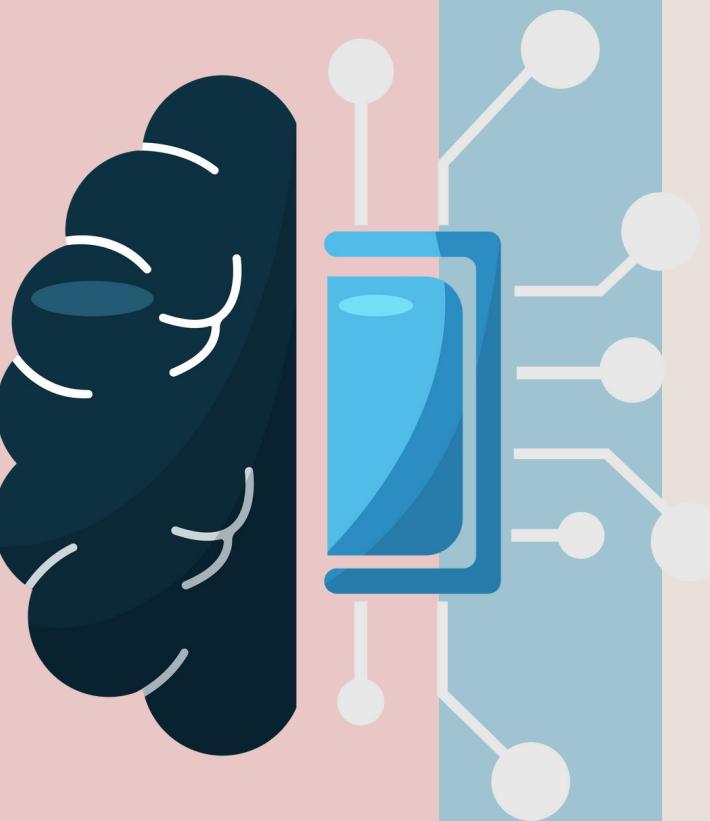


# ATTENTION IS ALL YOU

Revolutionizing Sequence Modeling in Natural Language

NEED  
Processing

CSCE 689 - Programming LLM



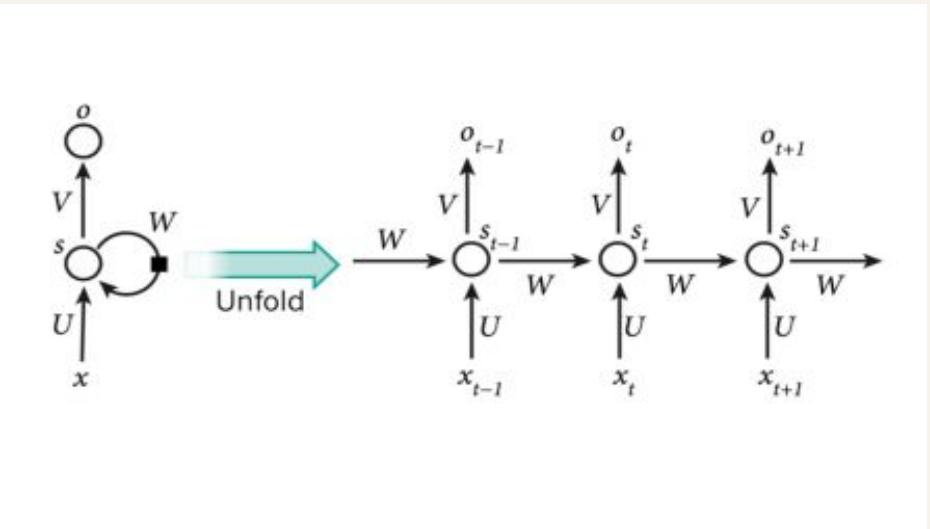
Presented By : Nikhitha Pasula

Ze Sheng

# Overview

- Background
- Context and Motivation
- Model
- Self-Attention
- Scaled Dot-Product Attention
- Multi-Head Attention
- Positional Encoding
- Encoder & Decoder
- Full-Architecture

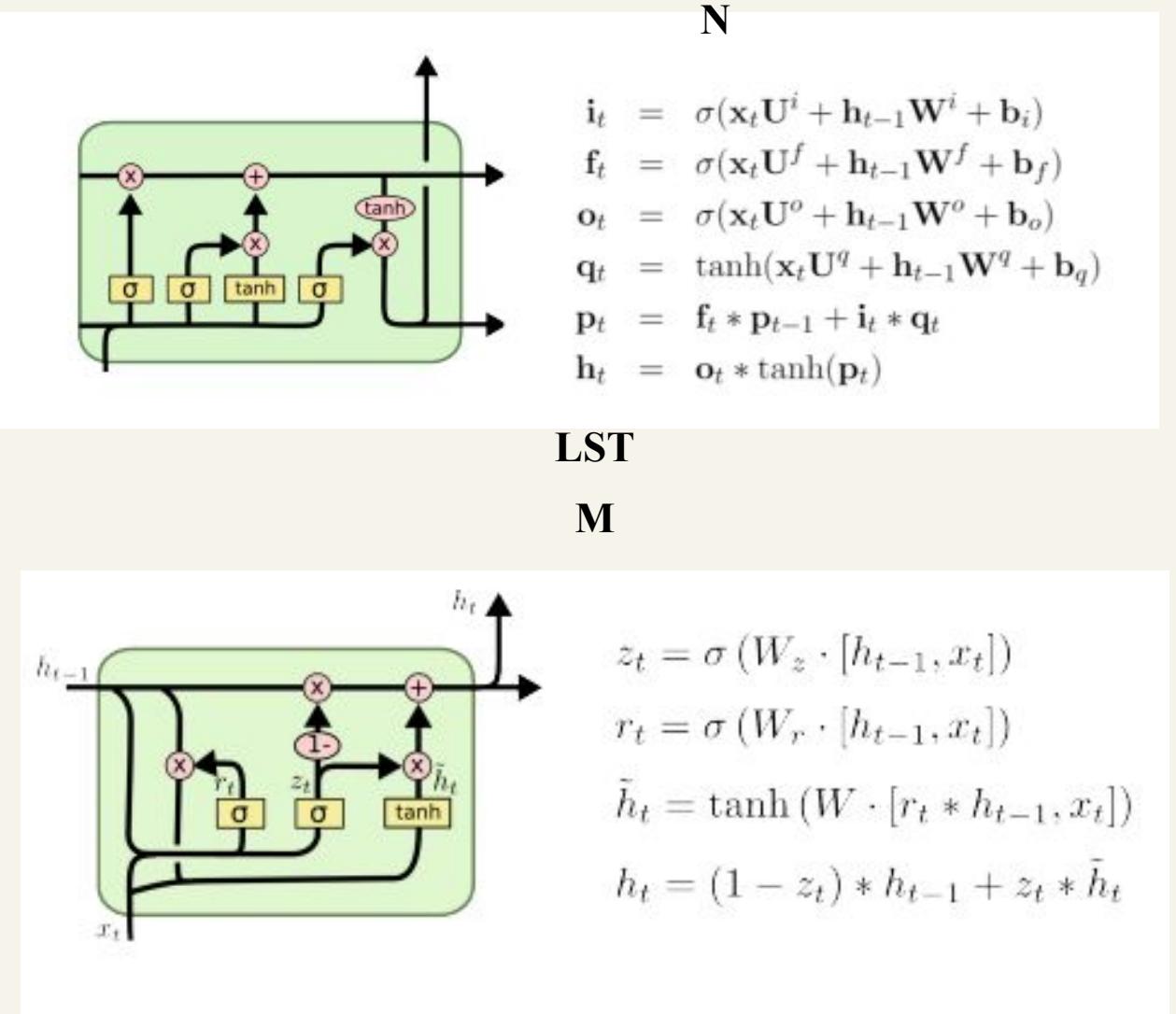
# Background



## Neural Networks and Limitations

### Traditional Neural Network Models:

- **Feedforward, Convolutional, Recurrent Networks:**
  - Commonly used for tasks like NLP, computer vision, speech recognition, etc.
- **Recurrent Networks (RNN, LSTM, GRU):**
  - Effective for sequence modeling but have key challenges:
    - Sequential Processing: Limited parallelization makes them slow to train.
    - Gradient Issues: Susceptible to exploding or vanishing gradients.
    - Long-Term Dependencies: Struggle to retain early context, leading to loss of critical information over long sequences.
- **Convolutional Approaches for Text:**
  - Faster than RNNs but limited by filter sizes and local context, making it challenging to capture long dependencies.

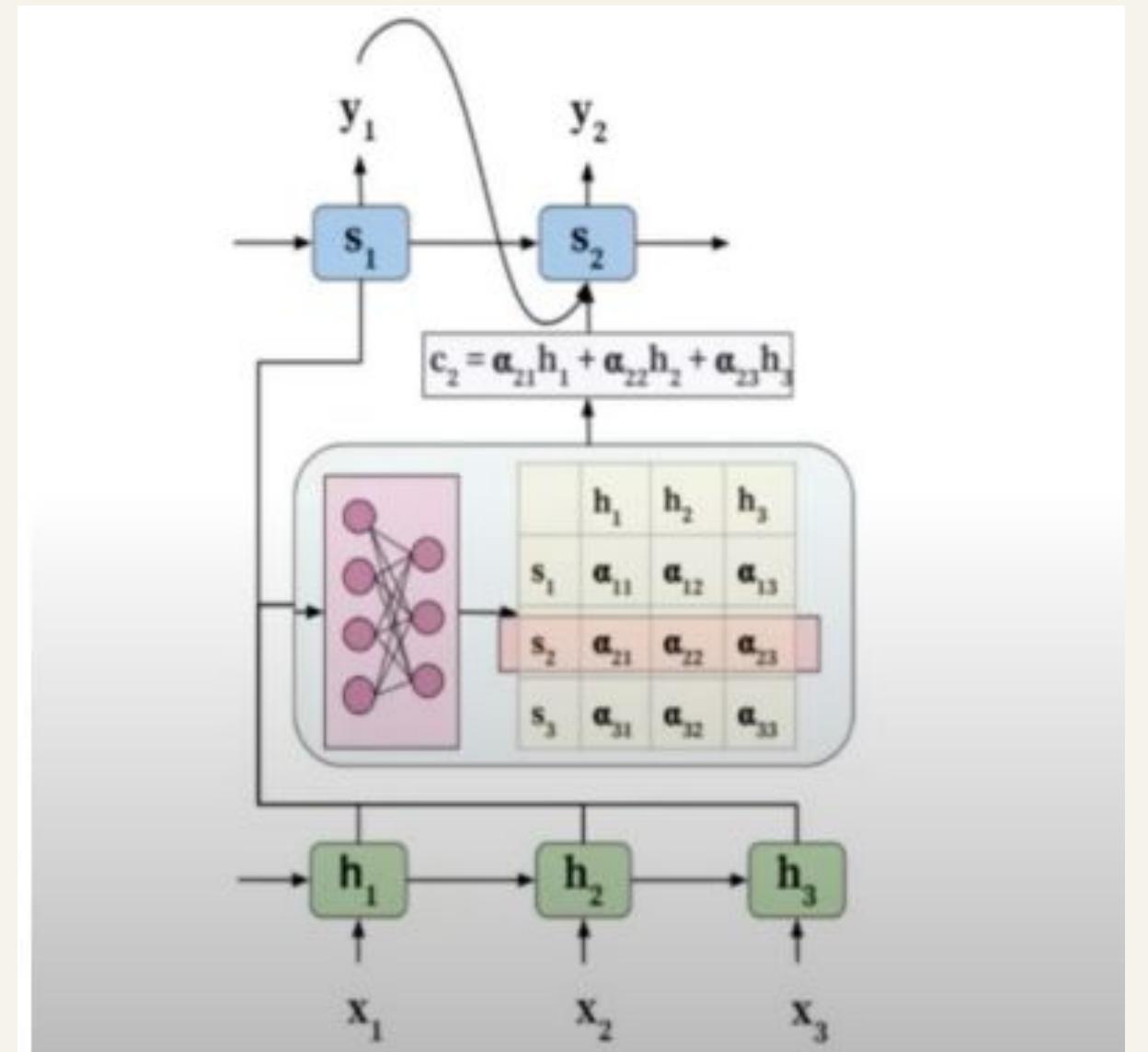


# Background

## Attention Mechanisms and Evolution

### Attention Mechanisms:

- Bahdanau et al. (2014) introduced an attention-based model for neural machine translation:
  - **Architecture:** Bidirectional RNN encoder, RNN decoder with an attention mechanism in between.
  - **Function:** Focuses on relevant hidden states for each output token.
  - **Benefit:** Improved translation accuracy, but still reliant on RNN structures, limiting parallelization.
- Transformers' Innovation:
  - Fully attention-based architecture, eliminating recurrence and convolution.
  - Directly models relationships across all tokens, achieving superior parallelization and efficiency.



Encoder-Decoder Model with  
Attention

# Context and Motivation

## The Problem:

- Traditional models like RNNs, LSTMs, and GRUs excel at sequence modeling but face key issues:
  - **Slow and Inefficient:** Sequential processing limits parallelization, making training resource-intensive.
  - **Gradient Challenges:** Prone to exploding or vanishing gradients, affecting stability.
  - **Long-Term Dependencies:** Struggle to retain early sequence information due to reliance on the last hidden state.
- **Attention Mechanisms:**
  - **Captures Relationships:** Models dependencies across tokens regardless of distance.
  - **Independence from RNNs:** The Transformer leverages attention alone, redefining NLP capabilities and setting new performance benchmarks.
- **Key Takeaway:**
  - "Attention is All You Need" led to the **Transformer**—revolutionizing sequence modeling through efficient, parallelizable attention mechanisms.

"In CSCE -689 today, Professor Huang assigned us to read the Attention is All You Need Paper. The paper proposes a new simple neural network architecture called the Transformer which relies upon attention. It outperforms RNNs and doesn't rely on recurrence or convolution. <Additional unrelated text>. I am so happy that \_\_\_ picked this paper."

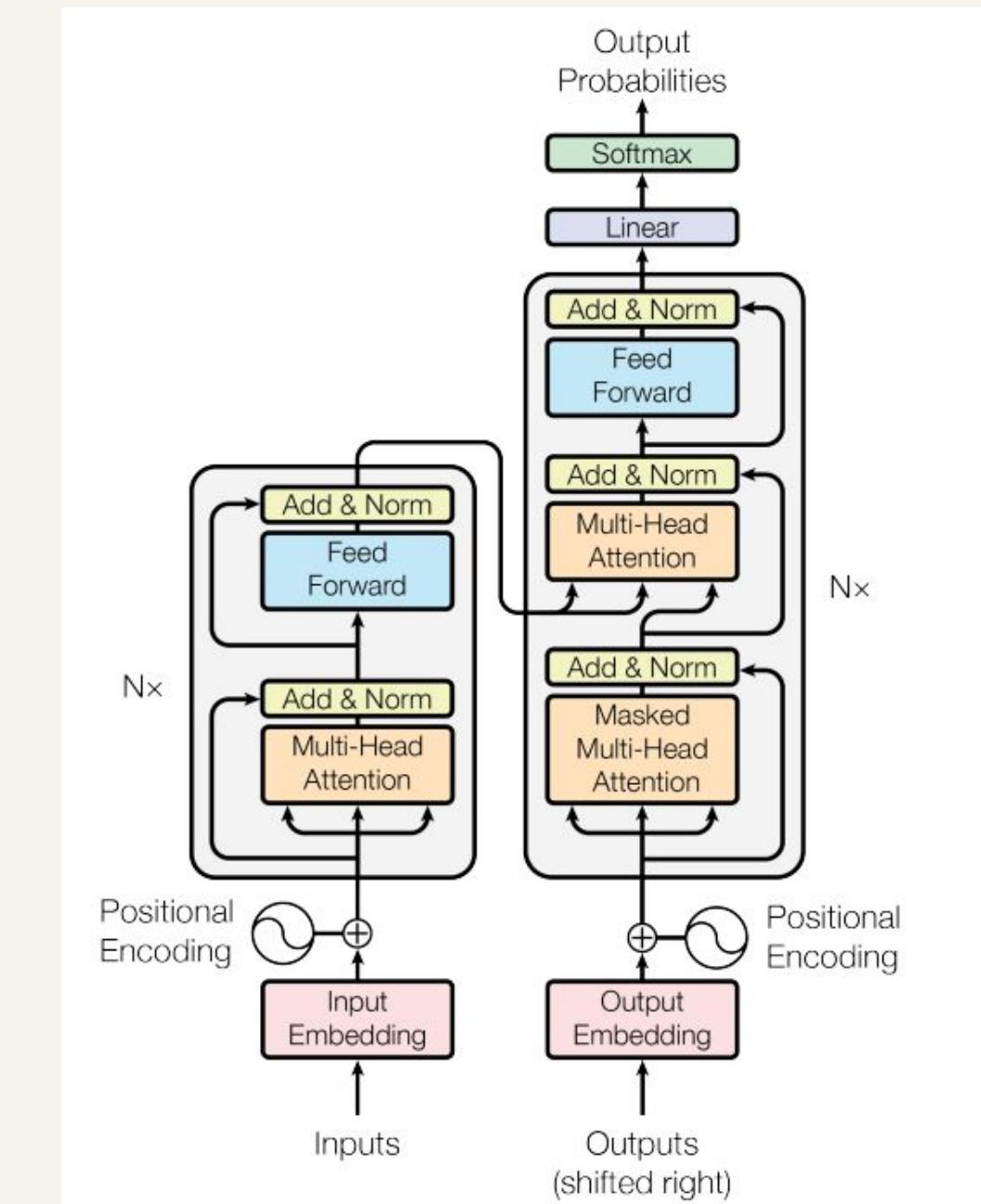
How would a RNN, LSTM, or GRU fill in the word mask?

1. Encode name inside cell state
2. Transfer cell state from one word to another
3. Decode name from previous cell state

Definitely not ideal! There has to be a better way...

# Model

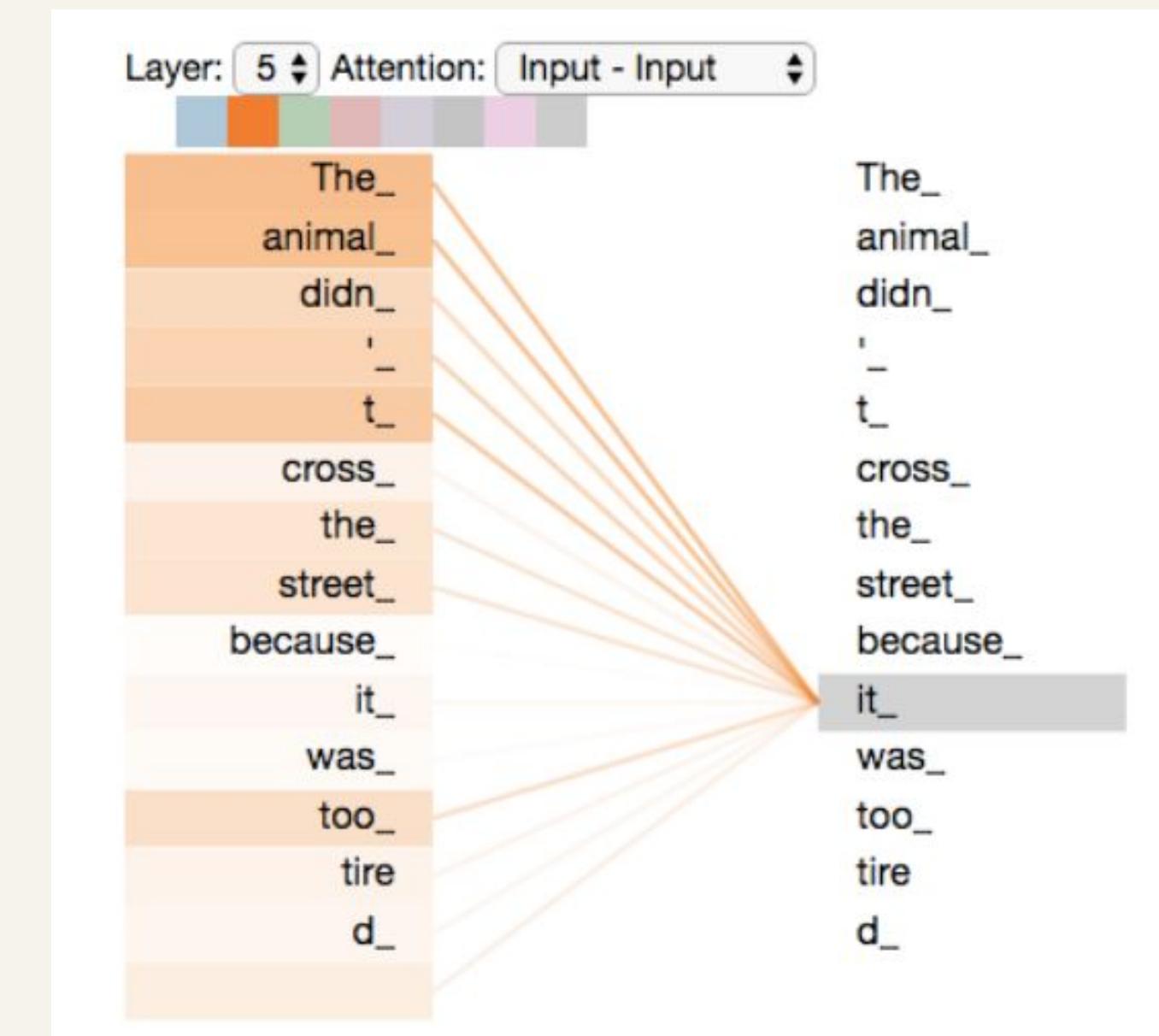
- Encoder is on the left;
  - N=6 blocks
  - Multi-head attention
  - Position-wise feed-forward network
- decoder is on the right.
  - Masked multi-head attention
    - Prevent positions from attending to subsequent positions
  - Same with Encoder
- These layers are stacked 6 times in the Transformer model.
- The decoder looks at the output of the encoder (and the previously generated words)



**The Transformer - model architecture.**

# Self-Attention

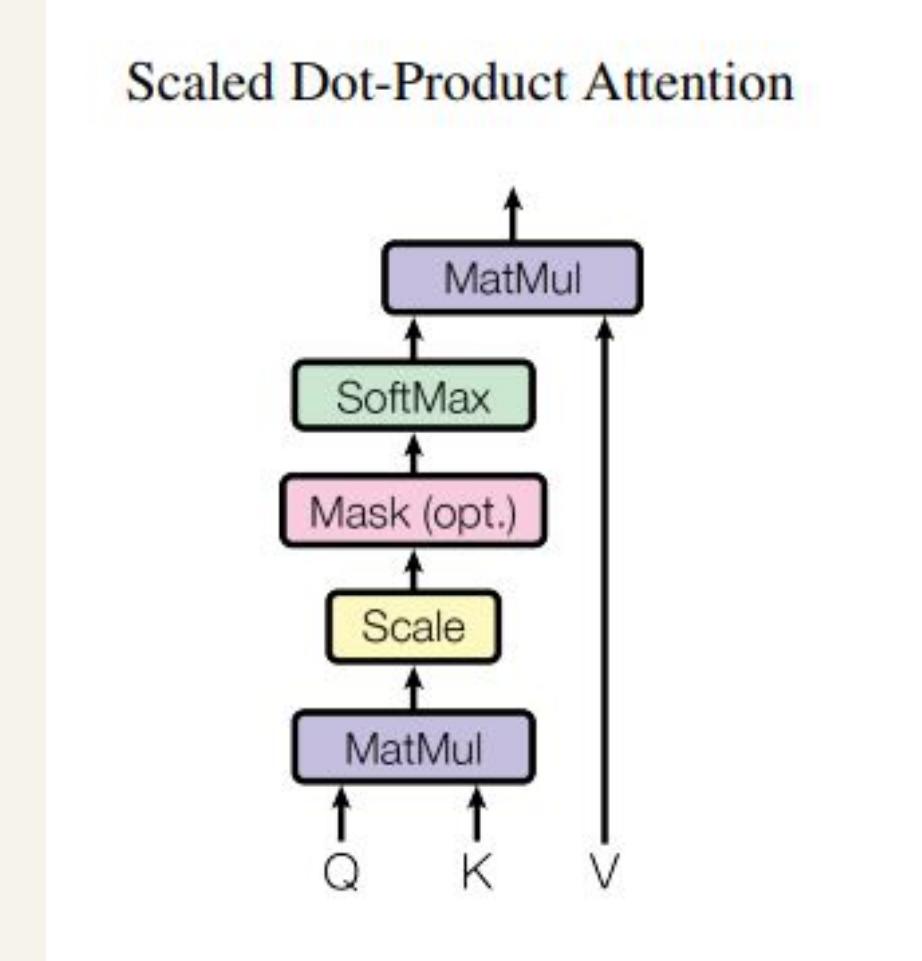
- Each element attends to every other element
- It is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence
- Each element becomes query, key, and value from the input embeddings by multiplying by a weight matrix



<https://jalammar.github.io/illustrated-transformer/>

# Scaled Dot-Product Attention

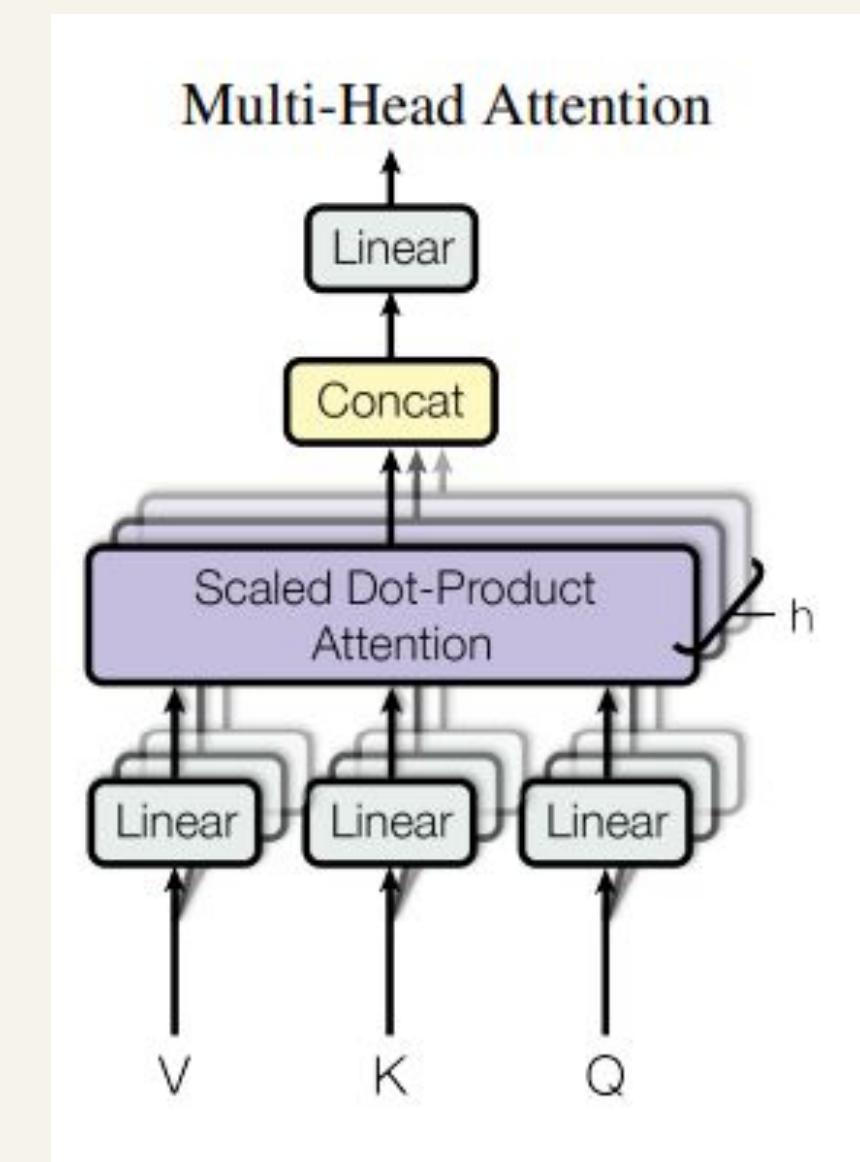
- Learn projections from input representation to
  - Query (Q) (dimension  $d_k$ )
  - Key (K) (dimension  $d_k$ )
  - Value (V) (dimension  $d_v$ )
- Matmul between Q and K are logits for how much attention is needed.
  - Attention weights: how likely each query matches key (used for weighted sum of values).
  - Softmax is used to compute weights to average the value representation.
- The paper introduces scaled dot-product attention •
  - Dot product attention (multiplicative) had been used without scaling.
  - Observation: Dot product grows too large in magnitude for large number of dimensions, so divide by  $\sqrt{d_k}$



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

# Multi-Head Attention

- Idea:
  - Stack linear layers (weight matrices without biases) that are independent each for keys, queries, values.
  - Concatenate output of attention heads to form (plus non-linearity) output layer
- Why?
  - Allows for model to focus on different positions
  - Gives attention layer multiple “representation subspaces”
  - No longer need to oversaturate one attention mechanism
- Averaging inhibits single-head attention from looking at different representation subspaces
- Instead, split single-attention into multiple attentions!
  - Each attention head is computed in parallel



# Multi-Head Attention Cont.

## ■ Key points:

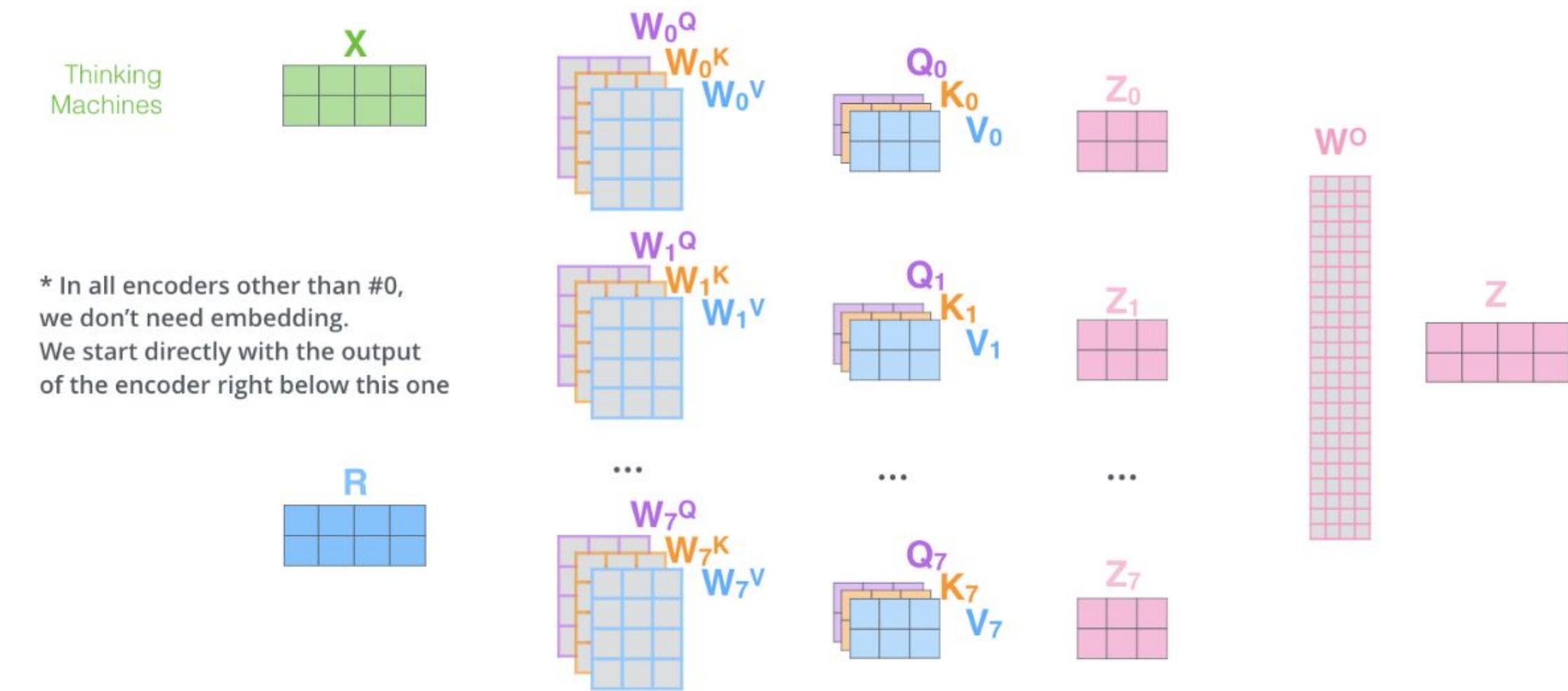
- We calculate 8 different attention heads, but we need to combine them
- Attention heads are independent of each other

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- $h = 8$  attention heads are used in Transformer.

- 1) This is our input sentence\*
- 2) We embed each word\*
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



<https://jalammar.github.io/illustrated-transformer/>

# Positional Encoding

- Need for information about the position and order of tokens in a sequence
- Positional Embedding: Vector that represents position of each token
  - Element wise addition the position embedding to the word embedding vector
  - Positional embedding be fixed or learned

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset  $k$ ,  $P$

$E_{pos}+k$  can be represented as a linear function of  $P$   
E<sub>pos</sub>."

For every sine-cosine pair corresponding to frequency  $\omega_k$ , there is a linear transformation  $M \in \mathbb{R}^{2 \times 2}$  (independent of  $t$ ) where the following equation holds:

$$M \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (t + \phi)) \\ \cos(\omega_k \cdot (t + \phi)) \end{bmatrix}$$

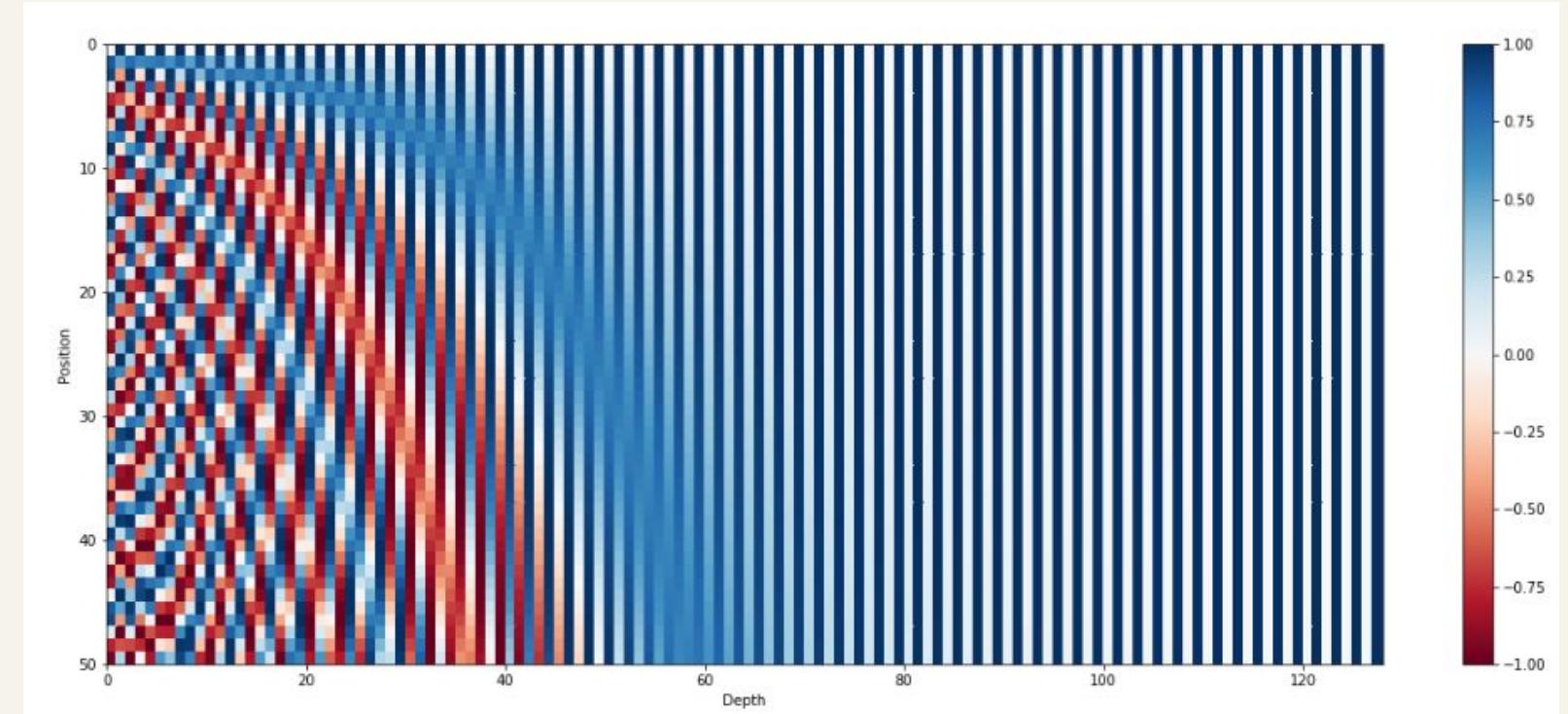
So the final transformation matrix  $M$  is

$$M_{\phi,k} = \begin{bmatrix} \cos(\omega_k \cdot \phi) & \sin(\omega_k \cdot \phi) \\ -\sin(\omega_k \cdot \phi) & \cos(\omega_k \cdot \phi) \end{bmatrix}$$

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

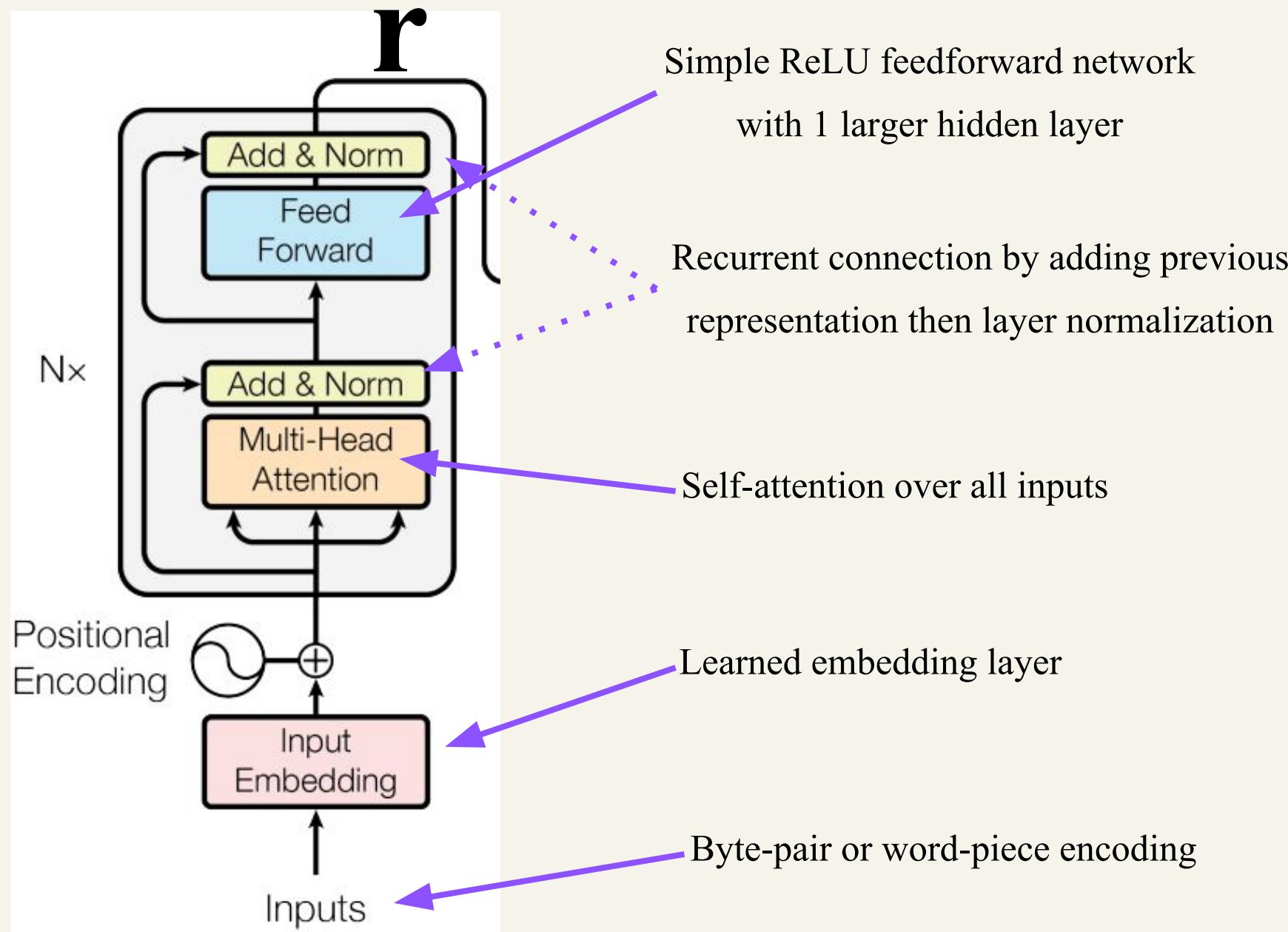
Pos is the position and  $i$  is the dimension



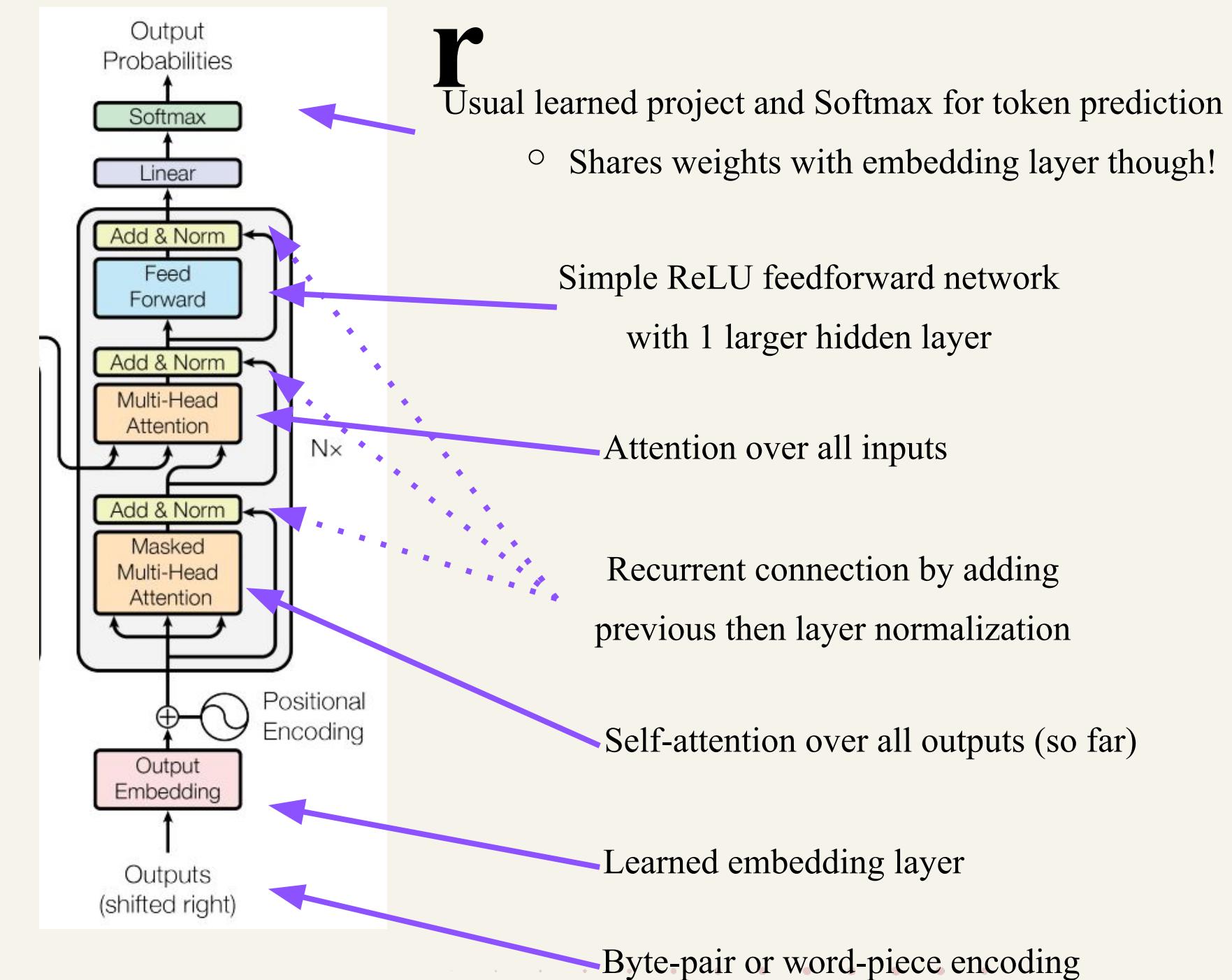
You can also imagine the positional embedding  $P_t \rightarrow$  as a vector containing pairs of sines and cosines for each frequency :

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

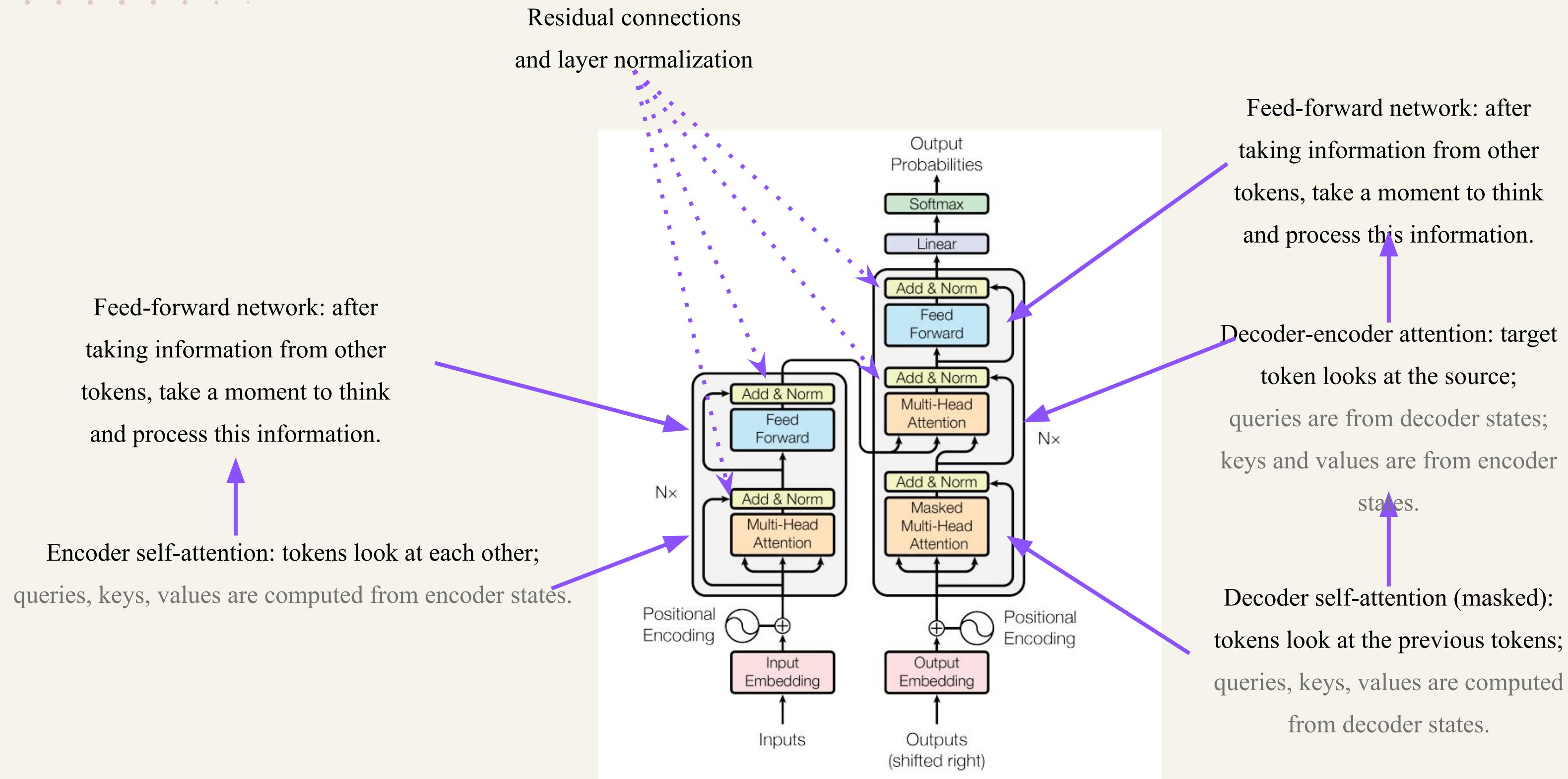
# Encode



# Decode



# FULL-ARCHITECTURE



# TRAINING

## Hardware:

- 8 NVIDIA P100 GPUs
- 30,000 steps (3.5 days)

## Optimizer

- Adam with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.98$  and  
 $\epsilon = 10^{-9}$

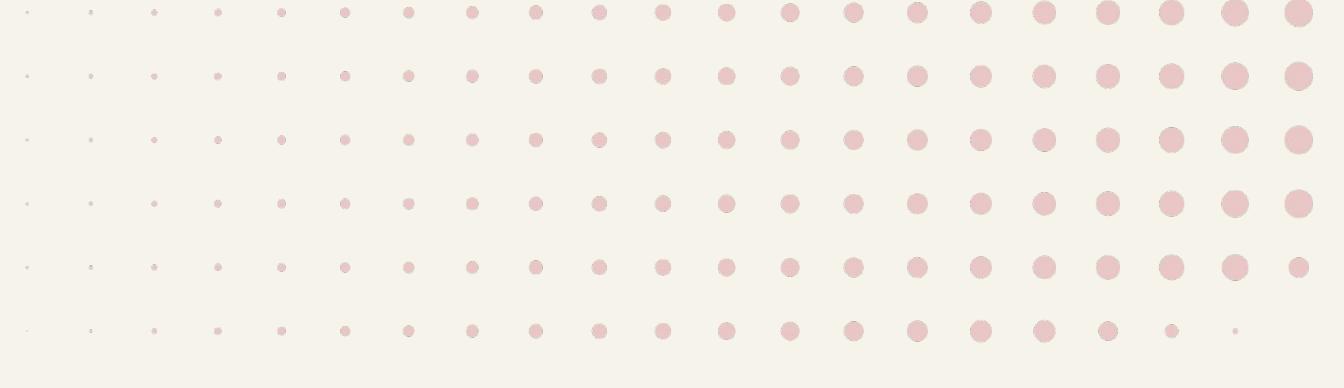
$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

## Dataset1: WMT 2014 English-German

- 4.5 million sentence (byte-pair encoded) pairs
- vocabulary: 37000 tokens

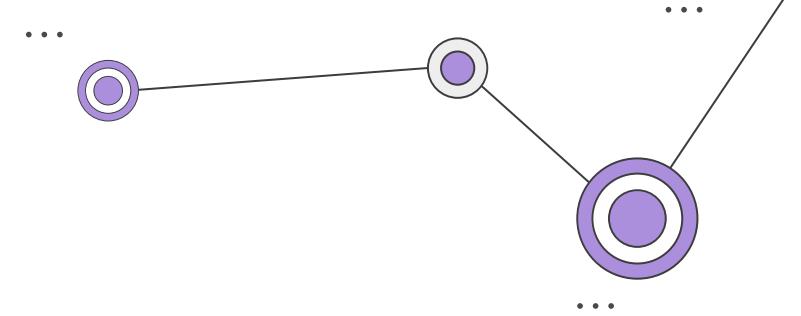
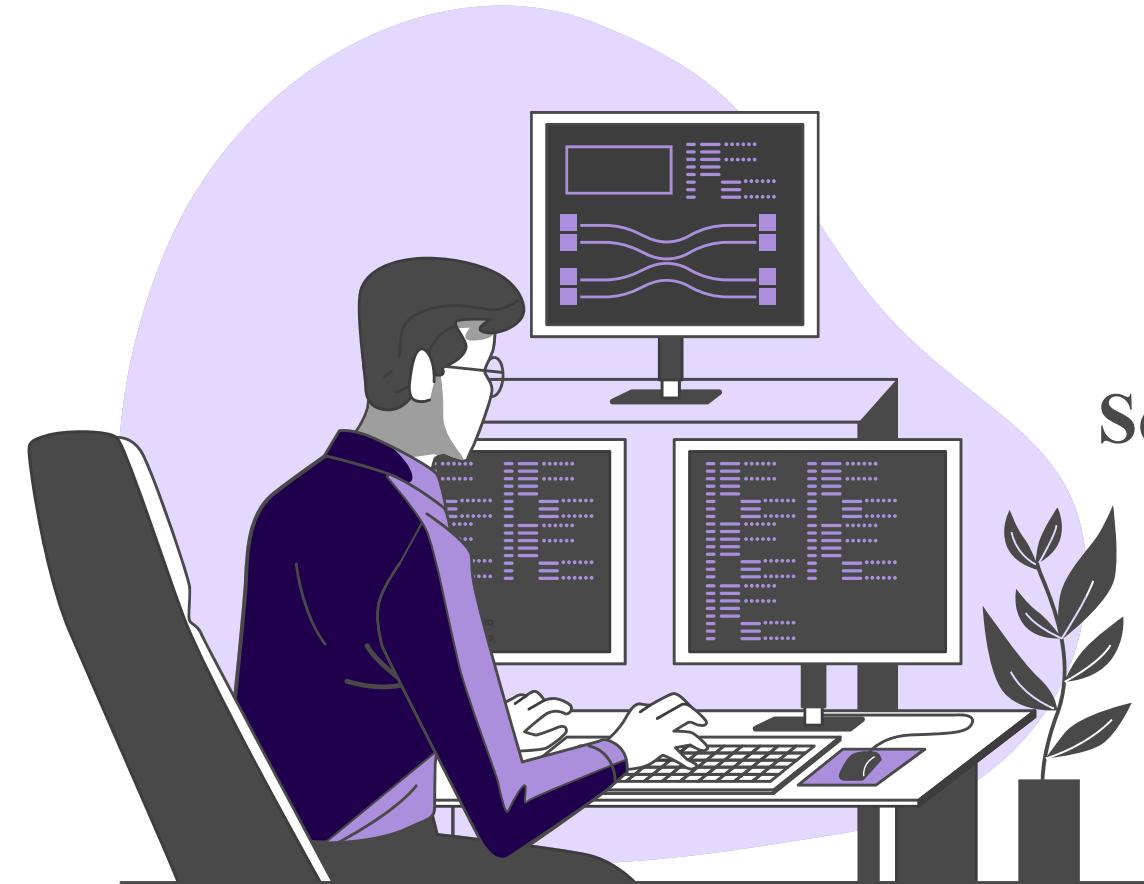
## Dataset2: WMT 2014 English-French

- 36M sentence pairs
- vocabulary size: 32000 tokens



Texas A&M University | 2024

# THANK YOU



Fall 24 CSCE 689

# Scaling Laws for Neural Language Models

Presented by:

Vishal Vardhan Adepu  
Venkat Pitta



# 1. Agenda



1. Research Overview & Motivation
2. Key Findings Summary
3. The Power Law Relationship
4. Model Architecture Independence
5. Sample Efficiency
6. Optimal Training Strategy
7. Generalization Performance
8. Mathematical Relations
9. Conclusion

## 2. Research Overview & Motivation

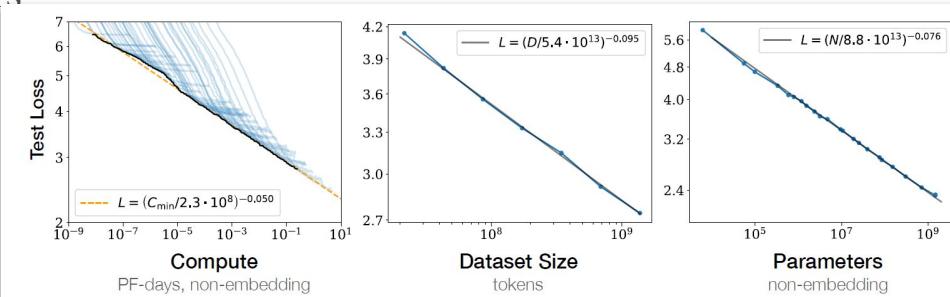
- Focus: Empirical scaling laws for language model performance.
  - Examines how performance improves with:
    - Model size (N)
    - Dataset size (D)
    - Compute power (C)
  - Performance governed by predictable power-law trends.
- Goal:
  - Establish universal principles for:
    - Efficient scaling of models.
    - Optimal allocation of compute resources.
    - Understanding the trade-offs between model size, data size, and compute.

# 3. Key Findings Summary

- Performance follows predictable power-law scaling:
  - Smooth power-law trends observed with model size (N), dataset size (D), and compute budget (C).
  - Extends across several orders of magnitude.
- Larger models are more sample-efficient:
  - Achieve equivalent performance with fewer optimization steps and less data.
- Minimal dependence on architectural details:
  - Performance depends more on overall scale than specifics like depth or width.
- Optimal training strategies:
  - Large models trained with moderate data and early stopping maximize compute efficiency.

# 4: The Power Law Relationship

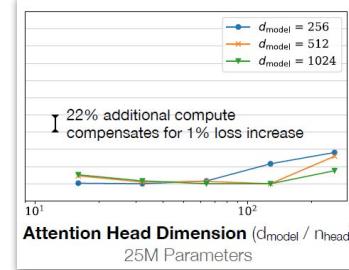
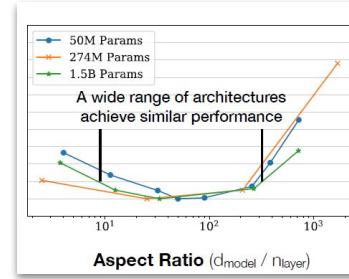
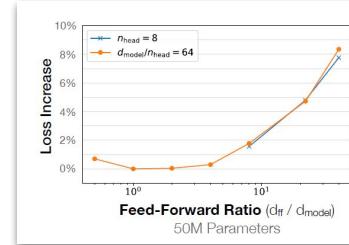
- Key Insight: Language modeling performance improves predictably with scale.
- Performance follows a power-law relationship with:
  - Compute (C): PF-days of training.
  - Dataset size (D): Total tokens.
  - Model size (N): Non-embedding parameters.
- Core Observations:
  - Trends span multiple orders of magnitude (see graph).
  - Performance scales predictably until bottlenecked by one of the three factors



**Figure 1** Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute<sup>2</sup> used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

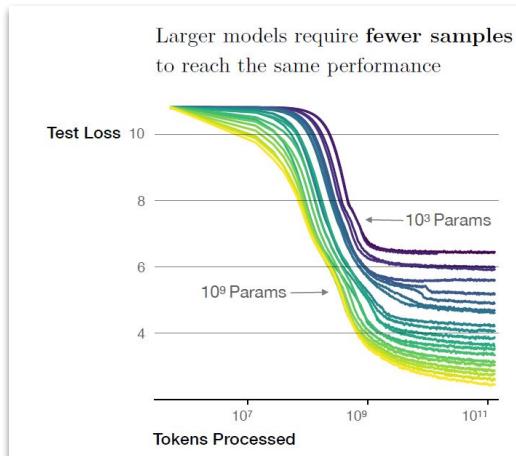
# 5: Model Architecture Independence

- Key Finding:
  - Performance depends minimally on model shape when total non-embedding parameters are fixed.
- Insights:
  - Variations in feed-forward ratio, aspect ratio, and attention head dimension cause only minor changes in loss (1-3%).
  - Compute can compensate for architectural changes:
    - Example: A 22% increase in compute offsets a 1% increase in loss.
- Implications:
  - Achieving optimal performance focuses on scaling model size, not fine-tuning architecture.



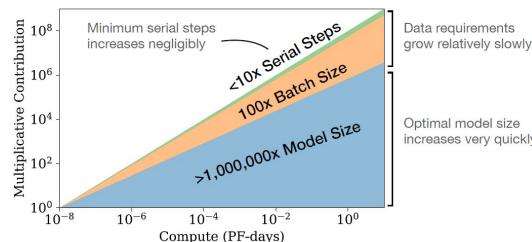
# 6: Sample Efficiency

- Key Insight: Larger models require fewer samples to achieve the same test loss.
  - Example: A  $10^9$  - parameter model needs significantly fewer tokens compared to a  $10^3$  - parameter model.
- Implications:
  - Training Efficiency: Larger models save compute time and resources during training.
  - Enables faster convergence without compromising performance.



# 7: Optimal Training Strategy

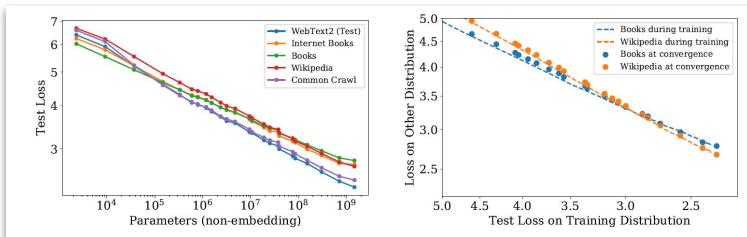
- Key Insights:
  - Optimal training involves:
    - Scaling model size for better performance.
    - Using larger batch sizes to improve parallelism.
    - Stopping training before full convergence to save compute.
- Recommendations:
  - Focus on scaling models rather than excessively increasing data or training steps.
  - Ensure training remains compute-efficient by balancing model size and batch size.



**Figure 3** As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. We illustrate this for a billion-fold increase in compute. For optimally compute-efficient training, most of the increase should go towards increased model size. A relatively small increase in data is needed to avoid reuse. Of the increase in data, most can be used to increase parallelism through larger batch sizes, with only a very small increase in serial training time required.

# 8: Generalization Performance

- Key Insight:
  - Smooth improvement across distributions: Model generalization improves predictably with model size.
- Findings:
  - Loss on new datasets is closely correlated with training loss.
  - A small, constant offset is observed when evaluating on out-of-distribution data.
- Implications:
  - Larger models transfer better across diverse datasets.
  - Generalization depends on training performance, not the duration or phase of training.



**Figure 8** **Left:** Generalization performance to other data distributions improves smoothly with model size, with only a small and very slowly growing offset from the WebText2 training distribution. **Right:** Generalization performance depends only on training distribution performance, and not on the phase of training. We compare generalization of converged models (points) to that of a single large model (dashed curves) as it trains.

# 9: Mathematical Relations

Parameters	Data	Compute	Batch Size	Equation
$N$	$\infty$	$\infty$	Fixed	$L(N) = (N_c/N)^{\alpha_N}$
$\infty$	$D$	Early Stop	Fixed	$L(D) = (D_c/D)^{\alpha_D}$
Optimal	$\infty$	$C$	Fixed	$L(C) = (C_c/C)^{\alpha_C}$ (naive)
$N_{\text{opt}}$	$D_{\text{opt}}$	$C_{\min}$	$B \ll B_{\text{crit}}$	$L(C_{\min}) = (C_c^{\min}/C_{\min})^{\alpha_C^{\min}}$
$N$	$D$	Early Stop	Fixed	$L(N, D) = \left[ \left( \frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D}$
$N$	$\infty$	$S$ steps	$B$	$L(N, S) = \left( \frac{N_c}{N} \right)^{\alpha_N} + \left( \frac{S_c}{S_{\min}(S, B)} \right)^{\alpha_S}$

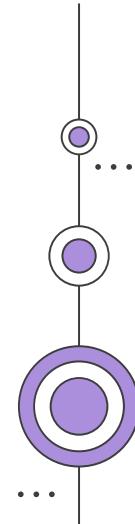
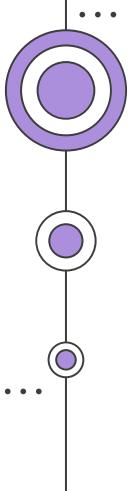
**Table 4**

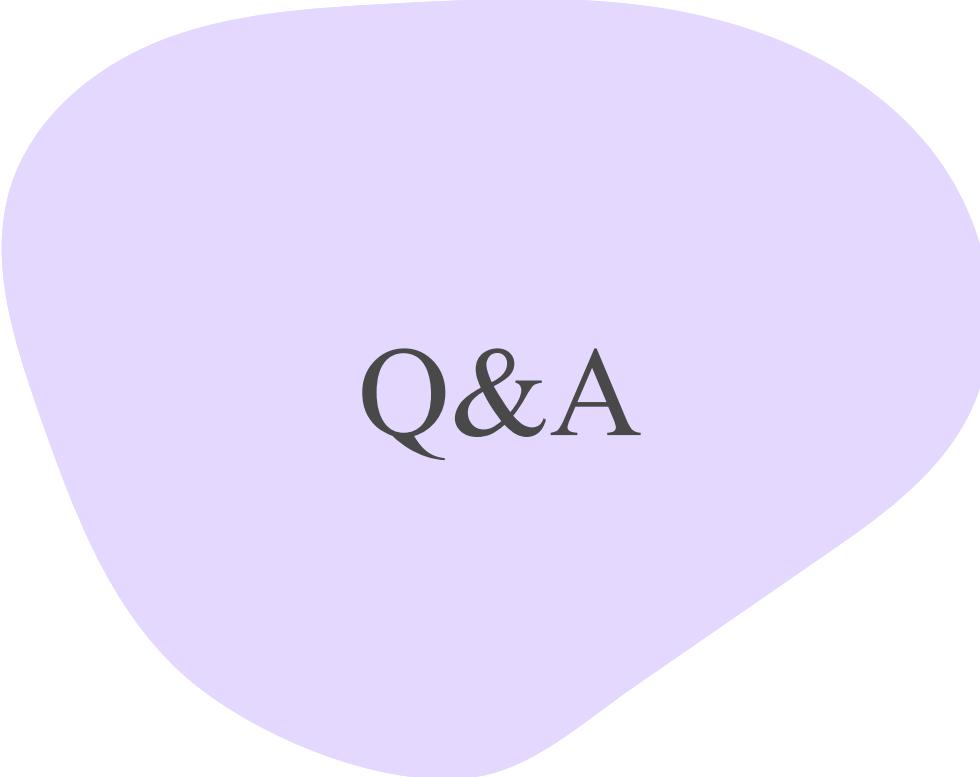
# 9: Conclusion

- Larger models consistently outperform smaller ones, highlighting the importance of scale.
- Sample efficiency increases significantly with model size, requiring fewer data points for comparable performance.
- Model architecture has a minimal impact when scaled appropriately.
- These findings suggest a clear path for future AI development: prioritize scaling up models while optimizing data and compute use.

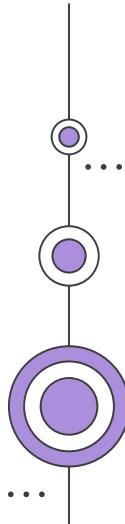
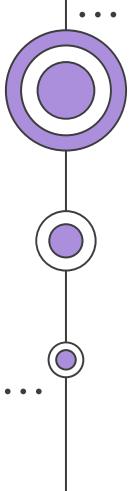


Thank  
you!





Q&A



# *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*

Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter,  
Fei Xia, Ed H. Chi, Quoc V. Le, Denny Zhou

NIPS 2022

Jing Cao, Sushil Vemuri 11/19/2024



TEXAS A&M UNIVERSITY  
Engineering

# Introduction

- NLP landscape revolutionized by language models.
- Scaling up language model size improves performance and sample efficiency.
- Scaling alone is **insufficient** for high performance **on tasks like arithmetic, commonsense, and symbolic reasoning**.

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. X

# *Motivating Work*

## **Current Approaches for Natural Language Rationale Generation:**

- Training from scratch
- Fine-tuning a pretrained model
- Neuro-symbolic methods using formal languages
- Costly to create a large set of high quality rationales

## **Few-Shot Learning via Prompting:**

- Large language models allow for in-context learning without fine-tuning a separate model checkpoint.
- A few input-output examples demonstrate the task.

# *The Idea*

**Maybe generating natural language rationales helps in arithmetic reasoning ?**

**Can we make models generate intermediate steps leading to the final answer ?**

# Method

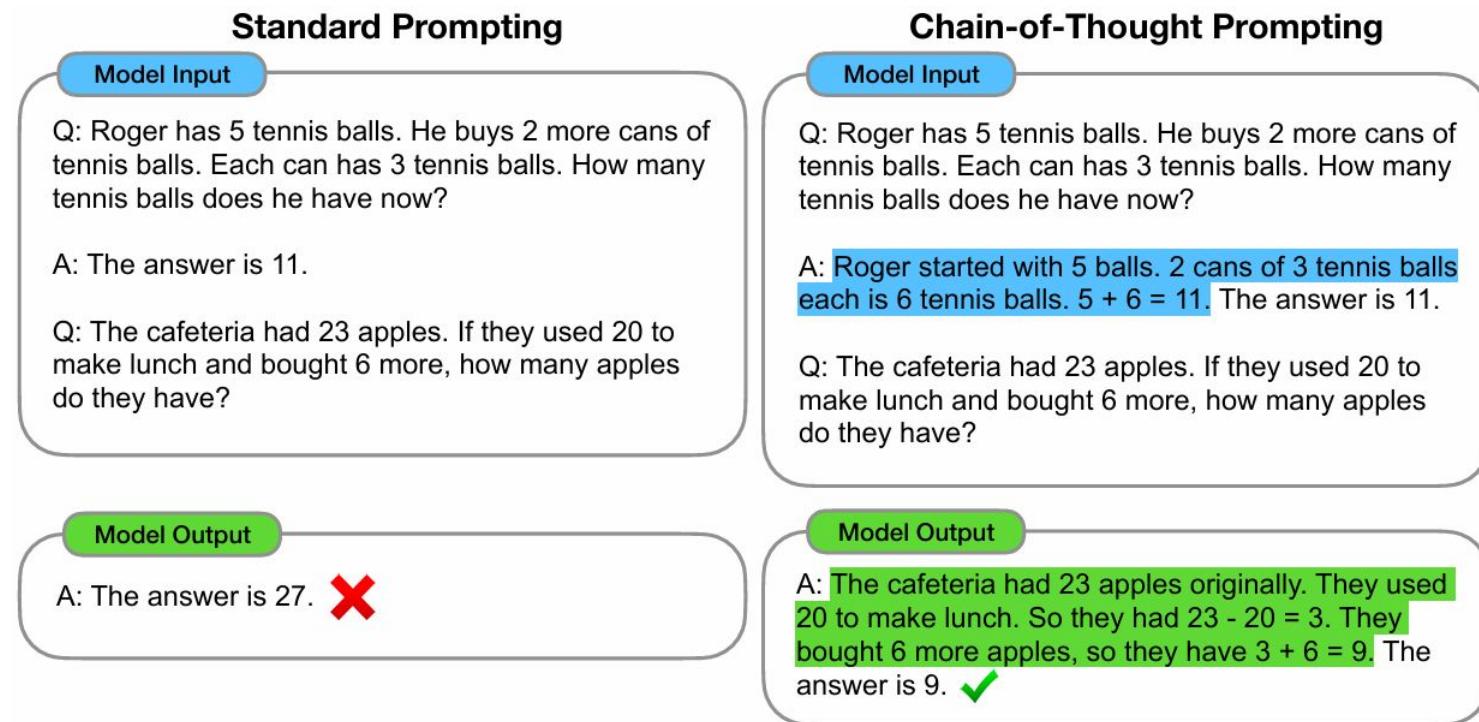


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

# *Experimental setup*

## **Benchmark:**

- **Math problem** benchmarks in different difficulty level
- **commonsense reasoning** benchmarks in diverse range
- **Symbolic Reasoning tasks**: easy for human, but challenge for llms

## **Model Selections:**

Evaluate five large language models in **different scales**. For example, they use GPT-3 models of 350M, 1.3B, 6.7B, and 175B parameters.

# Control group

- Standard few-shot prompting
- Chain of thought prompting  
(highlighted text)

<p><b>Math Word Problems (free response)</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. <math>5 + 6 = 11</math>. The answer is 11.</p>	<p><b>Math Word Problems (multiple choice)</b></p> <p>Q: How many keystrokes are needed to type the numbers from 1 to 500? Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788</p> <p>A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. <math>9 + 90(2) + 401(3) = 1392</math>. The answer is (b).</p>	<p><b>CSQA (commonsense)</b></p> <p>Q: Sammy wanted to go to where the people were. Where might he go? Options: (a) race track (b) populated areas (c) desert (d) apartment (e) roadblock</p> <p>A: The answer must be a place with a lot of people. Race tracks, desert, apartments, and roadblocks don't have a lot of people, but populated areas do. So the answer is (b).</p>
<p><b>StrategyQA</b></p> <p>Q: Yes or no: Would a pear sink in water?</p> <p>A: The density of a pear is about 0.6 g/cm<sup>3</sup>, which is less than water. Thus, a pear would float. So the answer is no.</p>	<p><b>Date Understanding</b></p> <p>Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?</p> <p>A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.</p>	<p><b>Sports Understanding</b></p> <p>Q: Is the following sentence plausible? "Joao Moutinho caught the screen pass in the NFC championship."</p> <p>A: Joao Moutinho is a soccer player. The NFC championship is part of American football, not soccer. So the answer is no.</p>
<p><b>SayCan (Instructing a robot)</b></p> <p>Human: How would you bring me something that isn't a fruit?</p> <p>Explanation: the user wants something to eat that isn't a fruit. An energy bar is not a fruit, so I will bring the user an energy bar.</p> <p>Plan: 1. find(energy bar) 2. pick(energy bar) 3. find(user) 4. put(energy bar) 5. done().</p>	<p><b>Last Letter Concatenation</b></p> <p>Q: Take the last letters of the words in "Lady Gaga" and concatenate them.</p> <p>A: The last letter of "Lady" is "y". The last letter of "Gaga" is "a". Concatenating them is "ya". So the answer is ya.</p>	<p><b>Coin Flip (state tracking)</b></p> <p>Q: A coin is heads up. Maybelle flips the coin. Shalonda does not flip the coin. Is the coin still heads up?</p> <p>A: The coin was flipped by Maybelle. So the coin was flipped 1 time, which is an odd number. The coin started heads up, so after an odd number of flips, it will be tails up. So the answer is no.</p>

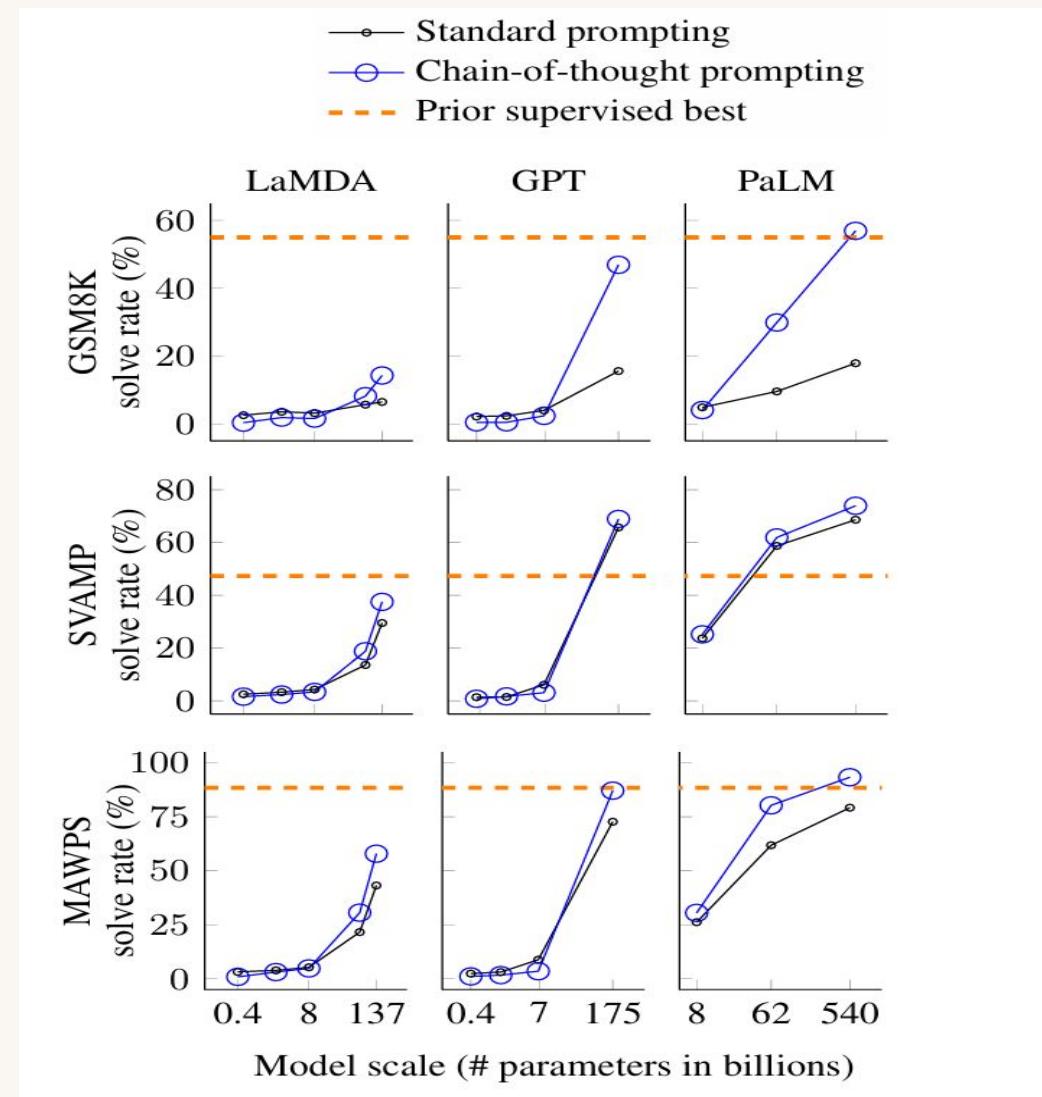
Figure 3: Examples of  $\langle$ input, chain of thought, output $\rangle$  triples for arithmetic, commonsense, and symbolic reasoning benchmarks. Chains of thought are highlighted. Full prompts in Appendix G.

# The results and evaluations

## 1. Math Problems

**GSM8K** is the hardest benchmark with lowest baseline performance.

**MAWPS** is the easiest benchmark which only requires a single step to solve.



# The results and evaluations

## 2. commonsense reasoning tasks

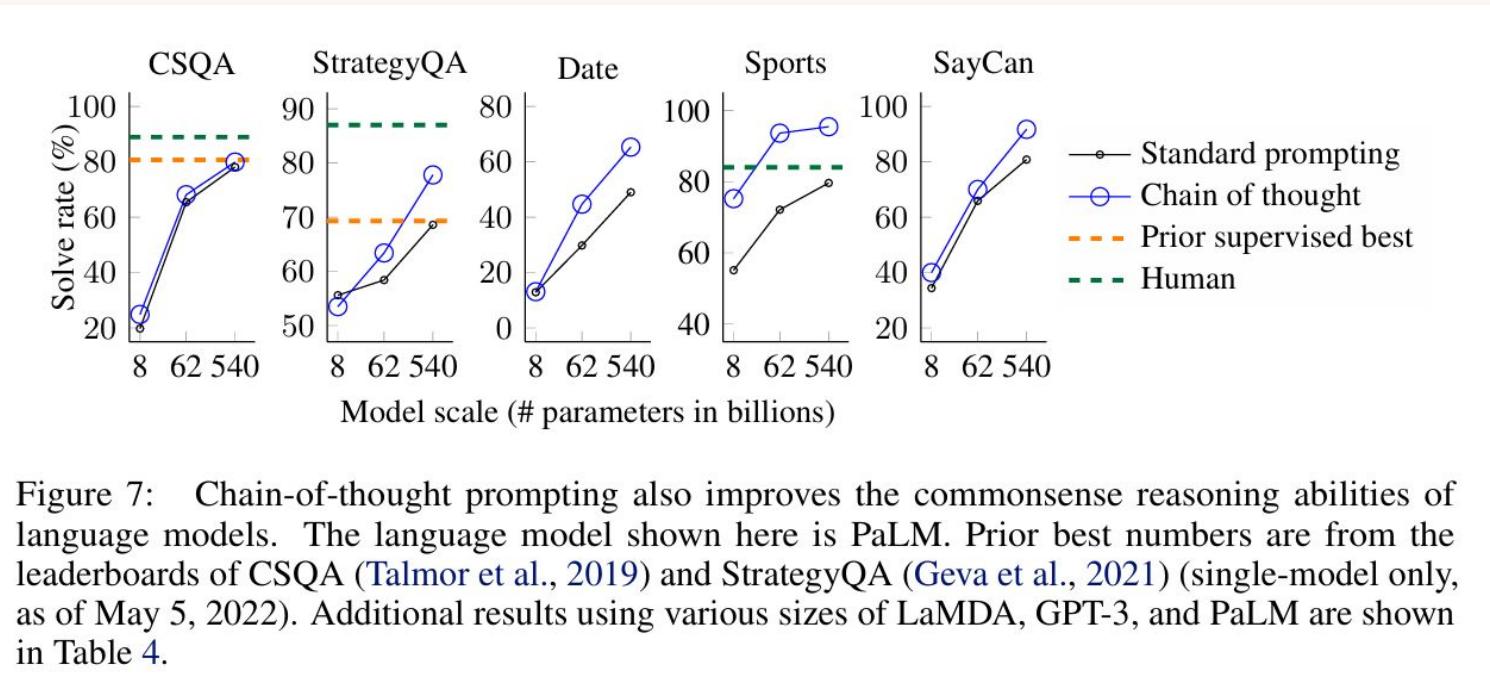


Figure 7: Chain-of-thought prompting also improves the commonsense reasoning abilities of language models. The language model shown here is PaLM. Prior best numbers are from the leaderboards of CSQA (Talmor et al., 2019) and StrategyQA (Geva et al., 2021) (single-model only, as of May 5, 2022). Additional results using various sizes of LaMDA, GPT-3, and PaLM are shown in Table 4.

- **CSQA:** Commonsense questions about the world involving complex semantics that require prior knowledge.
- **StrategyQA:** requires models to infer a multi-hop strategy to answer questions.
- **Date:** which involves inferring a date from a given context.
- **Sports:** Involves determining whether a sentence relating to sports is plausible or not.
- **SayCan:** involves mapping a natural language instruction to a sequence of robot actions

# The results and evaluations

## 3. Symbolic Reasoning tasks

1). Asks the model to **concatenate the last letters of words** in a name.

(e.g., “Amy Brown” “yn”)

2) **Coin flip**. This task asks the model to answer whether a coin is still heads up after people either flip or don't flip the coin.

(e.g., “A coin is heads up. Tom flips the coin. Jack does not flip the coin. Is the coin still heads up?” “no”).

**in-domain:** evaluation test cases have the **same number** of steps as the prompt example.

**OOD:** evaluation test cases have **more number** of steps than the prompt example.

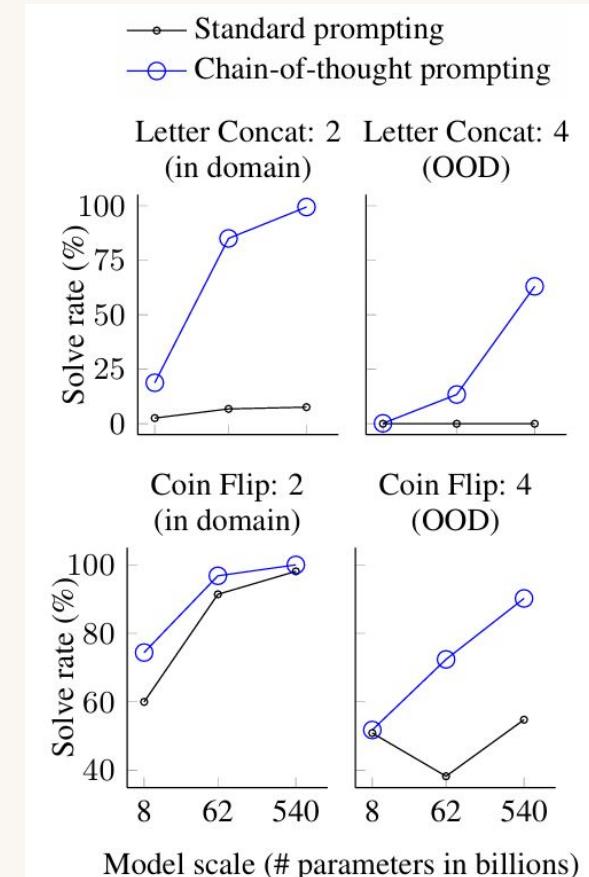


Figure 8: Using chain-of-thought prompting facilitates generalization to longer sequences in two symbolic reasoning tasks.

# *The results and evaluations*

**Compare to standard prompt:** standard prompting has a flat scaling curve (scaling law), but chain of-thought prompting leads to dramatically increasing scaling curves.

**Scale:** chain-of-thought prompting **works better on large scale models** exceed 100B parameters. The possible reason is models of smaller scale produced fluent but **illogical** chains of thought, leading to lower improvement

**Commonsense:** chain-of-thought prompting can also improve performance on tasks requiring a range of commonsense reasoning abilities.

**Symbolic Reasoning:** chain-of-thought prompting can facilitates length generalization and significantly improve the performance on long symbolic reasoning.

# Robustness

## 1. Varying Exemplar Sets

- Works effectively with different sets of exemplars, even if they come from independent sources or are not from the same dataset distribution as the test examples.

## 2. Exemplar Order

- Performance remains stable even when the order of exemplars changes.
- Notable exception is coin-flip task, where exemplar order influences performance.

## 3. Numbers of Examples

- Performance improvements hold even with a varying number of exemplars.
- No significant improvement when the number of exemplars is increased beyond a certain point.

## 4. Multiple Annotators

- Chain-of-thought prompting works well regardless of who writes the reasoning steps, even if different annotators are involved.

## 5. Writing Style

- The model benefits from chain-of-thought prompting, regardless of linguistic style, whether the annotations are detailed or concise.
- Success does not depend on a particular writing style or complexity.

## 6. Prompt Engineering Still Matters

- Chain-of-thought prompting is robust but still sensitive to how prompts are crafted.
- Variability exists in performance, underscoring the importance of fine-tuning prompt engineering for optimal results.

# *Critical thinking*

- Chain-of-thought reasoning works better at models in large scales, makes it **expensive** to serve in real-world applications; We could explore how to bring efficient reasoning in smaller models.
- Simple prompt adjustments (rather than **model changes**) can significantly improve reasoning is innovative and applicable to my project focused on optimizing LLMs.
- Applying this prompting technique to other domains, such as AI tutors or medical diagnosis, could explore its utility in broader areas by **providing reasoning step-by-step**.



---

TEXAS A&M UNIVERSITY  
Engineering

**THANK YOU**

# Paper Review: ReAct

CSCE 689 Programming LLM  
Yuanlong Zhou, Shuning Gu

# Menu

- Introduction
- Contributions
- Methodology
- Results and Findings



# Introduction

**ReAct:** A paradigm that combines reasoning and action in LLMs to enhance task solving

**Limitations** of traditional methods in LLM reasoning and acting :

- Chain-of-thought Reasoning: lacks interaction with the external world and leads to hallucination
- Transforming multi-modal observations into text: doesn't enable the models to perform abstract reasoning to aid in action.

# Introduction

## (1) Hotspot QA

**Question:** Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

### (1a) Standard

**Answer:** iPod



### (1b) CoT (Reason Only)

**Thought:** Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

**Answer:** iPhone, iPad, iPod Touch



### (1c) Act-Only

**Act 1:** `Search[Apple Remote]`

**Obs 1:** The Apple Remote is a remote control ...

**Act 2:** `Search[Front Row]`

**Obs 2:** Could not find [Front Row]. Similar: ...

**Act 3:** `Search[Front Row (software)]`

**Obs 3:** Front Row is a discontinued media center software ...

**Act 4:** `Finish[yes]`



## (1d) ReAct (Reason + Act)

**Thought 1:** I need to search Apple Remote and find the program it was originally designed to interact with.

**Act 1:** `Search[Apple Remote]`

**Obs 1:** The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

**Thought 2:** Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

**Act 2:** `Search[Front Row]`

**Obs 2:** Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

**Thought 3:** Front Row is not found. I need to search Front Row (software) .

**Act 3:** `Search[Front Row (software)]`

**Obs 3:** Front Row is a discontinued media center software ...

**Thought 4:** Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.

**Act 4:** `Finish[keyboard function keys]`



# Introduction

## (2) AlfWorld

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 1, a coffee machine 1, a countertop 3, a stove burner 1, and a toaster 1.

**Your task is to:** Put some pepper shaker on a drawer.

## (2a) Act-Only

**Act 1:** Go to drawer 1

**Obs 1:** The drawer 1 is closed.

**Act 2:** Open drawer 1

**Obs 2:** You open the drawer 1. The drawer 1 is open. In it, you see a dishponge 2, and a spoon 1.

**Act 3:** Go to sinkbasin 1

**Obs 3:** On the sinkbasin 1, you see a dishponge 3, a spatula 1, and a spoon 2.

**Act 4:** Take peppershaker 1 from sinkbasin 1

**Obs 4:** Nothing happens.

**Act 5:** Take peppershaker 1 from sinkbasin 1

**Obs 5:** Nothing happens.



## (2b) ReAct (Reason + Act)

**Act 1:** **Think**[First I need to find a pepper shaker...more likely to appear in cabinets (1-6), countertops (1-3), ... ]

**Act 2:** Go to cabinet 1

**Obs 2:** On the cabinet 1, you see a vase 2.

(Here the agent go to cabinet 1, then cabinet 2, 3, then countertop 1 and 2)

**Act 6:** Go to countertop 3

**Obs 6:** On the countertop 3, you see a apple 1, a bread 1, ..... a pepper shaker 1, and a vase 3.

**Act 7:** Take pepper shaker 1 from countertop 3

**Obs 7:** You pick up the pepper shaker 1 from the countertop 3.

**Act 8:** **Think**[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]

**Act 9:** Go to drawer 1

**Obs 9:** Drawer 1 is closed.

**Act 10:** Open drawer 1

**Obs 10:** You open Drawer 1 ...

**Act 11:** Put pepper shaker 1 in/on drawer 1

**Obs 11:** You put pepper shaker 1 in/on the drawer 1.





# Contributions

- Introduction of ReAct
- Demonstrating effectiveness of ReAct over state-of-the-art baselines in several benchmarks
- Systematic exploration of the interplay between reasoning and acting
- analyzing the limitations of ReAct under the prompting setup and perform initial finetuning experiments



# Methodology: ablate study

Select baseline models from these aspects:

- Standard prompt: remove all thoughts, actions, observations
- Chain-of-thought prompt (CoT): remove actions and observations, only keep reasoning.
  - Self consistent COT (CoT-SC): sampling a number of CoT and adopt the majority answer to have boosted performance
- Acting-only prompt (Act): remove thoughts



# Methodology: test cases

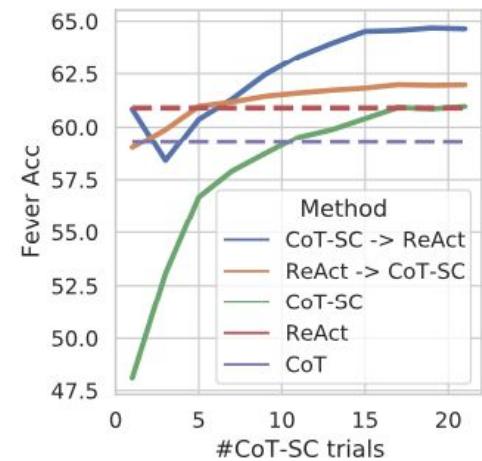
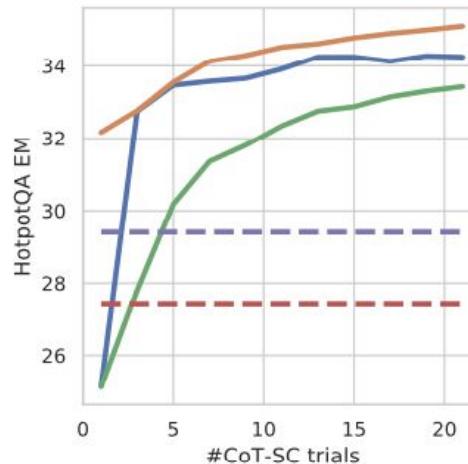
Test ReAct on two categories of task benchmarks:

- Knowledge-intensive reasoning tasks
  - HotPotQA: multi-hop question answering benchmark
  - FEVER: a fact verification benchmark
- Decision-making tasks
  - ALFWorld: text-based game
  - WebShop: online shopping website



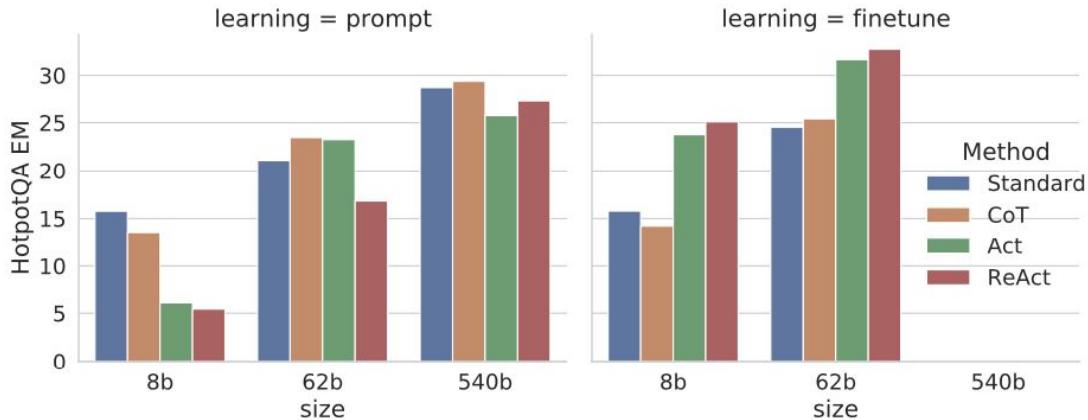
# Results

Prompt Method <sup>a</sup>	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	<b>64.6</b>
ReAct → CoT-SC	<b>35.1</b>	62.0
<b>Supervised SoTA<sup>b</sup></b>	67.5	89.5





# Results



	Type	Definition	ReAct	CoT
Success	True positive	Correct reasoning trace and facts	94%	86%
	False positive	Hallucinated reasoning trace or facts	6%	14%
Failure	Reasoning error	Wrong reasoning trace (including failing to recover from repetitive steps)	47%	16%
	Search result error	Search return empty or does not contain useful information	23%	-
	Hallucination	Hallucinated reasoning trace or facts	0%	56%
	Label ambiguity	Right prediction but did not match the label precisely	29%	28%



# Results

Method	Pick	Clean	Heat	Cool	Look	Pick 2	All
Act (best of 6)	88	42	74	67	72	<b>41</b>	45
ReAct (avg)	65	39	83	76	55	24	57
ReAct (best of 6)	<b>92</b>	58	<b>96</b>	86	<b>78</b>	<b>41</b>	<b>71</b>
ReAct-IM (avg)	55	59	60	55	23	24	48
ReAct-IM (best of 6)	62	<b>68</b>	87	57	39	33	53
BUTLER <sub>g</sub> (best of 8)	33	26	70	76	17	12	22
BUTLER (best of 8)	46	39	74	<b>100</b>	22	24	37

Table 3: AlfWorld task-specific success rates (%). BUTLER and BUTLER<sub>g</sub> results are from Table 4 of Shridhar et al. (2020b). All methods use greedy decoding, except that BUTLER uses beam search.

Method	Score	SR
Act	62.3	30.1
ReAct	<b>66.6</b>	<b>40.0</b>
IL	59.9	29.1
IL+RL	62.4	28.7
Human Expert	82.1	59.6

Table 4: Score and success rate (SR) on Webshop. IL/IL+RL taken from Yao et al. (2022).



# Conclusion

- ReAct successfully combines reasoning and acting
- ReAct is proven to be effective over baseline models:
  - Standard prompt, CoT, Act-only
- ReAct performs generally good on the benchmarks:
  - Knowledge-intensive reasoning tasks
  - Decision-making tasks
- ReAct performs best for fine-tuning and relies on accurate information retrieval

# Fast Inference from Transformers via Speculative Decoding

*Authors: Yaniv Leviathan, Matan Kalman, Yossi Matias*

Presented By-

Arunim Samudra  
Surya Dhaneshwar L

# Problem

- Single decoding step in large models is slow
- Decoding is sequential—decoding  $K$  tokens requires  $K$  serial steps



# Problem

- Latency becomes a bottleneck for real-time applications

# Core Idea

70 B Model

The Wright brothers made their first successful flight in the ..



Easy!

70 B Model

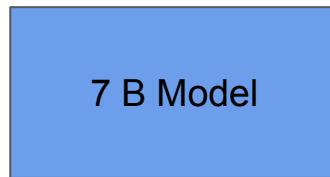
The Wright brothers made their first successful flight in the year..



Difficult!

# Core Idea

Take a draft model which is smaller in size, and generate tokens



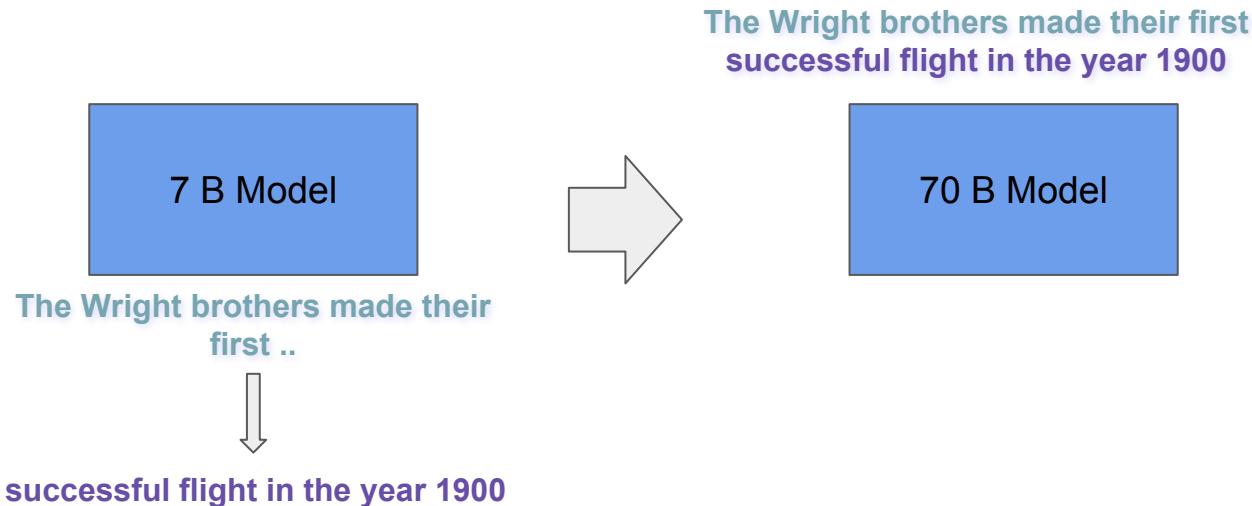
The Wright brothers made their  
first ..



successful flight in the year 1900

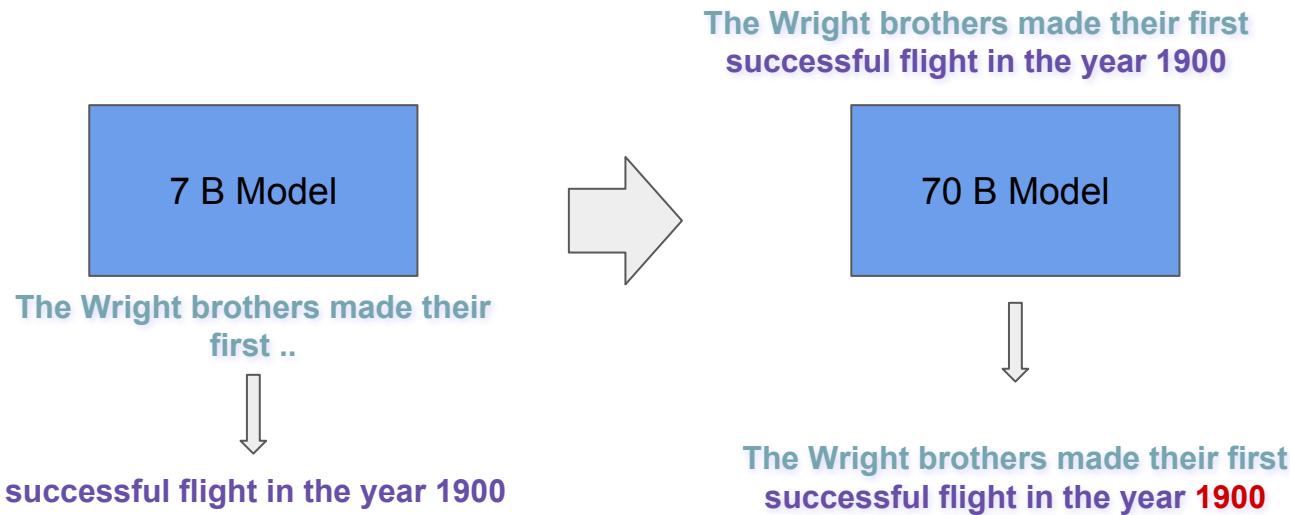
# Core Idea

Validate these tokens in parallel using the larger model



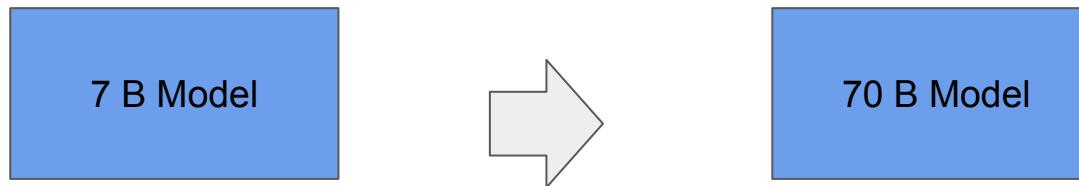
# Core Idea

Accept or reject tokens based on validation



# Core Idea

If a token is rejected, adjust the distribution and re-sample



**The Wright brothers made their first  
successful flight in the year 1903...**

# Speculative Sampling

Traditional sampling techniques:

- **Argmax:** Select the token with the highest probability (deterministic)
- **Top-k:** Choose from the top  $k$  tokens
- **Standard Sampling:** Probabilistic, influenced by temperature

Speculative Sampling:

- While sampling a token, we take a decision based on the probability distributions generated by the draft model and the target model
- We keep the token if they match, otherwise we resample

# Speculative Decoding Algorithm

Some basic terminologies -

$M_q$  - draft Model

$M_p$  - target model

$q(x)$  - probability distribution from draft model

$p(x)$  - distribution of target model

$\gamma$  - number of selection made from draft model

---

## Algorithm 1 SpeculativeDecodingStep

---

**Inputs:**  $M_p, M_q, prefix$ .

▷ Sample  $\gamma$  guesses  $x_{1,\dots,\gamma}$  from  $M_q$  autoregressively.

**for**  $i = 1$  to  $\gamma$  **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

**end for**

▷ Run  $M_p$  in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$

$M_p(prefix), \dots, M_p(prefix + [x_1, \dots, x_\gamma])$

▷ Determine the number of accepted guesses  $n$ .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

▷ Adjust the distribution from  $M_p$  if needed.

$p'(x) \leftarrow p_{n+1}(x)$

**if**  $n < \gamma$  **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$

**end if**

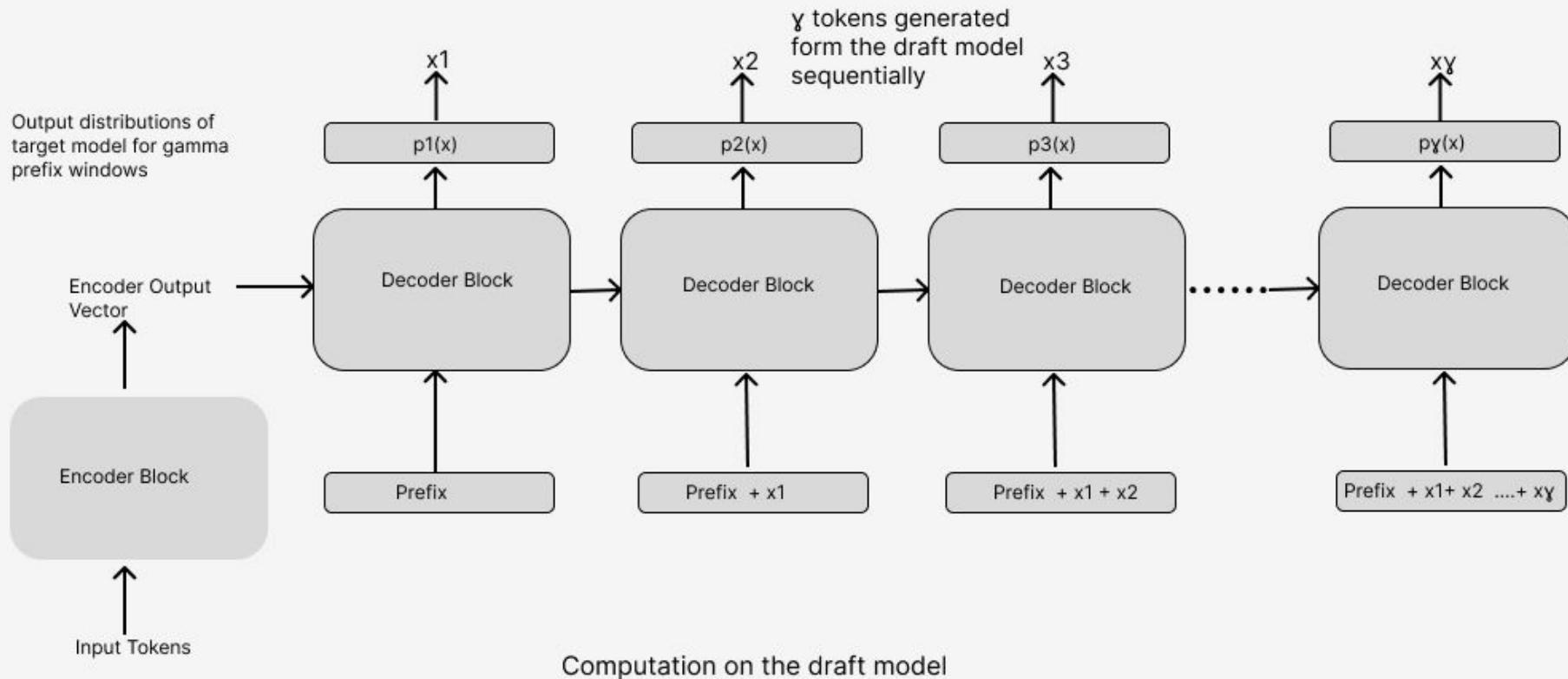
▷ Return one token from  $M_p$ , and  $n$  tokens from  $M_q$ .

$t \sim p'(x)$

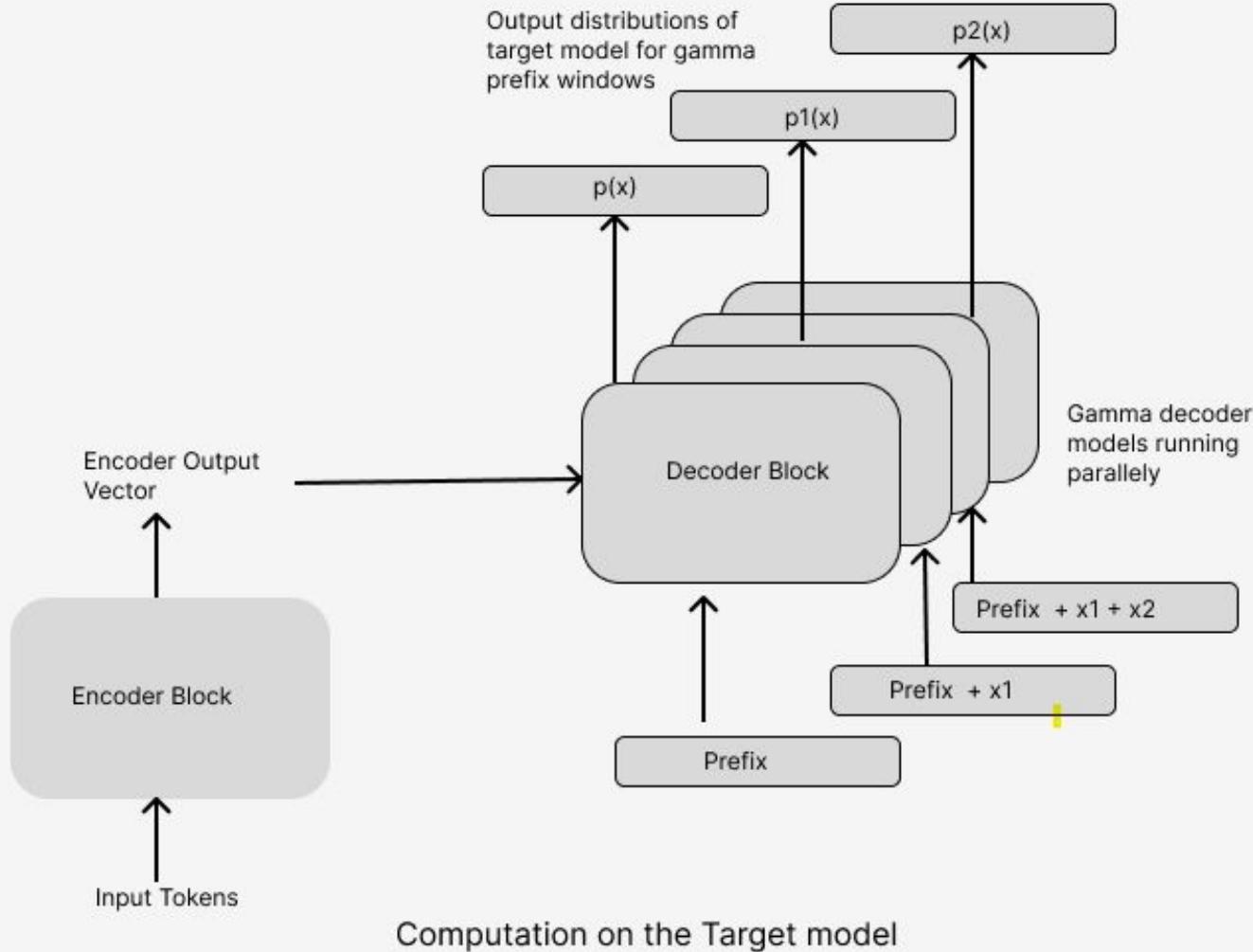
**return**  $prefix + [x_1, \dots, x_n, t]$

---

# Speculative Decoding Algorithm



# Speculative Decoding Algorithm



# Speculative Decoding Algorithm

- We have the two output distribution  $q(x)$  from draft model , and  $p(x)$  from the target model.
- When considering  $i^{\text{th}}$  token, if  $p_i(x) \geq q_i(x)$ , we accept the token  $x_i$ . If  $p_i(x) < q_i(x)$ , we accept it with a probability  $1 - (p_i(x)/q_i(x))$ .
- In case we reject the  $i^{\text{th}}$  token, we then sample the next suggestion token from target model using the adjusted distribution

$$p'(x) = \text{norm}(\max(0, (p_i(x) - q_i(x))))$$

- We repeat the process with the initial prefix along with all the accept and suggested token so far, until we encounter a end special token.

# Experiment Setup

- Models: T5-XXL as Target Model, T5-small/base/large as Draft Models
- Tasks: English-to-German translation and text summarization
- Hardware: Single TPU-v4, batch size = 1

# Results

Speedups:

Up to 3.4x for translation and 3.1x for summarization

Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

TASK	$M_q$	TEMP	$\gamma$	$\alpha$	SPEED
ENDE	T5-SMALL ★	0	7	0.75	<b>3.4X</b>
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	<b>2.6X</b>
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	<b>3.1X</b>
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	<b>2.3X</b>
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

# Key Takeaways

- Faster inference through speculative decoding
- No architectural changes or retraining required
- Preserves output distribution fidelity

Thank you



# ORCA: A Distributed Serving System for Transformer-Based Generative Models

---

**Authors:** Gyeong-In Yu and Joo Seong Jeong, Seoul National University; Geon-Woo Kim, FriendliAI and Seoul National University; Soojeong Kim, FriendliAI; Byung-Gon Chun, FriendliAI and Seoul National University

**Conference:** OSDI'22

**Presenters:** Chia-Chun Shen, Hector Cardenas



# Introduction

- Transformer-based generative models (e.g., GPT, LLaMA) are widely used.
  - ChatGPT reached about 180.5 million monthly users in March 2024[1].
- These models are computationally intensive, making efficient serving a significant challenge.
- This paper focuses on **improving the throughput** of serving transformer-based generative models.

[1] [How Many Users Does ChatGPT Have? Statistics & Facts \(2024\)](#)

# Inference Server Architecture

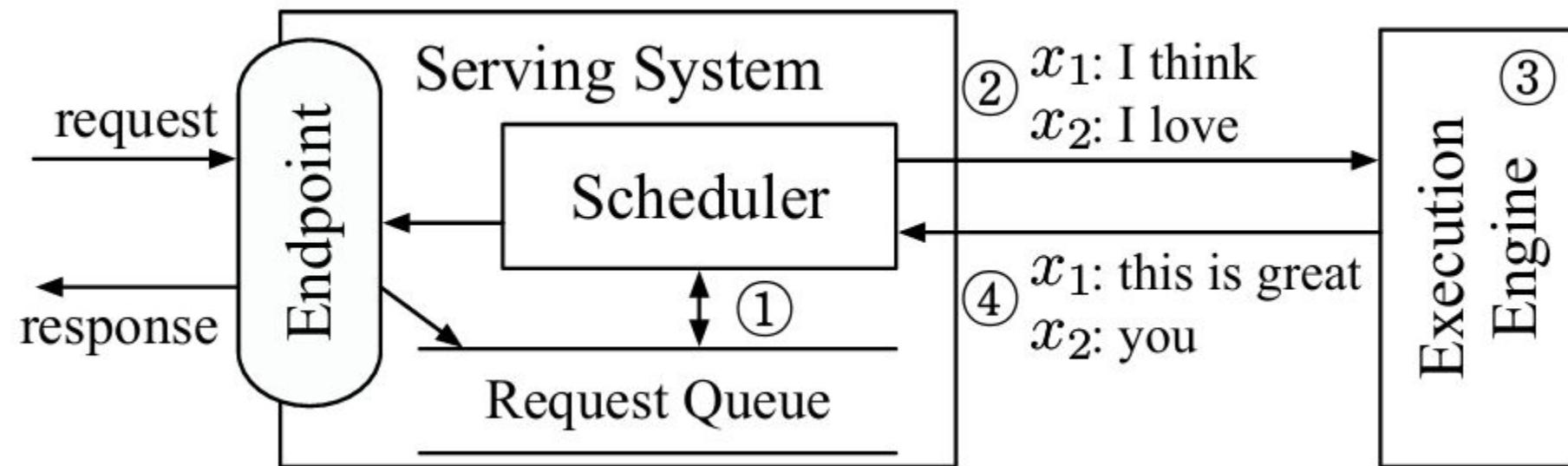


Image Source: Figure 2 in paper



# Challenge 1

- Existing systems process requests at request level.
  - Requests completed early must wait for the entire batch.
  - New requests wait until the current batch finishes.

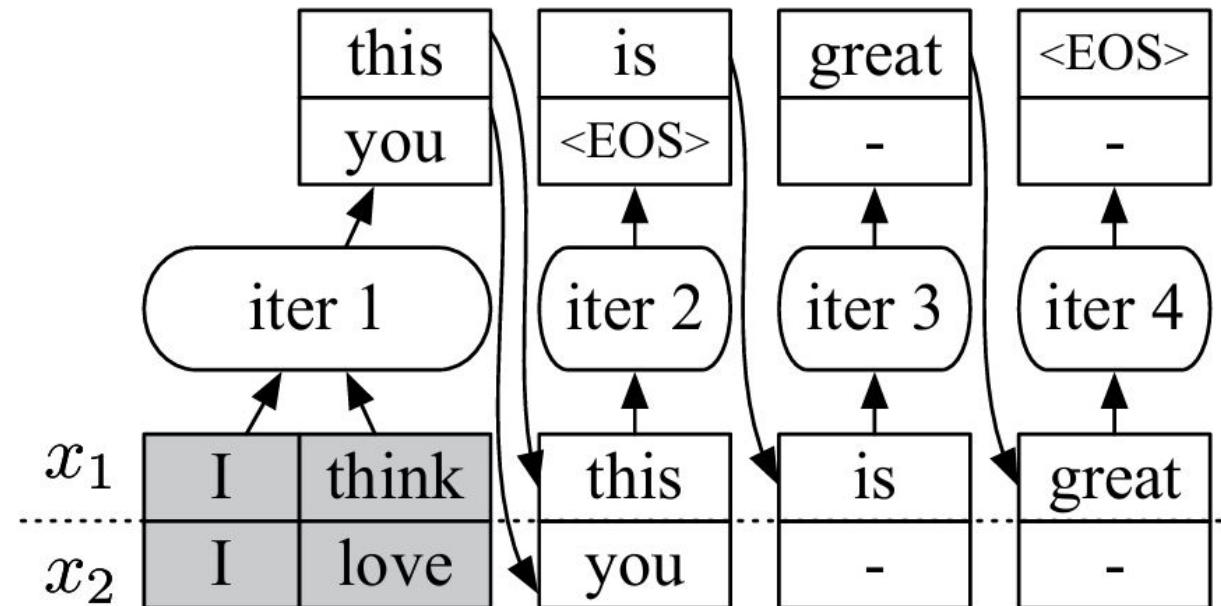


Image Source: Figure 3 in paper



# Solution 1

# Iteration-Level Scheduling

- Schedule executions at the granularity of iteration.
  - Execution engine run **one iteration** for the selected requests.

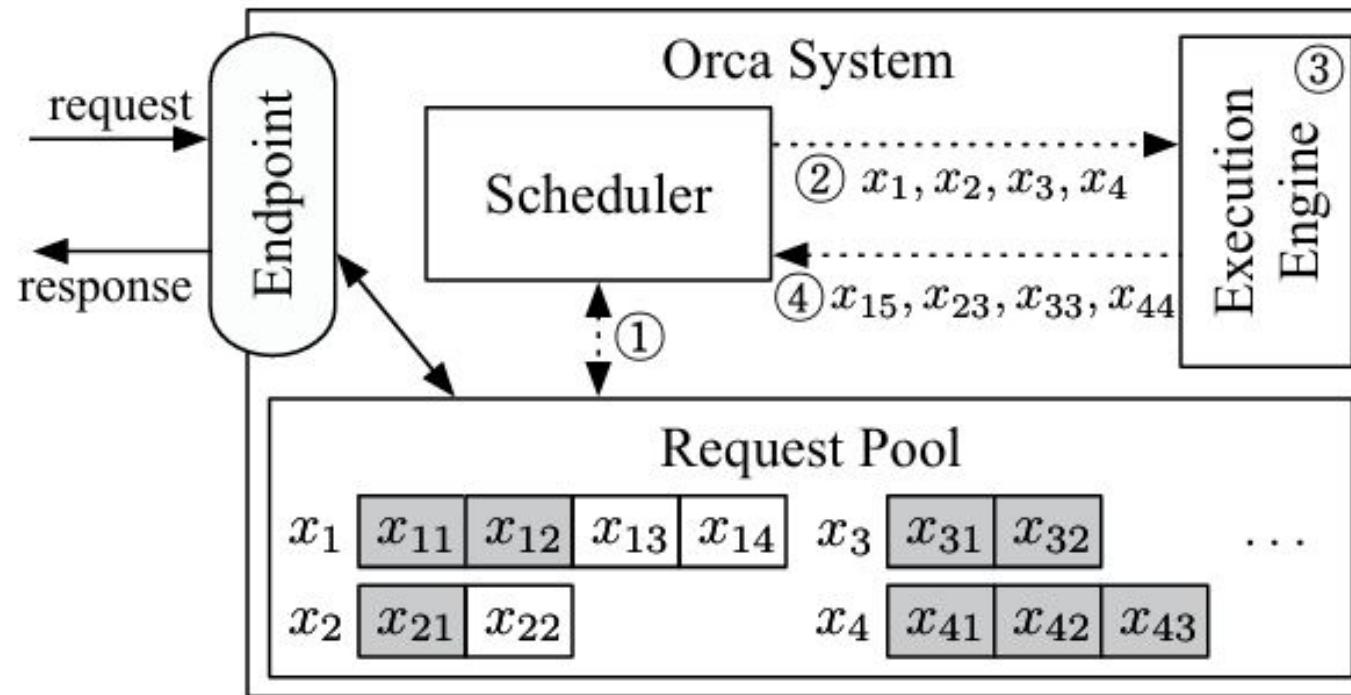


Image Source: Figure 4 in paper

# Iteration-Level Scheduling

- The scheduler can return result once a request is completed.
- New requests can start processing after the current iteration.

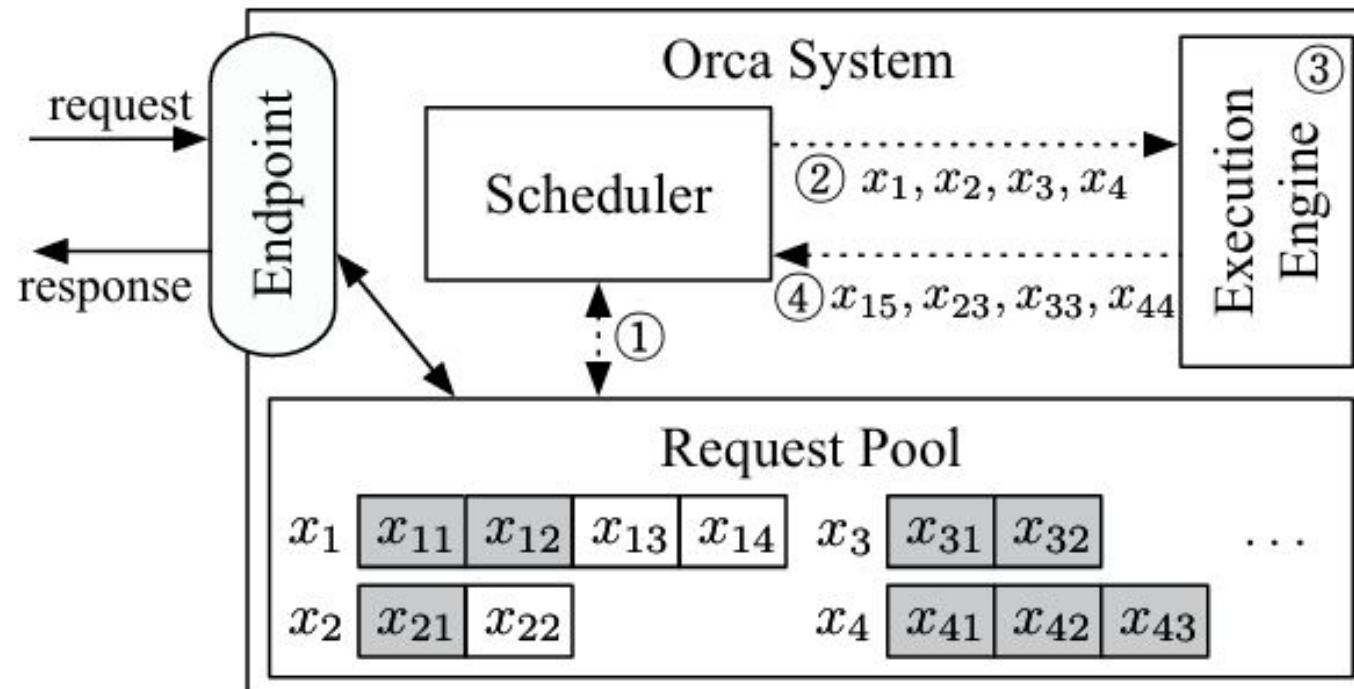


Image Source: Figure 4 in paper



# Challenge 2

# I Batching



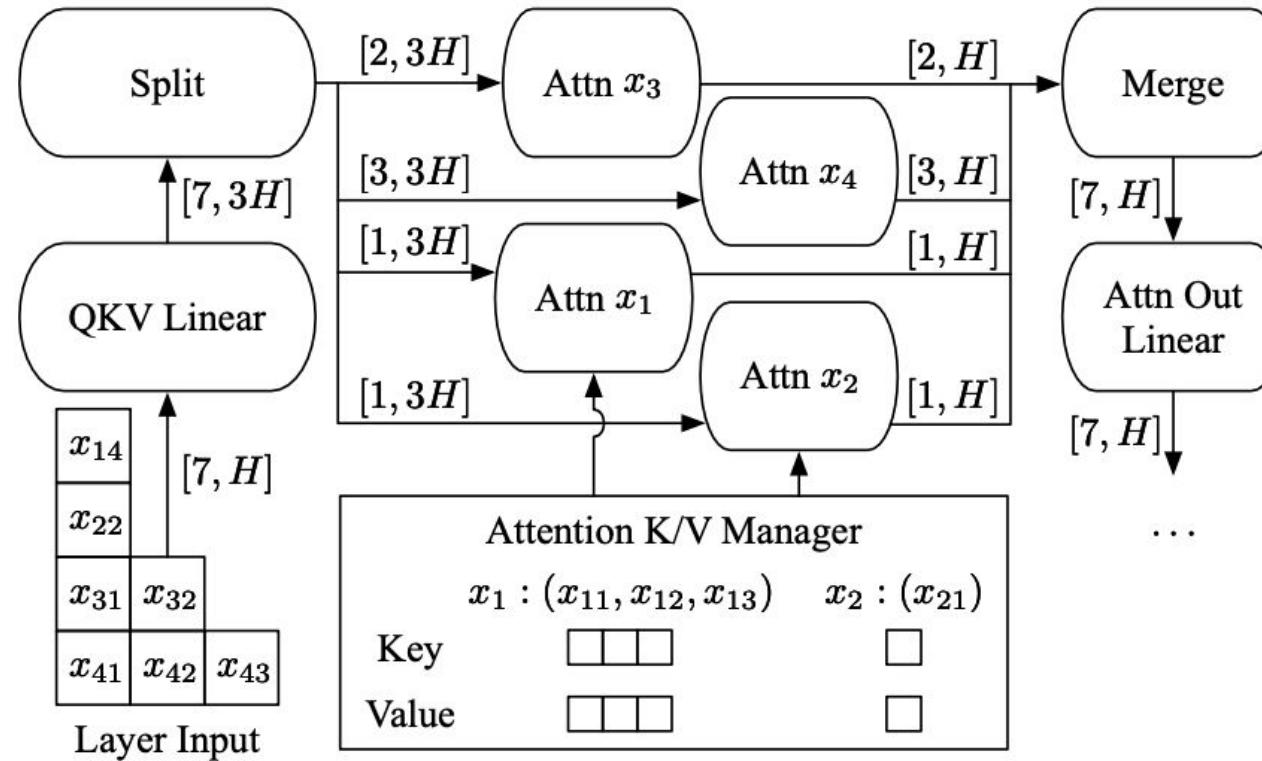
- Iterative scheduling means that requests will have different lengths and/or be at different stages.
- Thus, no guarantees that requests can be safely batched.



# Solution 2

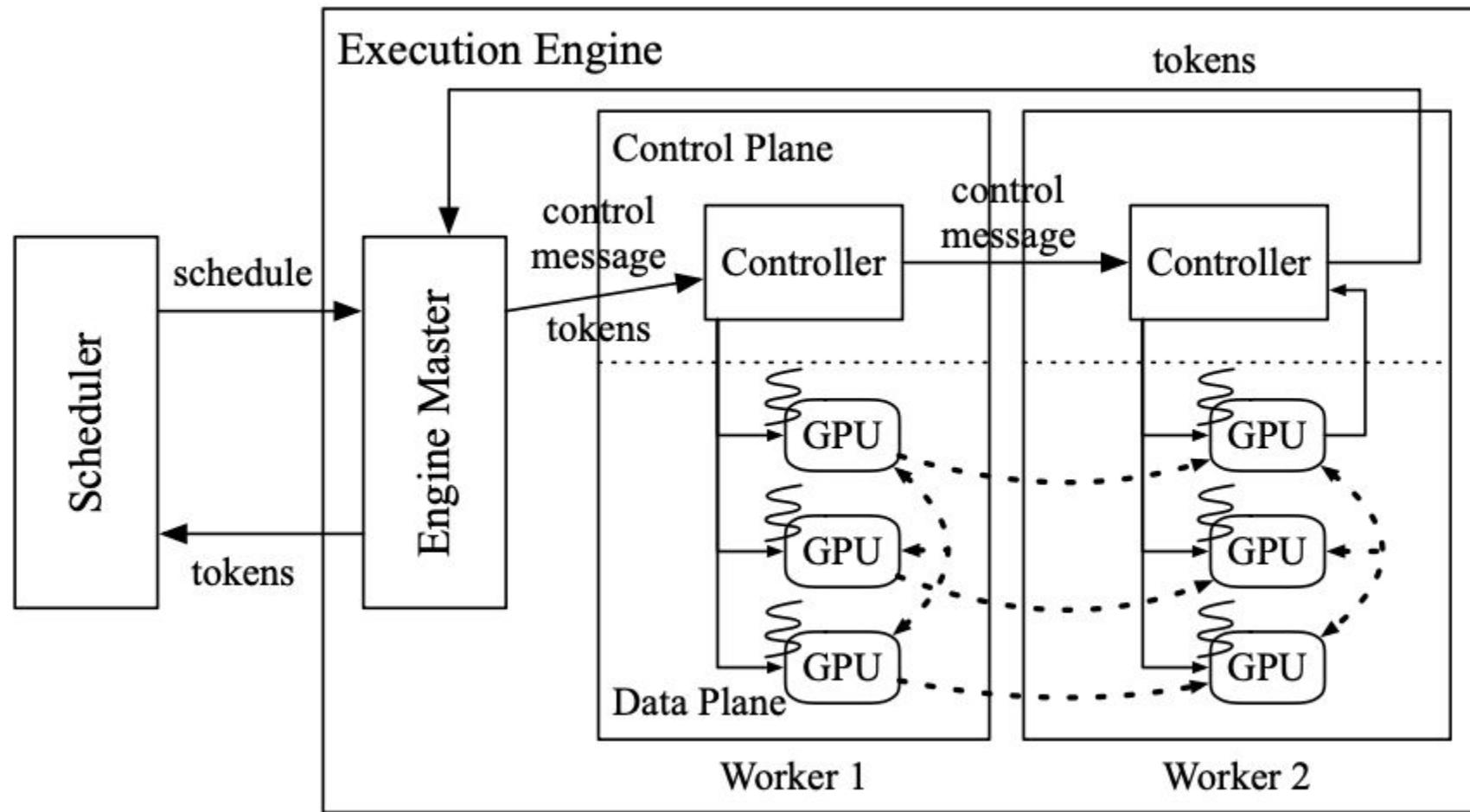
# Selective Batching

- Flattens irregular tensors for non-Attention operations.
- Each request is handled individually for Attention.





ORCA





# Evaluation

- Compared ORCA with FasterTransformer using GPT models with varying configurations.
- Microbenchmark:
  - Comparable for smaller models.
  - Up to 47% performance boost for bigger models.
- End-to-End:
  - Mostly outperformed with higher throughput and lower latency.
  - Achieved up to a 36.9x speedup.



# Conclusion

# I Conclusion



- Proposed **iterative-level scheduling with selective batching**.
- Created **ORCA**, a distributed serving system that significantly improves performance.



THANK YOU



TEXAS A&M UNIVERSITY  
Engineering

CSCE-689

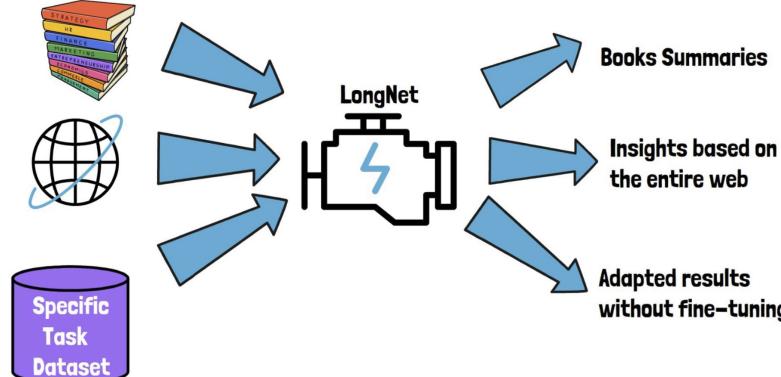
# Paper Review Presentation

LongNet: Scaling Transformers to 1,000,000,000 Tokens

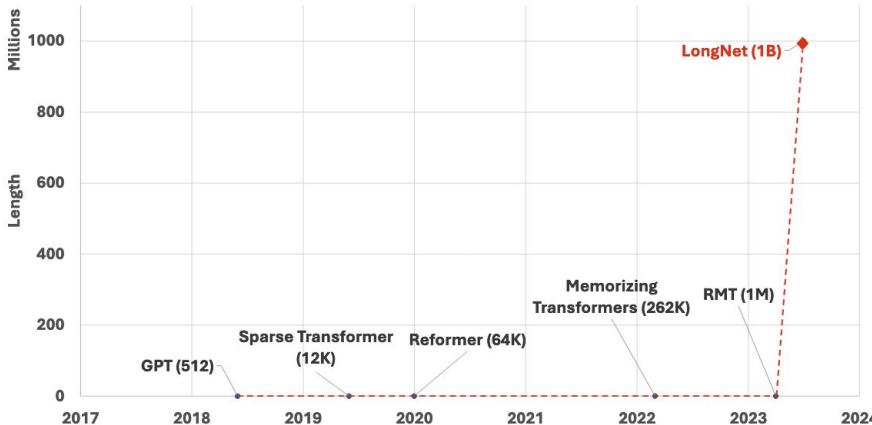
Presenter:  
Kevin Tieu  
Pavitra Malhotra

# Introduction

- Current LLMs have a limited sequence length
    - Small context window
    - Limited in-context learning
    - Short dependency
- Extend the limitation of sequence length



- Scale the sequence length to 1 billions tokens
- Replace vanilla transformer with dilated attention
  - Linear computational complexity
  - Distributed training



Method	Computation Complexity
Recurrent	$\mathcal{O}(Nd^2)$
Vanilla Attention	$\mathcal{O}(N^2d)$
Sparse Attention	$\mathcal{O}(N\sqrt{Nd})$
<b>Dilated Attention (This Work)</b>	$\mathcal{O}(Nd)$

- Self-attention
  - Maps Q, K, V to output
  - Struggles with long sequences
  - Is computational inefficient

$$O = \text{softmax}(QK^T)V$$

- Sparse attention
  - Restricts Q's access to subsets of Ks and Vs

$$O = \text{softmax}(QK^T \odot \mathbb{1}_S)V$$

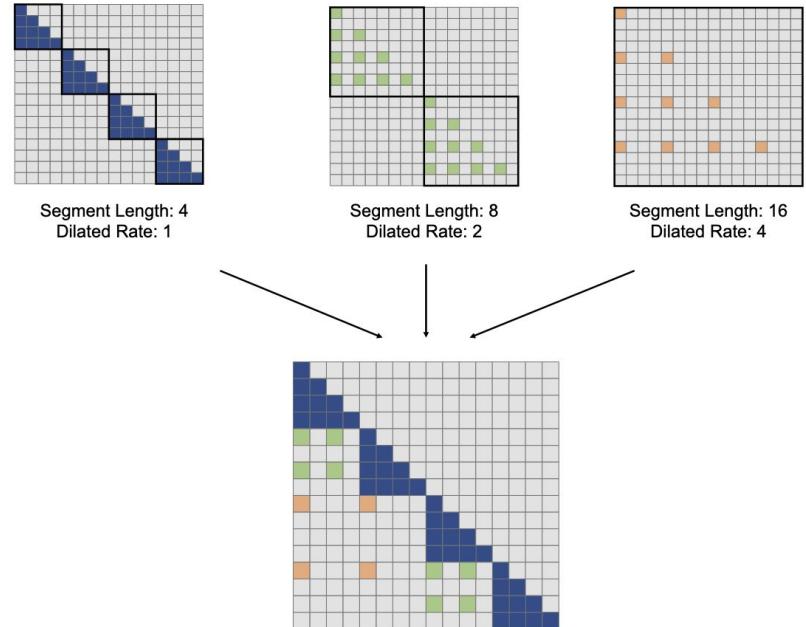
# Dilated attention

- Split (Q,K,V) into segments
  - Each with length w
  - Each is sparsified by interval r
- Send parsfied segments into the attention in parallel
- Scatter and concatenate segments as the output

$$\tilde{O}_i = \text{softmax}(\tilde{Q}_i \tilde{K}_i^T) \tilde{V}_i$$

$$\hat{O}_i = \{\tilde{O}_{i,j} | j \bmod r = 0; 0 | j \bmod r \neq 0\}$$

$$O = [\hat{O}_0, \hat{O}_1, \dots, \hat{O}_{\frac{N}{w}-1}]$$



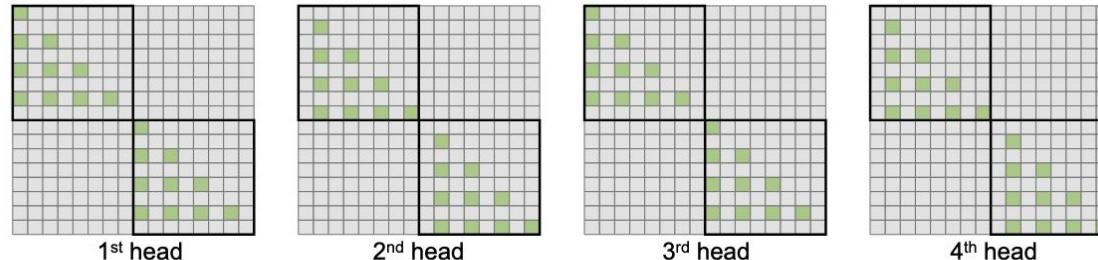
# Multi-head dilated attention

- Sparsify different parts of QKV pair

$$\tilde{Q}_i = [Q_{iw+s_j}, Q_{iw+s_j+r}, Q_{iw+s_j+2r}, \dots, Q_{(i+1)w+s_j-1}]$$

$$\tilde{K}_i = [K_{iw+s_j}, K_{iw+s_j+r}, K_{iw+s_j+2r}, \dots, K_{(i+1)w+s_j-1}]$$

$$\tilde{V}_i = [V_{iw+s_j}, V_{iw+s_j+r}, V_{iw+s_j+2r}, \dots, V_{(i+1)w+s_j-1}]$$



Segment Length: 8  
Dilated Rate: 2  
Heads: 4

- Flops of the attention computation extending to dilated attention with varying segment sizes and dilation rates

$$FLOPs = 2Nd \sum_{i=1}^k \frac{w_i}{r_i^2}$$

- Token dependency

$$\mathcal{O}(\log N)$$

This section explains the distributed training algorithm used in **LongNet** to handle extremely large sequence lengths efficiently by distributing computation across multiple GPUs. The figure illustrates the runtime comparison between **Dilated Attention** and **Vanilla Attention** for sequences of increasing length, showcasing the efficiency of LongNet's distributed algorithm.

## Distributed Algorithm for LongNet

The distributed algorithm divides the sequence dimension across multiple GPUs, leveraging **linear complexity** and efficient communication to process extremely large sequences. Here's how it works:

### 1. Splitting the Sequence Across Devices:

- The input sequence  $X$  is split across devices along the **sequence dimension**:

$$X = [X_1, X_2]$$

- $X_1$  is processed on GPU 1, and  $X_2$  on GPU 2. This approach ensures that each GPU handles only a portion of the sequence, reducing memory requirements.

# LongNet as Distributed Trainer

## 2. Projection into Queries, Keys, and Values:

- Each GPU independently projects its segment into **query (Q)**, **key (K)**, and **value (V)** embeddings:

$$[Q_1, K_1, V_1] = [W_Q, W_K, W_V]X_1, \quad [Q_2, K_2, V_2] = [W_Q, W_K, W_V]X_2$$

- $W_Q, W_K, W_V$  are weight matrices for the projections.
- These projections allow each GPU to compute local attention for its segment.

# LongNet as Distributed Trainer

## 3. Handling Short and Long Segments:

- **Local Attention:** For segments with lengths  $w_i \leq l$  (where  $l$  is the local sequence length on a GPU), attention is computed locally using the standard dilated attention mechanism.
- **Global Attention:** For segments with lengths  $w_i > l$ , keys and values are distributed across GPUs, requiring communication between devices.

## 4. Communication Between GPUs:

- To handle global attention:
  - An **all-gather operation** collects key-value pairs from all devices:

$$\widetilde{K} = [\widetilde{K}_1, \widetilde{K}_2], \quad \widetilde{V} = [\widetilde{V}_1, \widetilde{V}_2]$$

- These combined key-value pairs are then used to compute attention across the full sequence.
- In the **backward pass**, the all-gather operation becomes a **reduce-scatter operation**, ensuring constant communication cost regardless of sequence length.

# LongNet as Distributed Trainer

## 5. Computing Cross-Attention:

- Each GPU computes cross-attention using its local queries and the global key-value pairs:

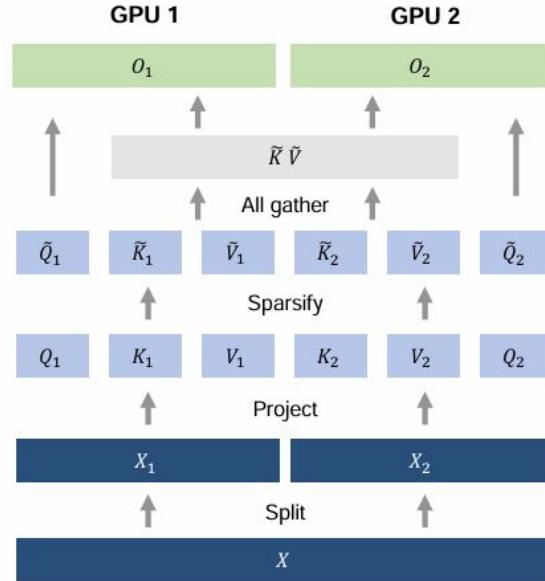
$$\tilde{O}_1 = \text{softmax}(\tilde{Q}_1 \tilde{K}^T) \tilde{V}, \quad \tilde{O}_2 = \text{softmax}(\tilde{Q}_2 \tilde{K}^T) \tilde{V}$$

## 6. Final Output Concatenation:

- The outputs from all devices are concatenated to form the final attention output

$$\tilde{O} = [\tilde{O}_1, \tilde{O}_2]$$

# LongNet as Distributed Trainer



## Key Advantages of the Distributed Algorithm

1. **Scalability:**
  - By splitting the sequence across devices, the algorithm scales to arbitrarily large sequences without overwhelming a single GPU.
2. **Constant Communication Cost:**
  - Unlike vanilla attention, the size of the gathered keys and values is independent of sequence length, keeping communication efficient.
3. **Compatibility with Other Parallelisms:**
  - This distributed approach is orthogonal to other parallelisms:
    - **Data Parallelism:** Splits the batch dimension.
    - **Model Parallelism:** Splits the hidden dimension.
    - **Pipeline Parallelism:** Splits across layers.

## Scaling up to 1B Tokens

This section discusses the scalability of LongNet to handle sequences up to **1 billion tokens** by leveraging a distributed training algorithm and the efficiency of dilated attention. The table and runtime data highlight how LongNet maintains its performance and efficiency while scaling, compared to traditional vanilla attention and sparse transformers.

## Experiment Setup

1. **Goal:** Test the feasibility of processing sequences up to 1 billion tokens using modern distributed systems.
2. **Methodology:**
  - Sequence lengths were incrementally scaled from **8K to 1 billion tokens**.
  - To ensure GPU memory constraints were not exceeded, the **batch size** was reduced proportionally to maintain 1 billion tokens per batch.
  - Each model uses **3 segment lengths**:
    - **2,048 tokens** (local device limit),
    - The number of tokens per GPU,
    - The total sequence length.
  - Both **vanilla attention** and **dilated attention** were implemented using the FlashAttention Kernel to save memory and improve computation speed.

# LongNet as Distributed Trainer

## Key Results and Insights

Model	Length	Batch	Github		
			2K	8K	32K
Transformer [VSP <sup>+</sup> 17]	2K	256	4.24	5.07	11.29
Sparse Transformer [CGRS19] <b>LONGNET (ours)</b>	8K	64	4.39 4.23	3.35 3.24	8.79 3.36
Sparse Transformer [CGRS19] <b>LONGNET (ours)</b>	16K	32	4.85 4.27	3.73 3.26	19.77 3.31
Sparse Transformer [CGRS19] <b>LONGNET (ours)</b>	32K	16	5.15 4.37	4.00 3.33	3.64 3.01

Table 2: Perplexity of language models for LONGNET and the baselines.

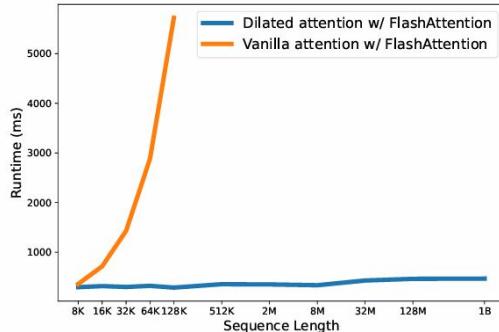
### Perplexity (Table 2):

- **Metric:** Perplexity measures how well the language model predicts the sequence, with lower values indicating better performance.
- **Findings:**
  - LongNet achieves **lower perplexity** than vanilla Transformers across all sequence lengths, demonstrating better generalization and predictive accuracy.
  - At **longer sequences (8K, 16K, and 32K tokens)**:
    - Vanilla Transformers struggle, showing significant increases in perplexity as sequence length grows (e.g., 11.29 at 32K tokens).
    - LongNet maintains much better performance, with perplexity stabilizing even at **32K tokens** (e.g., 3.01 on GitHub data).
  - Sparse Transformers also perform well, but LongNet achieves comparable or better perplexity with greater efficiency.

# LongNet as Distributed Trainer

## Runtime Comparison (Figure 5):

- **Dilated Attention:**
  - Maintains **constant runtime** across sequence lengths, demonstrating linear computational complexity ( $O(Nd)$ ).
  - Scales effectively to 1 billion tokens by leveraging distributed training and FlashAttention.
- **Vanilla Attention:**
  - Suffers from **quadratic complexity ( $O(N^2)$ )**, resulting in **exponentially increasing latency** as sequence length grows.
  - Becomes impractical at longer sequences due to unbounded memory and computational requirements.



## Key Observation:

- The efficiency of LongNet's distributed algorithm allows it to process massive sequences without significant latency increases, while vanilla attention lacks an effective distributed algorithm for handling such large-scale data.

## Advantages of LongNet's Approach

- Linear Complexity:**
  - LongNet's dilated attention reduces computational cost from quadratic ( $O(N^2)$ ) to linear ( $O(Nd)$ ), enabling scaling to sequences up to 1 billion tokens.
- Distributed Algorithm:**
  - The ability to partition sequences across GPUs ensures scalability without overwhelming memory limits.
  - Efficient communication between GPUs keeps latency low and ensures near-constant runtime even as sequence length grows.
- FlashAttention Kernel:**
  - Further enhances memory efficiency and speed, allowing LongNet to handle extreme sequence lengths with minimal performance trade-offs.

# LongNet as Distributed Trainer

## Takeaways

- **Scalability:** LongNet proves capable of scaling up to 1 billion tokens, maintaining efficiency and accuracy, making it ideal for ultra-long-sequence tasks.
- **Comparison:** While vanilla attention fails to scale efficiently due to quadratic complexity, LongNet leverages linear complexity and distributed training to remain performant even at extreme sequence lengths.
- **Implications:** LongNet's ability to handle such massive sequences opens doors for applications like large-scale document analysis, whole genome sequencing, and internet-scale text modeling, where sequence lengths are extremely large.

# Experiments on Language Modeling

This section presents experimental results showcasing **LongNet's** efficiency, scalability, and performance on language modeling tasks. The experiments compare LongNet with vanilla Transformers and sparse Transformers across different sequence lengths, model sizes, and context windows, validating its superiority in processing extremely long sequences.

## 4.1 Setup

### 1. Architecture:

- LongNet is implemented using the **MAGNETO** architecture with XPOS relative position encoding.
- Hidden dimension: **768**, Number of attention heads: **12**, Decoder layers: **12**.
- Standard attention layers are replaced with **dilated attention**.

### 2. Dataset:

- The **Stack dataset** is used, consisting of source code from 300+ programming languages.
- Preprocessed with the **tiktoken tokenizer** using **cl100k\_base** encoding.

### 3. Training:

- Batch size: **0.5 million tokens**.
- Training steps: **300K**.
- Experiments conducted with the **TorchScale** framework.

### Comparison Models:

- Vanilla Transformer: Dense, fully connected attention layers.
- Sparse Transformer: Fixed sparse patterns with block size 2,048.
- LongNet: Dilated attention with segment lengths  $w=\{2048, 4096, 8192, 16384, 32768\}$  and dilated ratios  $r=\{1, 2, 4, 6, 12\}$ .

# Experiments on Language Modeling

## 4.2 Results

1. **Key Metric:**
  - **Perplexity (PPL):** Lower perplexity indicates better language modeling performance.
2. **Findings:**
  - **Table 2:** LongNet achieves consistently lower perplexity than vanilla and sparse Transformers across sequence lengths (2K to 32K).
    - Vanilla Transformers struggle with longer sequences, showing much higher perplexity (e.g., 11.29 at 32K tokens).
    - LongNet performs well even at **32K tokens**, with perplexity stabilizing (e.g., 3.01 for GitHub data).
  - LongNet consistently outperforms baselines, proving its **scalability** and **efficiency** in handling long sequences.
3. **Blockwise Causal Attention (BCA):**
  - Used during inference for extrapolating sequence lengths beyond training limits.
  - Effective for LongNet but less impactful for baselines when lengths significantly exceed their training limits.

# Experiments on Language Modeling

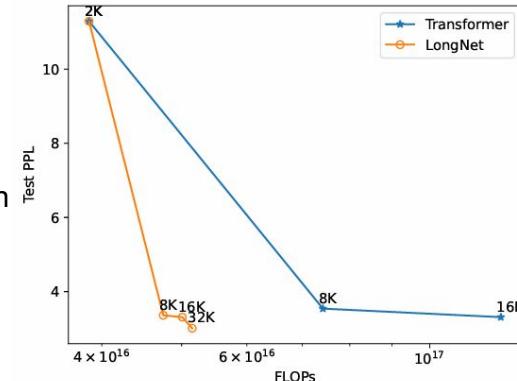
## 4.3 Scaling Curves of Sequence Length

### 1. Experiment:

- Models were trained on sequences of varying lengths (1K to 32K).
- Blockwise causal attention was applied during inference for better generalization

### 2. Results (Figure 6):

- **LongNet:**
  - Achieves **lower test loss** with a smaller number of FLOPs.
  - Demonstrates efficient scaling as context length increases.
- **Vanilla Transformers:**
  - Benefit from longer training inputs but require significantly higher compute (FLOPs) to achieve comparable results.
- **Conclusion:**
  - LongNet's ability to efficiently scale context length during training makes it a more effective approach for modeling long-range dependencies compared to extrapolation.



# Experiments on Language Modeling

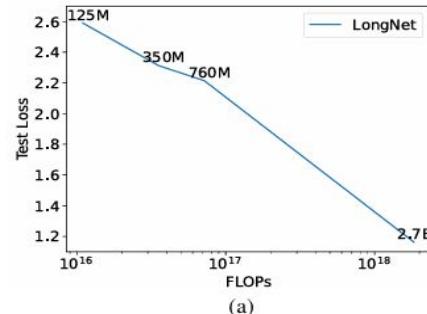
## 4.4 Scaling up Model Size

### 1. Experiment:

- Models with sizes ranging from **125 million** to **2.7 billion parameters** were trained and evaluated.
- Compute was measured in FLOPs, and perplexity was evaluated on the same test set.

### 2. Results (Figure 7a):

- LongNet follows a **power law scaling** for test loss with increasing compute and model size.
- Achieves significant performance gains as model size increases, comparable to dense Transformers but with **greater efficiency**.
- Confirms that dense Transformers are not a prerequisite for scaling language models.



# Experiments on Language Modeling

## 4.5 Long-Context Prompting

### 1. Objective:

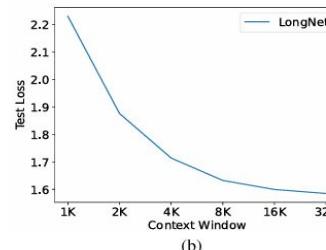
- Test whether LongNet can effectively leverage long-context prompts to improve language modeling.

### 2. Experiment:

- Prefixes of varying lengths (2K to 32K) were used as prompts, with a fixed suffix for evaluation.
- Models were compared based on their ability to utilize long contexts for better predictions.

### 3. Results (Figure 7b):

- LongNet's test loss **decreases** as the prompt length increases, demonstrating its ability to fully leverage long contexts.
- Highlights LongNet's superiority in processing long-context prompts compared to baseline models.



# Experiments on Language Modeling

## Takeaways

- **Scalability:** LongNet efficiently scales sequence length and model size, handling longer contexts with fewer computational resources than baselines.
- **Performance:** Consistently achieves lower perplexity and test loss across experiments, validating its effectiveness in language modeling.
- **Efficiency:** With linear complexity and distributed algorithms, LongNet is ideal for tasks requiring large-scale sequence processing.

# Conclusion and Future Work

## Conclusion and Future Work

### Conclusion:

- **LongNet Overview:**
  - LongNet is a **Transformer variant** designed to scale sequence lengths to **1 billion tokens and beyond**.
  - Unlike standard Transformers, LongNet achieves this without any loss in performance on shorter sequences, making it versatile for a wide range of tasks.
- **Key Innovation:**
  - The core innovation of LongNet is its **dilated attention mechanism**, which reduces the computational complexity of the attention mechanism from **quadratic ( $O(N^2)$ )** to **linear ( $O(Nd)$ )**.
  - This reduction in complexity allows LongNet to process extremely long sequences efficiently.
- **Distributed Training:**
  - LongNet leverages **distributed training algorithms** to parallelize the processing of sequence dimensions across multiple GPUs, overcoming hardware limitations for large-scale sequence modeling.
- **Experimental Validation:**
  - Through experiments, LongNet demonstrates **superior performance** over existing baseline models like vanilla Transformers and sparse Transformers, for both long and short sequence tasks.
  - These results validate LongNet as a scalable and efficient solution for handling large-scale sequence data.

# Future Work

## Future Work:

### 1. Support for Multimodal Tasks:

- Extend LongNet's capabilities to handle **multimodal large language modeling**, where data from multiple modalities (e.g., text, images, and audio) needs to be processed together.

### 2. BEiT Pretraining:

- Explore the use of LongNet for **BEiT (BERT Pretraining of Image Transformers)**, which is used for vision-language tasks and fine-tuning vision-based Transformers.

### 3. Genomic Data Modeling:

- Apply LongNet to **genomic data**, where sequences like DNA or RNA involve massive sequential data that require efficient long-range modeling.