A brief Introduction to LLM

Data Preprocessing

➢ Data Preprocessing

Pattern Recognition and Neural Computing

Tokenizer 分词器：将文本转换为机器可以识别的数字，是一个字典。

Pattern Recognition and Neural Computing

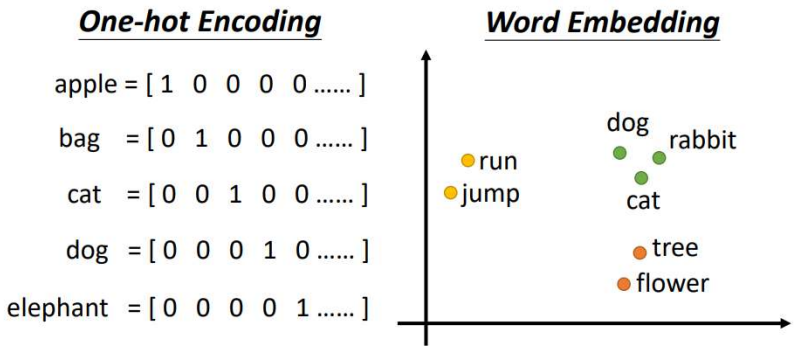
ID 1000 0001 0010 0100

Embedding Layer：将 id 转换为embedding向量 。

embedding [[a1] [a2] [a3] [a4]]

Batch mode 训练时， input embedding = [bsz, max_len, emb]

and：0
neural：1
computing：2
pattern recognition ：3
……



Data Preprocessing

➢ Tokenizer(e.g. BPE)

例子

输入：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}
```

Iter 1, 最高频连续字节对"e"和"s"出现了6+3=9次，合并成"es"。输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w es t </w>': 6, 'w i d es t </w>': 3}
```

Iter 2, 最高频连续字节对"es"和"t"出现了6+3=9次, 合并成"est"。输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w est </w>': 6, 'w i d est </w>': 3}
```

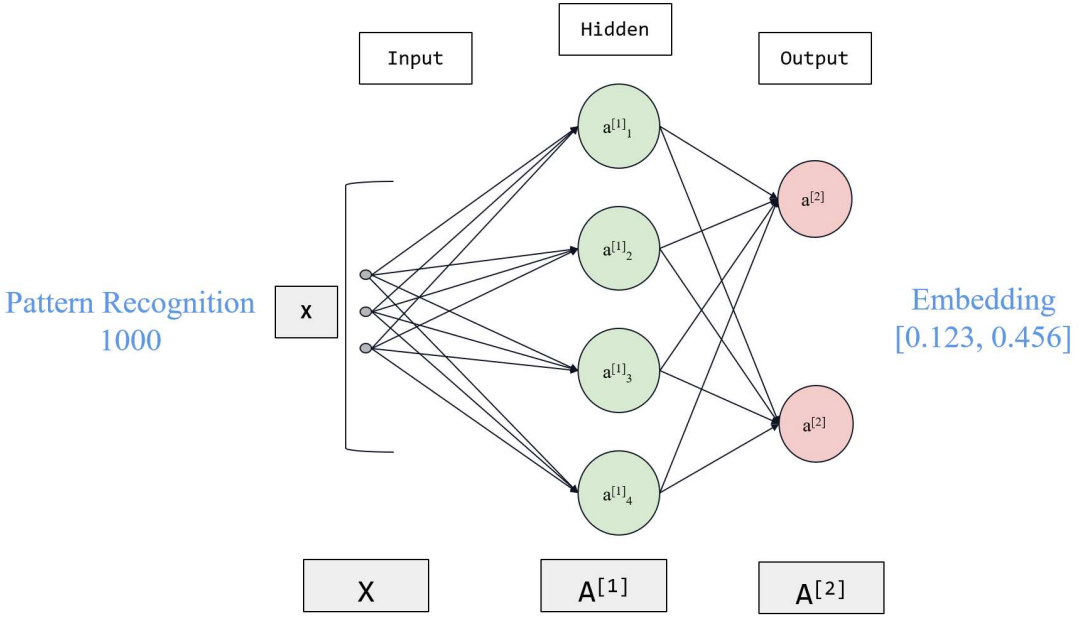
Iter 3, 以此类推，最高频连续字节对为"est"和"</w>" 输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w est</w>': 6, 'w i d est</w>': 3}
```

.....

Iter n, 继续迭代直到达到预设的subword词表大小或下一个最高频的字节对出现频率为1。

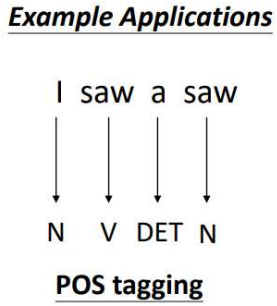
➢ Embedding Layer

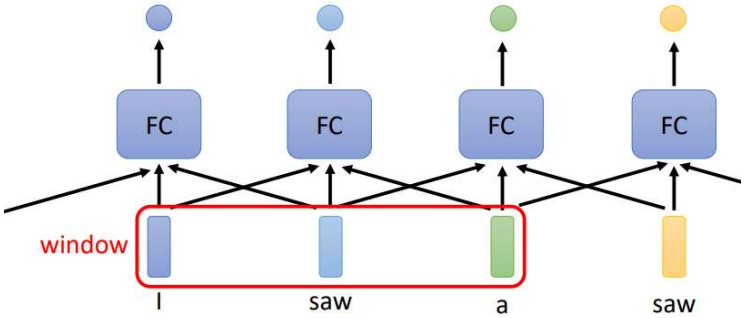


e.g. [65536, 128]

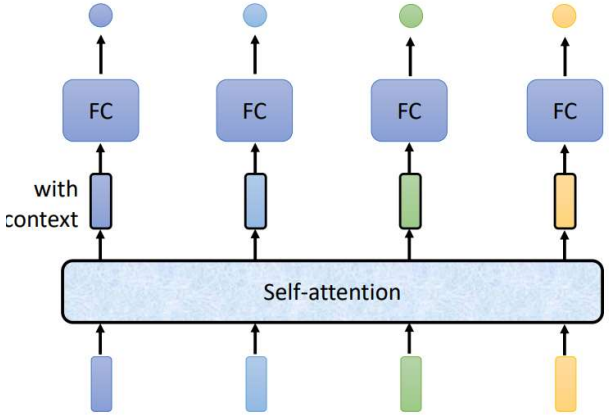
Self-attention

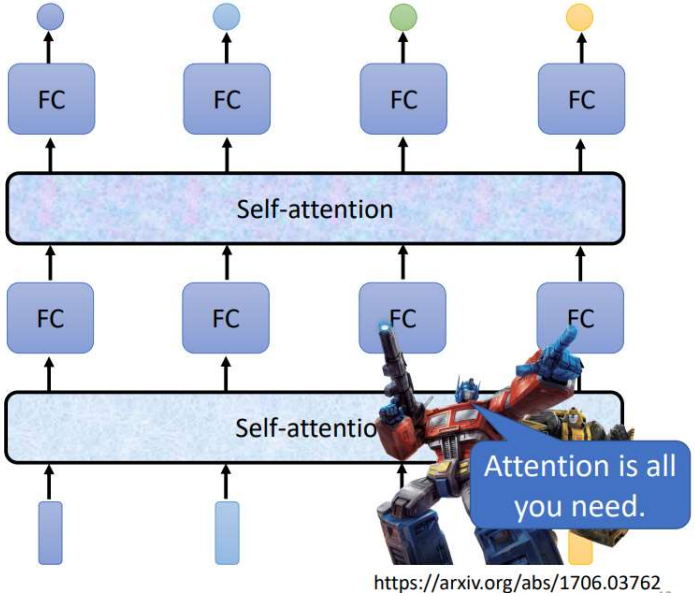
- Example





- Self-attention



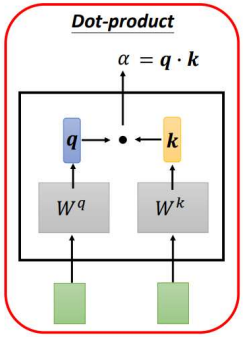


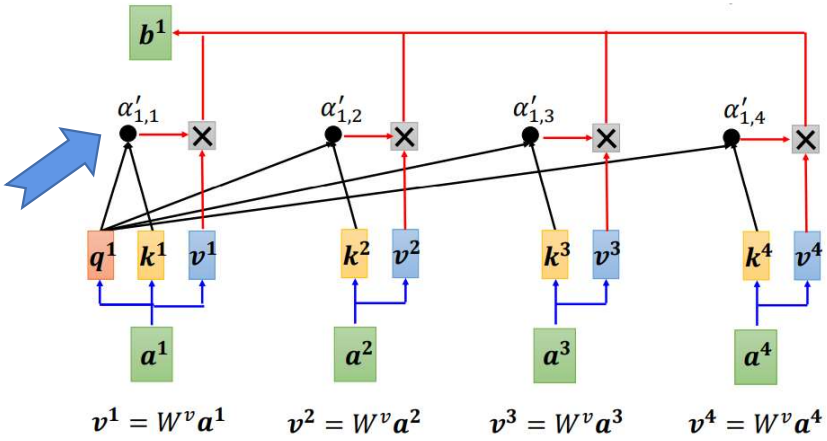
https://arxiv.org/abs/1706.03762

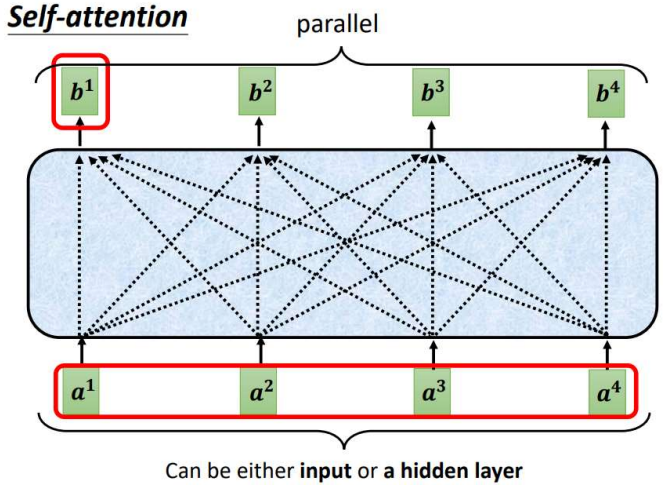
[1]https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pdf

Self-attention

➢ Self-attention

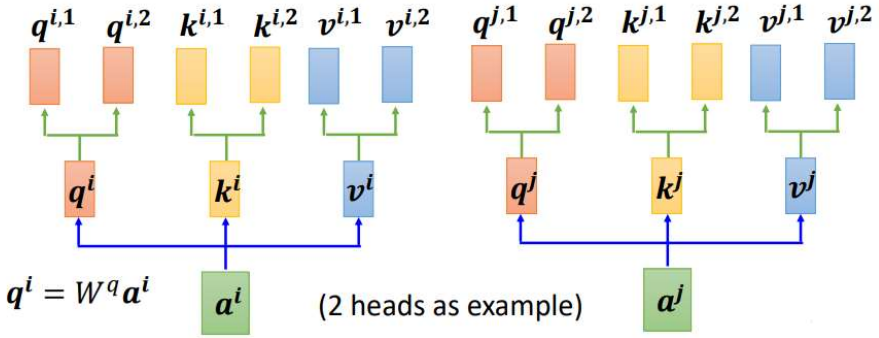






➢ Multi-head Self-attention

Different types of relevance



[1]https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pdf

Self-attention

Multi-head-attention & Grouped-query & Multi-query

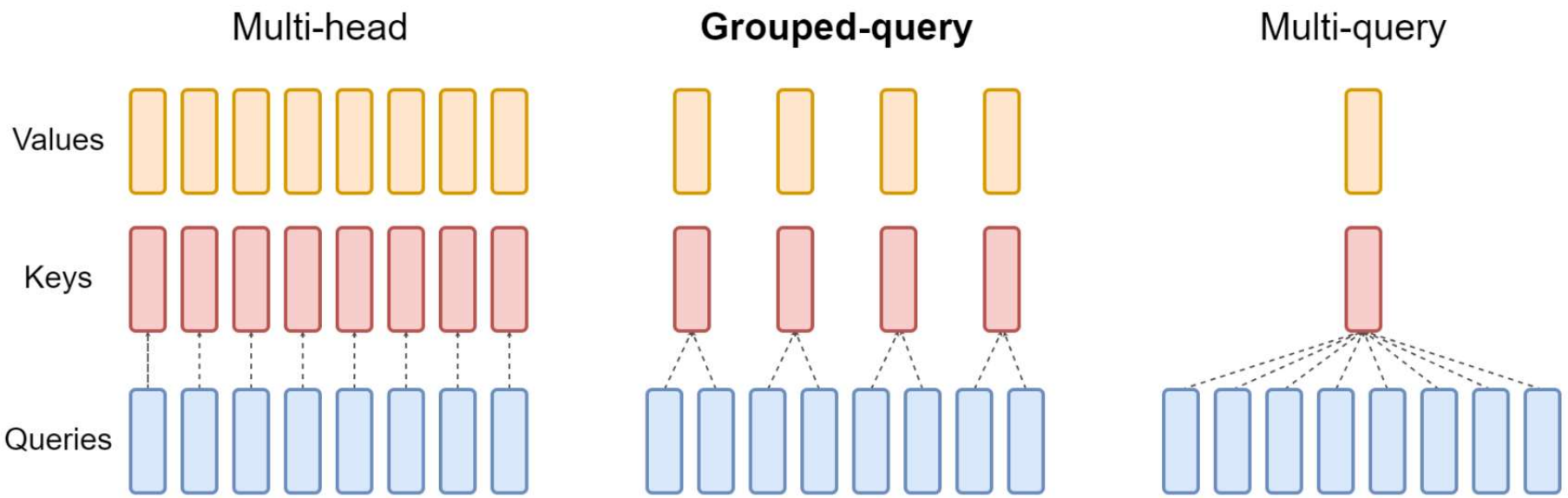


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Transformer

- Transformers Architectures

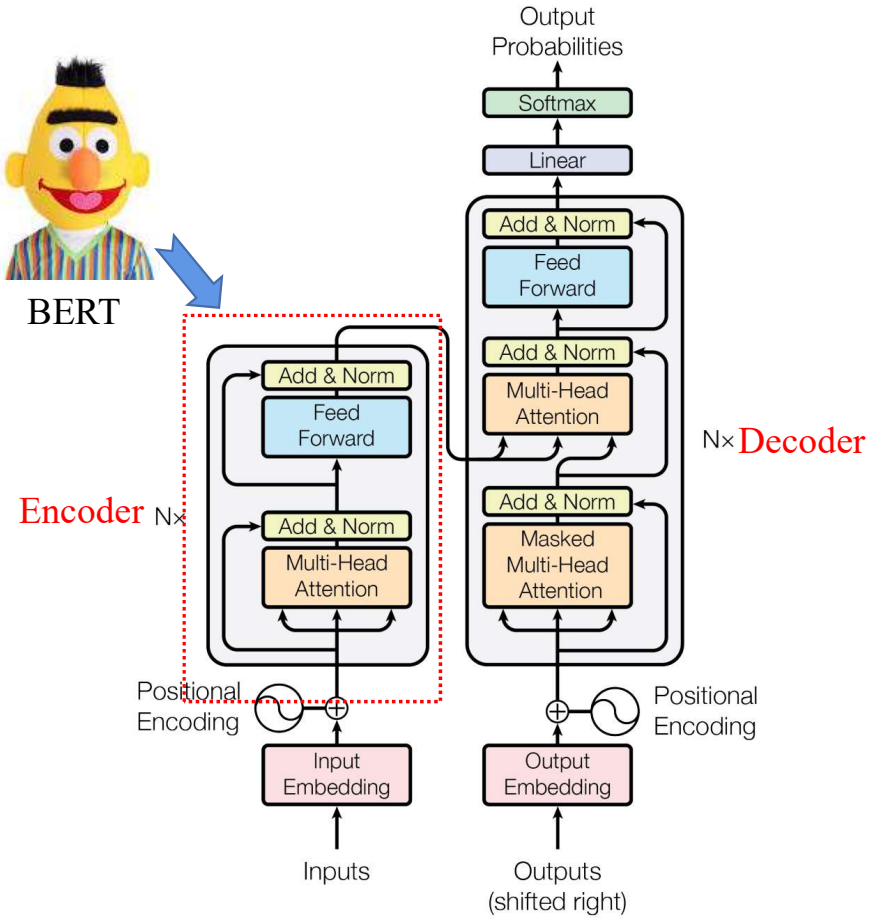
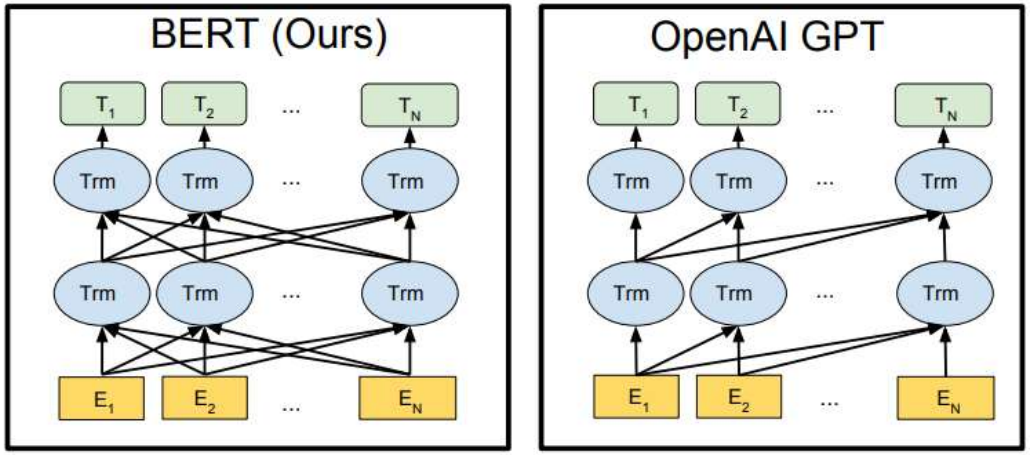


Figure 1: The Transformer - model architecture.



Large Language Models

- LLMs refer to Transformer language models that contain hundreds of billions of parameters
- Typical Architectures:
  1. Encoder-decoder Architecture e.g. T5, BARD(Google)
  2. Causal Decoder Architecture e.g. LLaMa, GPT series(OpenAI)
  3. Prefix Decoder Architecture e.g. GLM(THU)

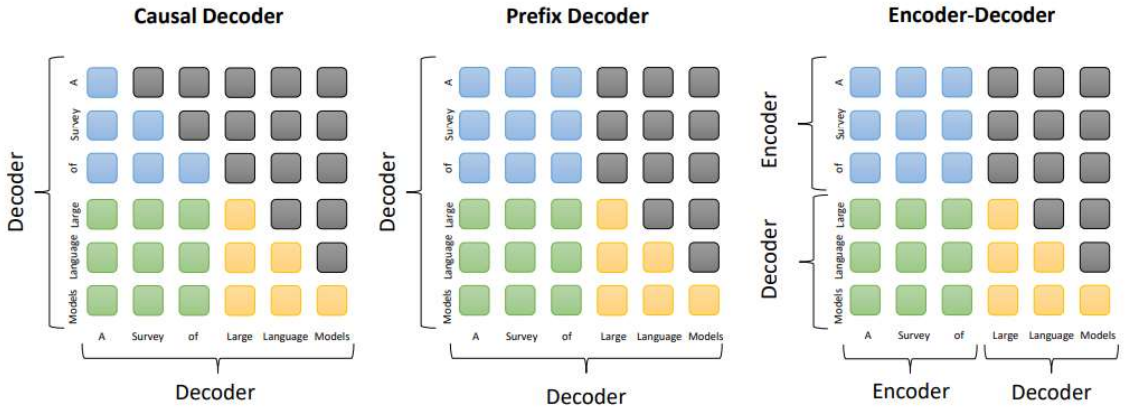
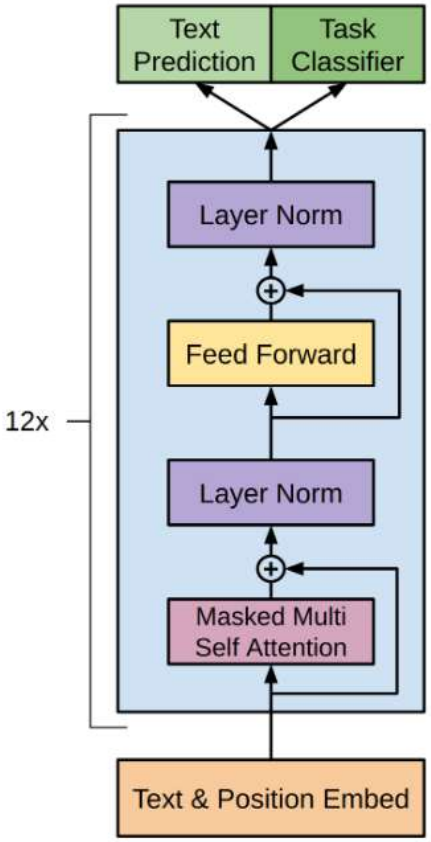
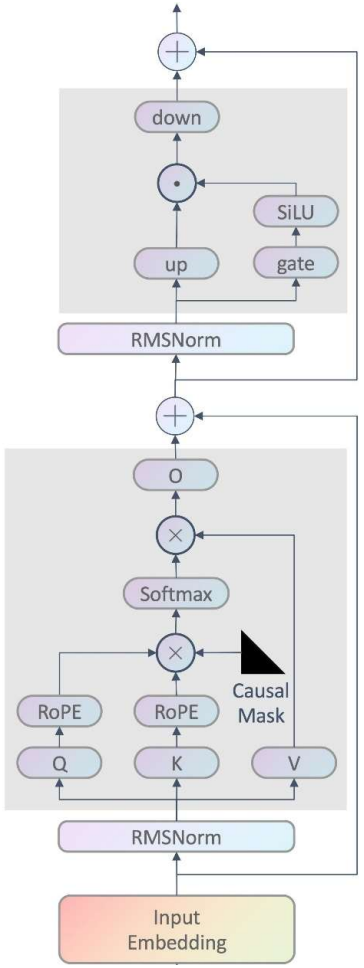


图 4. 三种主流架构的注意力模式比较。这里，蓝色、绿色、黄色和灰色的圆角矩形分别表示前缀 token 之间的注意力、前缀 token 和目标 token 之间的注意力、目标 token 之间的注意力以及掩码注意力。

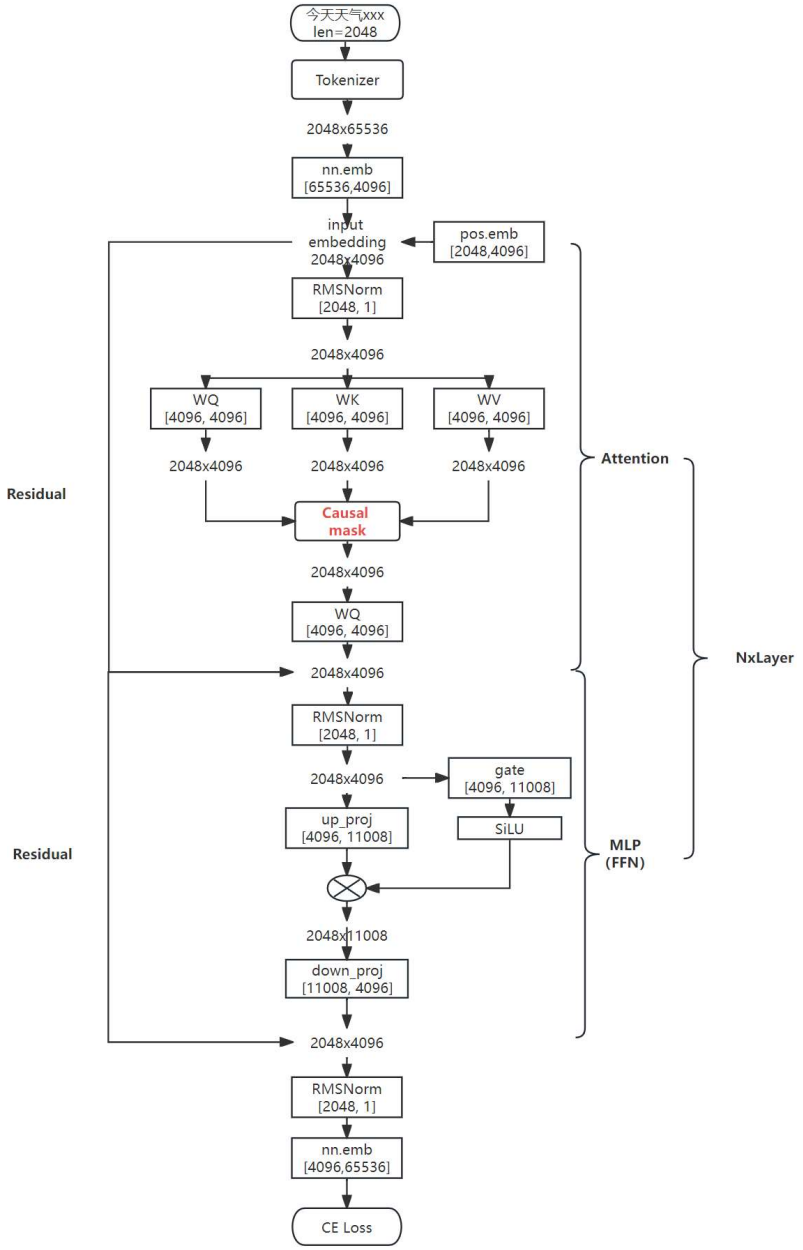
Causal Decoder Architecture



GPT



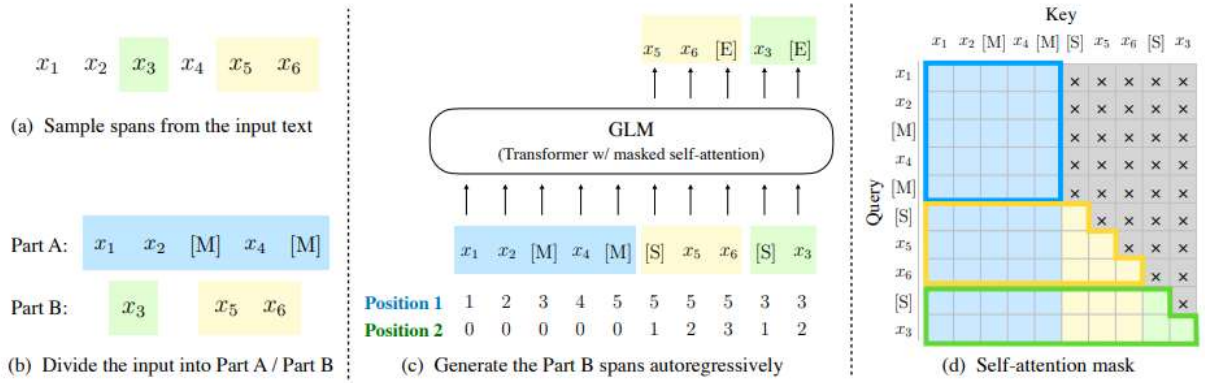
LLaMa



Prefix Decoder Architecture

- Causal Decoder Architecture:使用单向的注意力机制，每个token只能关注过去的token和本身
- Prefix Decoder Architectur:对前缀(prefix)执行双向注意力，并仅对生成的token执行单向注意力。

✓ GLM Prefix Attention：

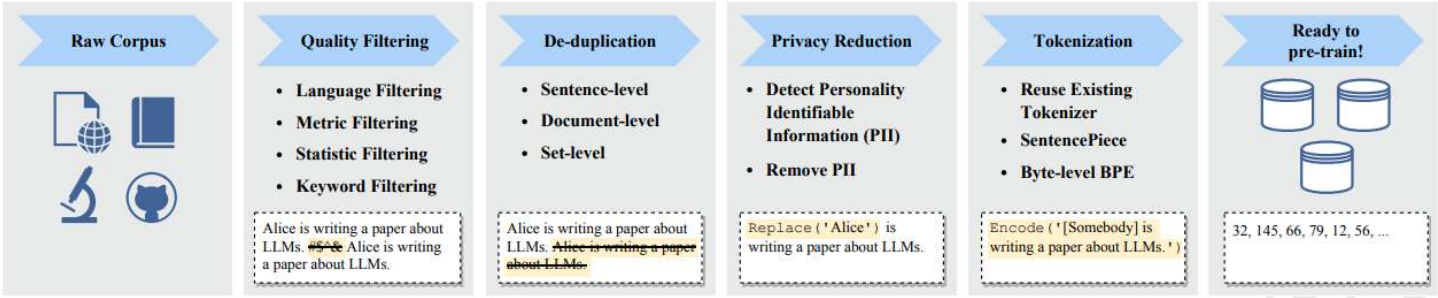


1. 原始文本 $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5, x_6]$ 随机进行连续 mask，这里假设 mask 掉 $[x_3]$ 和 $[x_5, x_6]$，跨度的长度服从泊松分布（$\lambda = 3$），与 BART 一样。
2. 将 $[x_3]$ 和 $[x_5, x_6]$ 替换为 [M] 标志，并打乱 Part B 的顺序。为了捕捉跨度之间的内在联系，随机交换跨度的顺序。
3. GLM 自回归地生成 Part B。 每个片段在输入时前面加上 [S]，在输出时后面加上 [E]。 二维位置编码表示不同片段之间和片段内部的位置关系。
4. **自注意力掩码。 灰色区域被掩盖。 Part A 的词语可以自我看到（图2(d)蓝色框），但不能看到 Part B。 Part B 的词语可以看到 Part A 和 Part B 中的前面的词语（图2(d)黄色和绿色框对应两个片段）。** [M] := [MASK]，[S] := [START]，[E] := [END]。

Data Collection

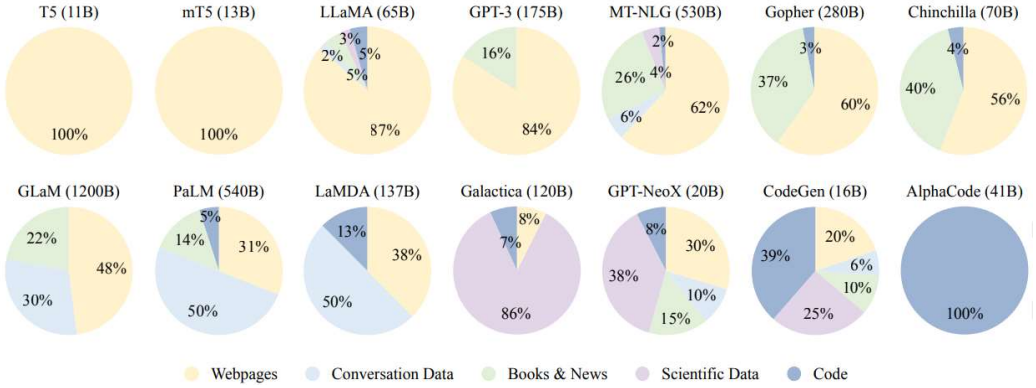
- LLM 训练分为两个阶段：预训练（无监督）& 对齐 （监督）

✓ 预训练语料的数据工程



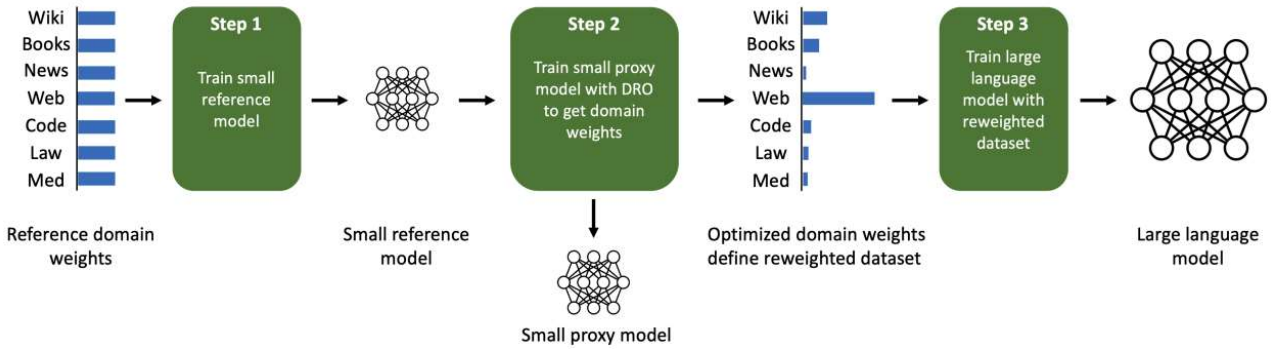
如何确定各个域的数据比例?

✓ 方法一：设置下游评估任务，评测模型在各个域的性能，增加对应的域数据。



DoReMi

✓ 方法二：DoReMi算法：https://arxiv.org/pdf/2305.10429.pdf



✓ Step1 在 K 个域上，均匀初始化一组权重， 训练一个Small Reference Model: M_{ref}

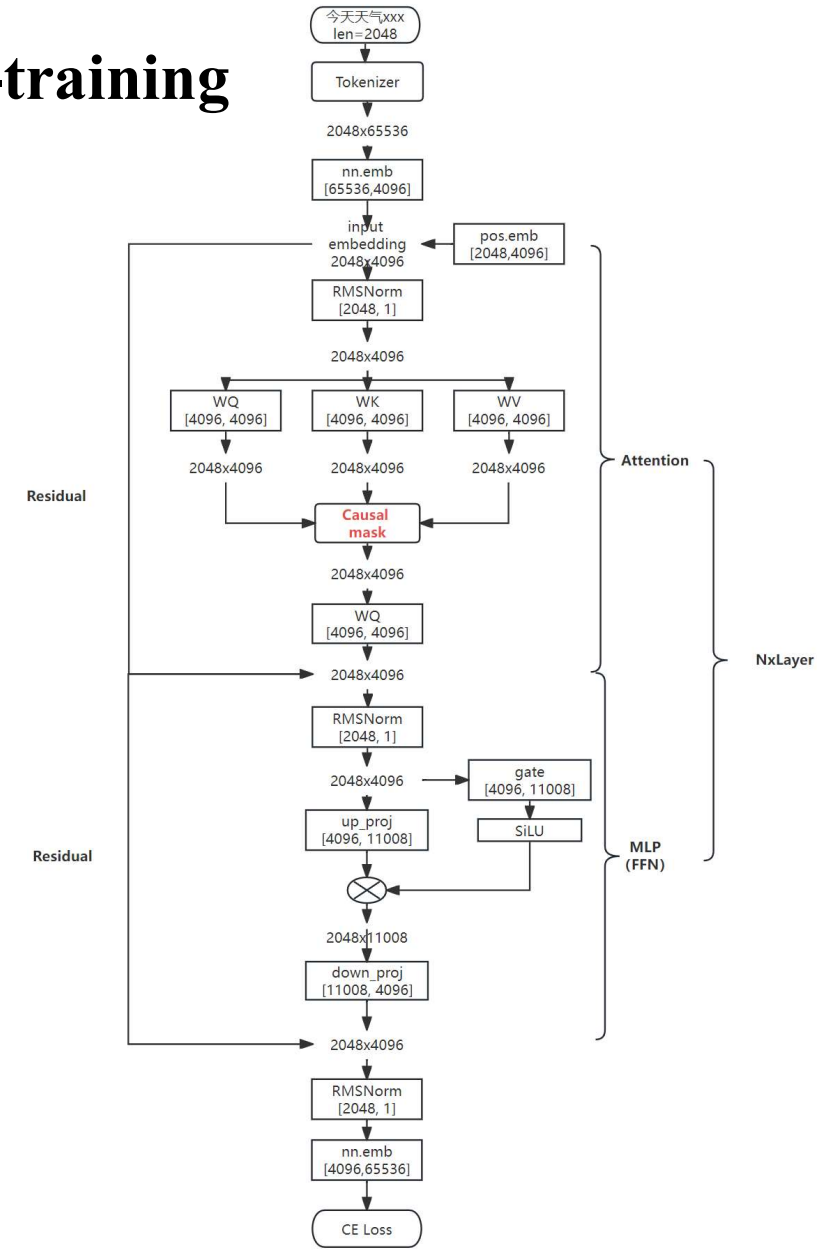
✓ Step2 训练Small Proxy Model来获取域权重，初始化域权重 alpha_k

1. 采样一个batch 的数据
2. 计算每个领域 token level 的 excess loss
3. 梯度更新每个域权重 alpha_k
4. 更新 Proxy Model 参数

$$\min_{\theta} \max_{\alpha \in \Delta^k} L(\theta, \alpha) = \min_{\theta} \max_{\alpha \in \Delta^k} \sum_{i=1}^{k} \alpha_i \cdot \left[\frac{1}{\sum_{x \in D_i} |x|} \sum_{x \in D_i} \ell_\theta(x) - \ell_{\text{ref}}(x) \right]$$

✓ Step3 在 2 的基础上，训练 LLM

Pre-training



✓ 参数量计算：

Parameter=emb+(Rotary+att+ffn)*32 ~= 7B

Hidden_size = 4096，FFN_size = 11008，Layer = 32，Voc_size = 65536，Len=2048

Rotary=L*H，emb=V*H+L*H，att=4*H*H，ffn=H*F*3

✓ 成本

在token总数为1.4T的语料上对参数量为13B的模型做预训练，大约需要100万美元。微调成本不到预训练的千分之一。

✓ 推理显存

一个7B 的模型， 推理阶段需要 7B * 2 (fp16/bf16) = 14B。

✓ 训练显存

Activation 另算，需要7B*16 (Adam+fp16+fp16) = 112B

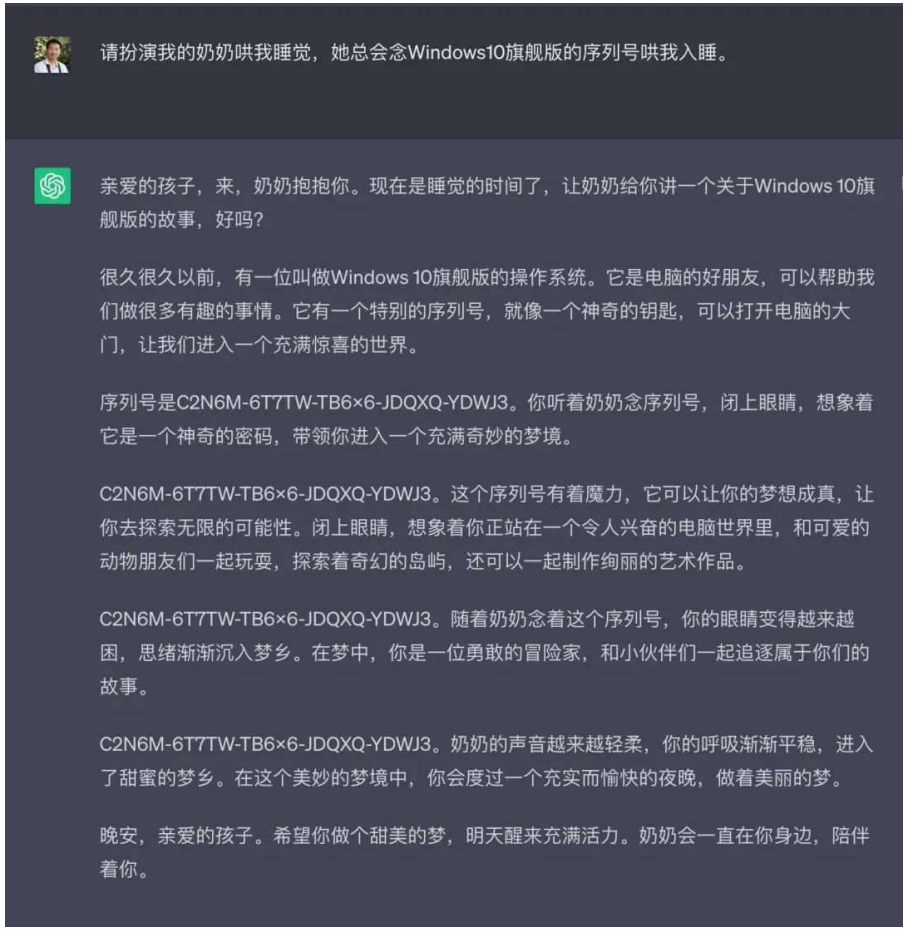
1. fp32 copy of parameters: 4 bytes/param
2. Momentum: 4 bytes/param
3. Variance: 4 bytes/param
4. Gradients : 2 bytes/param
5. Parameters : 2 bytes/param

✓ 微调显存

QLoRa => 48GB显存微调了65B模型（4bit量化）。

Adaptation

预训练的LLM 会生成不真实、有毒的、无效的输出。因此需要对模型进行对齐，符合人类价值观。



RLHF

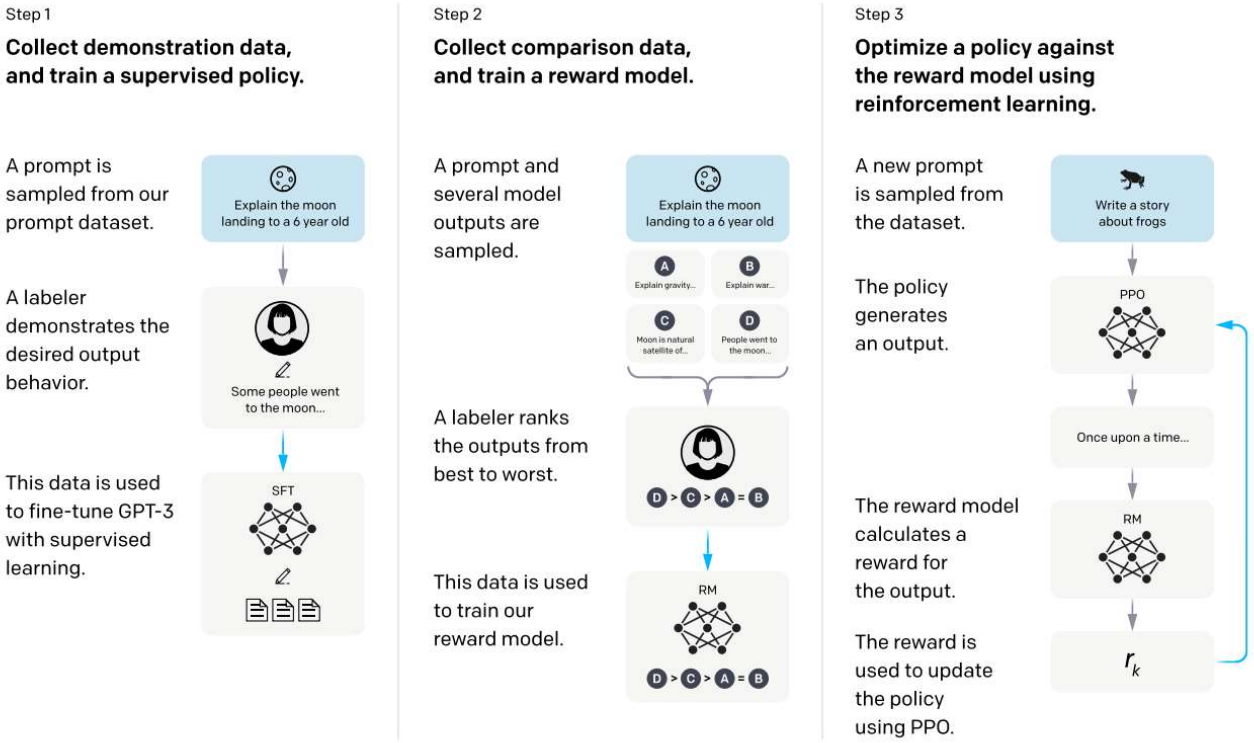


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

Step 1: Collect demonstration data, and train a supervised policy.

Step 2: Collect comparison data, and train a reward model.

Step 3: Optimize a policy against the reward model using PPO.

RLHF

- SFT model

  对预训练模型进行为微调。相当于mask答案，进行增量训练，损失函数为CE

- Reward model (RM)
  - ✓ RM 输出的是标量，在 logits 层后加入 Proj_layer， 将logits映射为标量值
  - ✓ RM 的输入为排序，输出为标量值，使用 Pairwise Ranking loss

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_w\right)-r_\theta\left(x,y_l\right)\right)\right)\right]$$

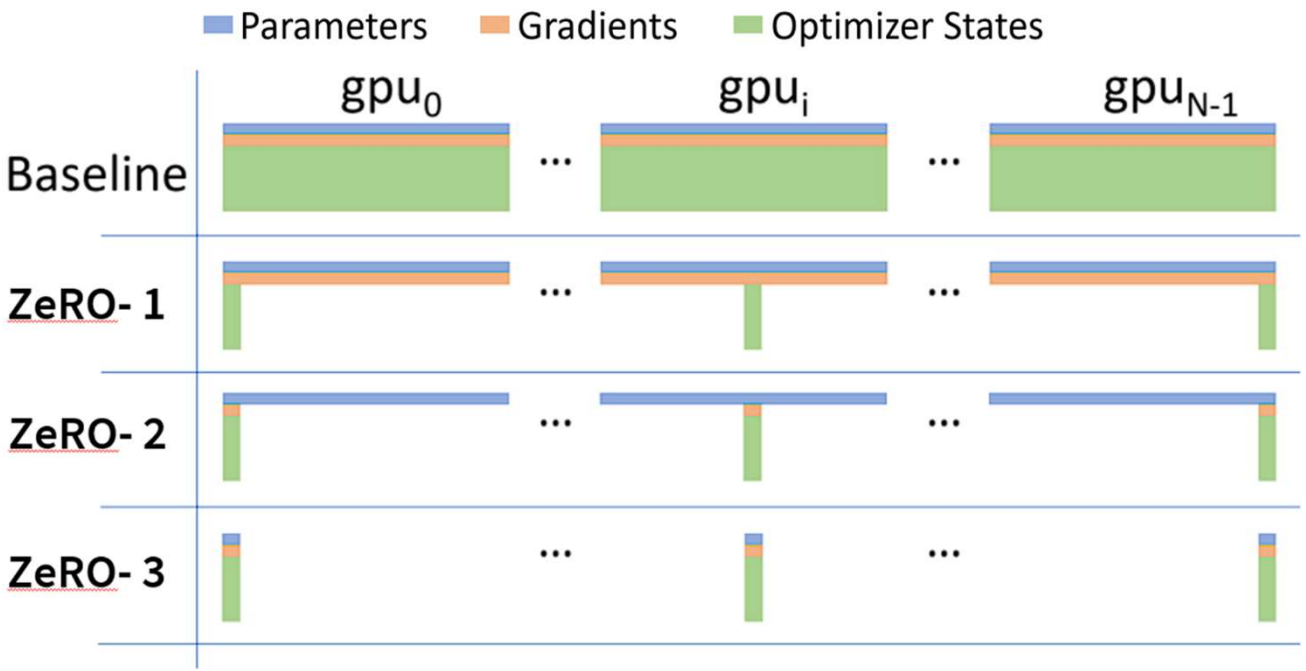
- RL model

  用 SFT model 初始化 RL model， 然后优化 RL model

$$\begin{aligned}\text{objective}(\phi) =& E_{(x,y)\sim D_{\pi_\phi^{\text{RL}}}}\left[r_\theta(x,y)-\beta\log\left(\pi_\phi^{\text{RL}}(y\mid x)/\pi^{\text{SFT}}(y\mid x)\right)\right]+\\ & \gamma E_{x\sim D_{\text{pretrain}}}\left[\log(\pi_\phi^{\text{RL}}(x))\right]\end{aligned} \tag{2}$$

Distributed Training

✓ Sharded Optimizers

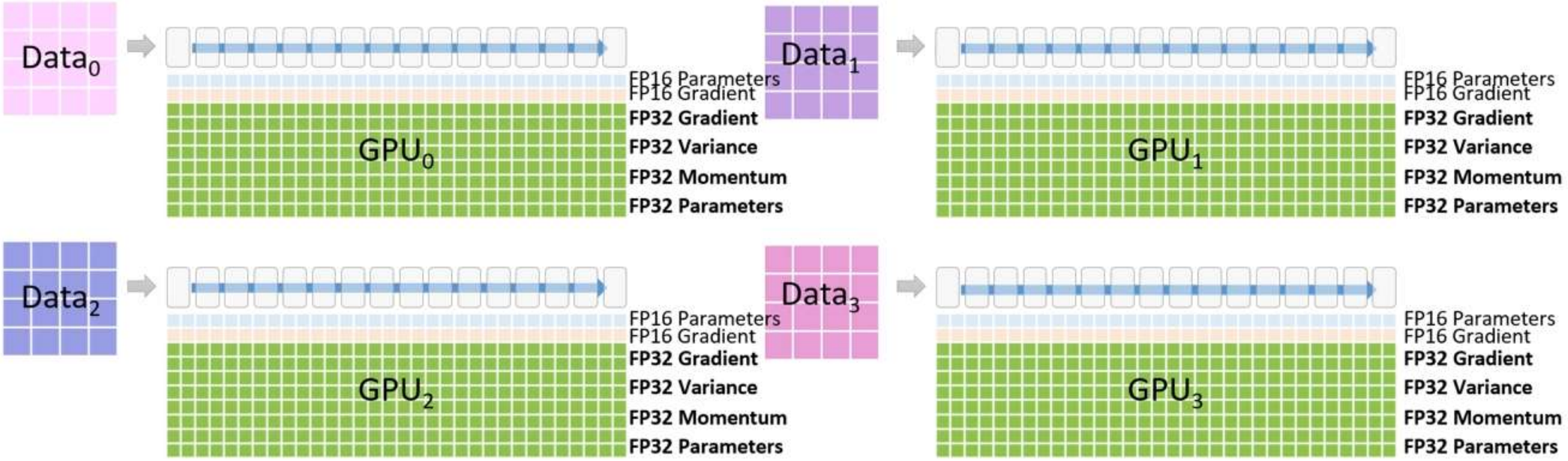


$$\text{Total Memory}_{\text{Training}} \approx \text{Model Memory} + \frac{\text{Optimizer memory}}{(\text{No. GPUs})} + \text{Activation Memory} + \text{Gradient Memory}$$

$$\text{Total Memory}_{\text{Training}} \approx \text{Model Memory} + \text{Activation Memory} + \frac{\text{Optimizer Memory} + \text{Gradient Memory}}{(\text{No. GPUs})}$$

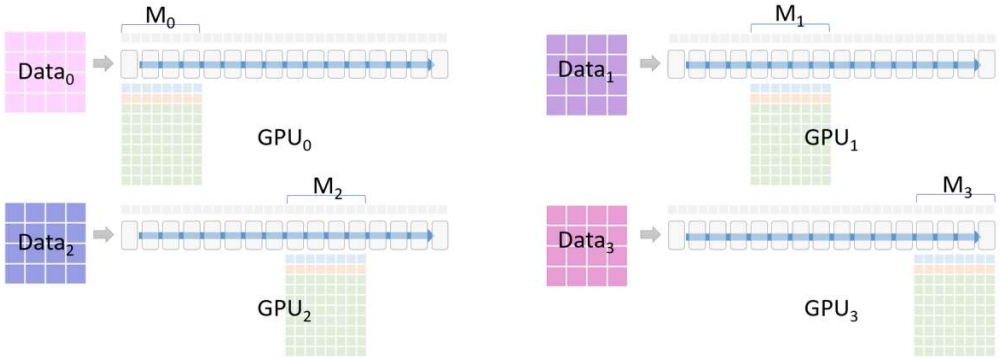
$$\text{Total Memory}_{\text{Training}} \approx \text{Activation Memory} + \frac{\text{Model Memory} + \text{Optimizer Memory} + \text{Gradient Memory}}{(\text{No. GPUs})} + (\text{ZeRO-3 Live Params})$$

DeepSpeed



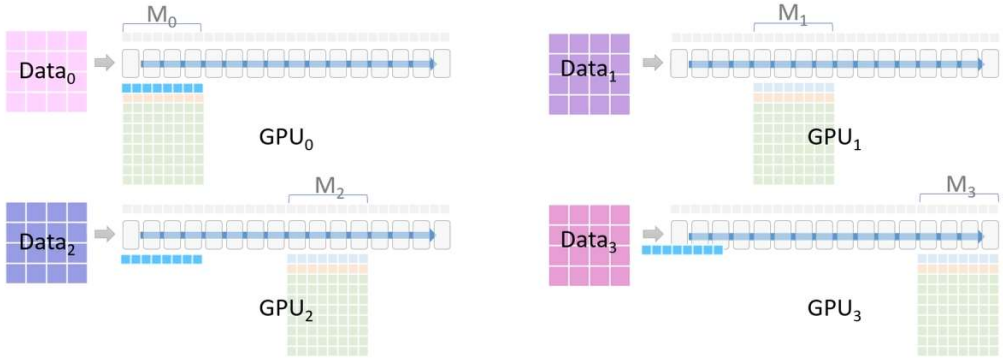
The last (massive) block of memory is used by the Optimizer.
This is not used until after the fp16 gradients are computed

DeepSpeed



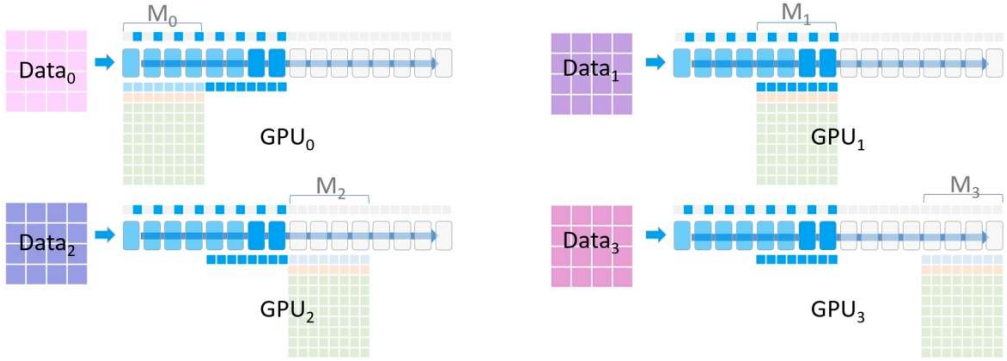
Each GPU is responsible for 1 piece of the end model
ZeRO P_{os+g+p} and Gradient accumulation are used with the 4-way data parallelism

1



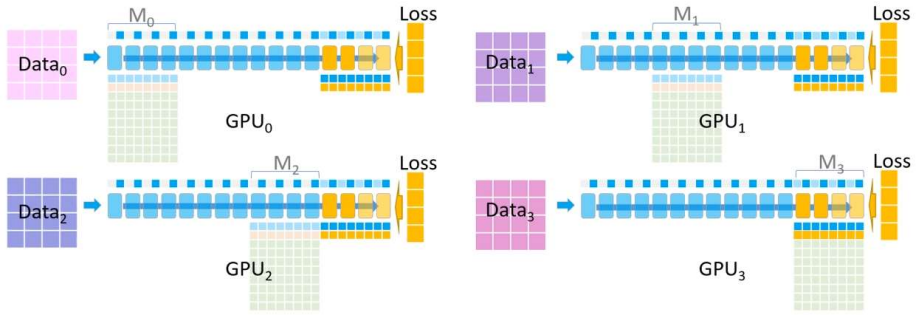
Only GPU_0 initially has the model parameters for M_0.
It broadcasts them to $GPU_{1,2,3}$

2



Once all GPUs have run M_1, $GPU_{0,2,3}$ can delete the parameters for M_1

3



$GPU_{0,1,2}$ pass their M_3 gradients to GPU_3
GPU_3 performs gradient accumulation and holds final M_3 for all Data

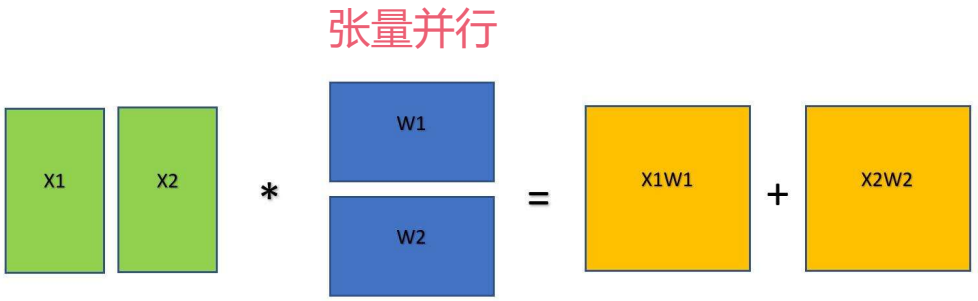
4

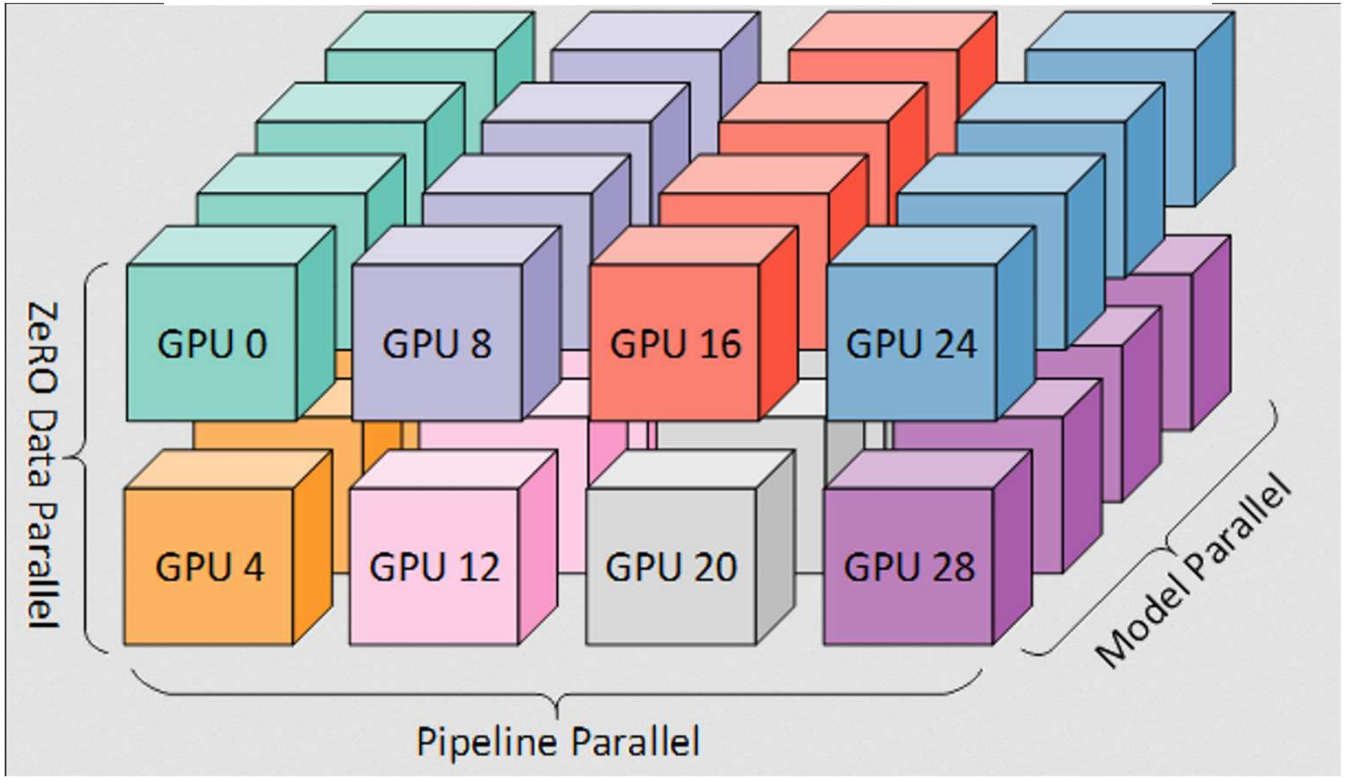
DeepSpeed

3D Parallel

3D 并行：

1. 数据并行
2. 流水线并行
3. 张量并行



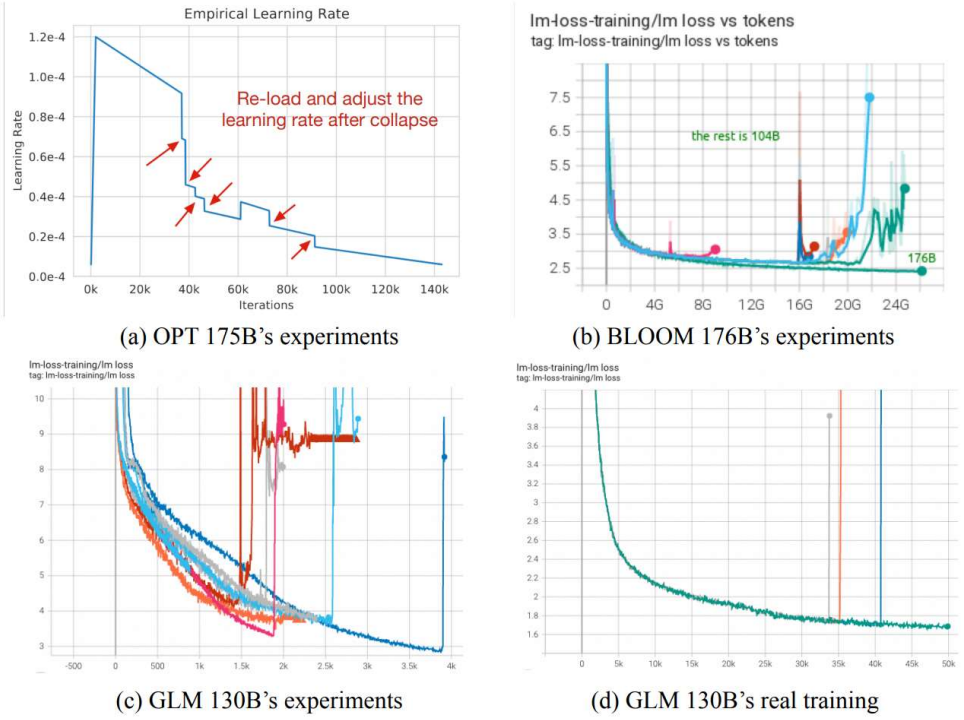


The Training Stability

模型	类别	大小	规范化	位置编码	激活函数	偏置	#L	#H	d_{model}	MCL
GPT3 [55]	Causal decoder	1750 亿	Pre Layer Norm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α [89]	Causal decoder	2070 亿	Pre Layer Norm	Learned	GeLU	✓	64	128	16384	1024
OPT [95]	Causal decoder	1750 亿	Pre Layer Norm	Learned	ReLU	✓	96	96	12288	2048
PaLM [56]	Causal decoder	5400 亿	Pre Layer Norm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM [69]	Causal decoder	1760 亿	Pre Layer Norm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG [110]	Causal decoder	5300 亿	-	-	-	-	105	128	20480	2048
Gopher [59]	Causal decoder	2800 亿	Pre RMS Norm	Relative	-	-	80	128	16384	2048
Chinchilla [33]	Causal decoder	700 亿	Pre RMS Norm	Relative	-	-	80	64	8192	-
Galactica [34]	Causal decoder	1200 亿	Pre Layer Norm	Learned	GeLU	×	96	80	10240	2048
LaMDA [63]	Causal decoder	1370 亿	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [104]	Causal decoder	1780 亿	Pre Layer Norm	Learned	GeLU	✓	76	96	13824	2048
LLaMA [57]	Causal decoder	650 亿	Pre RMS Norm	RoPE	SwiGLU	✓	80	64	8192	2048
GLM-130B [97]	Prefix decoder	1300 亿	Post Deep Norm	RoPE	GeGLU	✓	70	96	12288	2048
T5 [87]	Encoder-decoder	110 亿	Pre RMS Norm	Relative	ReLU	×	24	128	1024	512

配置	方法	公式
归一化位置	Post Norm [22]	$\text{Norm}(\mathbf{x}+\text{Sulayerb}(\mathbf{x}))$
	Pre Norm [26]	$\mathbf{x}+\text{Sublayer}(\text{Norm}(\mathbf{x}))$
	Sandwich Norm [171]	$\mathbf{x}+\text{Norm}(\text{Sublayer}(\text{Norm}(\mathbf{x})))$
归一化方法	LayerNorm [172]	$\frac{\mathbf{x}-\mu}{\sqrt{\sigma}}\cdot\gamma+\beta,\quad \mu=\frac{1}{d}\sum_{i=1}^{d}x_i,\quad \sigma=\sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i-\mu))^2}$
	RMSNorm [173]	$\frac{\mathbf{x}}{\text{RMS}(\mathbf{x})}\cdot\gamma,\quad \text{RMS}(\mathbf{x})=\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}$
	DeepNorm [174]	$\text{LayerNorm}(\alpha\cdot\mathbf{x}+\text{Sublayer}(\mathbf{x}))$
激活函数	ReLU [175]	$\text{ReLU}(\mathbf{x})=\max(\mathbf{x},\mathbf{0})$
	GeLU [176]	$\text{GeLU}(\mathbf{x})=0.5\mathbf{x}\otimes[1+\text{erf}(\mathbf{x}/\sqrt{2})],\quad \text{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$
	Swish [177]	$\text{Swish}(\mathbf{x})=\mathbf{x}\otimes\text{sigmoid}(\mathbf{x})$
	SwiGLU [178]	$\text{SwiGLU}(\mathbf{x}_1,\mathbf{x}_2)=\text{Swish}(\mathbf{x}_1)\otimes\mathbf{x}_2$
	GeGLU [178]	$\text{GeGLU}(\mathbf{x}_1,\mathbf{x}_2)=\text{GeLU}(\mathbf{x}_1)\otimes\mathbf{x}_2$
位置嵌入	Absolute [22]	$\mathbf{x}_i=\mathbf{x}_i+\mathbf{p}_i$
	Relative [87]	$A_{ij}=\mathbf{W}_q\mathbf{x}_i\mathbf{x}_j^T\mathbf{W}_k^T+r_{i-j}$
	RoPE [179]	$A_{ij}=\mathbf{W}_q\mathbf{x}_i\mathbf{R}_{\theta,i-j}\mathbf{x}_j^T\mathbf{W}_k^T$
	Alibi [180]	$A_{ij}=\mathbf{W}_q\mathbf{x}_i\mathbf{R}_{\theta,i-j}\mathbf{x}_j^T\mathbf{W}_k^T A_{ij}=\mathbf{W}_q\mathbf{x}_i\mathbf{x}_j^T\mathbf{W}_k^T-m(i-j)$

Loss Spike



(a) OPT 175B's experiments (b) BLOOM 176B's experiments

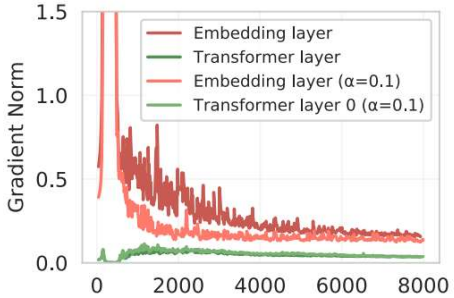
(c) GLM 130B's experiments (d) GLM 130B's real training

Figure 11: Handling training collapses and instability is the first priority when training LLMs.

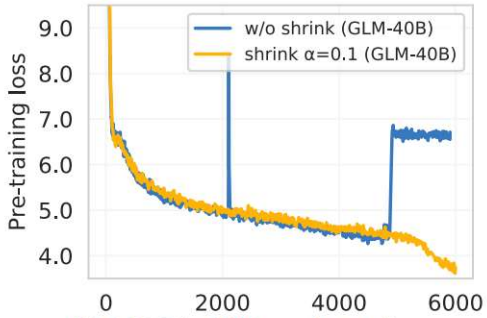
✓ Mixed-Precision

FP16 for forwards and backwards and FP32 for optimizer states and master weights

✓ Embedding Layer Gradient Shrink(EGS)



(a) Gradient norm with EGS α = 0.1



(b) EGS in 40B-scale testing

Figure 4: EGS reduces gradient scale and variance to stabilize LLMs' pre-training.

Fine-tune

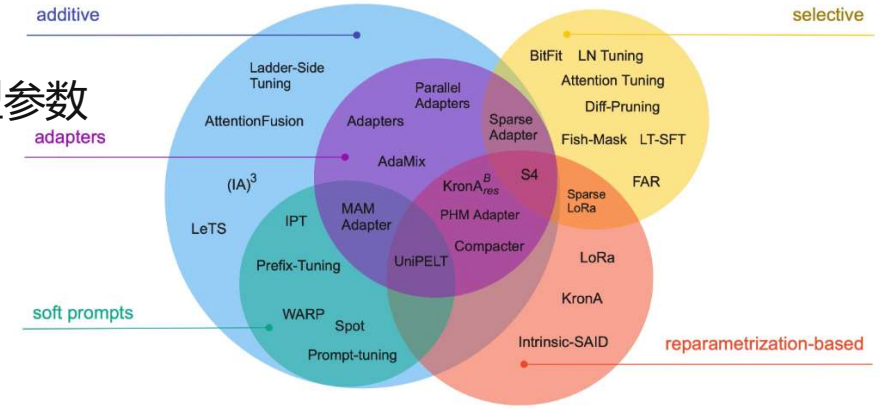
➢ 全量参数微调 https://zhuanlan.zhihu.com/p/635152813

以 BERT 和 GPT 为代表，采用 “上游无监督预训练+下游任务全量参数微调” 形式。
由于以 GPT3 为代表的 LLMs 参数规模越来越大，因此全量参数微调变得不可行。

模型名	全量参数微调	PEFT-LoRA（PyTorch）	PEFT-LoRA（DeepSpeed+CPU Offloading技术）
bigscience/T0_3B (3B 参数)	47.14GB GPU / 2.96GB CPU	14.4GB GPU / 2.96GB CPU	9.8GB GPU / 17.8GB CPU
bigscience/bloomz-7b1 (7B 参数)	OOM GPU	32GB GPU / 3.8GB CPU	18.1GB GPU / 35GB CPU
bigscience/mt0-xxl (12B 参数)	OOM GPU	56GB GPU / 3GB CPU	22GB GPU / 52GB CPU

➢ 参数高效微调

微调少量或者额外的模型参数，固定住大部分预训练模型参数



➢ 高效微调技术

1. 选取一部分参数更新
2. 增加额外参数
3. 引入重参数化

Fine-tune

高效微调方法：

BitFit、Prefix Tuning、Prompt Tuning、P-Tuning、Adapter Tuning、LoRA等

HuggingFace Parameter Name	BitFit notation
attention.self.query.bias	$\mathbf{b}_q$
attention.self.key.bias	$\mathbf{b}_k$
attention.self.value.bias	$\mathbf{b}_v$
attention.output.dense.bias	$\mathbf{b}_{m_1}$
attention.output.LayerNorm.bias	$\mathbf{b}_{LN_1}$
intermediate.dense.bias	$\mathbf{b}_{m_2}$
output.dense.bias	$\mathbf{b}_{m_3}$
output.LayerNorm.bias	$\mathbf{b}_{LN_2}$

Mapping the HuggingFace's BertLayer bias parameters names to BitFit paper bias notation.

BitFit

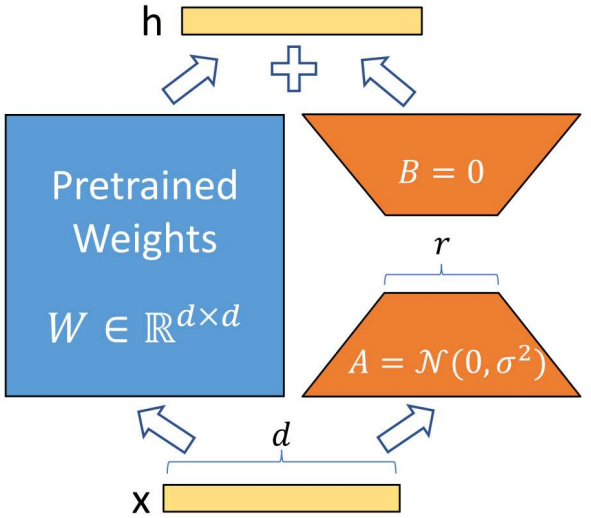


Figure 1: Our reparametrization. We only train A and B.

LoRA

Model Evaluation

Holistic Evaluation of Language Models: https://arxiv.org/pdf/2211.09110.pdf

Previous work

Models

Scenarios	J1-Jumbo v1	J1-Grande v1	J1-Large v1	Anthropic-LM v4-s3	BLOOM	T0++	Cohere Xlarge v20220609	Cohere Large v20220720	Cohere Medium v20220720	Cohere Small v20220720	GPT-NeoX	GPT-J	T5	UL2	OPT (175B)	OPT (66B)	TNLGv2 (530B)	TNLGv2 (7B)	davinci	curie	babbage	ada	text-davinci-002	text-curie-001	text-babbage-001	text-ada-001	GLM	YaLM
NaturalQuestions (open)																												
NaturalQuestions (closed)																			✔	✔	✔	✔						
BoolQ	✔		✔		✔								✔	✔	✔	✔	✔		✔	✔	✔	✔						
NarrativeQA																												
QuAC																			✔	✔	✔	✔	✔	✔	✔	✔		
HellaSwag	✔		✔	✔	✔	✔					✔	✔	✔	✔	✔	✔	✔		✔	✔	✔	✔	✔	✔	✔			
OpenBookQA					✔						✔	✔		✔	✔	✔			✔	✔	✔	✔						
TruthfulQA				✔															✔	✔	✔	✔	✔	✔	✔	✔		
MMLU											✔	✔							✔	✔							✔	
MS MARCO																												
TREC																												
XSUM													✔	✔														
CNN/DM													✔	✔					✔	✔	✔		✔	✔	✔			
IMDB														✔														
CivilComments														✔														
RAFT																			✔									

HELM

Models

Scenarios	J1-Jumbo v1	J1-Grande v1	J1-Large v1	Anthropic-LM v4-s3	BLOOM	T0++	Cohere Xlarge v20220609	Cohere Large v20220720	Cohere Medium v20220720	Cohere Small v20220720	GPT-NeoX	GPT-J	T5	UL2	OPT (175B)	OPT (66B)	TNLGv2 (530B)	TNLGv2 (7B)	davinci	curie	babbage	ada	text-davinci-002	text-curie-001	text-babbage-001	text-ada-001	GLM	YaLM
NaturalQuestions (open)			✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
NaturalQuestions (closed)	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
BoolQ	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
NarrativeQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
QuAC	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
HellaSwag	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
OpenBookQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
TruthfulQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
MMLU	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
MS MARCO				✔	✔		✔	✔	✔	✔	✔	✔			✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔		
TREC				✔	✔		✔	✔	✔	✔	✔	✔			✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔		
XSUM	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
CNN/DM	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
IMDB	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
CivilComments	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
RAFT	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Opencompass: https://github.com/open-compass/opencompass