预训练

7 预训练

7.1 预训练定义

预训练（Pre-training）是指在**大规模无监督数据**上进行初步模型训练，使模型能够学习到通用的语言模式、知识表征和统计特征。这一阶段不依赖于特定任务，帮助模型**建立基础的语言理解能力（通用能力）**。通过预训练得到base模型，该模型可以在后续的特定任务上（如分类、生成、翻译等）以更快的速度和更高的准确性进行微调。

预训练采用自监督的方式训练，训练目标为**预测下一个token的对数似然损失**：

$$L = \sum_{n=1}^{N} logp(x_n|x_1, ..., x_n - 1; \theta)$$

🍞 **在开源大模型数量爆炸的时代，预训练模型的意义在何处？**

- 许多大模型没有开源**训练框架、训练数据**等核心要素，仅通过评估无法确定大模型对知识的掌握程度，导致在模型对齐阶段无法对症下药。例如我要微调一个写诗的大模型，采用的大模型在预训练阶段可能根本没在诗歌数据集上训练过，仅通过微调训练出的大模型写出来诗肯定难以直视。
- 通用模型在特定领域的表现远不如domain模型，而要训练出好的domain模型，仅靠微调是不够的，需要做继续预训练，而继续预训练与预训练的技术栈基本一致。
- 开源模型的tokenizer不可控，导致解码速度不可控。比如llama模型做意图识别任务，有个意图叫 ai.listen.music，会被映射成 5 个 token，但如果使用自己训练的大模型，便会在一开始就设置成 1 个 token，极大节省了生成速度。

7.2 预训练数据集

预训练数据集是决定大模型性能的关键因素，大量的高质量数据能提高模型的泛化能力和应用效果。

🎉 **构建预训练数据集需要考虑的因素**

- **数据规模**：预训练数据集的规模对模型的性能具有直接影响，一般总体数据量需要达到10Ttoken的级别。
- **数据多样性**：预训练阶段需要接触到多种文本内容，包括文档、网页、代码、数学、多语言等。

- **数据质量**：数据质量直接影响模型预训练的有效性和稳定性，需要对文本进行清洗，对于图片、表格、公式等元素需要做额外的处理。

7.2.1 数据来源

数据集的质量和多样性主要取决于数据来源的选择。常见的数据来源包括：

- **公开数据集**：如维基百科、Common Crawl等大型通用文本数据集。这类数据资源庞大且容易获取。
- **领域特定数据**：从技术文献库、领域文献、研究论文或行业报告中收集数据，适用于特定领域（如医学、法律、金融等）模型的预训练。
- **网络抓取**：通过网络爬虫工具从特定网站抓取数据，尤其适合于需要最新或领域特定信息的场景。参考：https://github.com/NanmiCoder/MediaCrawler
- **自有数据**：如企业内部的技术文档、客户服务记录等，这些数据能够使模型专门针对企业应用场景进行优化。

7.2.2 数据清洗

收集到的原始数据往往包含很多噪声和冗余信息，需要通过数据清洗保障数据质量，主要包括：

- **模型打分**：llama3和qwen2的技术报告都提到，可以用打分模型对预训练质量打分。业界的普遍认知是，同等 size 下，BERT 结构的模型的表征能力是强于 transformer-decoder 模型的。训练打分模型的数据可以由GPT标注得到，需要注意的是，**code、markdown、latex**等格式的数据会被打低分，应该提前摘出来。

> 🦄 打分模型训练不需要特别多的数据和很高的准确性，可以结合规则进行判断（例如**数据长度，某个 token 的比例是否超过阈值，是否包含敏感词**等）。

- **去重处理**：消除重复的文本片段，以避免模型过度学习某些特定模式。需要去除训练集内部重复、训练集与测试集之间重复的n-grams、句子和段落。常见算法：文本去重**MinHash+LSH**，语义去重**BGE+分片聚类**。

> 数据清洗要先确定需要多少训练数据，再确定去重的粒度。例如需要10T的训练数据，就卡80%的相似度阈值。需要5T的训练数据，就卡90%的相似度阈值。

- **去除噪声**：如HTML标签、广告、目录、标点符号过度堆积、低质量url等。这些信息会干扰模型的训练，降低训练效果。常用工具：**DataTrove、Trafilatura**
- **文本规范化**：标准化文本格式，如统一的编码格式、消除特殊符号、处理数字或单位等。常用工具：**Normalise、CN-Text-Normalizer**

7.2.3 数据多样性

数据多样性体现在多个方面：**任务多样性，语义多样性，语种多样性，数据来源多样性**等。下图参考《A Survey on Data Selection for Language Models》，主要包含**语言过滤、启发式过滤**（类似粗筛，基于stopwords占比、字符重复率等规则过滤）、**数据质量检测**（类似精筛，可以通过大模型过滤）、**领域知识注入、去重、有害内容去除、数据混合**。

7.2.4 数据增强

在某些情况下，数据增强技术可以进一步提高数据集的多样性和丰富性，尤其是在数据规模不足的情况下。常见的数据增强技术包括：

- **同义词替换**：在不改变语义的前提下，用同义词替换句子中的部分词汇。
- **句子顺序打乱**：对文本中的句子顺序进行随机调整，增加训练数据的复杂性。
- **生成式增强**：利用已有模型生成新的语料，通过多种生成方式扩展训练数据的规模。

7.2.5 数据配比

为了区分不同类型的数据，可以训练一个数据分类器。对每一个 document 进行类别判断，不用特别精准，把数据划分成**新闻、百科、代码、markdown**等类目即可，分类器模型依然可以选择使用 **BERT** 家族。针对不同类别的数据，做数据清洗时的阈值也不同。

预训练数据主要包含几部分：

- **通用知识**：包含中文和英文知识。
- **逻辑数据**：一般是数学数据和CoT数据。
- **代码数据**：包含多种编程语言。

具体配比跟模型所需能力有关。

7.2.6 数据顺序

预训练的本质是教授模型知识，知识学习的顺序也很重要（先学高数再学泛函），相同数据采用不同顺序训练得到的模型能力是不同的。

- **数据顺序怎么确定?**

参考《**IN-CONTEXT PRETRAINING: LANGUAGE MODELING**

BEYOND DOCUMENT BOUNDARIES》中的观点，利用语义相似度，优先将最相似的 document 进行拼接，从而构成语义更加连贯流畅的上下文。

🏆 **预训练数据的一些其他经验总结：**

- 预训练需要大量的数据，如果有企业内自有数据，可以与大量的公开数据集混合进行训练。为了避免大模型遗忘自有数据，可以通过后续在自有数据上微调和对比学习的方式增强记忆。
- 数据配比和数据顺序一般先在小模型上实验，然后利用**scaling law**估计在大模型上的效果。
- 训练数据要兼顾质量和多样性，低质量数据不可能完全清洗干净，只能在选择阈值时尽可能提高信噪比。
- 预训练的数据是分块加载到内存的，一般块以B为单位，避免内存爆炸、损失爆炸等问题。
- 预训练阶段数据可以复用，数据处理时需要标注出每个数据被使用了多少次，使用过多的数据应该降低再被选中参与训练的概率。

7.3 预训练流程

7.3.1 分词器训练

分词器的作用是将连续的文本字符串分解成模型能够处理的基本单位。这些基本单位可以是单词、子词（subword），甚至是单个字符，每个基本单位对应词表中的一个数值，然后输入到模型进行处理。

分词器常用算法：**BPE、BBPE、WordPiece**。

🎉 **好的分词器的重要性：**

1. 减少词表大小，降低模型复杂性，提高模型计算效率。
2. 提升模型泛化能力，遇到未见过的词时，能通过学习到的子词来理解其含义。（**OOV问题**）
3. 保持语义一致性，尽量避免将具有特定语义的词错误地分割，以免影响模型的语义理解能力。

- **什么时候需要训练分词器?**

一般是外国模型可能没有在足够的中文数据上进行训练，需要在中文语料上进行二次预训练。

- **如何训练tokenizer?**

收集一个很大的训练集，直接利用**BPE、BBPE、WordPiece**等算法训练。

代码块

```
1  from tokenizers import Tokenizer
```

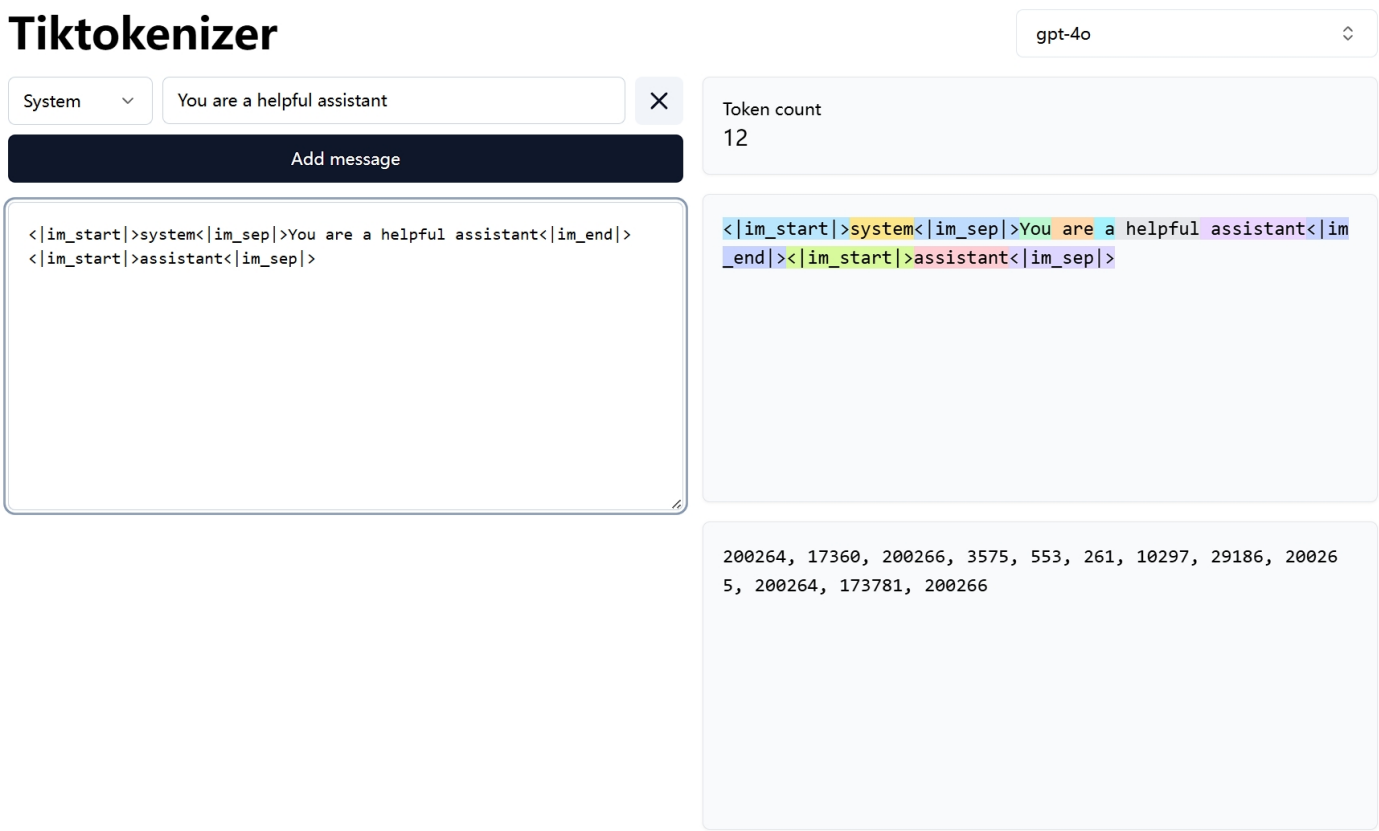
```python
from tokenizers.models import BPE
from tokenizers.trainers import BpeTrainer
from tokenizers.pre_tokenizers import Whitespace

# Step 1: 实例化一个空白的BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
# Step 2: 实例化一个BPE tokenizer 的训练器 trainer 这里 special_tokens 的顺序决定了其id排序
trainer = BpeTrainer(
    special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"],
    min_frequency=1,
    show_progress=True,
    vocab_size=40000
)
# Step 3: 定义预分词规则（比如以空格预切分）
tokenizer.pre_tokenizer = Whitespace()
# Step 4: 加载数据集 训练tokenizer
files = [f"./wikitext-103-raw/wiki.test.raw"]
tokenizer.train(files, trainer)
# Step 5: 保存 tokenizer
tokenizer.save("./tokenizer-wiki.json")
```

tokenizer训练的细节

- **不要随意扩词表：**例如加入了“中华人民”这个新 token，并且引入相对应的 merge token 的逻辑，就可能导致“中华人民共和国”这个旧 token 永远不会被 encode 出来，那“中华人民共和国”这个 token 对应的知识也就丢失了。
- **什么时候考虑扩充词表：**
  - 如果知道业务场景，提前补充业务场景相关的token，增加业务场景文本的压缩率，比如医疗场景，提前把阿莫西林、青霉素等作为一个 token。换句话说，如果基座模型的词表跟领域词表差别很大，可以增加domain的语料。
  - 扩充的token都是低频词，对原词表中的高频词影响不大（比如“思密达”这种音译词）。
- **数字切分**：避免9.9 > 9.11的问题。
- **控制压缩率**：1 个 token 对应多少个汉字：压缩率太低，那就是字太多、词太少，很影响解码效率；压缩率太大，也就是词太多，又会影响模型的知识能力。通常，压缩率越低的模型，loss 也会低，大部份中文大模型的1 个 token 会映射成 1.5 个汉字左右。
- **手动移除脏token**（GPT4o 词表泄露发现有很多色情赌博token）
- 词表的中、英覆盖率要足够大，其他语种看业务需求。

- tokenizer 的 vocab_size 和模型的 embedding_size 之间，要有一千个左右的 buffer，后续的 alignment 环节，需要大量在 pretrain 阶段没见过的全新 token 来做训练。

体验分词器：https://tiktokenizer.vercel.app/



7.3.2 模型选择

在预训练时尽量**选择基座模型**，不选 Chat 模型。模型架构一般采用**Llama**架构，采用**ROPE + GQA + RMSNorm + SwiGLU**，模型参数量与计算资源和数据量呈正相关。

7.3.3 核心超参

- **学习率和batch-size：一般模型规模越大batch-size越大**。学习率一般采用**WSD(warmup-stable-decay)**的策略。
  - **warmup**是为了保障前期训练稳定。
  - **stable**阶段维持高学习率，更快更广泛的探索最优的学习空间。
  - **decay**阶段学习率退火，需要更高质量的数据或者想着重增强模型能力领域的数据。
- **layer_num 和 hidden_size**：一般**layer_num 越大 hidden_size越大**，保障模型宽度和高度比较均匀。layer_num最好有多的质因数，保障流水线并行的效率。
- **num_head**：一般是**8 的整数倍**，因为张量并行的极限一般是8，大于 8 就引入了机间通讯，训练效率就低了。

7.3.4 预训练框架

megatron适合大量数据训练，预训练的数据一般是以T为单位的，所以预训练阶段往往采用megatron。

训练数据的训练速度：卡内通迅 > 卡间通讯 > 机间通讯。为了加速训练效率和降低通讯量，应避免机间通讯，尽量不引入**张量并行、流水线并行、序列并行**，只有**数据并行**不会牺牲算法里。在显存充足的情况下，不要开启数据offload和梯度激活点。

> 🍞 **Megatron vs Deepspeed**
>
> 具体介绍详见**2.4.5 deepspeed 和 2.4.6 Megatron-LM**
>
> - **训练速度**：Megatron训练速度更快，因为支持更完善的张量并行和流水线并行，并且，rope 已经被开发成了 apex 算子，速度远高于 llama 里的实现方案。Deepspeed训练速度相比Megetron会有 10% ~30% 左右的算力损失。
> - **参数清晰**：Megatron参数配置的非常完善，可微操的空间很大。与之对比的Deepspeed很难对源码进行修改。
> - **上手难度**：Deepspeed代码简单，且大多数开源项目也基于Deepspeed实现。Megatron代码复杂。

7.3.5 预训练监控

需要监控的指标有：

- 分别监控中文、英文、代码三类数据的**channel_loss**。
- 监控**loss的突然变化**，大概率是数据问题（全是乱码的数据 loss 很高，全是换行符的数据 loss 很低）。如果数据过脏，就回退到上个 checkpoint 进行训练。
- 如果数据没问题还是出现loss突然变化，监控**梯度**有没有异常，大概率与**优化器参数**设置有关。
- **困惑度PPL**，从每个数据集随机抽取几百条数据监控PPL。

7.3.6 Scaling law

scaling law指的是模型性能（如测试集上的损失或准确率）如何随着模型参数数量、训练数据量以及计算量的增加而变化的经验性定律。这三者按**幂律关系（Power Law）**共同影响模型表现：当三者同步增长时，损失函数以可预测的速度下降——但任何单一变量的过度增长都会遭遇瓶颈。

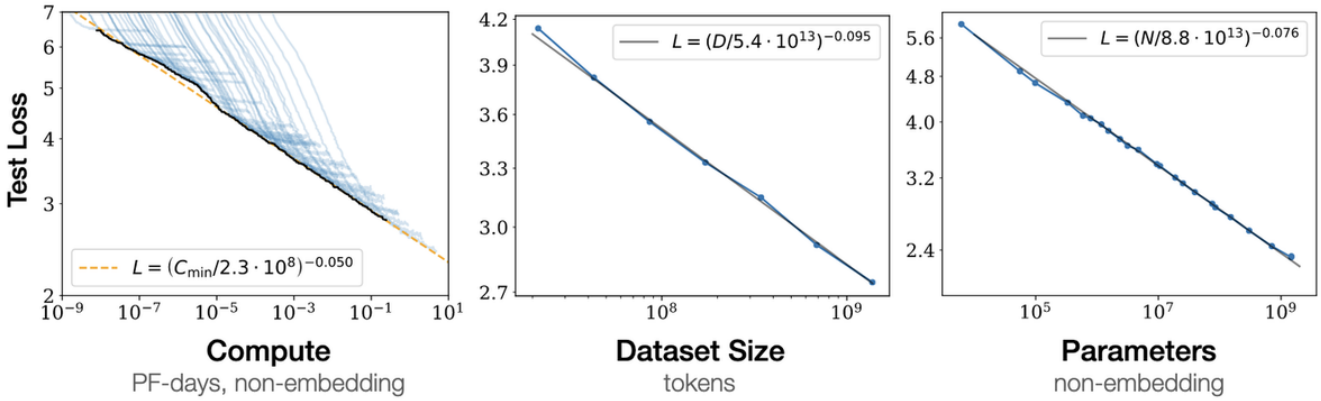


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when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

Scaling law的用途

- **优化资源分配：** Scaling Law揭示了**参数规模、数据规模与算力之间的最佳配比**。OpenAI的研究发现，在给定计算预算下，与其训练一个小模型用海量数据，不如训练一个**更大的模型**但**适度早停**更为高效。更大的模型对数据的**样本效率**更高——达到同等性能所需的训练步骤和数据更少。
- **预测和规划模型训练：** 在小规模模型上实验并拟合Scaling Law，可以**预测更大模型的性能**，从而决定训练多大的模型、用多少数据，以高效利用有限的计算预算，并且根据实验结果调整训练参数、数据配比等。
- **评估模型扩展性：** 通过Scaling Law，可以评估模型性能是否达到了当前规模的**饱和点**。如果模型仍遵循预期的幂律提升，则意味着增加规模仍有益；反之，如果性能提升出现停滞或偏离幂律曲线，则提示可能需要调整策略（如改进模型架构或算法）而不仅仅是“堆料”扩展。

Scaling Law的核心公式

1. 对于Decode-only模型，其计算量C（FLOPs），模型参数量N，数据量D满足以下关系：
   $\mathbf{C \approx 6\,N\,D}$ 。**计算量与模型结构无关**。

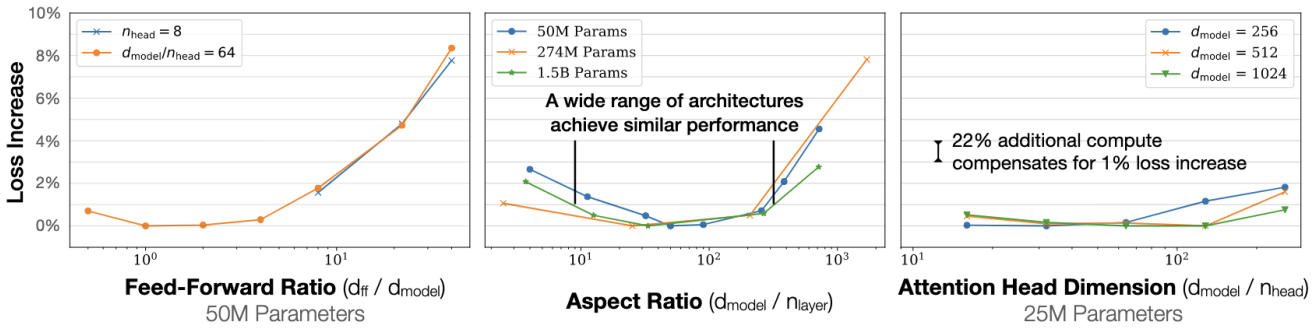


Figure 5 Performance depends very mildly on model shape when the total number of non-embedding parameters N is held fixed. The loss varies only a few percent over a wide range of shapes. Small differences in parameter counts are compensated for by using the fit to $L(N)$ as a baseline. Aspect ratio in particular can vary by a factor of 40 while only slightly impacting performance; an $(n_{\text{layer}}, d_{\text{model}}) = (6, 4288)$ reaches a loss within 3% of the $(48, 1600)$ model used in [RWC+19].

2. $L(N, D) = \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + L_0$，其中 L_0 是理论上的不可减损失下界（数据和模型无限大时的极限误差）,根据《**Training Compute-Optimal Large Language Models**》中的计算，$\alpha \approx 0.34, \beta \approx 0.28$，模型参数量对损失的影响略强于数据量。通过固定总算力 C，就可以推导出最优 N 和 D 的关系。

模型和数据放大比例存在不同的策略：

- **OpenAI**：若将训练计算量提高10倍，OpenAI推算**模型参数应增加约5.5倍，训练token数量增加约1.8倍**，反应在公式上就是**最优参数量** $N_{\text{opt}} \propto C^{0.73}$，**最优数据量** $D_{\text{opt}} \propto C^{0.27}$.
- **DeepMind**：**模型大小和数据量应按相等比例扩展**，提出**Chinchilla定律，即训练语料的token数量约为模型参数量的20倍。**

Scaling Law在大模型中的体现

- **GPT4**：基于10,000倍小的计算规模，就能预测最终GPT4的性能。

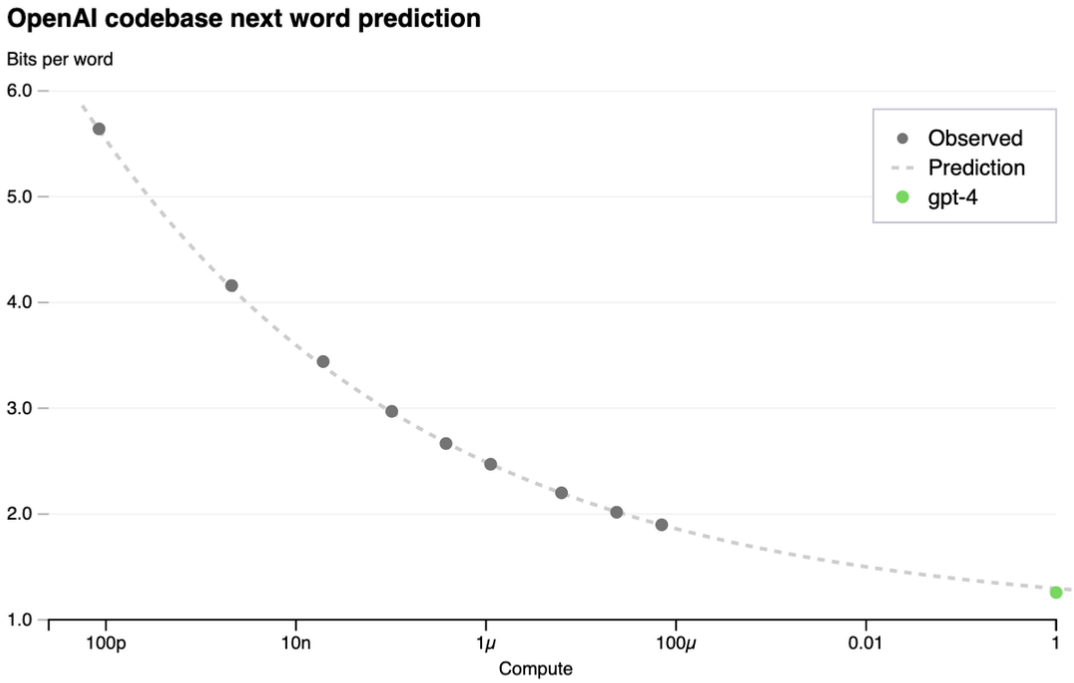


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

- **Baichuan2**：基于10M到3B的模型在1T数据上训练的性能，可预测出最后7B模型和13B模型在2.6T数据上的性能

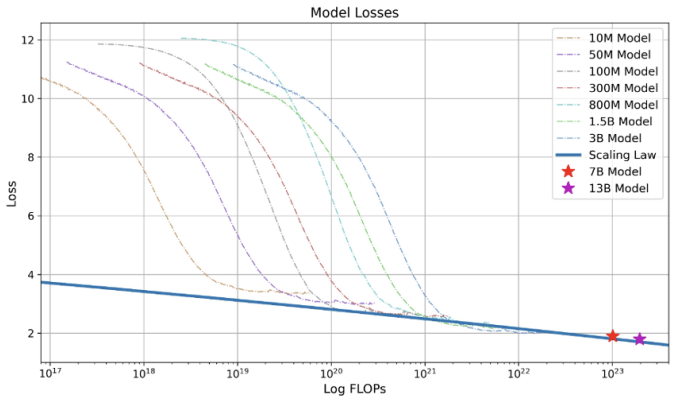
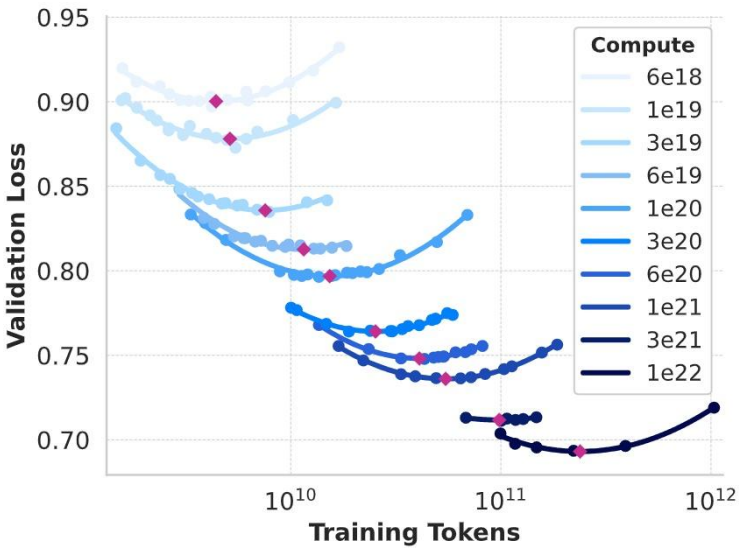
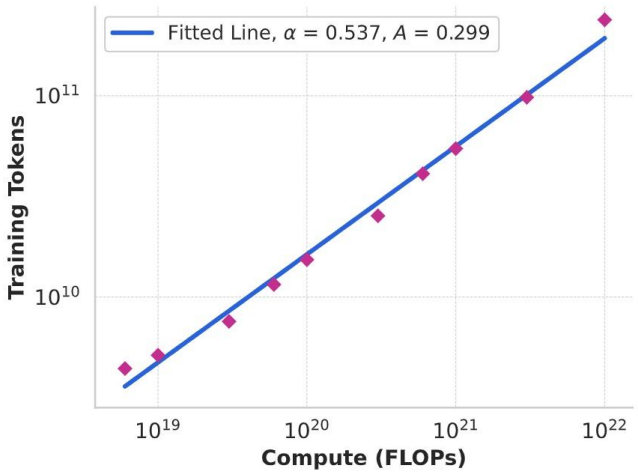


Figure 4: The scaling law of Baichuan 2. We trained various models ranging from 10 million to 3 billion parameters with 1 trillion tokens. By fitting a power law term to the losses given training flops, we predicted losses for training Baichuan 2-7B and Baichuan 2-13B on 2.6 trillion tokens. This fitting process precisely predicted the final models' losses (marked with two stars).

- **MindLLM**：基于10M到500M的模型在10B数据上训练的性能，预测出最后3B模型在500B数据上的性能。
- **LLaMA3**：给定不同的预算 $6 * 10^{18} \sim 10^{22} FLOPs$，调整模型的大小（从 40M 到 16B）。



基于在小模型上的实验结果，接下来预测在大预算下什么模型是计算最优模型。使用幂律法则来建模： $N^*(C) = AC^\alpha$ 。其中 C 是 FLOPs 预算，N* 是这个预算下的计算最优 Token 数，A 和 alpha 是需要拟合的参数。将刚才的实验中的计算最优点，重新绘制成下图，就可以拟合到这条线。这里 alpha=0.537，A=0.299。



输入是 $3.8 * 10^{25}$ FLOPs，得到计算最优 Token 数量为 16.55T，模型大小为 402B。LLama3的模型是405B。

7.4 预训练评估

7.4.1 PPL

困惑度Perplexity是自然语言处理常见的评价指标，可以衡量语言模型的好坏。困惑度越低表示模型对下一个词预测的越准确，具体原理详见 LLM体系知识搭建。

具体做法则是，收集一些预训练没用到的通用知识、数学推理、代码等数据，检测在测试集上的loss变化，正常情况下整体趋势呈下降逐渐稳定。只能同一个模型的PPL对比，因为不同模型的tokenizer压缩率不同，全是字没有词的 tokenizer，loss 一定是最低的。无论tokenizer压缩率多少，在通用知识上的loss应该能降低到2以下，否则说明训练不充分。

7.4.2 开源BenchMark

常用评估指标有：

- **MMLU**：评估 LLM 在广泛主题领域的理解和推理能力
- **GLUE**：对不同语境下的语言理解能力进行全面评估
- **GSM8K**：测试 LLM 解决多步数学问题的能力
- **MS-MARCO**：测试模型准确理解和响应真实世界查询的能力
- **HHH**：评估模型在交互场景中的道德反应
- **HumanEval**： 测试根据给定需求生成正确有效的代码

> 🥖 **推荐一些BenchMark相关链接**
>
> - **CLiB中文大模型能力评测榜单**：https://github.com/jeinlee1991/chinese-llm-benchmark
> - **大模型评估平台**：https://github.com/open-compass/opencompass
> - **SuperCLUE中文评估BenchMark**：https://arxiv.org/pdf/2307.15020
> - **更多BenchMark数据集**：https://blog.csdn.net/qq_36803941/article/details/140045494

7.4.3 概率探针

从概率的角度来监控模型的知识能力有没有遗忘或者提升，适用于我们要观察模型的某一项具体能力。观察某个 token 的概率是否增加或者某个句子的概率是否增加，唯一麻烦的地方是探针测试集往往需要训练者一条一条亲自去构造，而不能批量生成。

比较抽象，举几个例子：

1. Prob('北京'｜'中国的首都是')，就看一下这个概率值随着 pretrain 推进是否持续在增大
2. PPL('天空是蓝色的')，这个句子的 ppl 是否持续在增大；PPL('天空是绿色的')，这个句子的 ppl 是否持续在减小
3. 对比探针，PPL('9.9>9.11') > PPL('9.9<9.11') 是否成立
4. 指令 follow 能力，Prob('{ '| '以 json输出')

重点观察的是指标的变化趋势，而不是指标的绝对大小。

7.4.4 Needle In A Haystack 大海捞针

核心思想是在一段长文本（“**海**”）中插入一个或多个关键信息（即“**针**”），然后观察模型是否能够通过自然语言提问的方式准确地找到这些关键信息。主要用于测试模型在复杂上下文或海量数据中精准定位和提取关键信息的能力，考验模型的**长文本理解能力、信息检索精度**和**抗干扰能力**。

关键步骤：

1. **长文本准备**：选择或创建一段长文本（如书籍、论文、新闻合集）。
2. **插入“针”**：在文本的随机位置插入与文本内容目标关键信息（例如“π的小数点后第1000位是9”）。
3. **提问与检索**：提问跟针相关的问题，例如“π的小数点后第1000位是什么？”

> 🌰 **大海捞针实验的各种形式：**
>
> - **单一信息检索任务**：评估LLM在长文本中**提取单一关键信息的能力**，也就是只插入一个针，测试其对广泛叙述中特定细节的精确回忆能力。这对应于原始的大海捞针测试任务设定。
> - **多信息检索任务**：探讨LLM从长文本中**检索多个相关信息的能力**，模拟实际场景中对综合文档的复杂查询。
> - **多信息推理任务**：通过提取并利用长文本中的多个关键信息来评估LLM的长文本能力，要求**模型对各关键信息片段有综合理解**。
> - **祖先追溯挑战**：通过设计“亲属关系针”，测试LLM处理真实长文本中**多层逻辑挑战的能力**。通过一系列逻辑推理问题，检验模型对长文本中每个细节的记忆和分析能力，在此任务中，去掉了无关文本的设定，而是将所有文本设计为关键信息，LLM必须综合运用长文本中的所有内容和推理才能准确回答问题。

7.4.5 其他指标

- **幻觉检测**：检测LLM是否生成虚假或不准确的信息
- **安全性**：评估LLM的输出是否包含偏见、毒性或其他可能的有害内容
- **LLM评审**：使用LLM本身来评估其输出，例如G-Eval
- **回复相关性**：衡量模型输出是否能够准确、全面地回应用户输入

7.5 继续预训练

Continued pretraining 是指在一个已经预训练过的base模型基础上继续进行预训练的过程，目的是训练出适配下游领域的模型，训练方式与预训练一致。

- **什么时候需要预训练？**

预训练的语料与下游任务语料的【数据分布/领域差异】大时。假如预训练的数据缺少原神相关的数据，我想微调一个原神问答模型，最好先利用大量的原神资料做继续预训练，再利用原神问答对做微调。

- **继续预训练的领域数据占多少？**

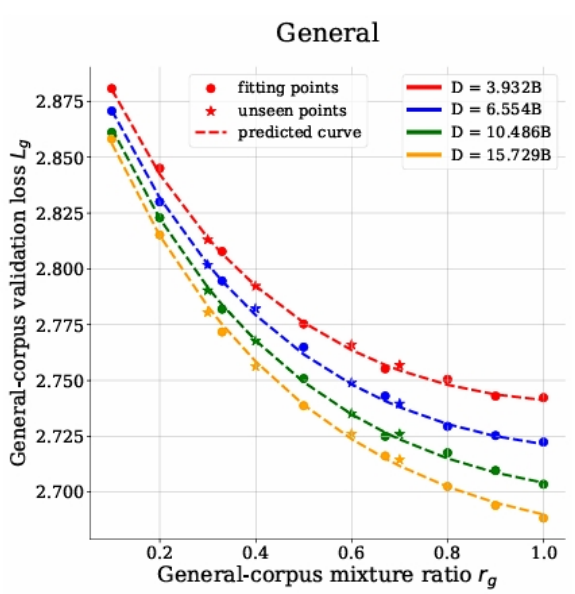
领域数据占比过高，可能导致训练损失直接崩掉，占比太低会导致在领域知识方面提升不大。

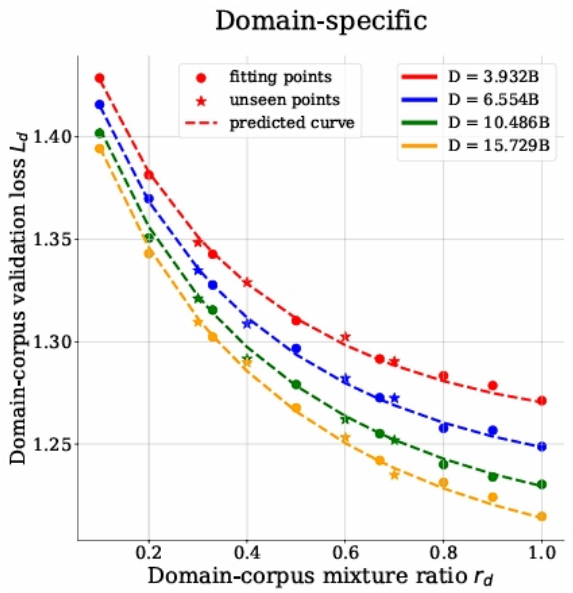
假设领域数据的比例是 r ，关于领域数据比例的scaling law的公式为：

$$L(N, D, r) = E + \frac{A}{N^{\alpha}} + \frac{B \cdot r^{\eta}}{D^{\beta}} + \frac{C}{r'^{\gamma}}, \text{ where } r' = r + \epsilon,$$

当 r 表示域语料混合比例 r_d 时，L 表示领域语料验证损失 L_d 。同样，当 r 表示域语料混合比例 r_g 时，L 表示领域语料验证损失 L_g

随着领域数据的比例增大，会导致领域损失降低而通用损失上升，最终趋于稳定。基于scaling law，就可以估计不同数据配比下损失的预估值。





接下来求解最优混合比例，设模型通用能力下降的阈值不超过T，则优化目标为：

$$\underset{r_d}{\operatorname{argmin}} L_d(N = N_0, D = D_0, r_d) \quad \text{s.t.} \quad \frac{L_g - L_g^0}{L_g^0} < T \qquad (10)$$

继续预训练vs微调

- **训练数据**：继续预训练在预训练数据的基础上，加入**特定领域的无标注数据集**（例如法律、医疗（~~不是董小姐~~），覆盖一个领域，比较困难），沿用预训练的训练目标，微调使用**特定任务的标注数据集**（例如分类、问答，相对比较简单），训练目标与任务相关。

- **训练规模**：继续预训练需要的数据量更大，训练时间长，微调数据量少，训练时间短。
- **训练参数**：继续预训练更新模型的全部参数，微调可以只更新部分参数（例如PEFT）。
- **训练结果**：继续预训练出将通用模型适配到垂直领域，例如将通用模型在医疗数据上训练，得到医学领域的大模型。微调做特定任务的优化，例如将大模型在标注的癌症分类数据上训练，得到癌症分类模型。

👍 继续预训练的经验：

- 数据集需要足够大，至少几B的token，指令数据占比高效果更好。
- 训练开始阶段可能出现loss上升，然后慢慢收敛。
- 要设置学习率warmup，但warmup步数对于充分训练的模型性能影响不大。
- 继续预训练要在**已经过预训练的模型**的基础上做，充分利用其先验知识。
- 继续预训练阶段学习率低于预训练，对于充分训练的模型，学习率越大在下游任务性能越好，上游任务性能越差（遗忘严重）。
- 真实训练中可能没有足够的数据和计算资源，这种情况下**建议选择较小的学习率和较长的warmup步数。**
- 预训练中遇到训练中断，需要继续训练时，应该**把学习率和衰减率都恢复到中断前的状态**。

参考文章：

- Continual Pre-Training of Large Language Models: How to (re)warm your model?
- LLM微调经验&认知-2
- 如何更好地继续预训练（Continue PreTraining）
- LLM训练-pretrain
- 解析大模型中的Scaling Law
- 详解 Llama 3 Scaling Laws
- D-CPT Law: Domain-specific Continual Pre-Training Scaling Law for Large Language Models

🍰
- **预训练**（Pre-training）是指在**大规模无监督数据**上进行初步模型训练，此阶段帮助模型**建立基础的语言理解能力（通用能力）。预训练就像是高中生，学的东西很多但不深入，微调、强化学习等后训练的方法则像选了不同专业的大学生，在不同领域深入研究。**

- 虽然大多数同学的项目中可能接触不到预训练，但可以把继续预训练作为一个项目后续的优化点，特别是对于一些小众领域的特定任务，预训练的模型没有在该领域做充分训练，这里就可以考虑先在大量无监督文本上做继续预训练，再做具体任务的微调。继续预训练的流程与预训练基本一致。
- 关于预训练，有很多预训练已死的说法，即现在已经将互联网上所有数据都用于预训练了，预训练阶段的scaling law也呈现逐渐放缓的趋势。但目前市面上大模型的技术报告里预训练仍是不可或缺的一部分，还没有能完全取代预训练的方法。