

Large Language Models

Contents

- Language Models
- Transformer
 - Attention Mechanism and Self-Attention, Transformer Architecture
- Recent Transformers
 - Normalization, Activation Function, Position Embedding
- Large Language Models
 - Instruction Tuning
 - Alignment Tuning

Language Models

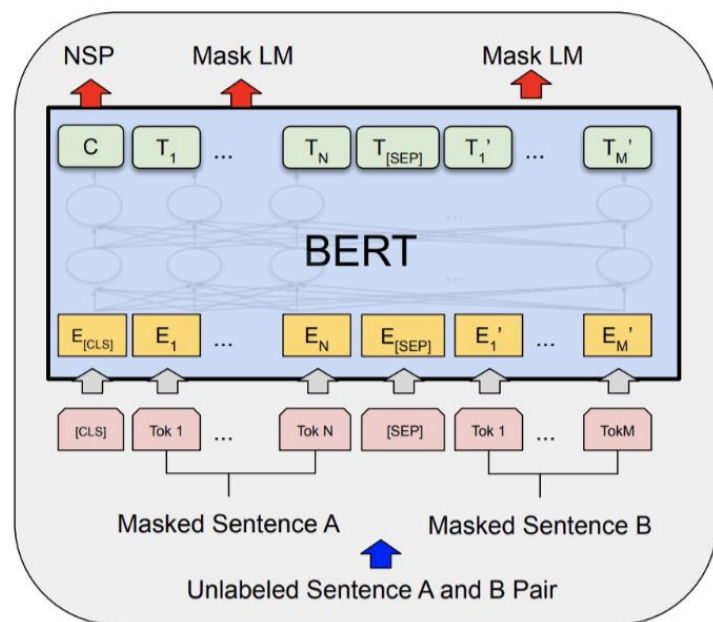
- Definition (Narrow sense)
 - A probabilistic model that assigns a probability $P(w_1, w_2, \dots, w_n)$ to every finite sequence w_1, w_2, \dots, w_n .
 - Using the chain rule of conditional probability,

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2|w_1) \cdots P(w_n|w_1, \dots, w_{n-1})$$

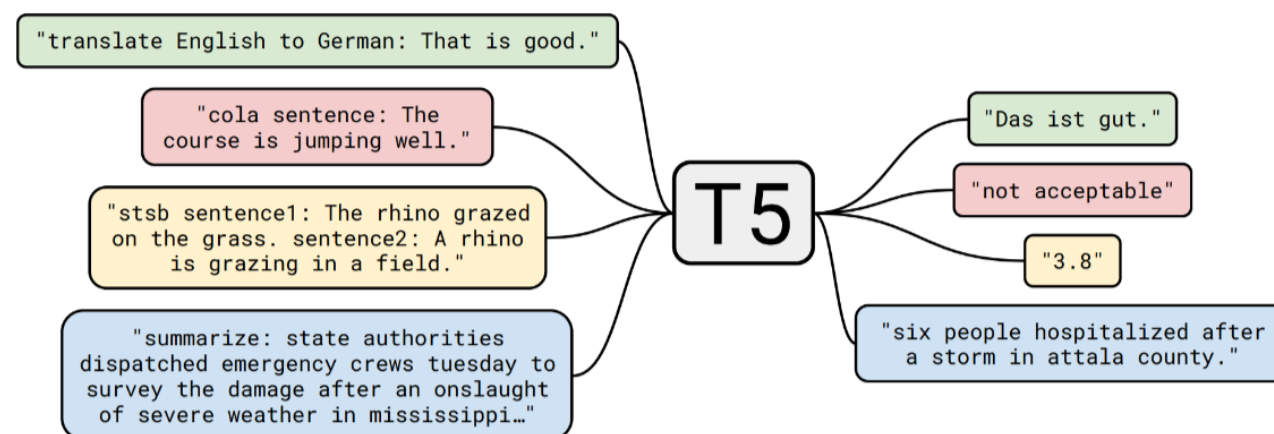
- The language modeling problem is equivalent to being able to predict the next word.

Language Models

- Definition (Broad sense)
 - Decoder-only models (GPT-x models)
 - Encoder-only models (BERT, RoBERTa, ELECTRA)
 - Encoder-decoder models (T5, BART)

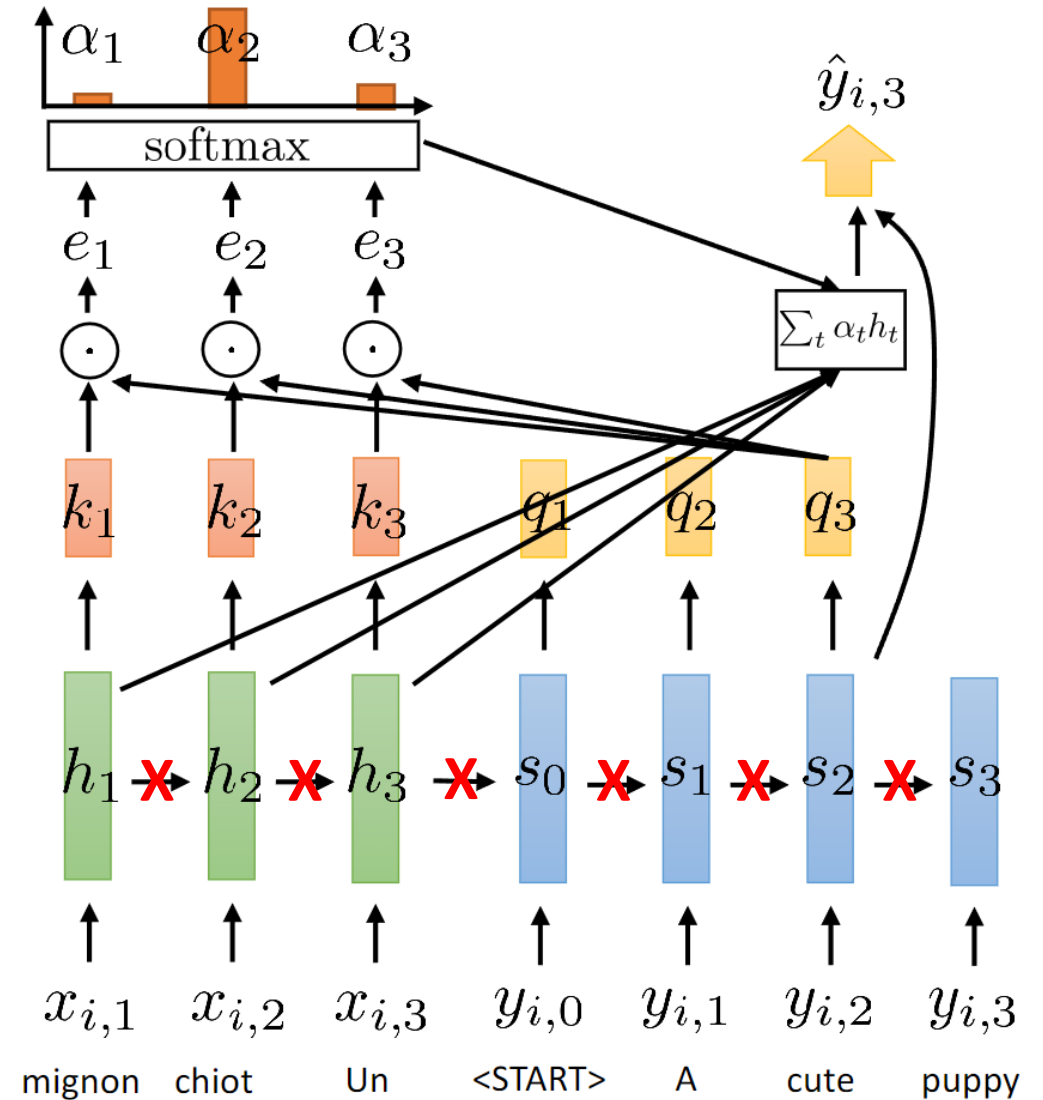


pretrain-then-finetune



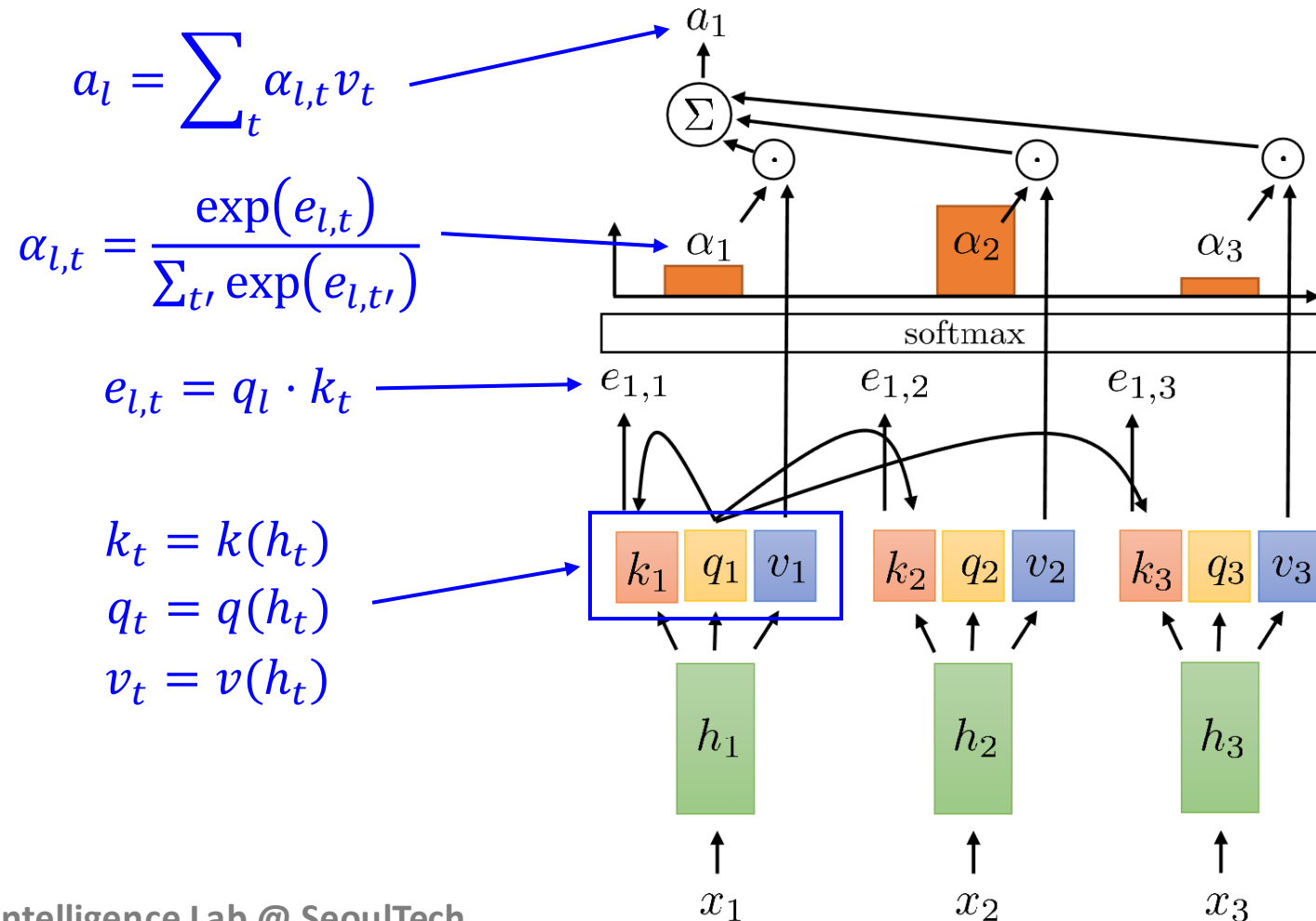
Attention Mechanism

- Questions
 - If we have attention, do we even need recurrent connections?
 - Can we transform our RNN into a purely attention-based model?
- Answers
 - Attention can access every time step.
 - It can in principle do everything that recurrence can, and more.
- However, there are some problems.
 - No temporal dependencies!



Self-Attention

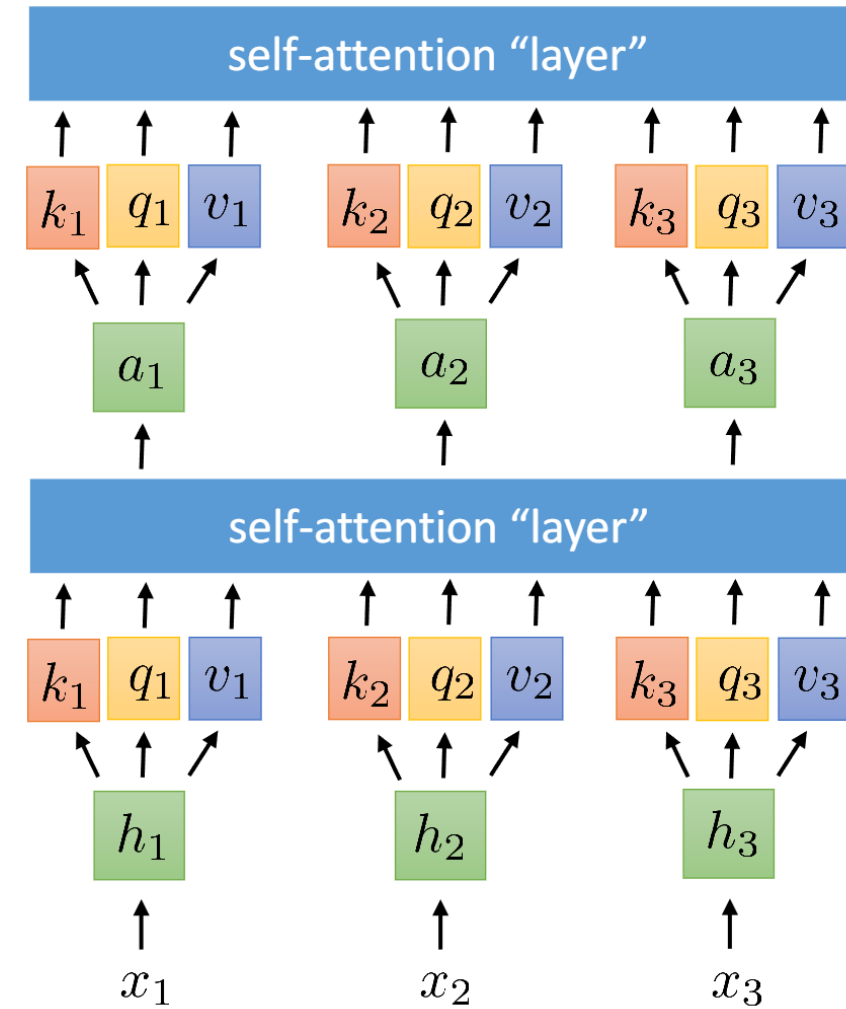
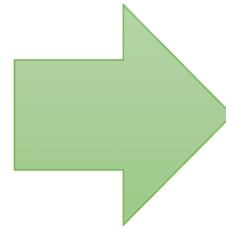
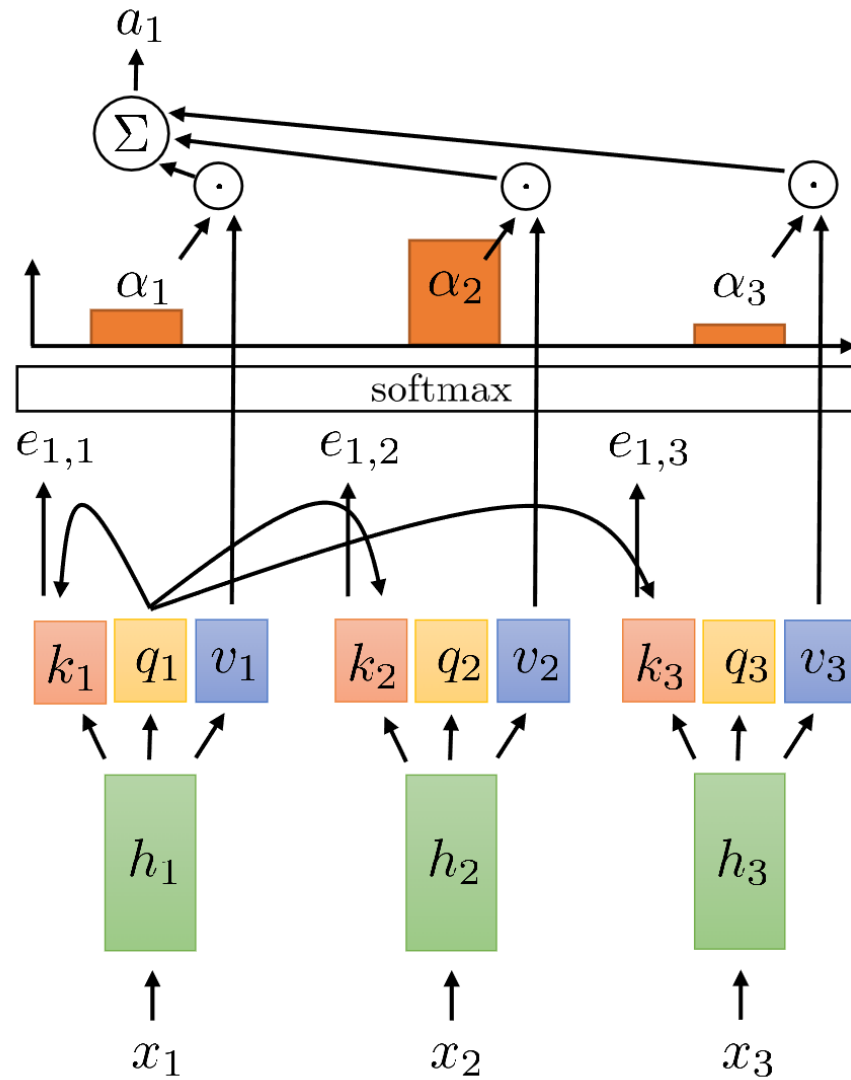
- Insert attention everywhere



- Typically, k_t , q_t , and v_t are linear transformations, e.g., $k_t = W_k h_t$.
- This is not a recurrent model, but still weight sharing:

$$h_t = \sigma(Wx_t + b)$$

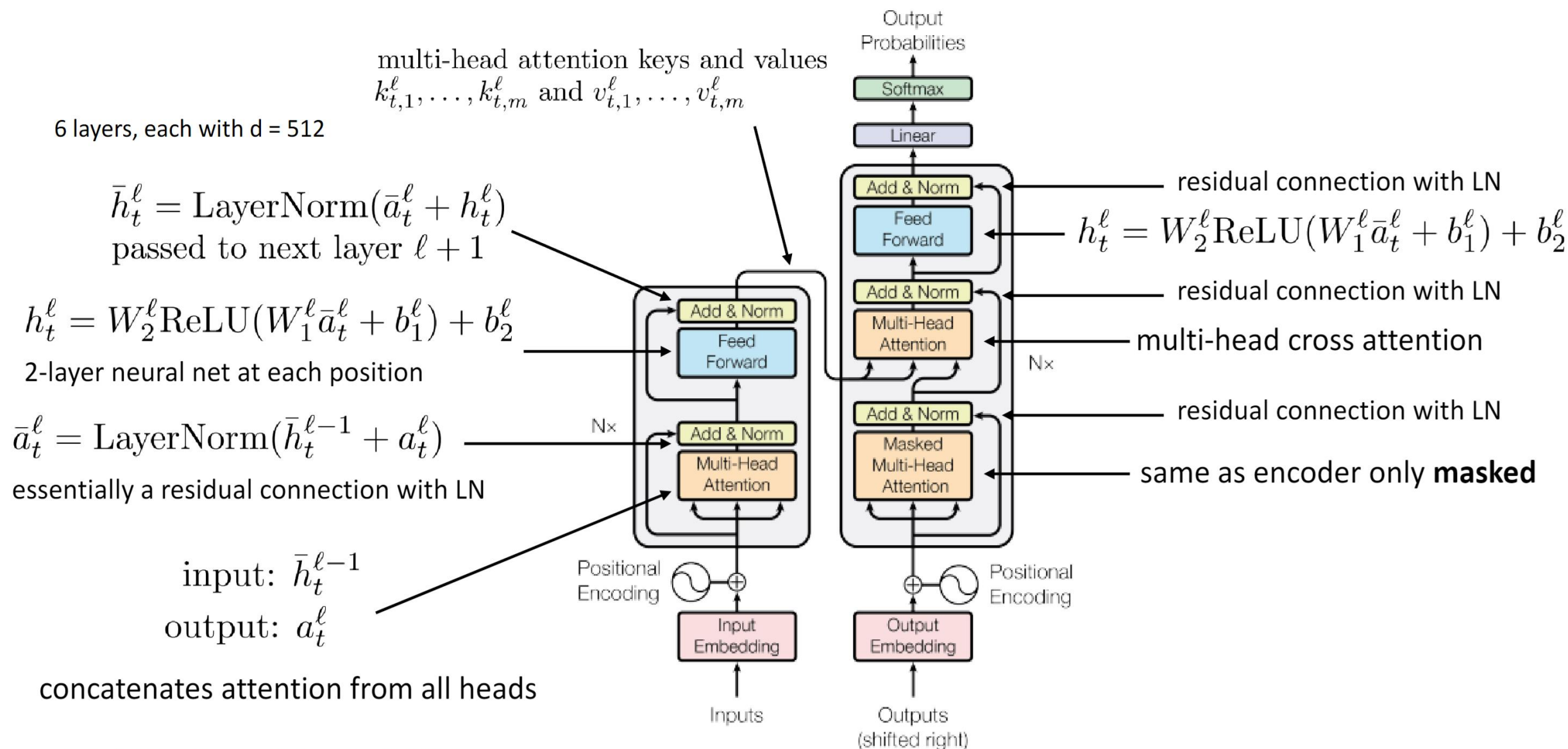
Self-Attention



From Self-Attention to Transformer

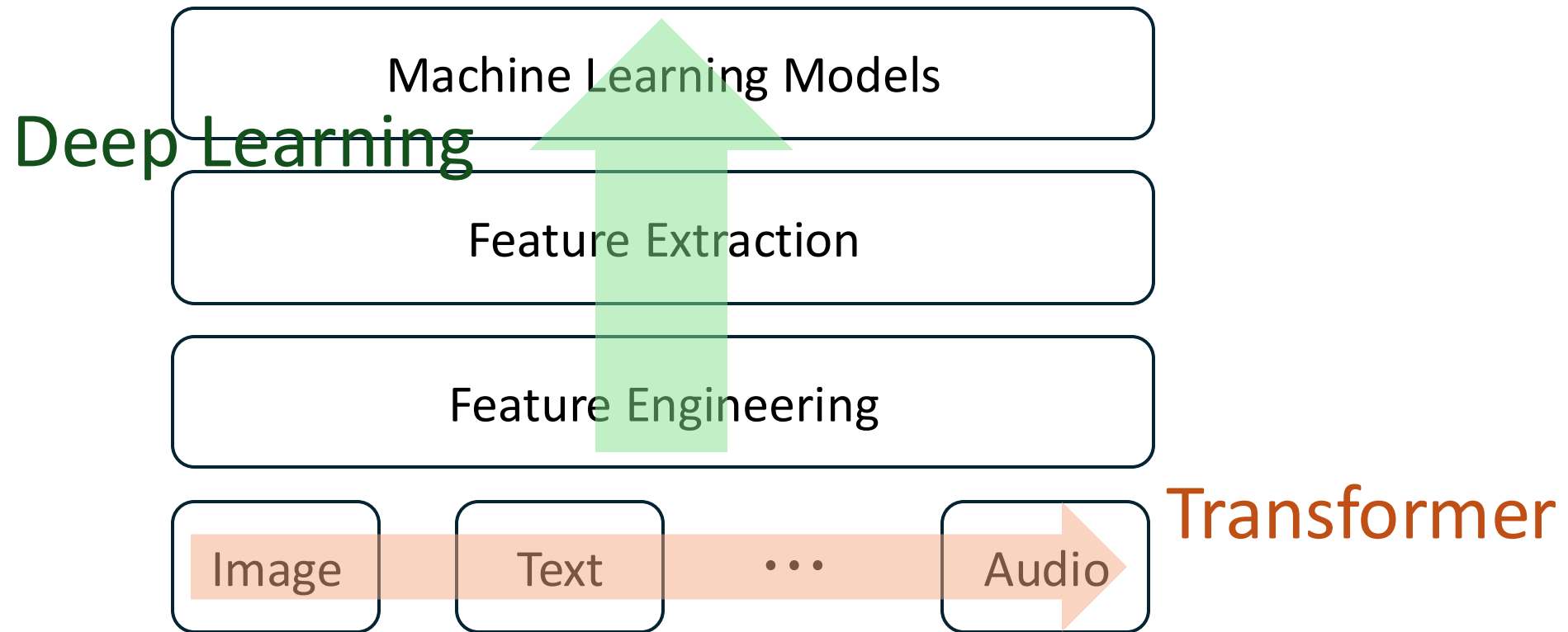
- The basic concept of self-attention can be used to develop a very powerful type of sequence model, called a Transformer.
- But to make this actually work, we need to address some fundamental limitations.
 - Addresses lack of sequence information → Positional embedding
 - Allows querying multiple positions → Multi-headed attention
 - Each successive is linear w.r.t. inputs → Adding nonlinearities
 - Prevent attention lookups into the future → Masked decoding

Putting It All Together



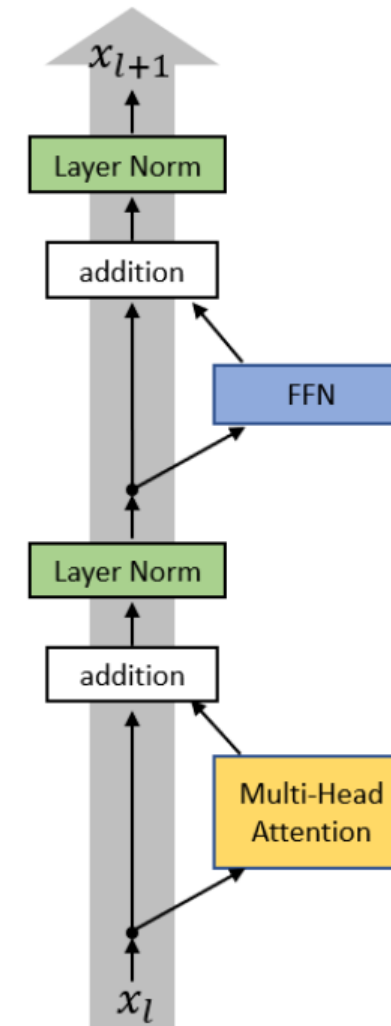
Why Transformers?

- (+) Better long-range connections, parallel computations
- (−) Attention computations are technically $O(n^2)$

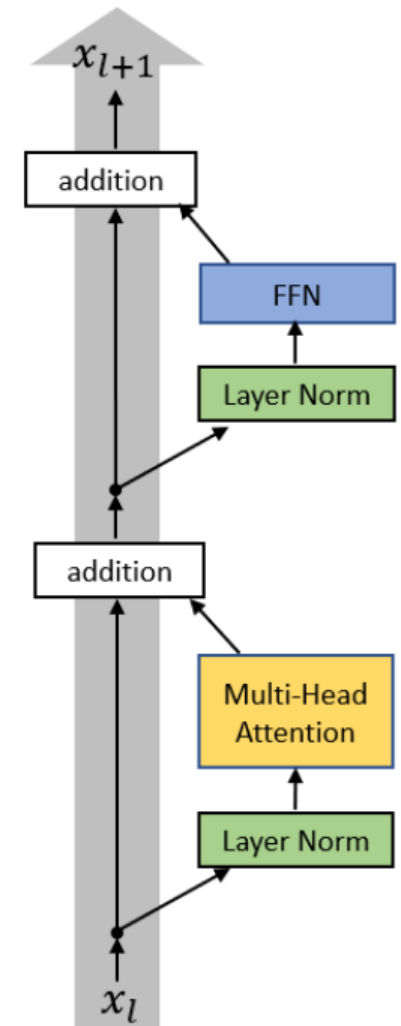


Normalization Position

- Post-LN
 - It is used in the vanilla Transformer, which is placed between residual blocks.
- Pre-LN (Xiong et al., 2020)
 - It is verified that Transformer with post-LN is instable due to the large gradients near the output layer.
 - LN is applied before each sub-layer, and an additional LN is placed before the final prediction.
 - Most LLMs adopt the pre-LN strategy.



Post-LN



Pre-LN

Normalization Methods

- In the vanilla Transformer, LayerNorm is employed.
- RMSNorm (Zhang and Sennrich, 2019)
 - It simplifies LayerNorm by removing the mean statistic.

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_{i=1}^n a_i^2}$$

- Used in Gopher, Chichilla, and recently, LLaMA 2/3.

Normalization in LLaMA 2/3

```
class TransformerBlock(nn.Module):
    def __init__(self, layer_id: int, args: ModelArgs):
        """
        Initialize a TransformerBlock.

        Args:
            layer_id (int): Identifier for the layer.
            args (ModelArgs): Model configuration parameters.

        Attributes:
            n_heads (int): Number of attention heads.
            dim (int): Dimension size of the model.
            head_dim (int): Dimension size of each attention head.
            attention (Attention): Attention module.
            feed_forward (FeedForward): FeedForward module.
            layer_id (int): Identifier for the layer.
            attention_norm (RMSNorm): Layer normalization for attention output.
            ffn_norm (RMSNorm): Layer normalization for feedforward output.

        """
        super().__init__()
        self.n_heads = args.n_heads
        self.dim = args.dim
        self.head_dim = args.dim // args.n_heads
        self.attention = Attention(args)
        self.feed_forward = FeedForward(
            dim=args.dim,
            hidden_dim=4 * args.dim,
            multiple_of=args.multiple_of,
            ffn_dim_multiplier=args.ffn_dim_multiplier,
        )
        self.layer_id = layer_id
        self.attention_norm = RMSNorm(args.dim, eps=args.norm_eps)
        self.ffn_norm = RMSNorm(args.dim, eps=args.norm_eps)
```

```
def forward(
    self,
    x: torch.Tensor,
    start_pos: int,
    freqs_cis: torch.Tensor,
    mask: Optional[torch.Tensor],
):
    """
    Perform a forward pass through the TransformerBlock.

    Args:
        x (torch.Tensor): Input tensor.
        start_pos (int): Starting position for attention caching.
        freqs_cis (torch.Tensor): Precomputed cosine and sine frequencies.
        mask (torch.Tensor, optional): Masking tensor for attention. Defaults to None.

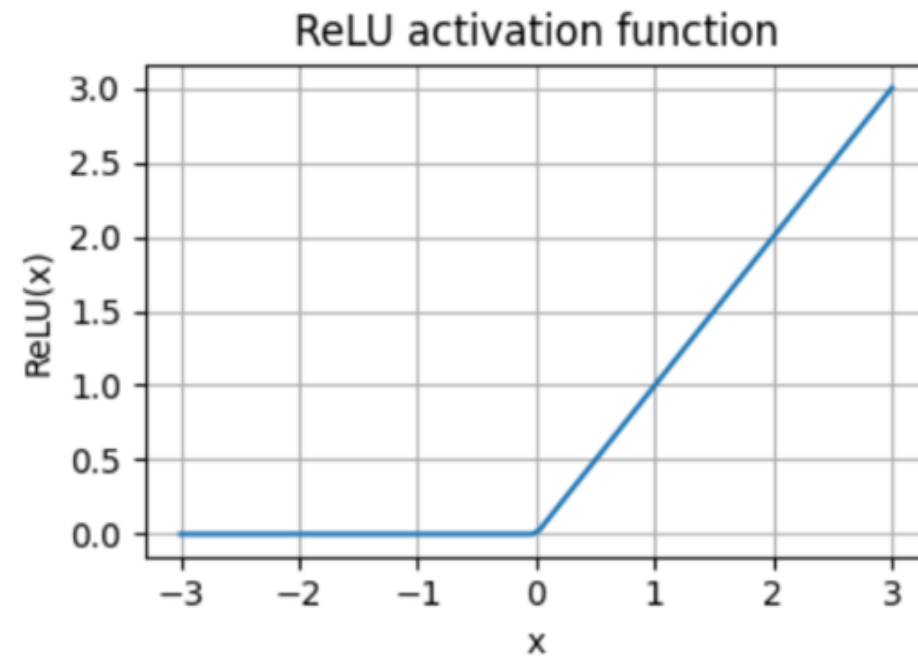
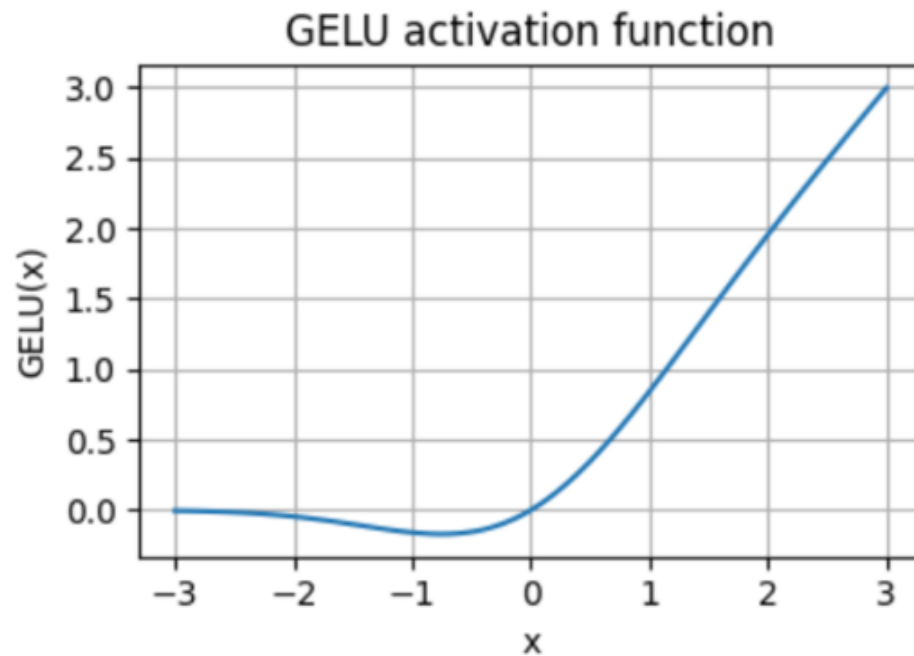
    Returns:
        torch.Tensor: Output tensor after applying attention and feedforward layers.

    """
    h = x + self.attention(
        self.attention_norm(x), start_pos, freqs_cis, mask
    )
    out = h + self.feed_forward(self.ffn_norm(h))
    return out
```

<https://github.com/meta-llama/llama/blob/main/llama/model.py>

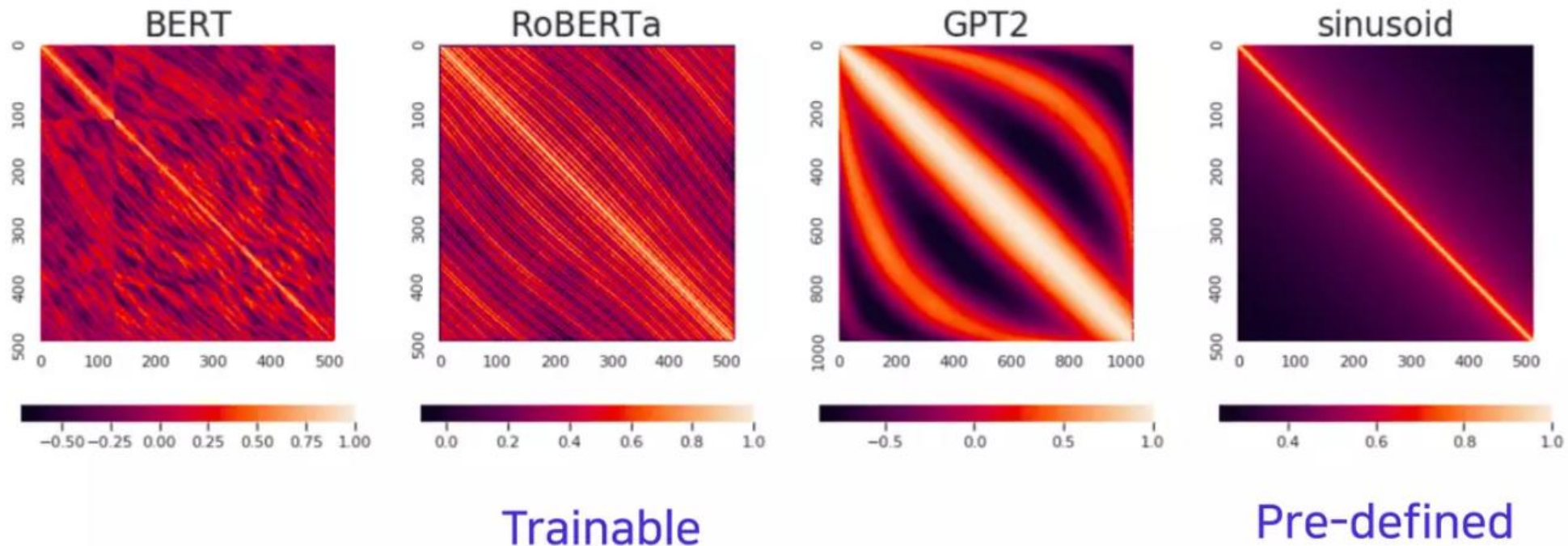
Activation Functions

- In the vanilla Transformer, ReLU is employed.
- In recent LLMs, GeLU is widely used.
- SwiGLU and GeGLU are often used, but they require extra parameters.



Positional Embedding

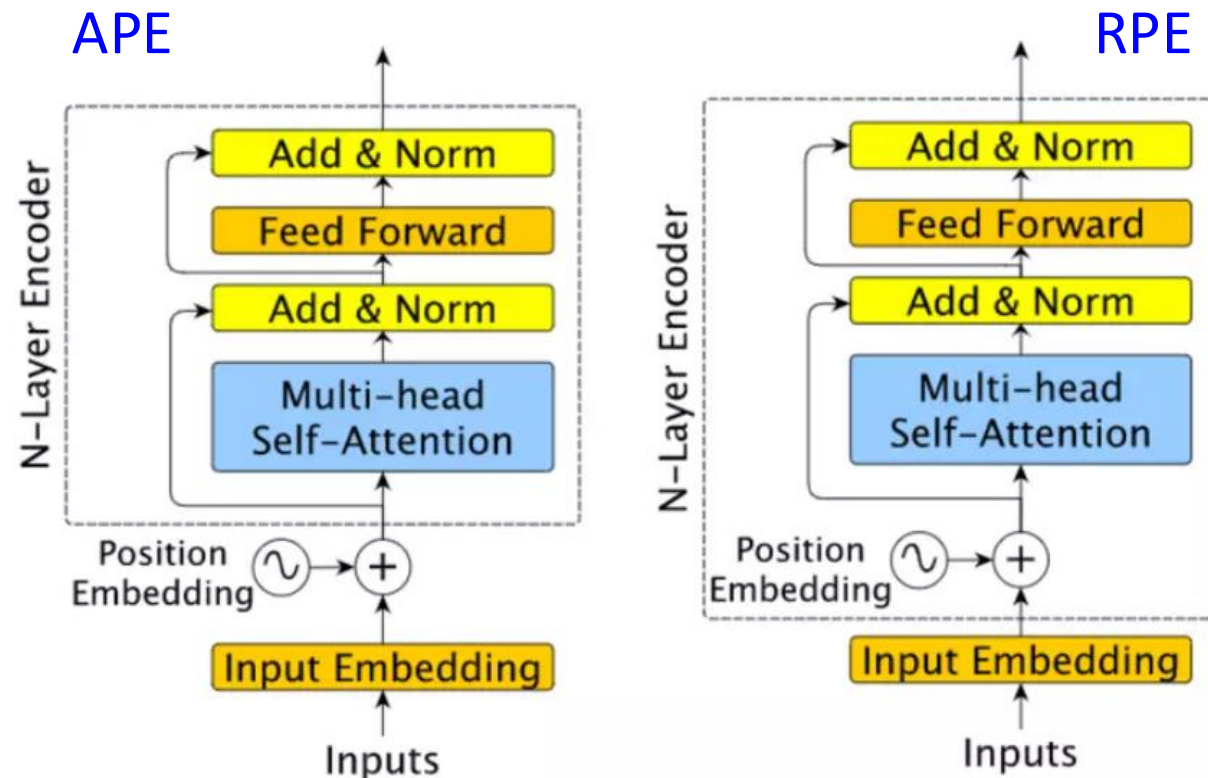
- In the vanilla Transformer, absolute position embedding is employed.
- Absolute position embedding (APE): added to context representations
 - BERT, ALBERT, GPT-1, GPT-2, ELECTRA



<https://www.slideshare.net/slideshow/roformer-enhanced-transformer-with-rotary-position-embedding/250482951>

Positional Embedding

- Relative position embedding (RPE)
 - Add position information to a model



LLaMA 2/3

```
class Attention(nn.Module):
    def forward(
        """
        Forward pass of the attention module.

        Args:
            x (torch.Tensor): Input tensor.
            start_pos (int): Starting position for caching.
            freqs_cis (torch.Tensor): Precomputed frequency tensor.
            mask (torch.Tensor, optional): Attention mask tensor.

        Returns:
            torch.Tensor: Output tensor after attention.

        """
        bsz, seqlen, _ = x.shape
        xq, xk, xv = self.wq(x), self.wk(x), self.wv(x)

        xq = xq.view(bsz, seqlen, self.n_local_heads, self.head_dim)
        xk = xk.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)
        xv = xv.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)

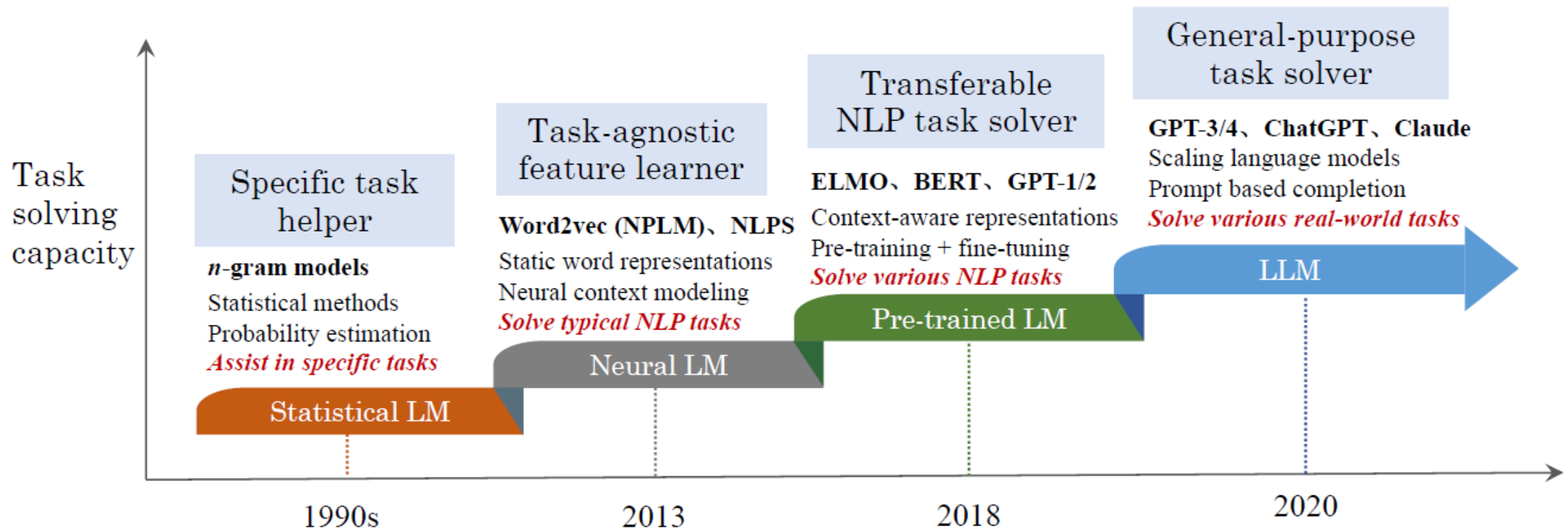
        xq, xk = apply_rotary_emb(xq, xk, freqs_cis=freqs_cis)
```


Summary

Configuration	Method	Equation
Normalization position	Post Norm [22]	$\text{Norm}(\mathbf{x} + \text{Sublayer}(\mathbf{x}))$
	Pre Norm [26]	$\mathbf{x} + \text{Sublayer}(\text{Norm}(\mathbf{x}))$
	Sandwich Norm [255]	$\mathbf{x} + \text{Norm}(\text{Sublayer}(\text{Norm}(\mathbf{x})))$
Normalization method	LayerNorm [256]	$\frac{\mathbf{x} - \mu}{\sigma} \cdot \gamma + \beta, \quad \mu = \frac{1}{d} \sum_{i=1}^d x_i, \quad \sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2}$
	RMSNorm [257]	$\frac{\mathbf{x}}{\text{RMS}(\mathbf{x})} \cdot \gamma, \quad \text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}$
	DeepNorm [258]	$\text{LayerNorm}(\alpha \cdot \mathbf{x} + \text{Sublayer}(\mathbf{x}))$
Activation function	ReLU [259]	$\text{ReLU}(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0})$
	GeLU [260]	$\text{GeLU}(\mathbf{x}) = 0.5\mathbf{x} \otimes [1 + \text{erf}(\mathbf{x}/\sqrt{2})], \quad \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$
	Swish [261]	$\text{Swish}(\mathbf{x}) = \mathbf{x} \otimes \text{sigmoid}(\mathbf{x})$
	SwiGLU [262]	$\text{SwiGLU}(\mathbf{x}_1, \mathbf{x}_2) = \text{Swish}(\mathbf{x}_1) \otimes \mathbf{x}_2$
	GeGLU [262]	$\text{GeGLU}(\mathbf{x}_1, \mathbf{x}_2) = \text{GeLU}(\mathbf{x}_1) \otimes \mathbf{x}_2$
Position embedding	Absolute [22]	$\mathbf{x}_i = \mathbf{x}_i + \mathbf{p}_i$
	Relative [82]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{x}_j^T \mathbf{W}_k^T + r_{i-j}$
	RoPE [263]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{R}_{\Theta, i-j} \mathbf{x}_j^T \mathbf{W}_k^T = (\mathbf{W}_q \mathbf{x}_i \mathbf{R}_{\Theta, i})(\mathbf{W}_k \mathbf{x}_j \mathbf{R}_{\Theta, j})^T$
	ALiBi [264]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{x}_j^T \mathbf{W}_k^T - m(i - j)$

Generations of Language Models

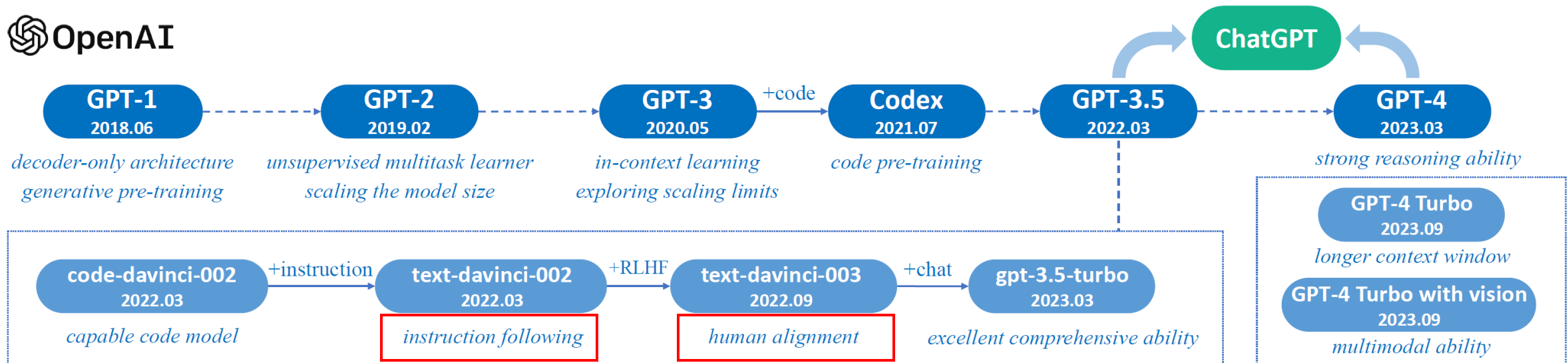
- An evolution process of the four generations of LM



Statistics of LLMs

	Model	Release Time	Size (B)	Base Model	Adaptation		Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation	
					IT	RLHF					ICL	CoT
Publicly Available	T5 [82]	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
	mT5 [83]	Oct-2020	13	-	-	-	1T tokens	-	-	-	✓	-
	PanGu- α [84]	Apr-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
	CPM-2 [85]	Jun-2021	198	-	-	-	2.6TB	-	-	-	-	-
	T0 [28]	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
	CodeGen [86]	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
	GPT-NeoX-20B [87]	Apr-2022	20	-	-	-	825GB	-	96 40G A100	-	✓	-
	Tk-Instruct [88]	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
	UL2 [89]	May-2022	20	-	-	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
	OPT [90]	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
	NLLB [91]	Jul-2022	54.5	-	-	-	-	-	-	-	✓	-
	CodeGeeX [92]	Sep-2022	13	-	-	-	850B tokens	-	1536 Ascend 910	60 d	✓	-
	GLM [93]	Oct-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
	Flan-T5 [69]	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
	BLOOM [78]	Nov-2022	176	-	-	-	366B tokens	-	384 80G A100	105 d	✓	-
	mT0 [94]	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
	Galactica [35]	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
	BLOOMZ [94]	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
	OPT-IML [95]	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
	LLaMA [57]	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
	Pythia [96]	Apr-2023	12	-	-	-	300B tokens	-	256 40G A100	-	✓	-
	CodeGen2 [97]	May-2023	16	-	-	-	400B tokens	-	-	-	✓	-
	StarCoder [98]	May-2023	15.5	-	-	-	1T tokens	-	512 40G A100	-	✓	✓
	LLaMA2 [99]	Jul-2023	70	-	✓	✓	2T tokens	-	2000 80G A100	-	✓	-
	Baichuan2 [100]	Sep-2023	13	-	✓	✓	2.6T tokens	-	1024 A800	-	✓	-
	QWEN [101]	Sep-2023	14	-	✓	✓	3T tokens	-	-	-	✓	-
	FLM [102]	Sep-2023	101	-	✓	-	311B tokens	-	192 A800	22 d	✓	-
	Skywork [103]	Oct-2023	13	-	-	-	3.2T tokens	-	512 80G A800	-	✓	-

GPT-Series of OpenAI



Motivation

- Language modeling \neq assisting users
 - Language models are not aligned with user intent: do completion instead of instruction following

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

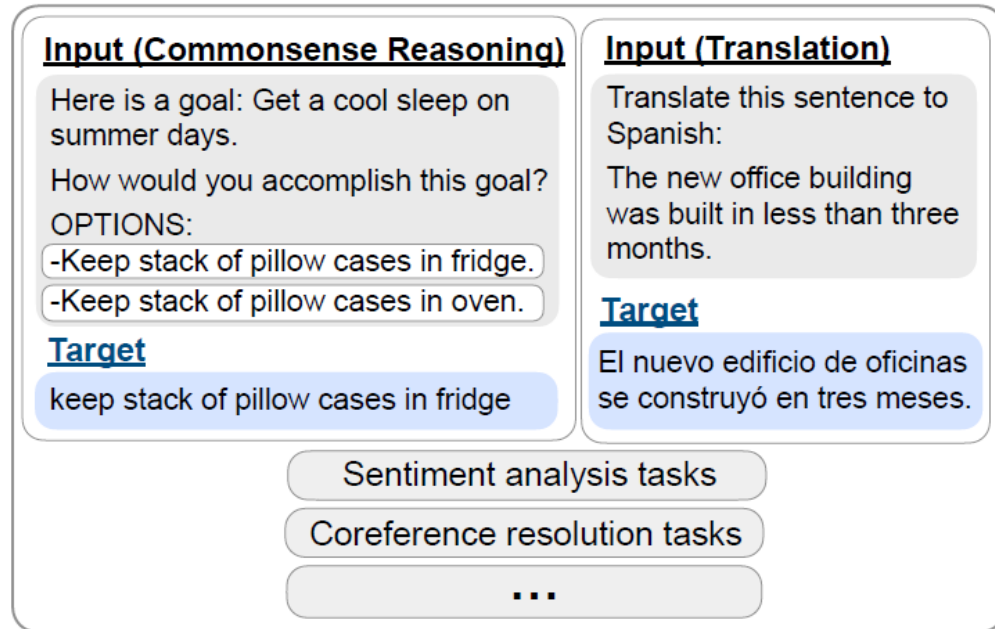
InstructGPT

People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.

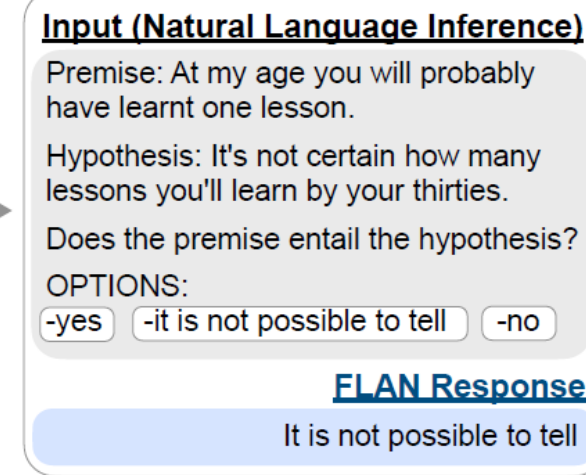
FLAN (Wei et al., 2022)

- Finetuning language models on a collection of datasets described via instructions → improves zero-shot performance on unseen tasks

Finetune on many tasks (“instruction-tuning”)



Inference on unseen task type



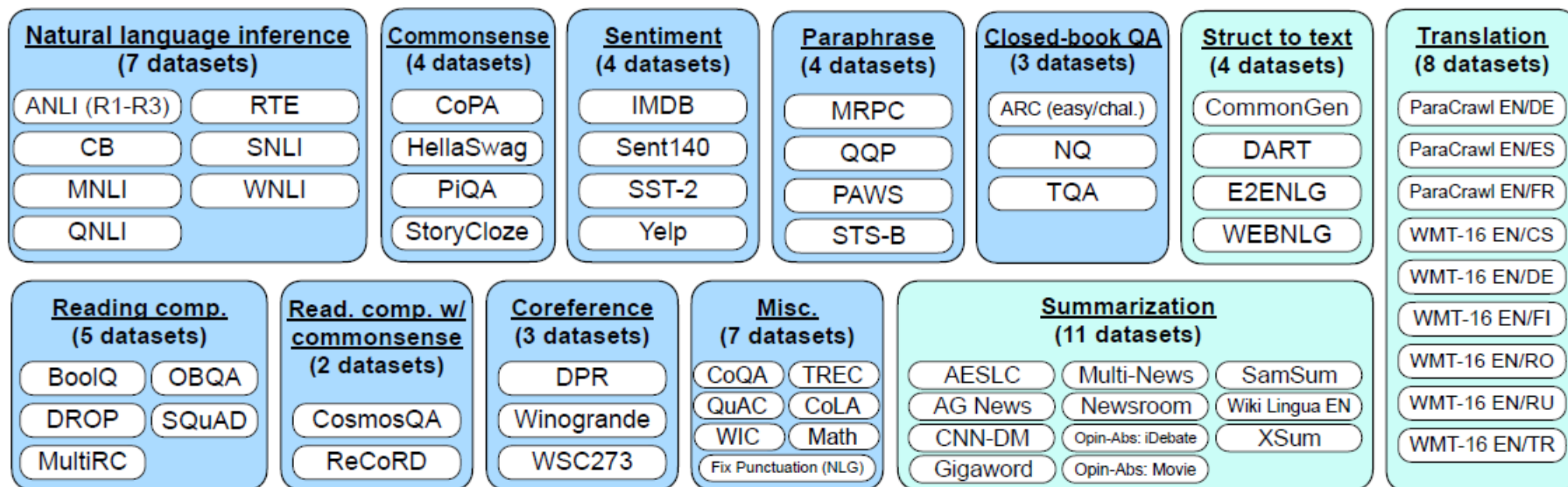
```
{
  "instruction": ...,
  "input": ...,
  "output": ...
}
```



Training with
Seq2Seq loss

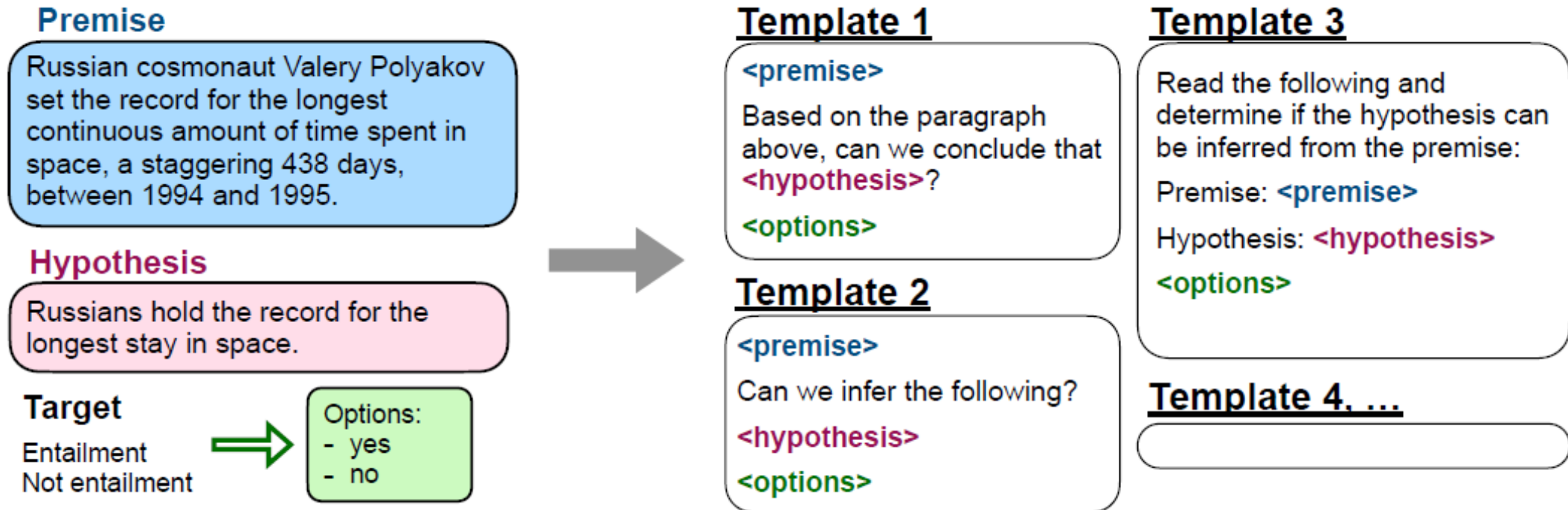
FLAN (Wei et al., 2022)

- Tasks and templates
 - Transform existing datasets into an instructional format



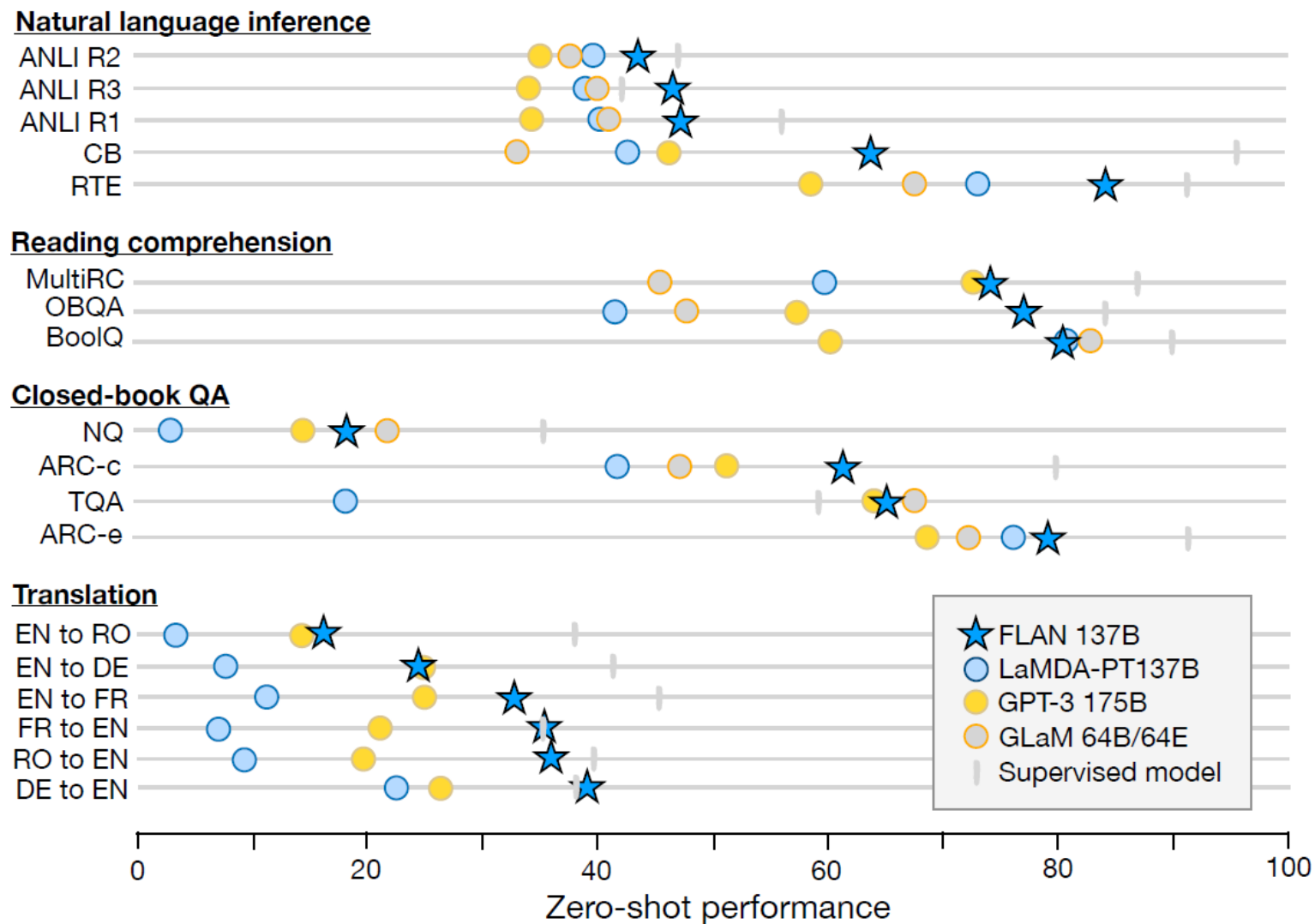
FLAN (Wei et al., 2022)

- Tasks and templates
 - Compose 10 unique templates to describe the task

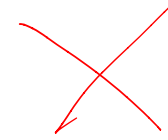


FLAN (Wei et al., 2022)

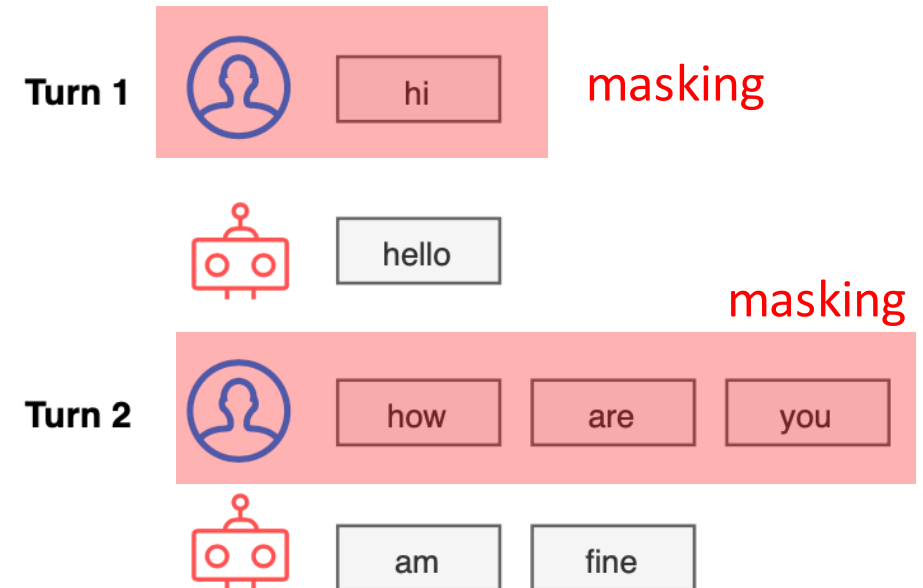
- Zero-shot performance



Training Multi-Turn Chat Data

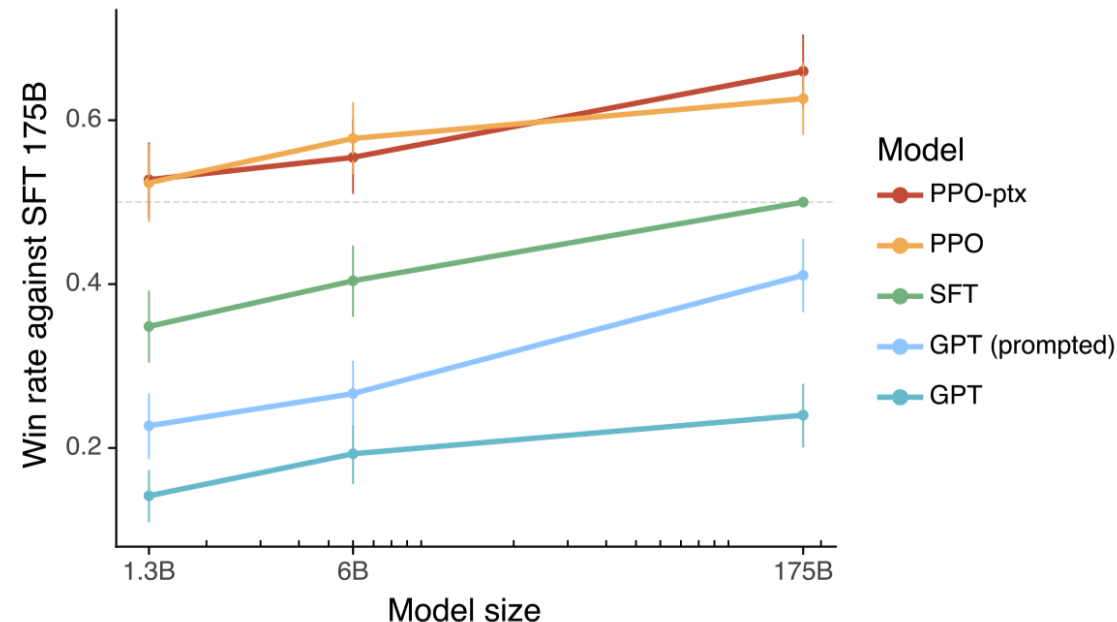


- Efficient training for multi-turn chat data
 - A straightforward way: split it into multiple context-response pairs for training
 - I.e., a LLM is fine-tuned to generate the response based on the corresponding context for all splits (at each utterance from the user).
- Vicuna (Chiang et al., 2023)
 - employs a loss mask that only computes the loss on the responses.
 - It significantly reduce the compute costs derived from the overlapped utterances.



InstructGPT

- Why do we need the alignment?
 - Language modeling objective (i.e., next token prediction) is misaligned with the objective “follow the user’s instructions helpfully and safely”.
- Learning good examples at sentence-level rather than token-level

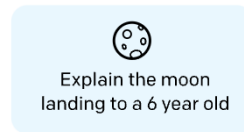


InstructGPT

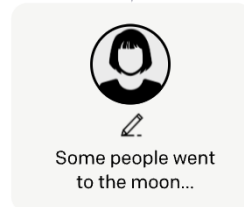
Step 1

Collect demonstration data, and train a supervised policy.

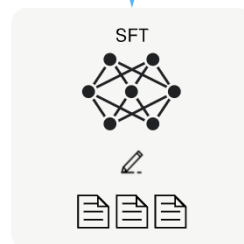
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



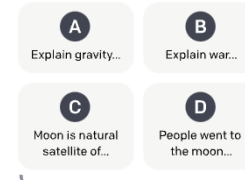
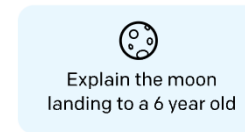
This data is used to fine-tune GPT-3 with supervised learning.



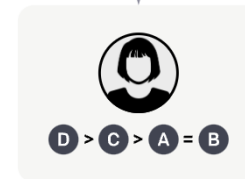
Step 2

Collect comparison data, and train a reward model.

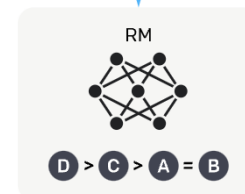
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



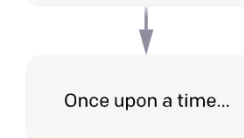
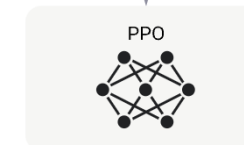
Step 3

Optimize a policy against the reward model using reinforcement learning.

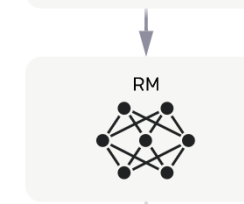
A new prompt is sampled from the dataset.



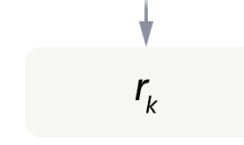
The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



Reference

- Survey of LLMs
 - <http://arxiv.org/abs/2303.18223>