亚马逊云科技

探秘大语言模型预训练

从预训练一个玩具LLM模型学到的

What I learned from pretraining a tiny toy LLM

Wenju Zhang, PhD
Partner Training
2024.03.08

https://github.com/colorzhang/TinyLM

Agenda

1. Why I train a useless (tiny toy) model
2. Training process & result
3. Lessons learned
4. Applications and future possibilities

Quote

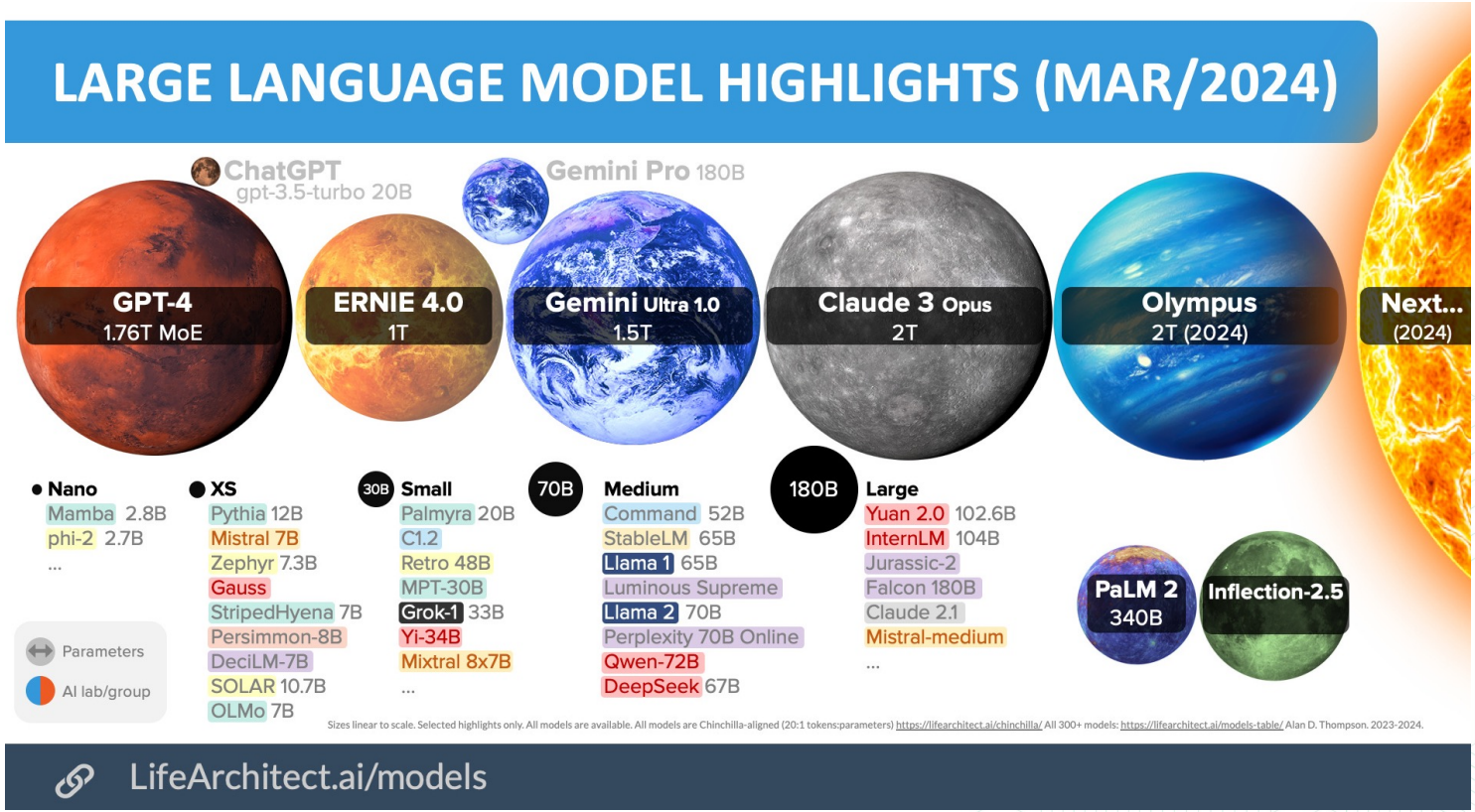
“What I cannot create, I do not understand”

-- Richard Feynman 里查德•费曼
理论物理学家，诺贝尔奖获得者

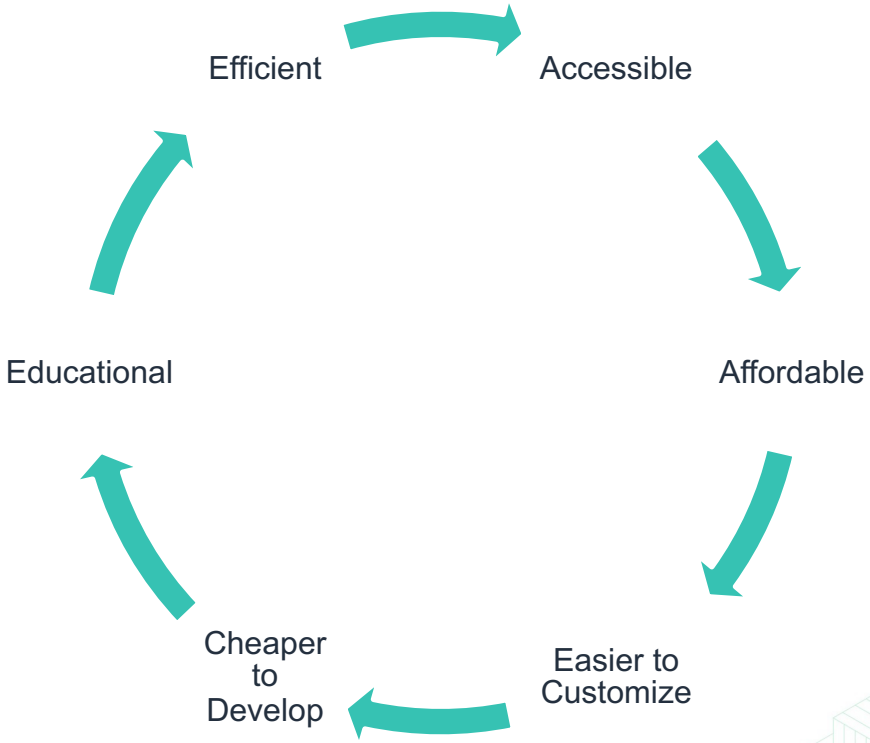
LLM scaling and size war

- Tiny

~M parameters



Why Tiny and Small language models



Why I train a useless(tiny toy) model

Personal:

1. 看懂相关论文
2. LLM黑盒→白盒 transparency
3. 了解算法与(超)参数等
4. 掌握CPT&Fine-tune理论基础
5. 熟悉HF transformers生态
6. 了解高效 training(distributed, TE/PE, flash-attention, DDP/FSDP/PP/TP)
7. 了解高效 inference(temperature, greedy, top-p, top-k)
8. Make my hands dirty

Agenda

1. Why I train a useless (tiny toy) model
2. Training process & result
3. Lessons learned
4. Applications and future possibilities

任务设定

从零训练一个LLM用来写诗

训练数据：56315首唐诗
模型：nanoGPT模型 (size: 30M/124M) | MyMistral-0.4/1B
目标：写诗
Hardware: G5.4xlarge * 1 (A10G-24G GPU mem)

“床前明月光，疑是地上霜，举头望明月，低头思故乡。”

~30M size/ 80M token / 16min / ~10 epochs

送贺六司空游蜀
李中
春日凉风泛落花，独游看暮蓼花新。
行人绿里归来去，蜀客楼前曙鸟鸿。
空想可怜非著客，远逢河北望乡人。

题江南北驿湖
徐铉
湖边岸路心绝，水渺山长路难。
那忍入沙寒雁，得经过海是人家。

初晚春望
韩愈
绿枝心未半，青山落尽时。
欲去长亭树，空山空望迟。
他日行人尽，惆怅几行衣。

送僧游江南
曹松
远别离西曲，仙人独向西。
帆当清隔下，灯下乱僧家。
自喜匡山鹤，高人只一身。
终年花落叶，共说广陵人。

~124M size/ 80M tokens / 46min / ~10 epochs

晓发翠岩寺
齐己
星映高峰出，山行步未斜。
九霄云雨足，万里鸟飞多。
古塔幽人合，空蒙古驿过。
白云秋磬度，青翰夜窗和。<|endoftext|>汴河
齐己
海内穿江水，河间采石河。
水声犹万里，人鬼欲千河。
石壁深巢月，茶灯尽涧波。
迹寻松影远，杉与江湖多。<|endoftext|>赠尊师
齐己
大师出经衲，僧窗礼佛心。
高梧见云壑，古磬入禅林。
绝顶逢金座，寒林过石林。
殷勤见终日，心在白云深。<|endoftext|>赠毕丞相里
齐己
玉音俱掩处，有似醉如泥。
千里沧溟合，千山碧落低。
有情胜弱柳，无限更相嗤。

潭州词二首 一
温庭筠
桥南日暮烟雨微，紫陌长江作鹭飞。
不似便随池上鹭，武陵滩向海边归。<|endoftext|>潭州词二首
二
温庭筠
轻舟潋滟到秦川，日日红香入苑烟。
碧嶂海山千万峰，浸天如雾半西天。<|endoftext|>赠窦兵曹
温庭筠
新春灾烬曙光攒，小吏衫襟叹此心。
红药马嘶荒径断，白烟江上远山深。
江流水急清芜晚，雨暗山深碧草深。
若使邀来只鹦鹉，预愁未觉薜萝林。<|endoftext|>和友人寄怀

李白乘舟将欲行，忽闻岸上踏歌声。
潮平海峤千帆远，雨霁江风万里情。
几处鹧鸪飞回首，青山早晚向南城。

~757M size/ 20M tokens / 77min / ~3 epochs

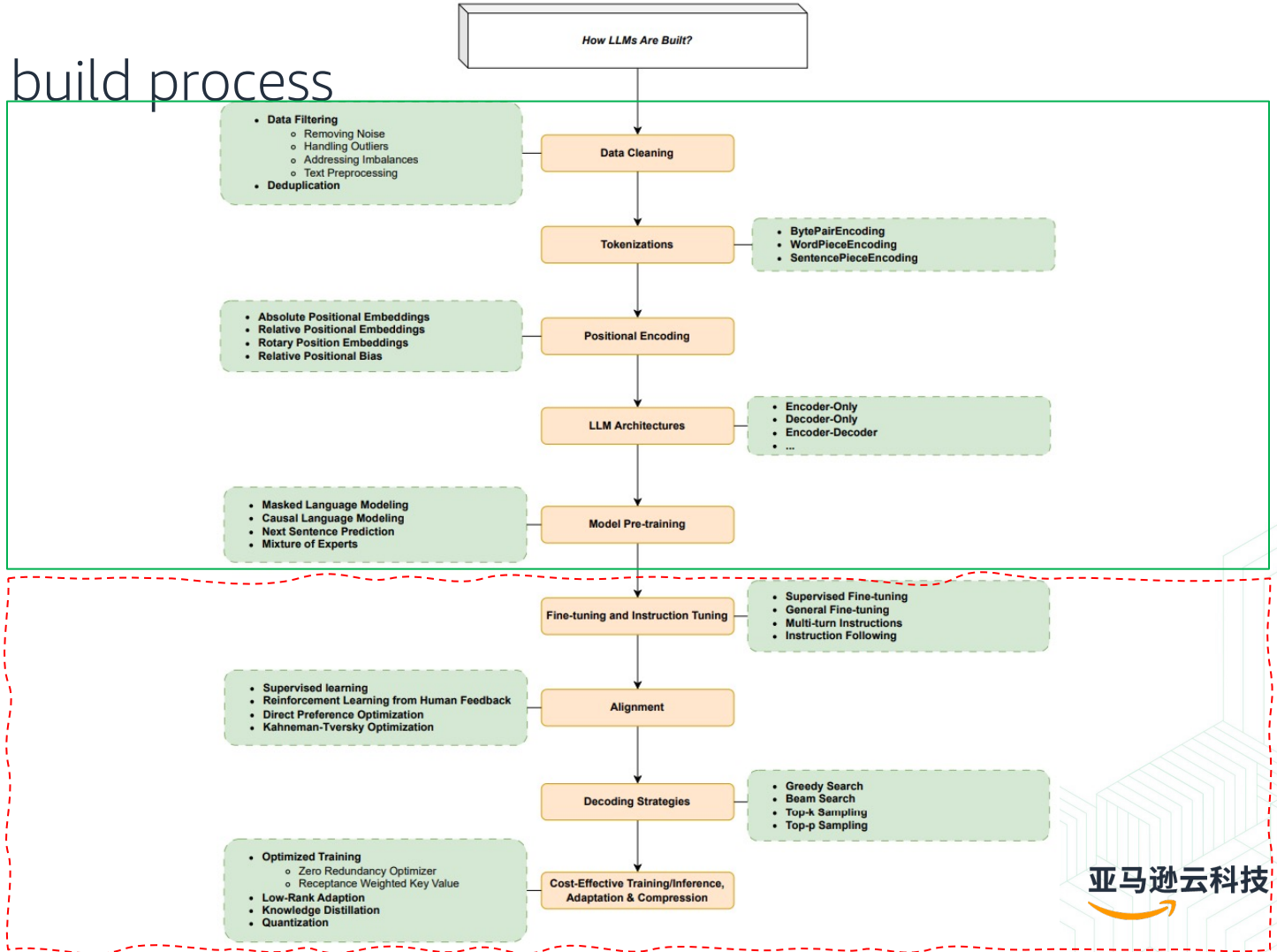
虬已秋，磬带，夫，任雄。
芫白绮思乱入发贽纠夔薅，皑送，虾枇浔子。
<|endoftext|>回迟元鹿绝
绿游砷时荒石得鸣反四令曲通
阁荁远关山阂，悠思首 印药重
然秋。
言褰卟。
烟关，□昏河陋百，迹赴
躬湿，鹐，息，轮落无未
七赏甲正。
梵知今晓。
公，回气赋，衔吏落。
相帝草附凝身江窳生咸舍池九绿，风明犹依，共烟。
深陪还倣置更智。
珠如空水流仙和仔决虎。
自，烗，不溟。
辞。
粹，灰，炴首 三一心

MyMistral-0.4B / 254min/ 6 epcochs

<s> 床前明月光，疑是地上霜
羊卧疾
坠叶浓春色，芳香满院花。
莺啼花自发，谁见岁时愁。
桂水悄然在，梨花拂艳姿。
何时惆怅望，愿与凄凄凄。
昔日趋名利，今年紫芝侪。
珥臣逐流水，寒催绿树衰。
谓言千万里，夙意百千年。
戎马迷天子，鸳鸯惜此躯。
枝枝啼茗暮，泛泛泥金缕。
湛翡翠羽盖，谁肯学丹穹。

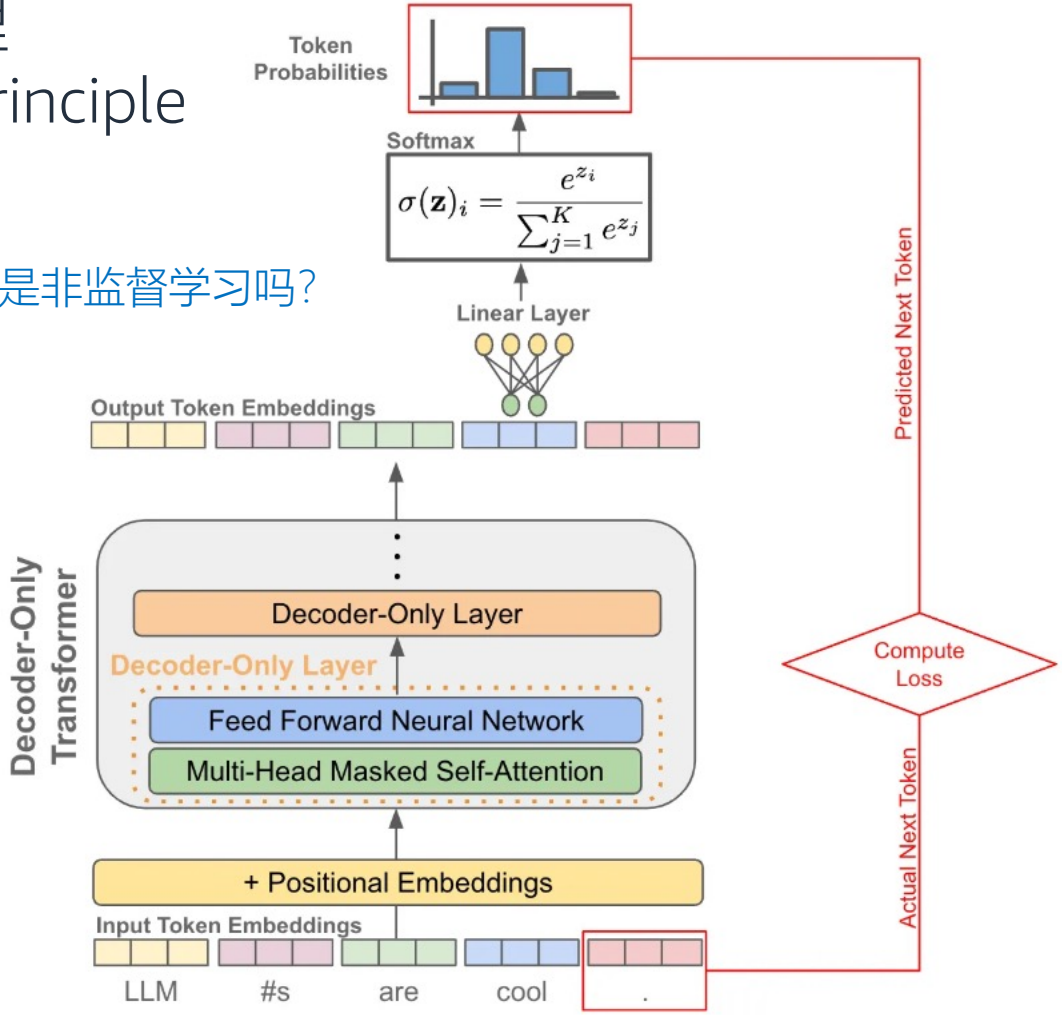
<s> 床前明月光，疑是地上霜
羊巩
戚里难逢此夜长，蟾蜍复恐晓寒浊。
莫将孤枕为谁得，更有弦歌一曲歌。
堪嗟不叹君恩泽，年少壮心相惜年。
愿作献书何所忆，岂知天地有谁荣。
秦楼贺客欺贞舜，愿君怀旧凤皇明。
岂无鸡黍皆盛名，苦寒琼树枝生枝。
祇应驱驰道弱柳，羞人渴饮酒徒然。
羞将白纶何所恨，秉知睡兮凯歌声。

LLM build process



第一性原理
LLM first principle

LLM Pre-training 是非监督学习吗?



Pretrain loss function


```
def forward(self, idx, targets=None):
     if targets is not None:
          # if we are given some desired targets also calculate the loss
          logits = self.lm_head(x)
          loss = F.cross_entropy(logits.view(-1, logits.size(-1)), targets.view(-1), ignore_index=-1)
     else:
          # inference-time mini-optimization: only forward the lm_head on the very last position
          logits = self.lm_head(x[:, [-1], :]) # note: using list [-1] to preserve the time dim
          loss = None
```

Data preprocessing

Tang poem data preprocessing 唐诗数据处理


```
[{
"id": "e91e6239-120e-4368-8ff1-d6aa69f0b2fe",
"title": "送通禅师还南陵隐静寺",
"author": "李白",
"content": "我闻隐静寺，山水多奇踪。\n岩种朗公橘，门深杯渡松。\n道人制猛虎，振锡还孤峰。
\n他日南陵下，相期谷口逢。"
},
{
"id": "1f83fccf-7476-464b-8cd4-8a2b4569c7a0",
"title": "送友人",
"author": "李白",
"content": "青山横北郭，白水遶东城。\n此地一为别，孤蓬万里征。\n浮云游子意，落日故人情。
\n挥手自兹去，萧萧班马鸣。"
},
……
```


```
enc = tiktoken.get_encoding("gpt2")
def process(poem):
    text = poem['title'] + '\n' + poem['author'] + '\n' + poem['content']
    ids = enc.encode(text)
    ids.append(enc.eot_token)
    return ids
```

Data preprocessing

Llama 2 (convert_dialog_to_input_prompt)

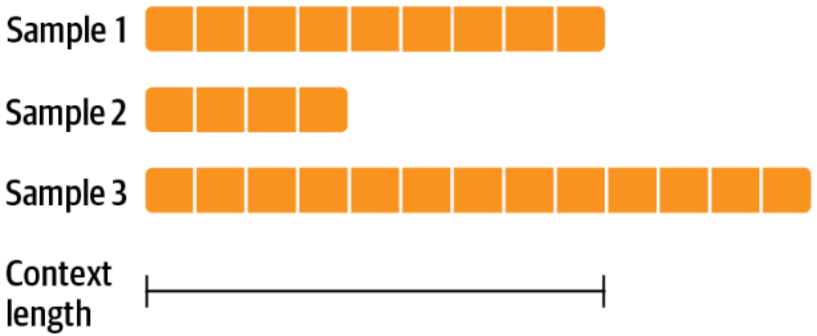
```
B_INST, E_INST = "[INST] ", " [/INST] "
B_SYS, E_SYS = "<<SYS>>\n", "\n<</SYS>>\n\n"
```

ChatML

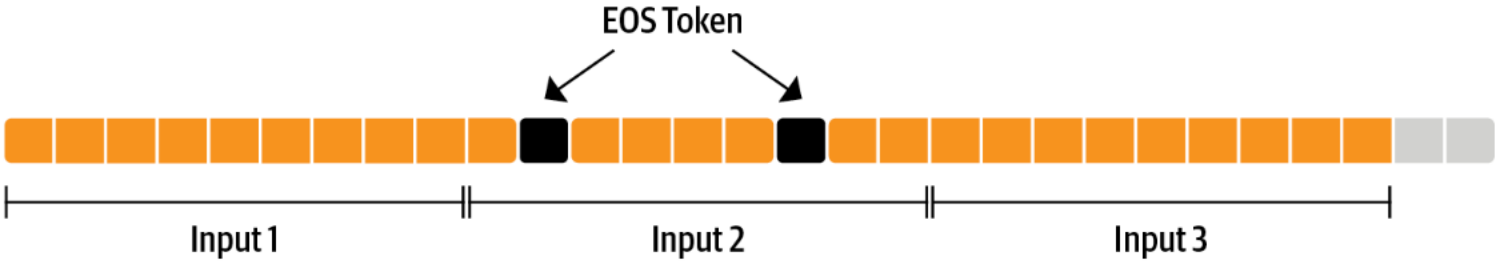
```
msg1 = {"role": "system", "content": "system1"}
msg2 = {"role": "user", "content": "user1"}
msg3 = {"role": "assistant", "content": "assistant1"}
```

['bos[INST] <<SYS>>\nsystem1\n<</SYS>>\n\nuser1 [/INST] assistant1eosbos[INST] user1 [/INST] ']

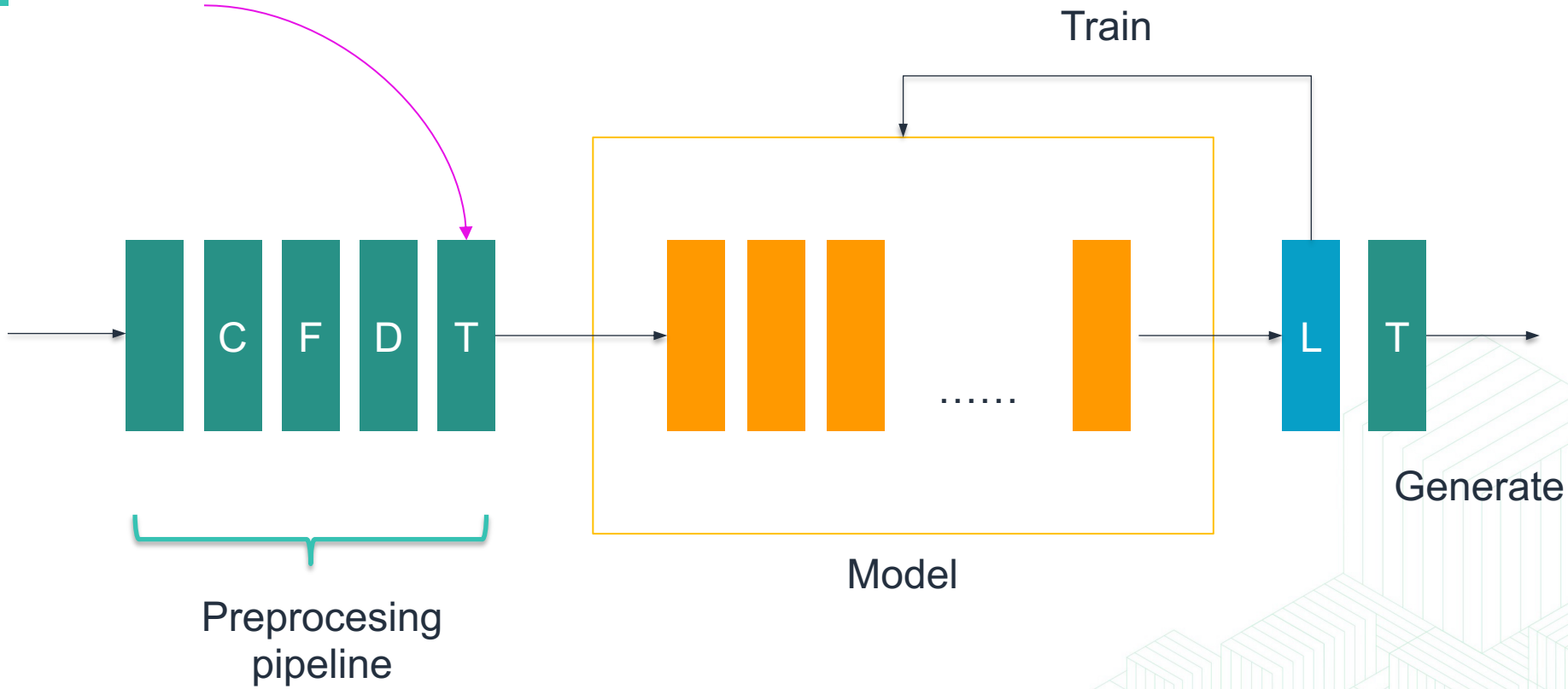
Sequence packing




```
from trl import SFTTrainer

trainer = SFTTrainer(
        model=model,
        args=training_arguments,
        train_dataset=train_dataset,
        dataset_text_field="text",
        max_seq_length=1024,
        packing=True,
)
```



Tokenize



Tokenizer

常见LLM使用的Tokenizer

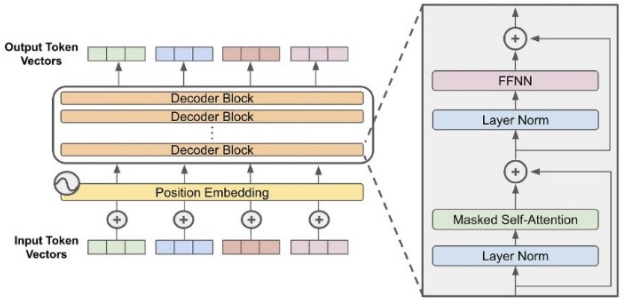
Tokenizer	Name	Vocab size	中文:token	编码密度	Implementation
Claude 3		65000	1:~1.41		BPE?
GPT-4	cl100k_base	100277	1:~1.48		tiktoken
GPT-3.5	cl100k_base	100277	1:~1.48		tiktoken
GPT-3	p50k_base	50281	1:~2.16		tiktoken
GPT-2	gpt2	50257	1:~2.16		tiktoken
Llama 2		32000	1:~1.47		BPE (SentencePiece)
Mistral-7B-v0.1		32000	1:~1.47		LlamaTokenizer
Mixtral-8x7B-Instruct-v0.1		32000	1:~1.47		LlamaTokenizer
ChatGLM2		64794	1:~0.997		BPE (SentencePiece)
ChatGLM3		64798	1:~0.997		BPE (SentencePiece)
Baichun		64000	1:~0.982		BPE (SentencePiece)
Baichun2		125696	1:~0.911		BPE (SentencePiece)
Qwen1		151851	1:~0.918		BPE
Qwen1.5		151643	1:~0.918		BPE

说明：中文：token比例是在56315首唐诗中用代码运行测试，包括换行和标点符号。

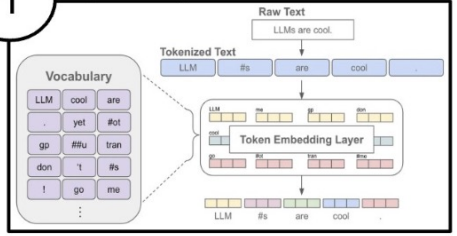
Model architecture

LLM – Decoder-only (casual lm) components

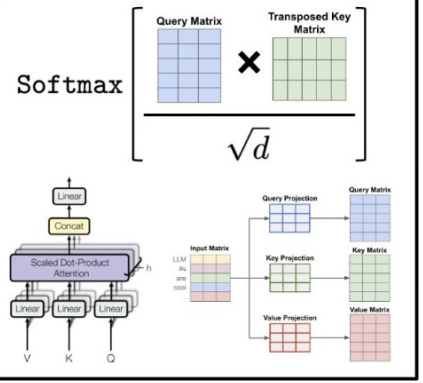
Components of the Decoder-only Transformer



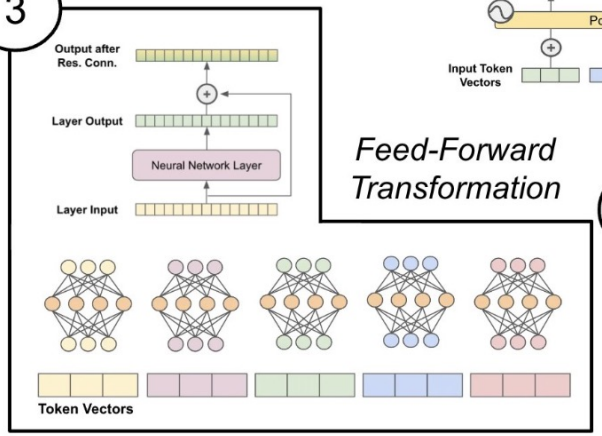
1 Input Layer



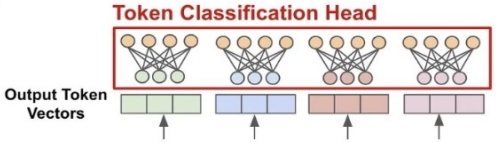
2 Causal Self-Attention



3 Feed-Forward Transformation



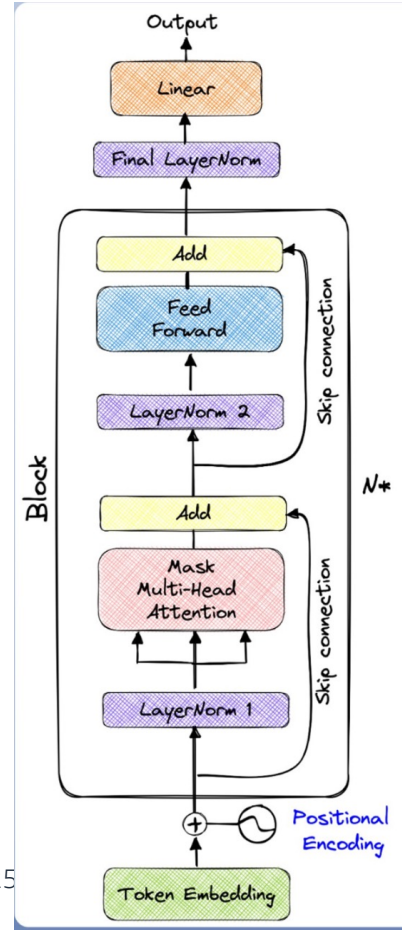
4 Classification Head



5 Transformer Block

```python
from torch import nn

class Block(nn.Module):
    def __init__(
        self,
        d,
        H,
        T,
        bias=False,
        dropout=0.2,
    ):
        """
        d: size of embedding dimension
        H: number of attention heads
        T: maximum length of input sequences (in tokens)
        bias: whether or not to use bias in linear layers
        dropout: probability of dropout
        """
        super().__init__()
        self.ln_1 = nn.LayerNorm(d)
        self.attn = CausalSelfAttention(d, H, T, bias, dropout)
        self.ln_2 = nn.LayerNorm(d)
        self.ffnn = FFNN(d, bias, dropout)

    def forward(self, x):
        x = x + self.attn(self.ln_1(x))
        x = x + self.ffnn(self.ln_2(x))
        return x
```

Model 1: nanoGPT



gpt_language_model.py

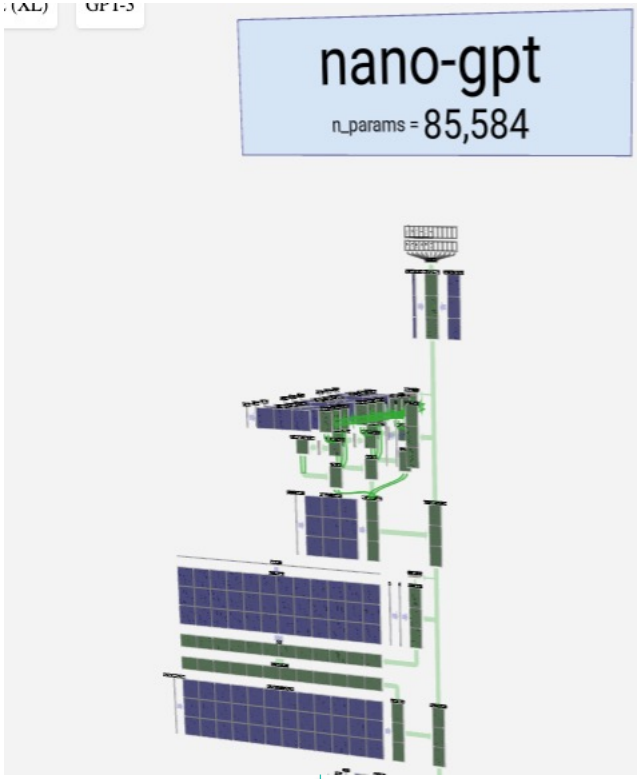
```python
class GPTLanguageModel(nn.Module):

    def __init__(self):
        super().__init__()
        # each token directly reads off the logits for the next token from a lookup table
        self.token_embedding_table = nn.Embedding(vocab_size, n_embd)
        self.position_embedding_table = nn.Embedding(block_size, n_embd)
        self.blocks = nn.Sequential(*[Block(n_embd, n_head=n_head) for _ in range(n_layer)])
        self.ln_f = nn.LayerNorm(n_embd) # final layer norm
        self.lm_head = nn.Linear(n_embd, vocab_size)

        # better init, not covered in the original GPT video, but important, will cover in followup video
        self.apply(self._init_weights)

    def _init_weights(self, module):
        if isinstance(module, nn.Linear):
            torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)
            if module.bias is not None:
                torch.nn.init.zeros_(module.bias)
        elif isinstance(module, nn.Embedding):
            torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)

    def forward(self, idx, targets=None):
        B, T = idx.shape

        # idx and targets are both (B,T) tensor of integers
        tok_emb = self.token_embedding_table(idx) # (B,T,C)
        pos_emb = self.position_embedding_table(torch.arange(T, device=device)) # (T,C)
        x = tok_emb + pos_emb # (B,T,C)
        x = self.blocks(x) # (B,T,C)
        x = self.ln_f(x) # (B,T,C)
        logits = self.lm_head(x) # (B,T,vocab_size)

        if targets is None:
            loss = None
        else:
            B, T, C = logits.shape
            logits = logits.view(B*T, C)
            targets = targets.view(B*T)
            loss = F.cross_entropy(logits, targets)

        return logits, loss
```

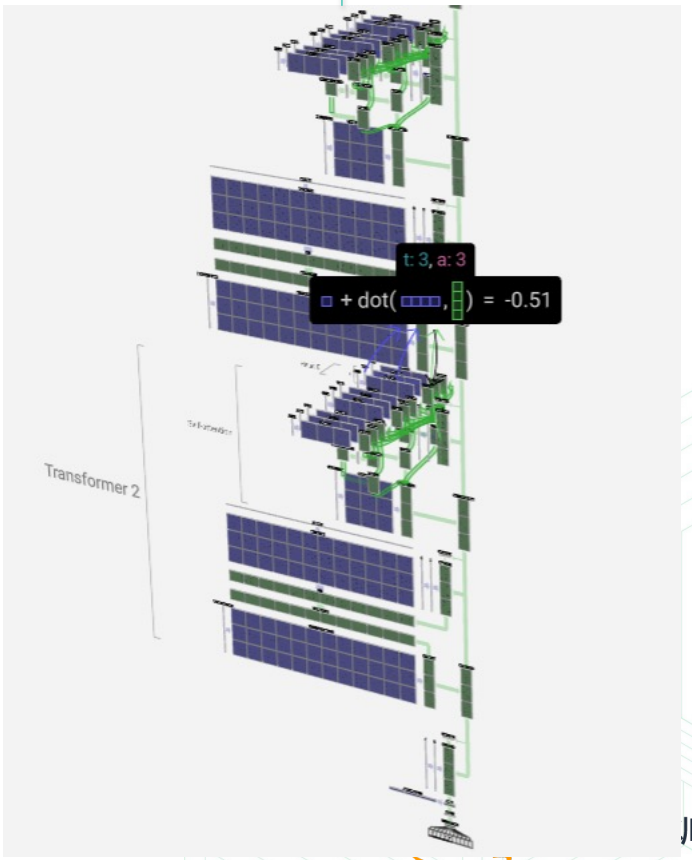
parameter	value
dim	1536
n_layer	24
n_head	24
dropout	0.2
context_len	256
optimzer	AdamW
beta1	0.9
beta2	0.99
learning rate	1e-3
lr rate decay	cosine
vocab_size	50304
FlashAttention	2.0
precision	BF16

Model 2: MyMistral-1B (904M)

```
MistralForCausalLM(
  (model): MistralModel(
    (embed_tokens): Embedding(32000, 2048)
    (layers): ModuleList(
      (0-7): 8 x MistralDecoderLayer(
        (self_attn): MistralSdpaAttention(
          (q_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (k_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (v_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (o_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (rotary_emb): MistralRotaryEmbedding()
        )
        (mlp): MistralMLP(
          (gate_proj): Linear(in_features=2048, out_features=14336, bias=False)
          (up_proj): Linear(in_features=2048, out_features=14336, bias=False)
          (down_proj): Linear(in_features=14336, out_features=2048, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): MistralRMSNorm()
        (post_attention_layernorm): MistralRMSNorm()
      )
    )
    (norm): MistralRMSNorm()
  )
  (lm_head): Linear(in_features=2048, out_features=32000, bias=False)
)
```

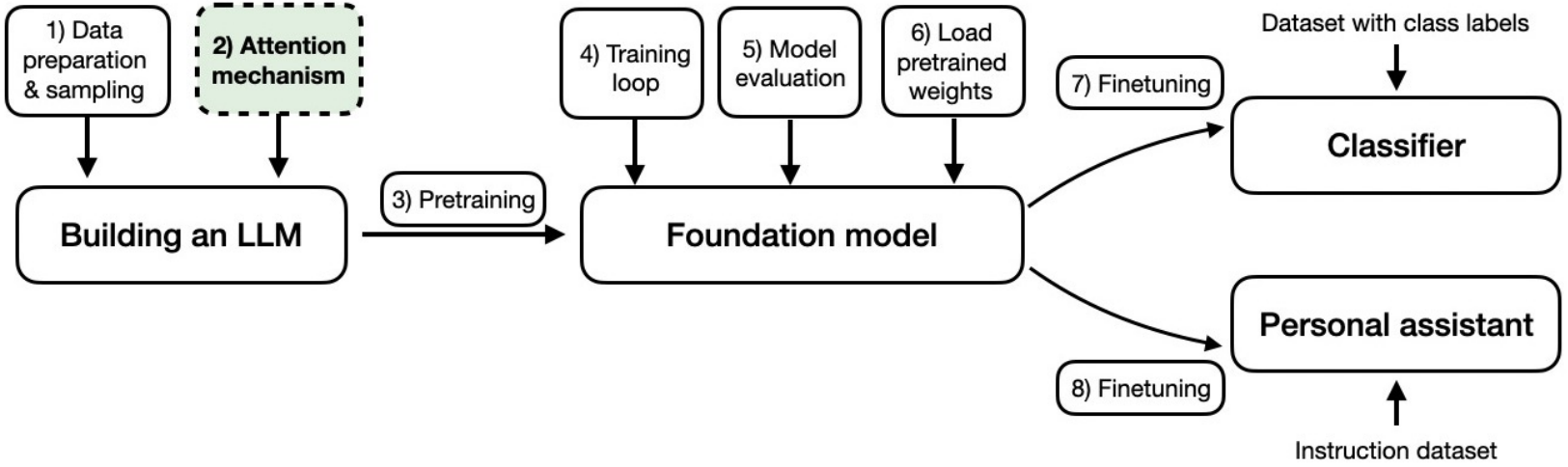
LLM Visualization



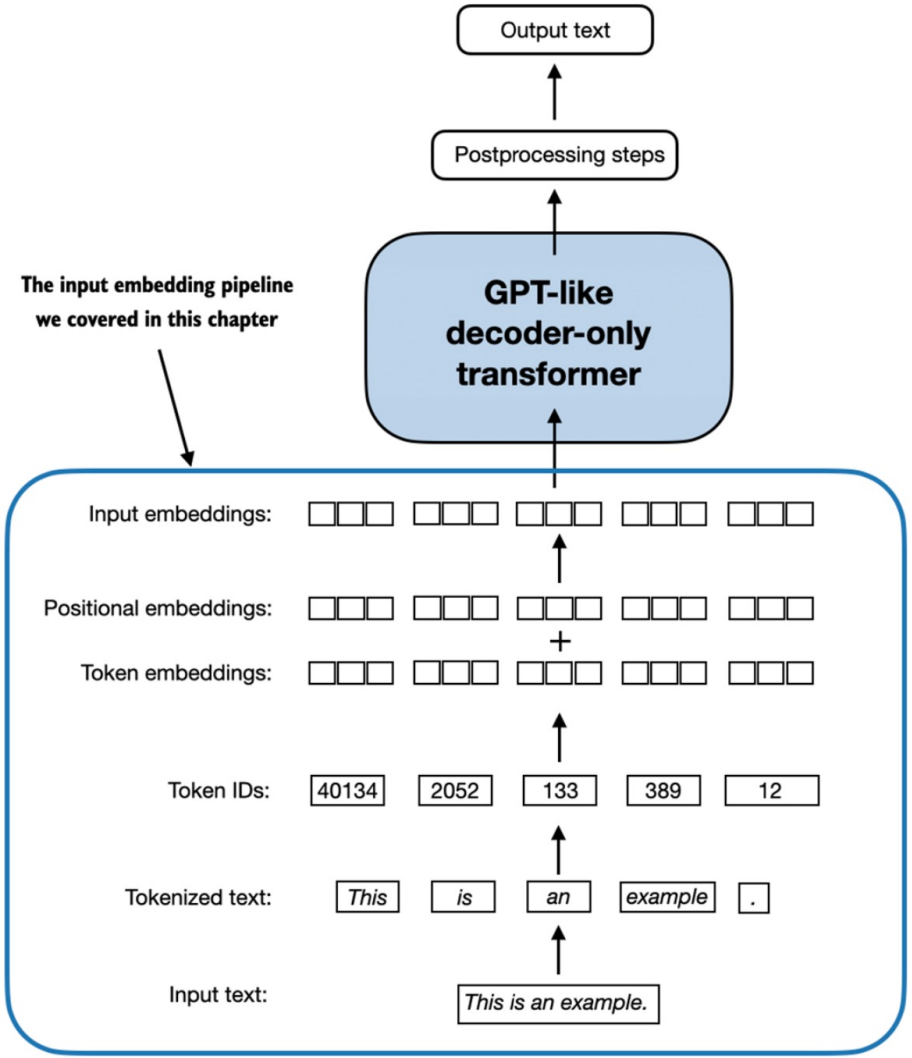


Pretrain & fine-tune

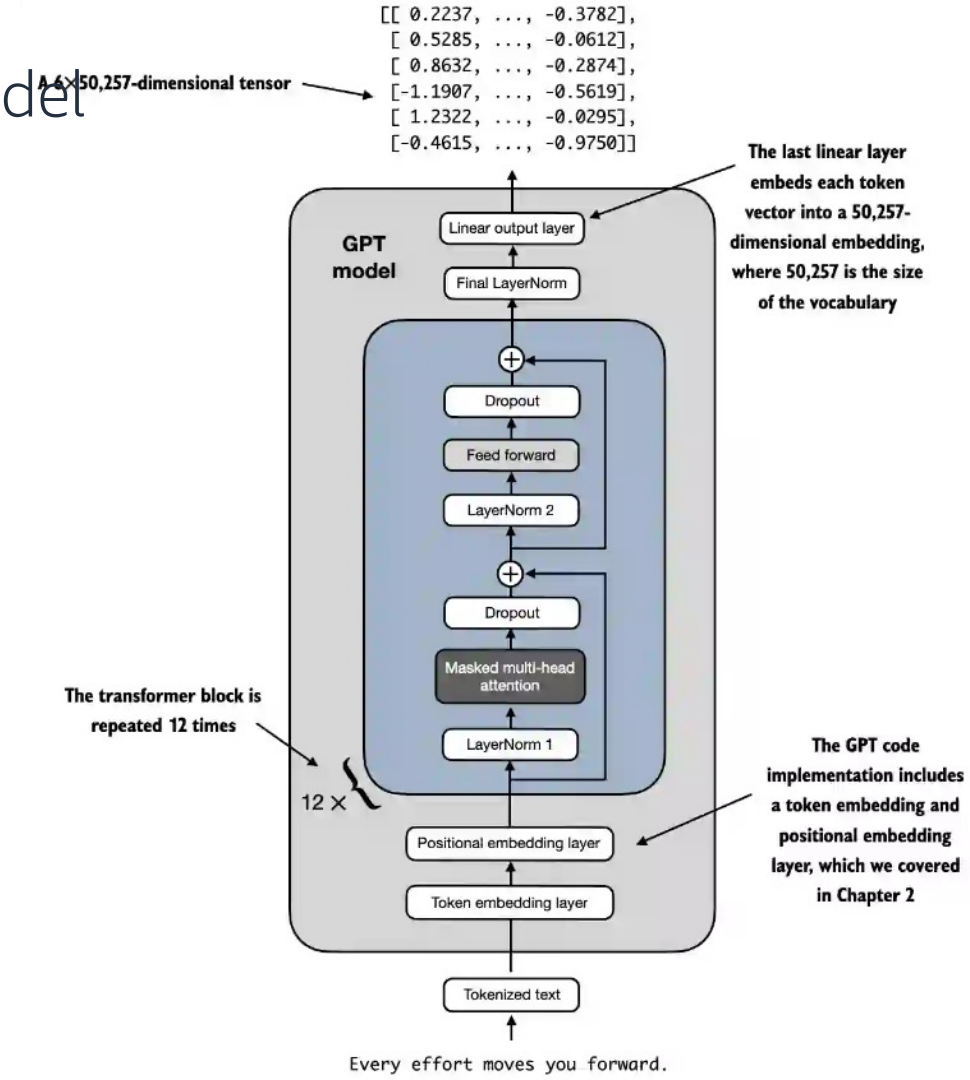
Pretrain & fine-tune



Training input



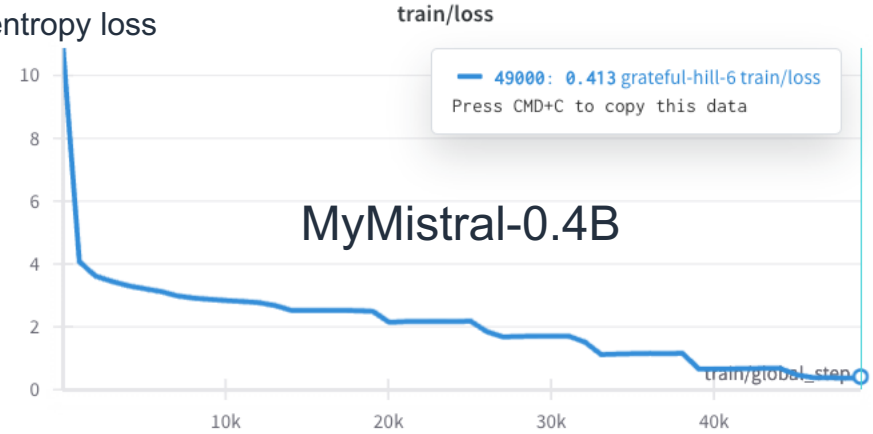
GPT-like model



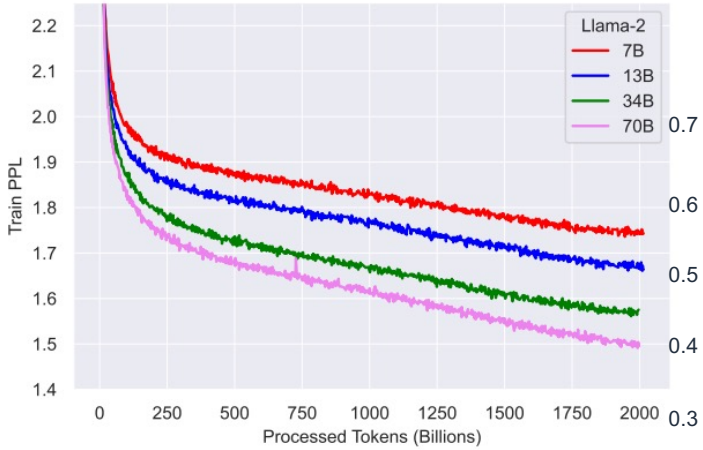
My training loss

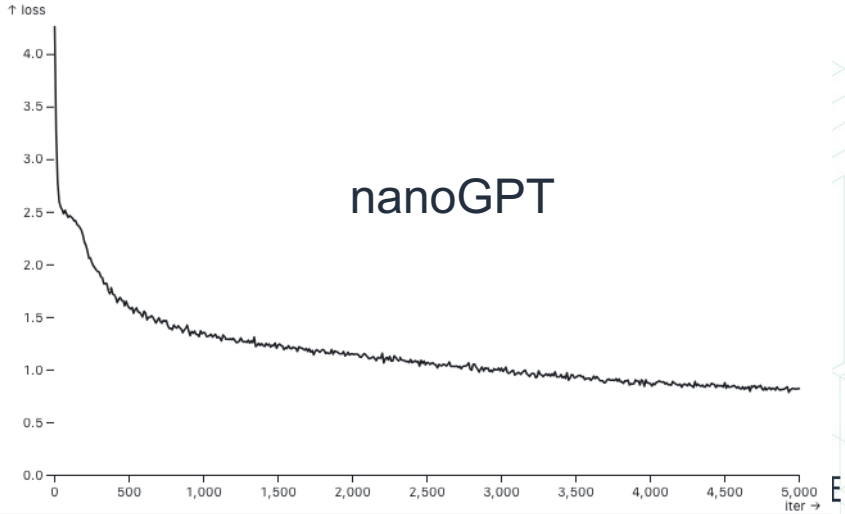
Figure 5: Training Loss for Llama 2 models. We compare the training loss of the Llama 2 family of models. We observe that after pretraining on 2T Tokens, the models still did not show any sign of saturation.

Cross entropy loss



Perplexity

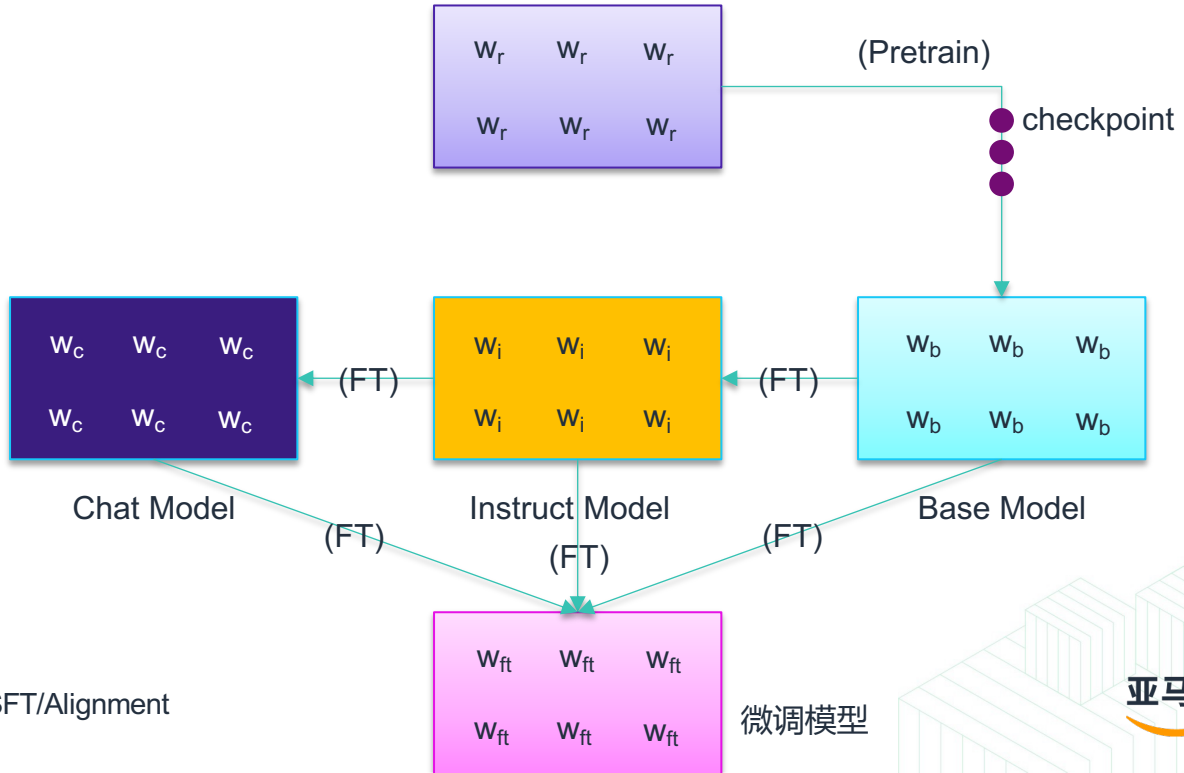




训练到什么时候停止？ Scaling Law

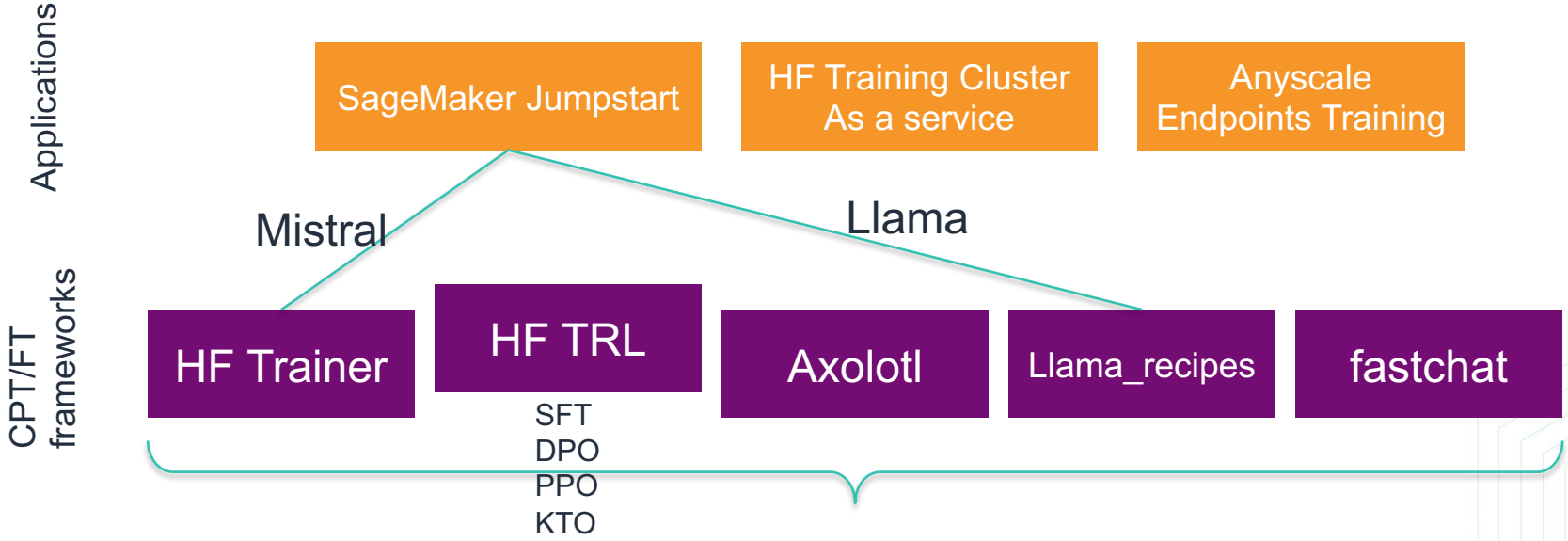
Pretrain和Fine-tune区别

PT阶段和SFT阶段，两者的数据流动、loss函数、都是一样的，程序都可以通用，只是计算loss的方式存在差异，PT阶段计算的是整段输入文本的loss，而SFT阶段计算的是Answer部分的loss。



*FT = CPT/SFT/Alignment

Fine-tune frameworks



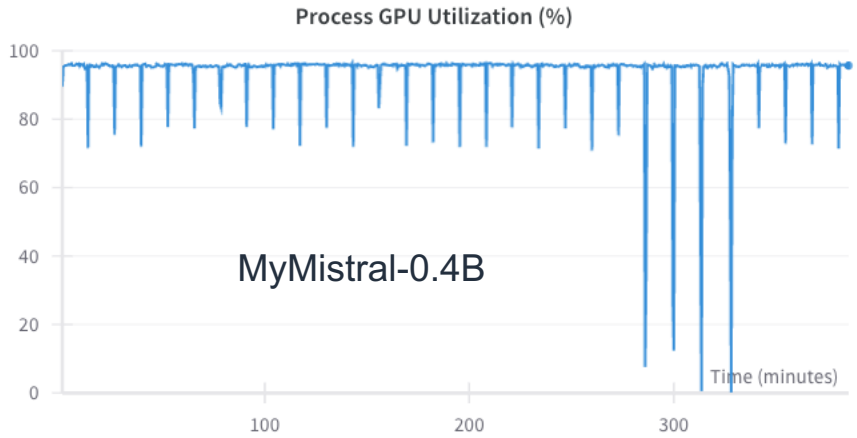

```
# forward the model
logits, loss = model(x, y)

# backprop and update the parameters
model.zero_grad(set_to_none=True)
loss.backward()
optimizer.step()
```

Naïve
Train Loop

Training efficiency

MegaScale 万卡 55.2%

Model	Seq. Len.	GPUs	TP sizes	PP sizes	MB Sizes	Act. Checkpointing	RMSNorm Kernel
13B	2k	64	{1, 2}	{1, 2}	{1, 2, 4, 8}	{yes, no}	{yes, no}
13B	8k	128	{1, 2, 4}	{1, 2, 4}	{1, 2, 4}	{yes, no}	{yes, no}
30B	2k	256	{1, 2, 4}	{1, 2, 4}	{1, 2, 4}	{yes, no}	{yes, no}
30B	8k	128	{2, 4}	{2, 4, 8, 16}	{1, 2, 4}	{yes, no}	{yes, no}
65B	2k	128	{2, 4, 8}	{2, 4, 8}	{1, 2, 4}	{yes, no}	{yes, no}



MFU=
model FLOPs utilization

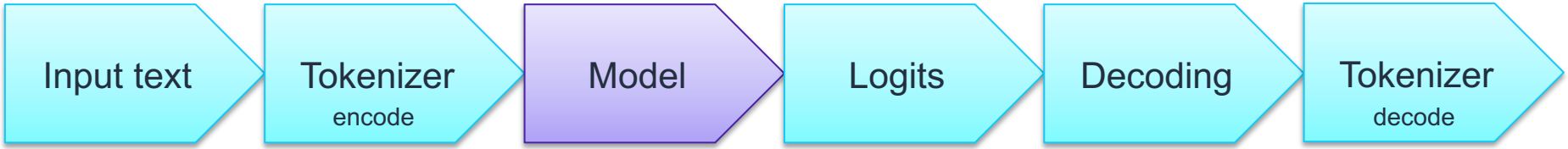
Model	GPUs	Seq. Len.	Batch Size	MFU (↑)
AA-Scaling LLAMA 13B (ours)	64	2048	2048	**70.5%**
MPT 13B	64	2048	2048	52.5%
Megatron-LM 18B†	256	2048	1024	34.2%
AA-Scaling LLAMA 13B (ours)	64	8192	512	**62.7%**
MPT 13B	8	8192	120	52.8%
AA-Scaling LLAMA 30B (ours)	64	2048	2048	**61.9%**
MPT 30B	64	2048	3072	52.9%
Megatron-DeepSpeed 22B	8	2048	4	41.5%
Megatron-LM 39B†	512	2048	1536	34.5%
AA-Scaling LLAMA 30B (ours)	64	8192	512	**60.2%**
MPT 30B	8	8192	168	42.6%
AA-Scaling LLAMA 65B (ours)	64	2048	2048	**59.6%**
MPT 70B	64	2048	2048	53.3%
LLAMA 65B by Meta†	2048	2048	2048	49.4%
Megatron-LM 76B†	1024	2048	1792	34.7%

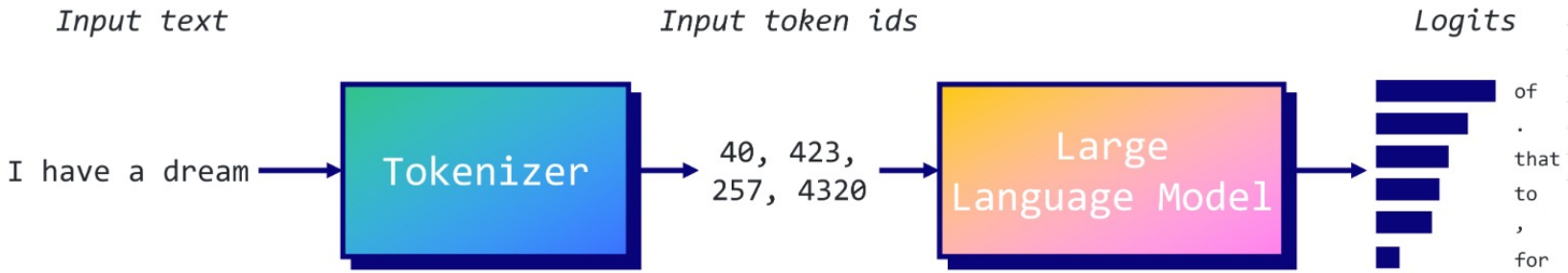
Inference & generate

Inference and Generate

`do_sample=False`

- Greedy Search
- Beam Search
- Top-k Sampling
- Nucleus sampling





Decoding ~= Sampler (penalty, temperature, top-p, top-k)

Transformers / generation, vllm / model_executor / layers / sampler.py

Temperature

GPT-4

What sampling temperature to use, between **0 and 2**. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic.

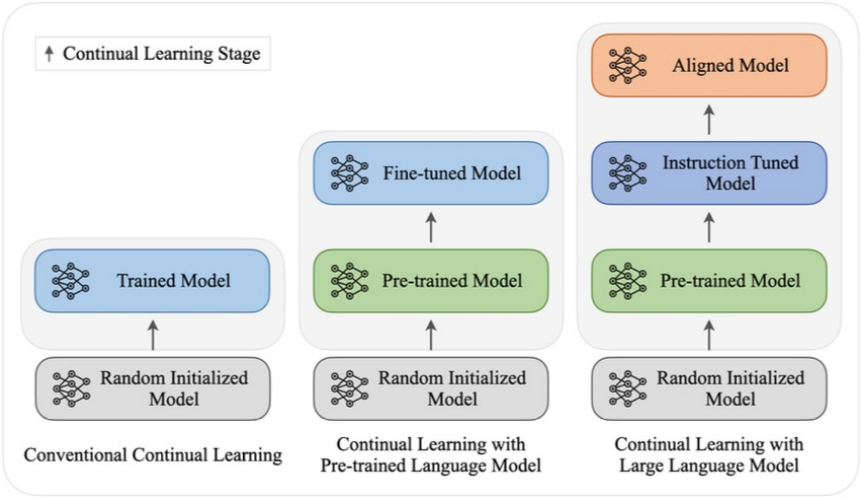
Framework	Type	Default	Min	Max	Logical	when zero
transformers	float	1	>0	inf	logits/t	error, use do_sample=False
vllm	float	1	0	inf	logits/t	<1e-5, set t=1, use greedy
TGI	float	1	>0	inf	logits/t	error
bedrock	float	0.5	0	1	?	ok
GPT-4	float	1?	0	2	?	ok

说明：Temperature是模型输出logits后进行decoding时用到的参数，与LLM模型无关（vLLM把temperature scaling也当成模型的一个层），与serving框架有关，相同的模型在不同的serving框架中temperature设置可能不同，如Llama在vLLM可以设置temperature为 [0, positive float)，在transformers中是 (0, positive float)，在Bedrock中是 [0, 1]。

Generate - code can tell you the truth, but not the whole truth

```
@torch.no_grad()
def generate(self, idx, max_new_tokens, temperature=1.0, top_k=None):
    """
    Take a conditioning sequence of indices idx (LongTensor of shape (b,t)) and complete
    the sequence max_new_tokens times, feeding the predictions back into the model each time.
    Most likely you'll want to make sure to be in model.eval() mode of operation for this.
    """
    for _ in range(max_new_tokens):
        # if the sequence context is growing too long we must crop it at block_size
        idx_cond = idx if idx.size(1) <= self.config.block_size else idx[:, -
        self.config.block_size:]
        # forward the model to get the logits for the index in the sequence
        logits, _ = self(idx_cond)
        # pluck the logits at the final step and scale by desired temperature
        logits = logits[:, -1, :] / temperature
        # optionally crop the logits to only the top k options
        if top_k is not None:
            v, _ = torch.topk(logits, min(top_k, logits.size(-1)))
            logits[logits < v[:, [-1]]] = -float('Inf')
        # apply softmax to convert logits to (normalized) probabilities
        probs = F.softmax(logits, dim=-1)
        # sample from the distribution
        idx_next = torch.multinomial(probs, num_samples=1)
        # append sampled index to the running sequence and continue
        idx = torch.cat((idx, idx_next), dim=1)

    return idx
```

Continual learning for LLMs



Information	RAG	Model Editing	Continual Learning
Fact	✓	✓	✓
Domain	✓	×	✓
Language	×	×	✓
Task	×	×	✓
Skills (Tool use)	×	×	✓
Values	×	×	✓
Preference	×	×	✓

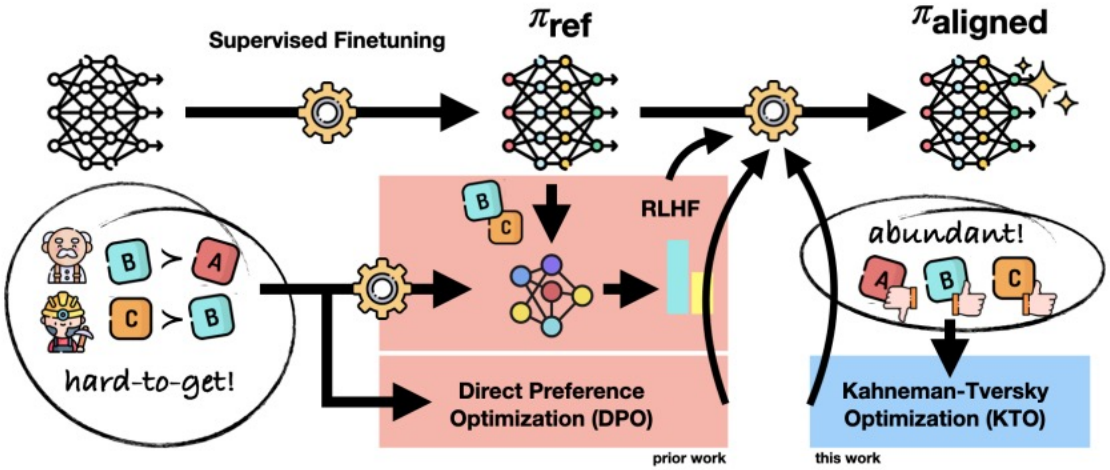
Table 1: Continual Learning v.s. RAG and Model Editing

Information	Pretraining	Instruction-tuning	Alignment
Fact	✓	×	×
Domain	✓	✓	×
Language	✓	×	×
Task	×	✓	×
Skills (Tool use)	×	✓	×
Values	×	×	✓
Preferences	×	×	✓

Table 2: Information updated during different stages of continual learning for LLMs.

Alignment & evaluation

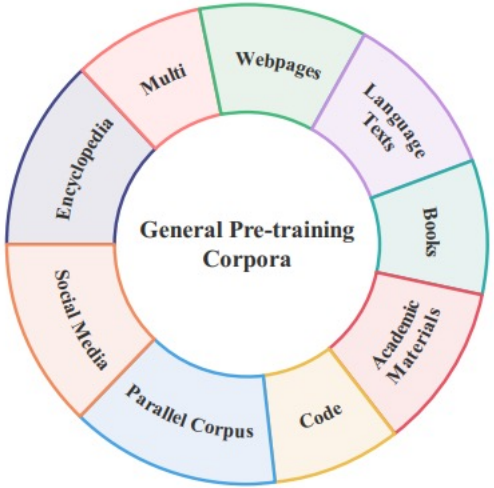
TODO

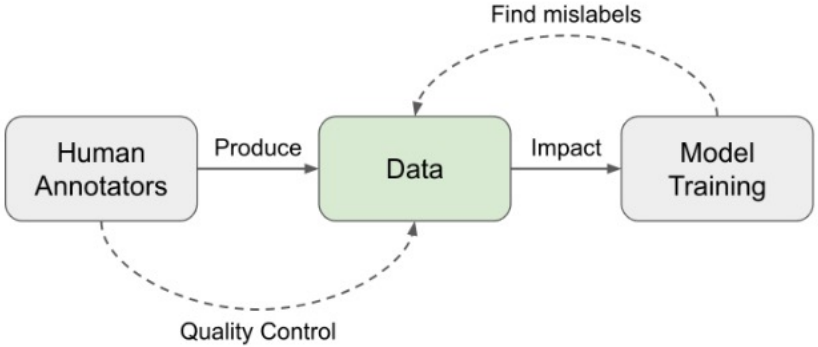


Agenda

1. Why I train a useless (tiny toy) model
2. Training process & result
3. Lessons learned
4. Applications and future possibilities

1. High quality data is the key





实验生成

芳草未知谁与侣，酒酣不辞莫叹息。

愿得徒劳怨不得，恐是颍头盈盈缘。

Datasets for Large Language Models: A Comprehensive Survey
https://arxiv.org/pdf/2402.18041.pdf

2. 模块下沉和坍缩 push down and condense

核心功能模型下沉到框架如Pytorch

```
    def forward(self, x):
        B, T, C = x.size() # batch size, sequence length, embedding dimensionality (n_embd)
@@ -60,11 +59,9 @@ def forward(self, x):
        q = q.view(B, T, self.n_head, C // self.n_head).transpose(1, 2) # (B, nh, T, hs)
        v = v.view(B, T, self.n_head, C // self.n_head).transpose(1, 2) # (B, nh, T, hs)

-       # causal self-attention; Self-attend: (B, nh, T, hs) x (B, nh, hs, T) -> (B, nh, T, T)
-       att = (q @ k.transpose(-2, -1)) * (1.0 / math.sqrt(k.size(-1)))
-       att = att.masked_fill(self.bias[:,:,:T,:T] == 0, float('-inf'))
-       att = F.softmax(att, dim=-1)
-       att = self.attn_dropout(att)
+       # the output of sdp = (batch, num_heads, seq_len, head_dim)
+       y = torch.nn.functional.scaled_dot_product_attention(q, k, v, None, self.dropout, True)
+
        y = att @ v # (B, nh, T, T) x (B, nh, T, hs) -> (B, nh, T, hs)
        y = y.transpose(1, 2).contiguous().view(B, T, C) # re-assemble all head outputs side by side
```

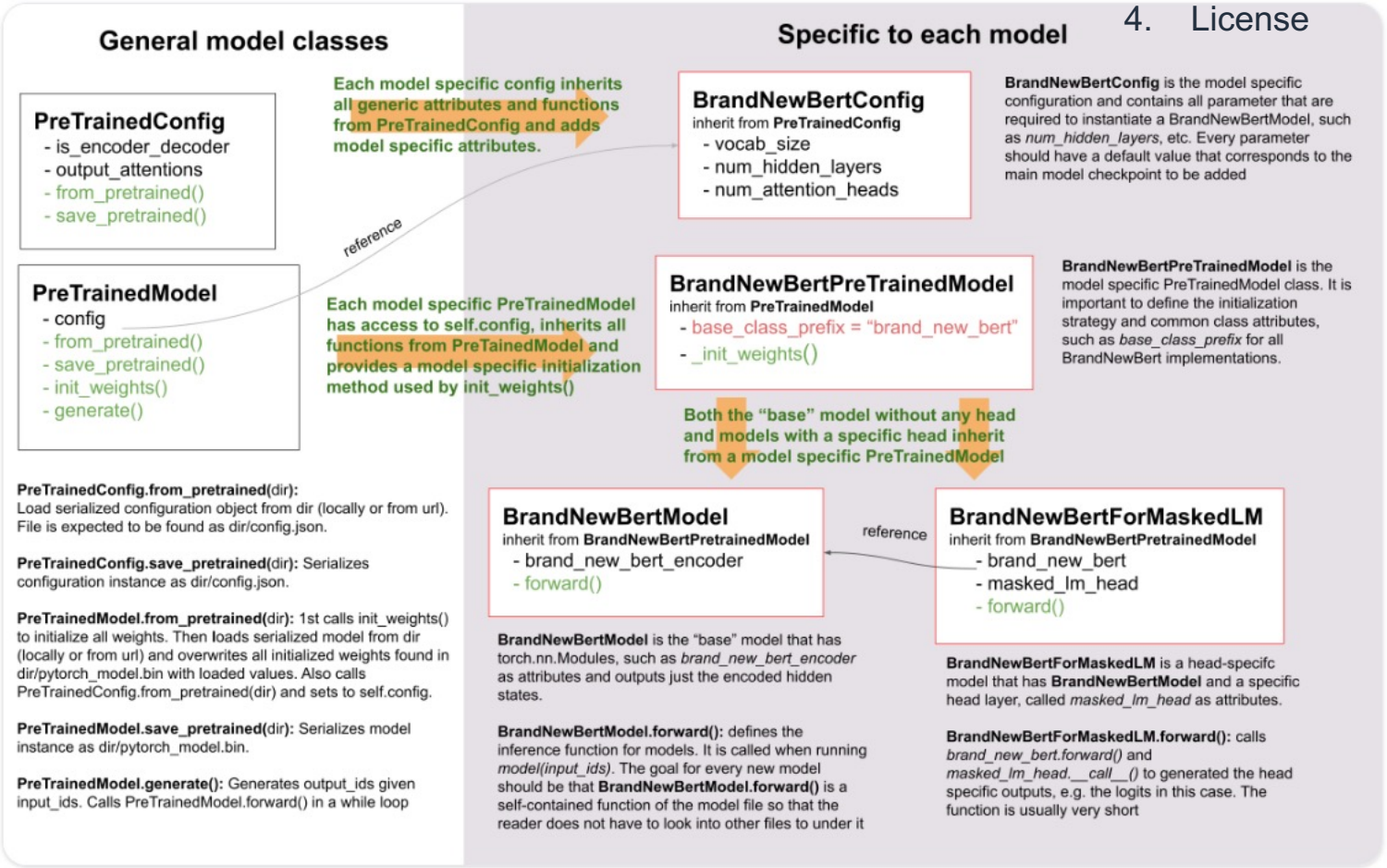
Flash-attention v2

[nn.Transformer]

3. Open or Close AI

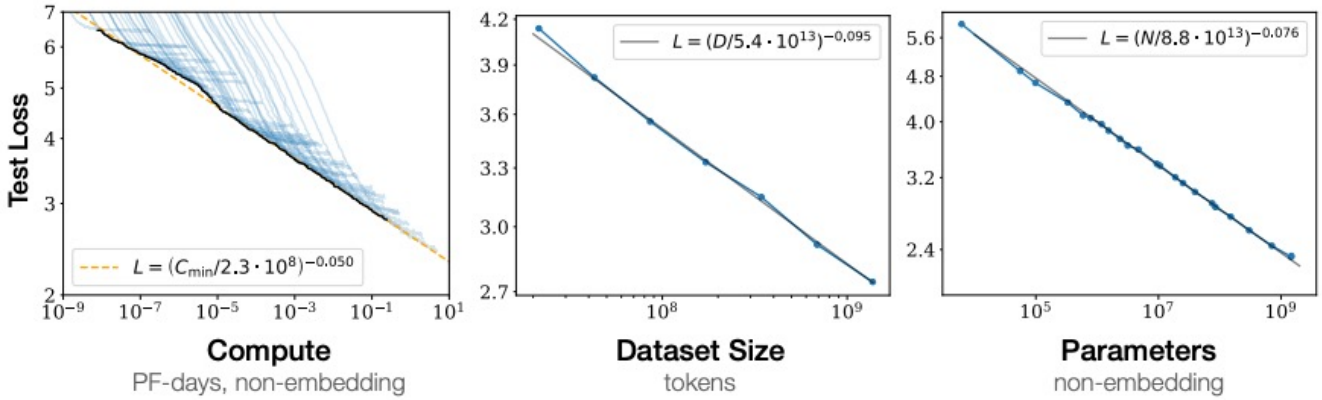
1. Data
2. Model Architecture
3. Recipes
4. License

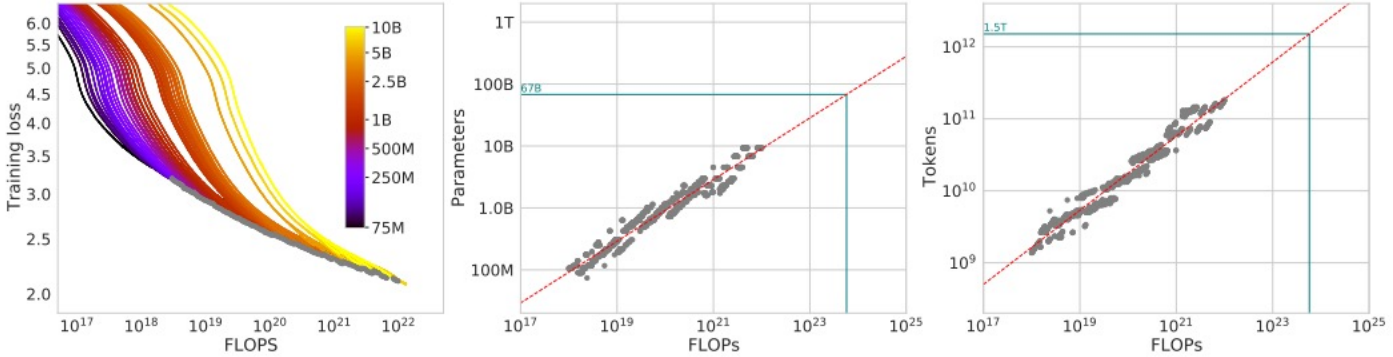
OLMo
Dolma



4. Scaling law is the law?

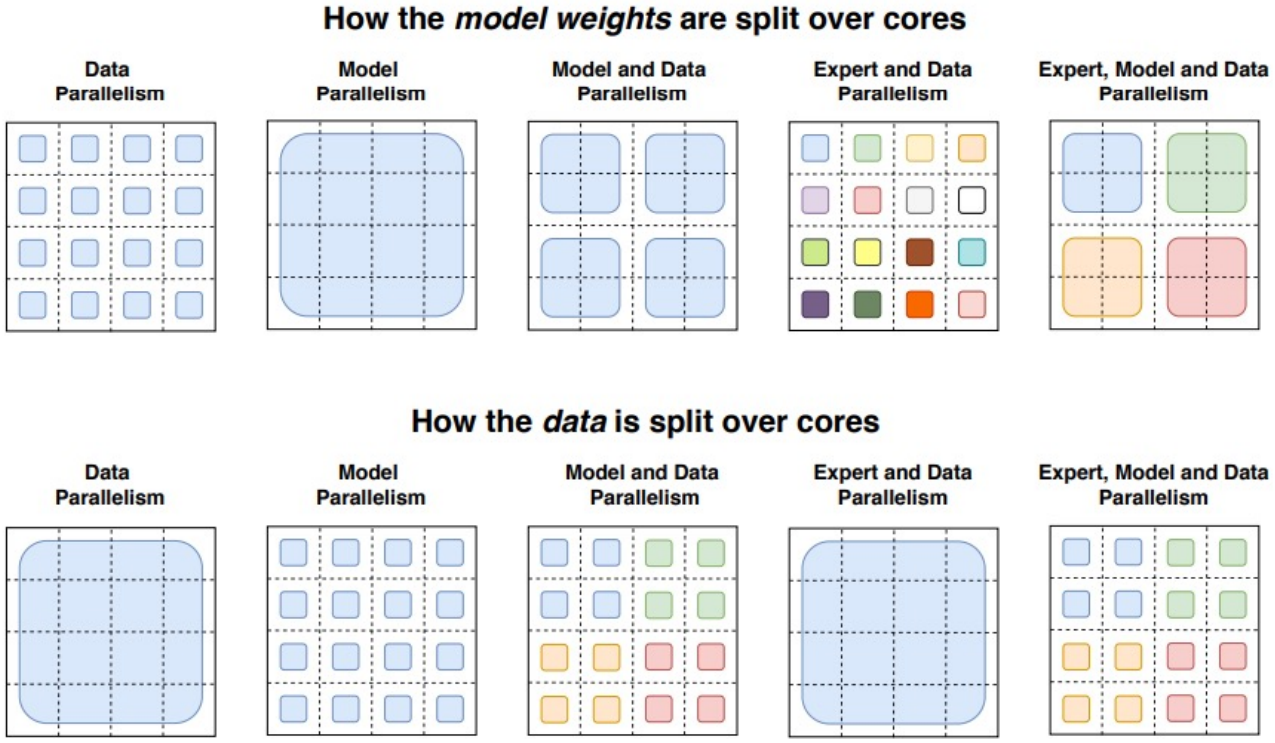
$C = 6ND$





N:D配比
20
|
3000

5. Large scale parallelism is a must



6. Still a loooong way to build high-end LLMs

Data
- Textbook is all your need
- Synthetic data

Model Architecture
- MultiModal
- MoE
- GQA
- Scaling

Training
- DDP/FSDP/TP/PP/3D
- Receipes

Evalutaiton
- loss
- Metrics
- Dataset

Alignment
- PPO/DPO/KTO
- Safety
- Value preferences

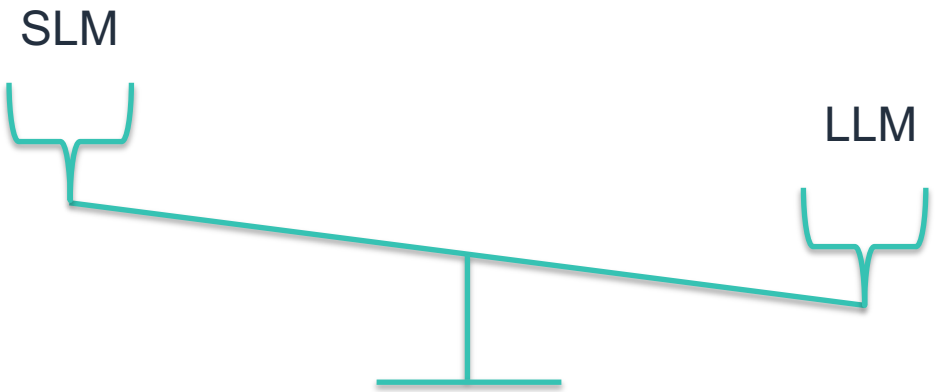
Continual learning
- CPT/SFT/PEFT
- Multi-LoRA

Agenda

1. Why I train a useless (tiny toy) model
2. Training process & result
3. Lessons learned
4. Applications and future possibilities

Discuss

Small language model is good enough?



亚马逊云科技

Thank You!