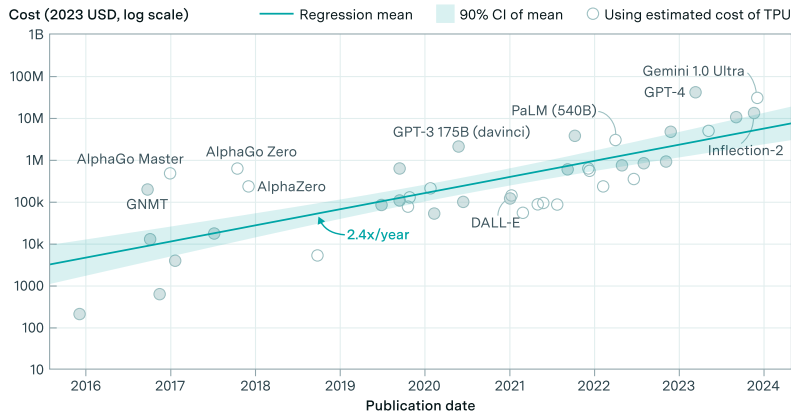
Same Training Compute

Which would you fund?



Why Not Train Both and Compare?

Training runs are expensive.



GPT-4 (2023)

~\$40M

Final run:
hardware
+ energy

Source: Cottier et al. (2025), Fig. 1. Historical estimates; amortized hardware + energy.

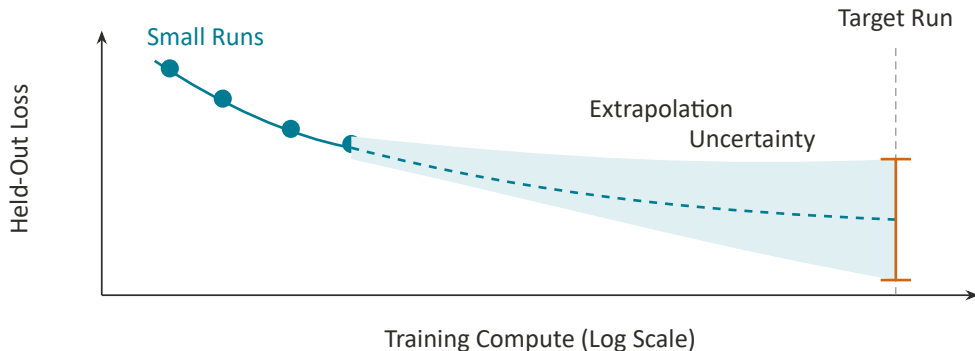


Can Small Runs Predict a Large One?



Can Small Runs Predict a Large One?

Idea: fit a trend on small runs, then extrapolate to the target.

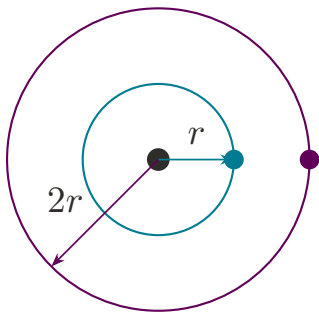


Do such trends exist, and do they hold at scale?



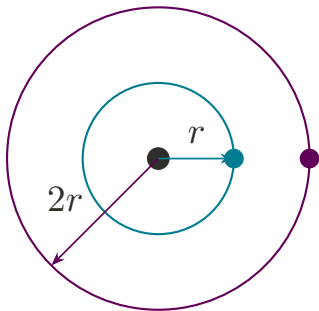
Scaling Laws in Other Fields

Double the distance. What happens to the force?



Scaling Laws in Other Fields

Double the distance. What happens to the force?



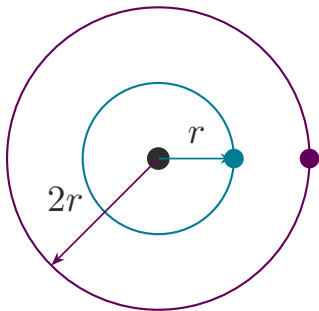
$$F(2r) = \frac{1}{4}F(r)$$

$$F \propto r^{-2}$$



Scaling Laws in Other Fields

Double the distance. What happens to the force?



$$F(2r) = \frac{1}{4}F(r)$$

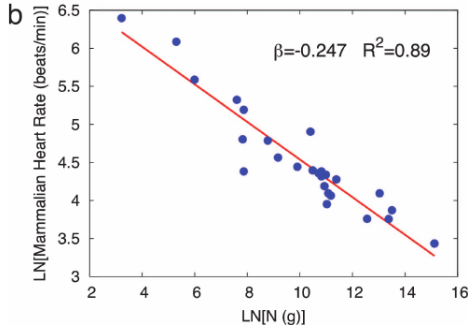
$$F \propto r^{-2}$$

Explanation: the same flux spreads over a sphere of area $4\pi r^2$.

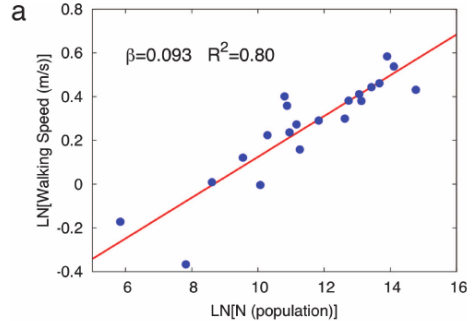
Prediction: with Newton's laws, it predicts planetary orbits.



Scaling Laws in Other Fields



Larger animals: slower heart rates.



Larger cities: faster walking.

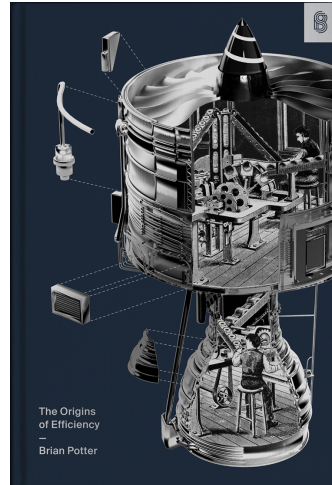
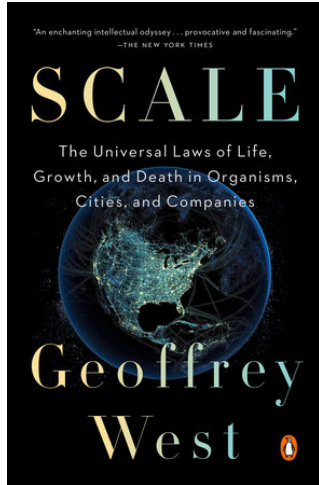
On log-log axes, each trend is a straight line: a power law $Y \propto X^a$.

Fitted exponents predict well; mechanisms are still debated.

Source: Bettencourt et al. (2007), Figs. 2a–b.
MS&E 319 Lecture 2



Scaling Laws in Other Fields

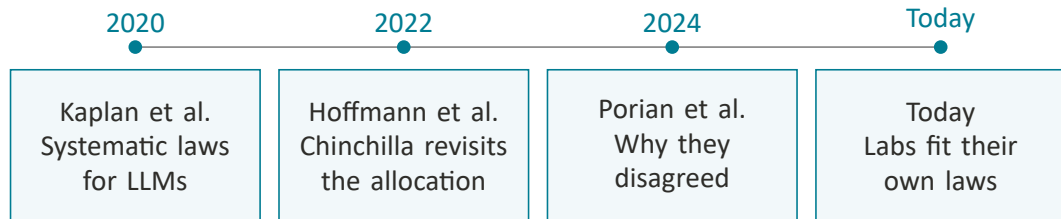


Sources: West, *Scale* (2017); Potter, *The Origins of Efficiency*.
MS&E 319 Lecture 2



Scaling Laws in Machine Learning

Power-law learning curves predate LLMs: Hestness et al. (2017) found them in language modeling, translation, vision, and speech.

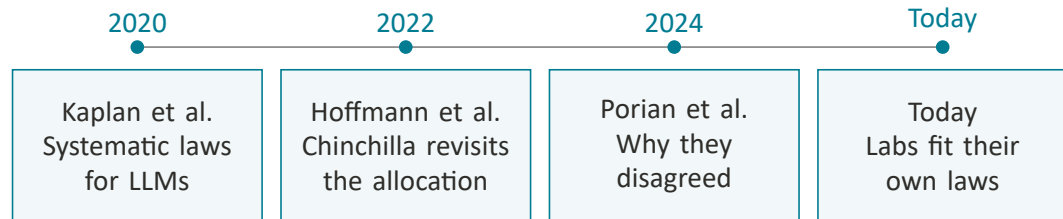


At each step: what does it say about A vs. B?



Scaling Laws in Machine Learning

Power-law learning curves predate LLMs: Hestness et al. (2017) found them in language modeling, translation, vision, and speech.



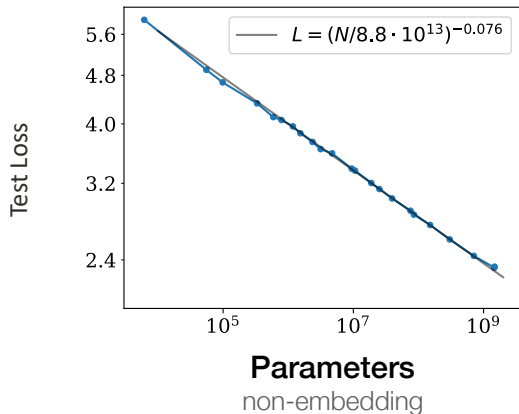
At each step: what does it say about A vs. B?

LLM scaling laws predict well; a general explanation of their exponents is still open.

Source: [Hestness et al. \(2017\)](#), [arXiv:1712.00409](#).
MS&E 319 Lecture 2



Kaplan: Predictable Loss at Scale



Vary Parameters

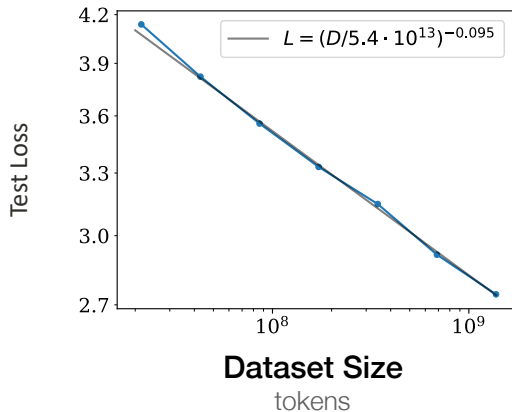
Enough data and training.

$$\mathcal{L} \propto N^{-0.076}$$

Source: Kaplan et al. (2020), Figure 1. Original panels; each has its own regime.



Kaplan: Predictable Loss at Scale



Vary Data

(dataset size D)

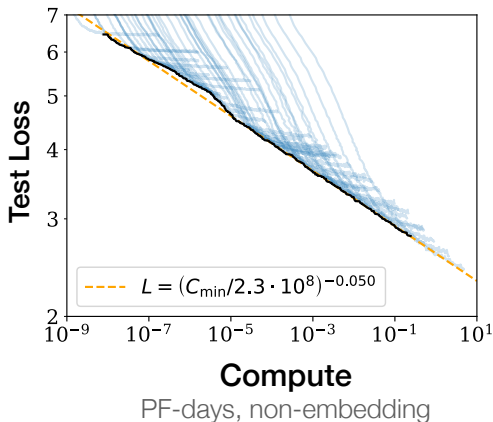
Large model, trained to best test loss.

$$\mathcal{L} \propto D^{-0.095}$$

Source: Kaplan et al. (2020), Figure 1. Original panels; each has its own regime.



Kaplan: Predictable Loss at Scale



Vary Compute

Best loss across model sizes and training runs.

$$\mathcal{L} \propto C_{\min}^{-0.050}$$

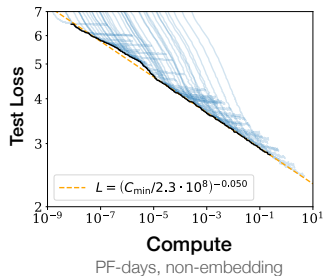
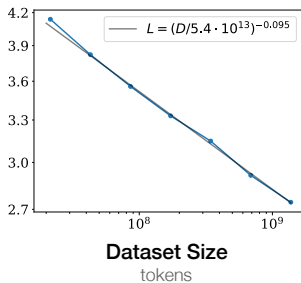
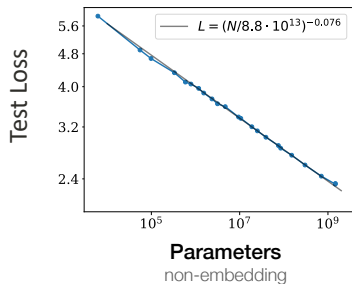
$C_{\min}$: compute at an efficient batch size

Source: Kaplan et al. (2020), Figure 1. Original panels; each has its own regime.



Kaplan: Predictable Loss at Scale

Smooth trends across many orders of magnitude.



$C_{\min}$: compute at an efficient batch size

Source: Kaplan et al. (2020), Figure 1. Original panels; each has its own regime.



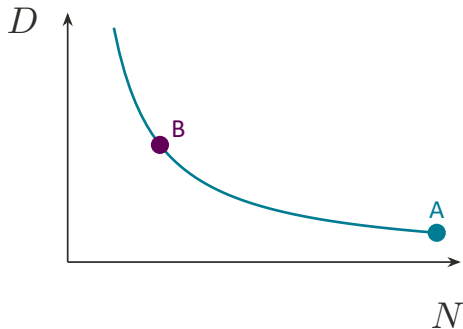
Parameters and Tokens Share a Budget

N : parameter count. D : processed training tokens.

C : training FLOPs.

$$C \approx 6ND$$

A larger model gets fewer tokens at fixed compute.



Which allocation gives the lowest loss?



Parameters and Tokens Share a Budget

N : parameter count. D : processed training tokens.

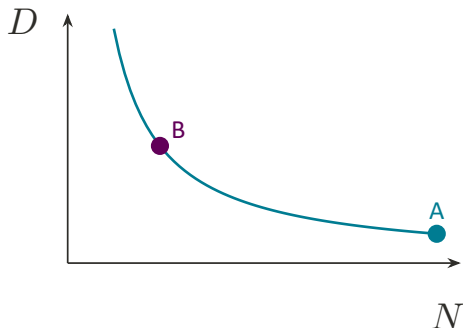
C : training FLOPs.

$$C \approx 6ND$$

A larger model gets fewer tokens at fixed compute.

A: $6 \times 280\text{B} \times 300\text{B} \approx 5.0 \times 10^{23}$.

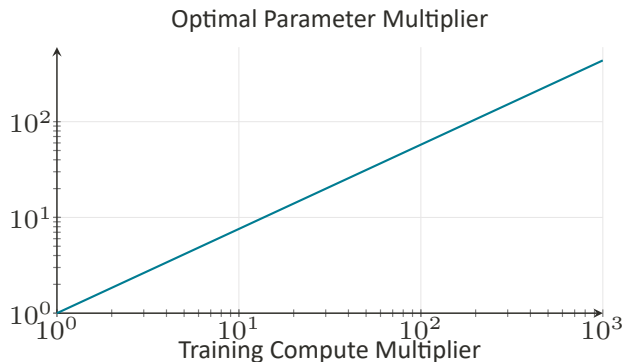
B: $6 \times 70\text{B} \times 1.4\text{T} \approx 5.9 \times 10^{23}$.



Which allocation gives the lowest loss?



Kaplan's Allocation Recommendation



$$N^* \propto C^{0.88}$$

10× compute: about
7.6× parameters, only
1.3× tokens.

Here, N excludes embeddings and the output head.

Source: [Kaplan et al. \(2020\)](#), Figure 14. Calculated curve of the reported unadjusted fit.



Back to Our Question: Kaplan

Favored

A: Larger Model
280B parameters
300B tokens

Kaplan's fit: loss 1.71

B: Smaller Model
70B parameters
1.4T tokens

Kaplan's fit: loss 1.74

Kaplan (2020): **A**

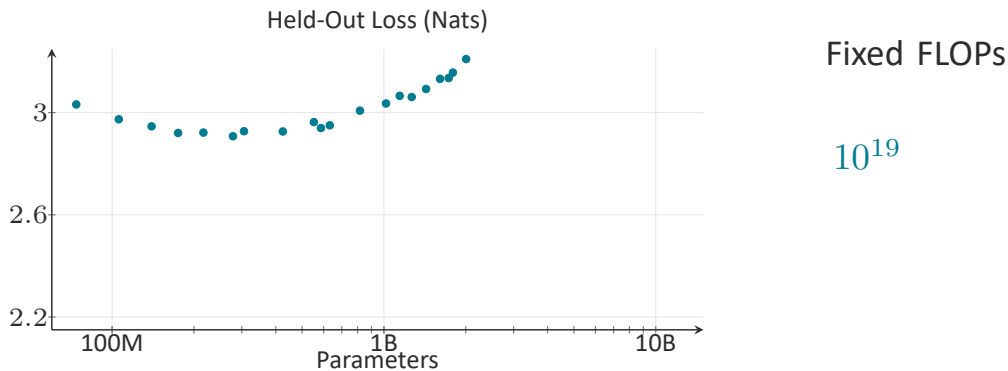
Chinchilla (2022): ?

Porian (2024): ?

Kaplan: put most extra compute into parameters. Favors A.



Chinchilla: IsoFLOP Experiments

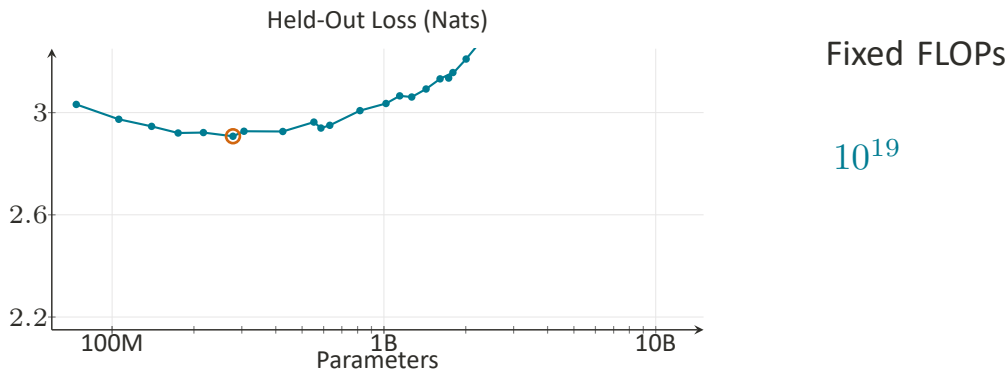


At fixed FLOPs, a larger N gets fewer tokens. Each curve's minimum is the best N for that budget.

Source: [Hoffmann et al. \(2022\)](#), Figure 3. Digitized observations.
MS&E 319 Lecture 2



Chinchilla: IsoFLOP Experiments

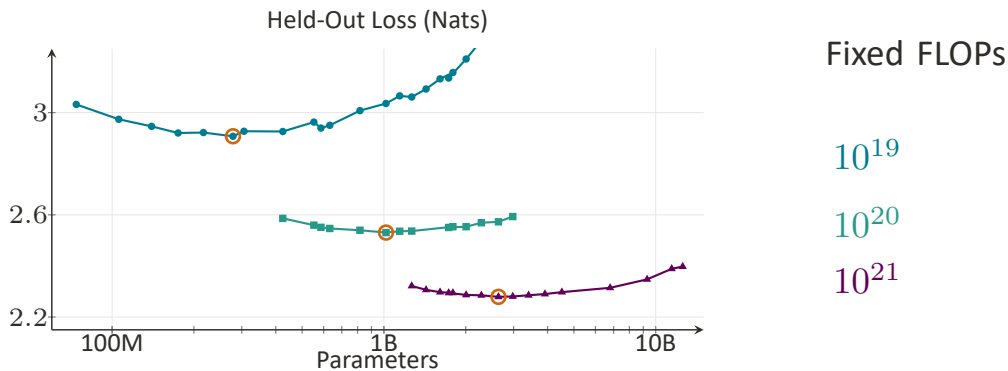


At fixed FLOPs, a larger N gets fewer tokens. Each curve's minimum is the best N for that budget.

Source: [Hoffmann et al. \(2022\)](#), Figure 3. Digitized observations.
MS&E 319 Lecture 2



Chinchilla: IsoFLOP Experiments

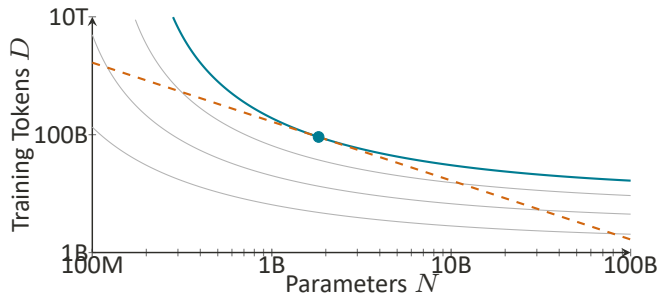


At fixed FLOPs, a larger N gets fewer tokens. Each curve's minimum is the best N for that budget.



From a Loss Fit to an Allocation

$$\widehat{\mathcal{L}}(N, D) = \mathcal{L}_{\infty} + AN^{-\alpha} + BD^{-\beta}$$



The best allocation is where the budget line touches the lowest contour.

Budget:
 $C \approx 6ND$

Source: Hoffmann et al. (2022), Equation 2. Calculated contours from the rounded fit.



Balancing Parameters and Tokens

Substitute $D = C/(6N)$, then differentiate with respect to $\log N$.

$$\alpha AN^{-\alpha} = \beta BD^{-\beta}$$

The gain from extra capacity balances the cost of fewer tokens.



Balancing Parameters and Tokens

Substitute $D = C/(6N)$, then differentiate with respect to $\log N$.

$$\alpha AN^{-\alpha} = \beta BD^{-\beta}$$

The gain from extra capacity balances the cost of fewer tokens.

$$N^* \propto C^{\beta/(\alpha+\beta)}, \quad D^* \propto C^{\alpha/(\alpha+\beta)}$$



Balancing Parameters and Tokens

Substitute $D = C/(6N)$, then differentiate with respect to $\log N$.

$$\alpha AN^{-\alpha} = \beta BD^{-\beta}$$

The gain from extra capacity balances the cost of fewer tokens.

$$N^* \propto C^{\beta/(\alpha+\beta)}, \quad D^* \propto C^{\alpha/(\alpha+\beta)}$$

Chinchilla's fit: $\alpha \approx 0.34$, $\beta \approx 0.28$

$$\Rightarrow N^* \propto C^{0.45}, \quad D^* \propto C^{0.55}.$$



Chinchilla's Allocation

Follow the best allocation as the compute budget grows.

$$N^* \propto C^{1/2}, \quad D^* \propto C^{1/2}$$

Their large run used about 20 training tokens per parameter.

Source: [Hoffmann et al. \(2022\)](#). Approximate equal scaling; estimates vary by fitting approach.



Chinchilla's Allocation

Follow the best allocation as the compute budget grows.

$$N^* \propto C^{1/2}, \quad D^* \propto C^{1/2}$$

10× Compute: 3.2× Parameters, 3.2× Tokens

Their large run used about 20 training tokens per parameter.

Source: [Hoffmann et al. \(2022\)](#). Approximate equal scaling; estimates vary by fitting approach.



Back to Our Question: Chinchilla

Favored

A: Larger Model
280B parameters
300B tokens

B: Smaller Model
70B parameters
1.4T tokens

Chinchilla's fit: loss 1.99
About 1 token per parameter

Chinchilla's fit: loss 1.94
20 tokens per parameter

Kaplan (2020): **A**

Chinchilla (2022): **B**

Porian (2024): ?

Chinchilla: about 20 tokens per parameter. Favors B.



Testing the Prediction at Scale

DeepMind trained both. A was Gopher; B was Chinchilla.

Gopher (A)

280B parameters

300B training tokens

Chinchilla (B)

70B parameters

1.4T training tokens



Testing the Prediction at Scale

DeepMind trained both. A was Gopher; B was Chinchilla.

Gopher (A)

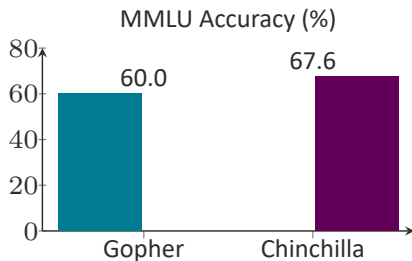
280B parameters

300B training tokens

Chinchilla (B)

70B parameters

1.4T training tokens



Better benchmark performance at a similar training budget.

Source: [Hoffmann et al. \(2022\)](#), Tables 1 and 6. MMLU five-shot average.
MS&E 319 Lecture 2



Porian: Explaining the Disagreement

Kaplan's reported exponent: 0.88.

Chinchilla: about 0.50.

Porian et al. reproduce a Kaplan-style setup, then change one ingredient at a time.

The reproduction starts at 0.835 on RefinedWeb.

Source: [Porian et al. \(2024\)](#), Figure 1 and Table 1. Corrections accumulate down the table.

$$N^* \propto C^a$$

Baseline	0.835
----------	-------

Output Head

Warmup

Optimizer



Counting the Output Head

The head has $|\mathcal{V}|d$ weights.

$d = 512, L = 8$

Body Head



Head: 50.4% of weight FLOPs

$d = 8192, L = 80$



Head: 0.63% of weight FLOPs

$$N^* \propto C^a$$

Baseline	0.835
Output Head	0.706
Warmup	
Optimizer	

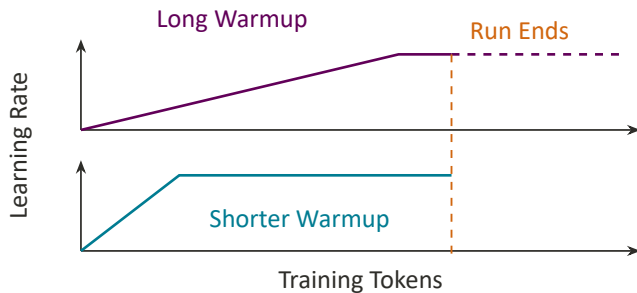
Omitting it makes small models look cheaper than they are.

Source: [Porian et al., Section 3.2](#). Calculated bars: $|\mathcal{V}| = 50,000$, body $12Ld^2$.
MS&E 319 Lecture 2



Matching Warmup to the Run

Warmup ramps up the learning rate at the start of training. A fixed warmup takes a large share of a short run.



$$N^* \propto C^a$$

Baseline	0.835
Output Head	0.706
Warmup	0.602

Optimizer

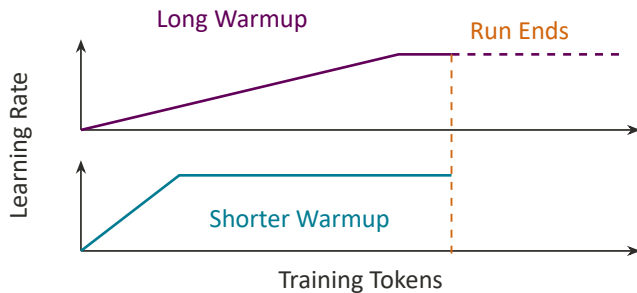
Source: [Porian et al. \(2024\)](#), Section 3.3. Schematic schedules for the same short run.
MS&E 319 Lecture 2



Matching Warmup to the Run

Warmup ramps up the learning rate at the start of training. A fixed warmup takes a large share of a short run.

At fixed compute, larger models get the shortest runs, so they are hurt most at small budgets. This inflates the exponent.



$$N^* \propto C^a$$

Baseline	0.835
Output Head	0.706
Warmup	0.602
Optimizer	

Source: [Porian et al. \(2024\)](#), Section 3.3. Schematic schedules for the same short run.
MS&E 319 Lecture 2



Tuning the Optimizer at Each Scale

Tune the batch size, learning rate, and other optimizer settings at each scale.

The allocation exponent reaches 0.497.

$$N^* \propto C^a$$

Baseline	0.835
Output Head	0.706
Warmup	0.602
Optimizer	0.497

Source: [Porian et al. \(2024\)](#), Table 1. RefinedWeb, cumulative corrections.



Back to Our Question: Porian

Favored

A: Larger Model
280B parameters
300B tokens

B: Smaller Model
70B parameters
1.4T tokens

Kaplan-style setup: $\alpha \approx 0.84$

Corrected setup: $\alpha \approx 0.50$

Kaplan (2020): **A**

Chinchilla (2022): **B**

Porian (2024): **B**

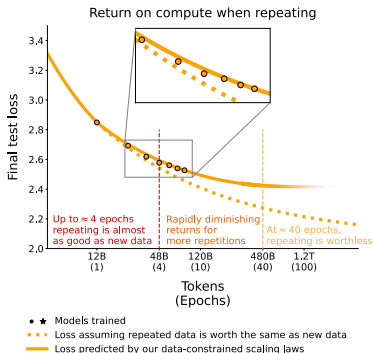
Porian: counting and tuning led Kaplan to A. Corrected: B.

Source: [Porian et al. \(2024\)](#), Table 1. RefinedWeb, cumulative corrections.
MS&E 319 Lecture 2



Reusing a Fixed Corpus

So far every token was fresh. What if data is limited?



U unique tokens; E passes.

$$D = EU, C \approx 6NEU.$$

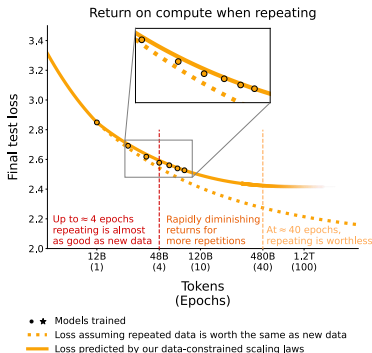
Repeated tokens count fully toward compute.

Up to about 4 passes is nearly as good as fresh data; after that, returns diminish.



Reusing a Fixed Corpus

So far every token was fresh. What if data is limited?



U unique tokens; E passes.

$$D = EU, C \approx 6NEU.$$

Repeated tokens count fully toward compute.

Up to about 4 passes is nearly as good as fresh data; after that, returns diminish.

If only 350B unique tokens exist: A uses 300B (under one pass);
B needs $1.4T = 4$ passes, still near the fresh-data curve.

Source: [Muennighoff et al. \(2023, revised 2025\), Figure 1.](#)

MS&E 319 Lecture 2



Training Cost and Lifetime Cost

$$C_{\text{life}} \approx \underbrace{6ND}_{\text{Training}} + \underbrace{2NR}_{\text{Inference}}$$

R counts lifetime prompt and generated-token positions.

Heavy use can favor a smaller model trained on more tokens to reach the same quality.



Training Cost and Lifetime Cost

$$C_{\text{life}} \approx \underbrace{6ND}_{\text{Training}} + \underbrace{2NR}_{\text{Inference}}$$

R counts lifetime prompt and generated-token positions.

Each served token costs $2N$ FLOPs: A 560 GFLOPs, B 140 GFLOPs.

Serving about 1T tokens costs A as much as its training run ($R = 3D$).

Heavy use can favor a smaller model trained on more tokens to reach the same quality.

Llama 3 8B: about 15T tokens, roughly 1,900 per parameter.

Sources: [Sardana et al. \(2024\)](#); [Llama Team \(2024\)](#). Classroom FLOP approximation.
MS&E 319 Lecture 2



How Labs Use Scaling Laws

Fit scaling laws on small runs for many training choices.

Report	What They Fit and Used
GPT-4 ('23)	Final loss predicted from runs with up to $10^4 \times$ less compute
DeepSeek ('24)	Learning rate, batch size, N/D split; 7B/67B loss
Llama 3 ('24)	Flagship model size; benchmark accuracy via task loss
Qwen2.5 ('24)	Learning rate and batch size vs. model and data size

Closed labs disclose little; this is the public evidence.

Sources: [GPT-4, §3](#); [DeepSeek LLM, §§3.1–3.3](#); [Llama 3, §§3.1.2–3.2.1](#); [Qwen2.5, §3.2](#).
MS&E 319 Lecture 2



How Labs Use Scaling Laws

The fits vary with setups: DeepSeek's allocation exponent ranged from 0.45 to 0.58 across datasets.

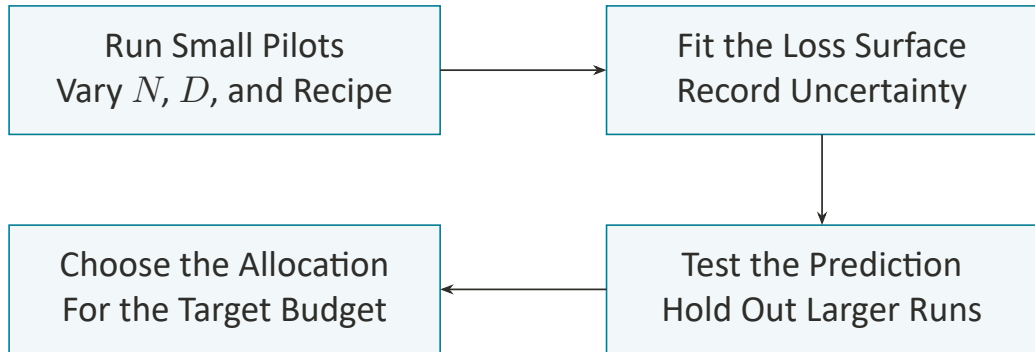
Report	What They Fit and Used
GPT-4 ('23)	Final loss predicted from runs with up to $10^4 \times$ less compute
DeepSeek ('24)	Learning rate, batch size, N/D split; 7B/67B loss
Llama 3 ('24)	Flagship model size; benchmark accuracy via task loss
Qwen2.5 ('24)	Learning rate and batch size vs. model and data size

Closed labs disclose little; this is the public evidence.

Sources: [GPT-4, §3](#); [DeepSeek LLM, §§3.1–3.3](#); [Llama 3, §§3.1.2–3.2.1](#); [Qwen2.5, §3.2](#).
MS&E 319 Lecture 2



Designing the Next Training Run



Recheck the fit when the data or training recipe changes.



Planning the Next Training Run

A: Larger Model
280B parameters
300B tokens

B: Chinchilla

B: Smaller Model
70B parameters
1.4T tokens

For this budget and fresh data: B.

In practice: fit your own laws on small runs, validate on larger pilots, and account for data limits and inference.

Where should the next unit of resource go?

Next lecture: making compute go further – optimizers and multi-token prediction.



Questions?

