【从零训练Steel-LLM】预训练数据收集与处理

战士金 炼钢AI 2024年08月18日 21:43 北京

此篇文章24年5月6日首发于我的zhi hu帐号“战士金”，内容有略微改动。今天(2024.8.18)正好是我们模型预训练完成的日子，后续还会做一些sft和评测的工作。

① 前言

自从工作变动之后就没咋更新了，在新地方需要学的东西太多，实在是抽不出来时间。新部门说实话做的东西和LLM关系不是非常大，只是出于个人兴趣吧，并且有了一些条件，打算最近从头训练一个LLM，我会和@lishu14共同来完成，记录项目过程中数据收集、数据处理、预训练框架思考、模型设计等各种细节，并且会开源所有的代码，让大家都能在有8~几十张卡的情况下使用T级别的数据预训练出一个小型号的（1~2B，对标TinyLlama）可用的中文LLM。tokenizer我们会选择qwen1.5的来用，具体模型的话暂时还没想好，应该不太会完全copy目前的LLM结构，既然都从头预训练了，可以尝试一下自己感兴趣的结构。LLM初步打算定名为steel(钢)，我最喜欢乐队万能青年旅店在做一专的时候条件有限，自称为是在“土法炼钢”，但却是摇滚圈里的一张神专（建议也听听除《杀死那个石家庄人》以外的歌曲）。我们训练LLM的条件也有限，但希望也能炼出好“钢”来。我和 @lishu14都是码农，基本只有周末才有时间弄这个项目，因此项目周期可能比较长（3个月+），也欢迎各位进群交流，群超过200人了，加vx：a1843450905拉群。

```
github地址：https://github.com/zhanshijinwat/Steel-LLM/tree/main
```

② 预训练数据收集

2.1 Steel-LLM使用的数据

Steel-LLM的预训练语料以中文为主，全部为开源数据，github项目Steel-LLM/data/download_data目录下有各数据集的下载脚本。有些数据在huggingface官网，如无法访问，可访问HF-Mirror-Huggingface 镜像站。使用的数据如下：

1. Skywork/Skypile-150B数据集：训练天工大模型使用的部分语料，从可公开访问的233M个网页获取的大型训练语料库。150B个token，大概600GB的数据，自称是开源最大的中文数据集。
2. wanjuan1.0(nlp部分)：由来自网页、百科、书籍、专利、教材、考题等不同来源的清洗后预训练语料组成，数据总量超过5亿个文档，数据大小超过1TB（578GB中文数据+434GB英文数据），经过细粒度的清洗、去重、价值对齐。wanjuan1.0还包含图文和视频数据集。
3. 中文维基过滤数据：数据集基于中文维基2023年7月20日的dump存档，过滤了特殊、低质量、敏感、富有争议的词条，仅保留了254547条质量较高的词条内容。
4. 百度百科数据：563w条百度百科数据，没有经过清洗，大概17GB的数据。
5. 百度百科问答数据：含有150万个预先过滤过的、高质量的百科类问题和答案。数据集总共有492个类别。
6. 知乎问答数据：100w条，1.5GB大小。

7. BELLE对话数据：我们选择在预训练阶段就加入对话数据，减少sft阶段学习对话模式的难度。使用train_2M_CN和train_3.5M_CN两个数据集，由ChatGPT产生的，未经过严格校验。
8. moss项目对话数据：包含110万中英文多轮对话数据
9. firefly1.1M：收集了23种常见的中文NLP任务的数据，并且构造了许多与中华文化相关的数据，如对联、作诗、文言文翻译、散文、金庸小说等。对于每个任务，由人工书写若干种指令模板，数据量为115万。
10. starcoder训练数据：包含 86 种编程语言，共783GB数据，Steel LLM只使用了python、java、cpp这3种语言的数据。预训练数据中加入代码能力可提升模型的逻辑推理能力。

2.2其他数据

这里我也给出调研过程中的发现其他数据，训练Steel-LLM时候并没有用到，如果您有更大的数据需求与算力也可以考虑使用。

中文

1. WuDao文本预训练数据集：采用20多种规则从100TB原始网页数据中清洗得出最终数据集，注重隐私数据信息的去除，源头上避免GPT-3存在的隐私泄露风险；包含教育、科技等50+个行业数据标签，可以支持多领域预训练模型的训练，开源了200GB的数据。这部分数据貌似只支持从web下载，不太方便弄到服务器上，遂放弃。
2. MNBVC：3个老哥发起的中文数据集收集项目，目标收集40T数据，截至2024.3.16号，已经收集80%了。数据包含的比较杂，甚至包含歌词、商品介绍、笑话、糗事、聊天记录等形式的文本数据。我们的算力训练不动这么大的数据量，同时这部分数据没有清洗过。
3. CLUECorpus2020：通过对Common Crawl的中文部分进行语料清洗，最终得到100GB的高质量中文预训练语料，数据有点老了

英文

1. wanjuan-cc：从CommonCrawl获取的一个 1T Tokens 的高质量英文网络文本数据集，开源了100B Tokens。
2. SlimPajama：TinyLlama的训练数据，由 59166 个 jsonl 文件组成，压缩为后895GB大小，是RedPajama的清洁和重复数据删除版本。
3. dolma：OLMo开源模型的训练数据，包含 3 万亿个代币的数据集，来自各种网络内容、学术出版物、代码、书籍和百科全书材料。
4. RefinedWeb:对 CommonCrawl 进行严格过滤和大规模重复数据删除构建的数据集，解压后2.8TB。

代码

1. the-stack-dedup：包含超过 6TB 的开代码文件，涵盖 358 种编程语言。

③ 数据处理

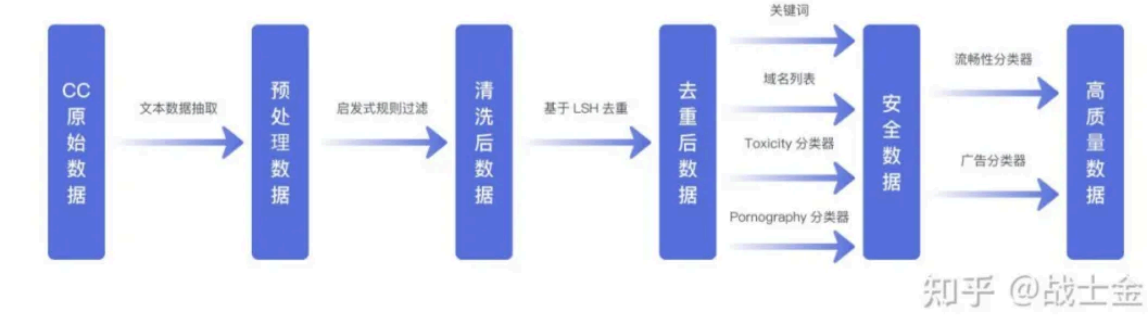
3.1其他数据的处理方式

我们的人手和时间有限，并且使用开源数据大多是有一些清洗和过滤的，因此目前没有实现有害内容检测等比较复杂的数据清洗流程，后面有时间再完善吧。但这里也向读者简单介绍一下一些优秀的项目的从原始数据开始的完整数据清洗方式。

1.WanjuanCC数据处理

```
论文地址：https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/2402.19282.pdf
```

(1) 从Common Crawl的WARC格式数据中提取文本，得到"原始数据"。
(2) 通过启发式规则对原始数据进行过滤，生成"清洗数据"。
(3) 利用基于MinHash（ datasketch 库）的去重方法对清洗数据进行处理，得到"无重复数据"。
(4)使用基于关键词和域名列表的过滤方法，以及基于Bert的有害内容分类器和淫秽内容分类器对"无重复数据"进行过滤，产生"安全数据"。
(5)采用基于Bert的广告分类器和流畅性分类器对"安全数据"进行进一步过滤，得到最终的"高质量数据"。



2.Dolma数据处理

```
论文地址：https://arxiv.org/abs/2402.00159
```

完全开源LLM OLMo的训练数据，论文有83页之多。
(1)语言过滤：使用CCNet的fastText语言试别模型进行语言试别并过滤，目标语言是英语，语言过滤永远不会完美，总会有一些其他语言过滤不掉。
(2)质量过滤：结合处理Gopher数据和 C4数据时的规则来实现质量过滤。
(3)内容过滤：过滤掉"有毒信息"（zz不正确、种族歧视、色情等）,同时使用规则（关键词、域名黑名单等）和文本分类器。个人隐私信息使用一系类正则表达式以及用 Jigsaw Toxic Comments训练过的fastText模型进行试别和剔除。
(4)数据去重：结合使用了 URL、文档和段落级重复数据删除，具体技术上使用了布隆过滤器。

3.ziya2数据处理

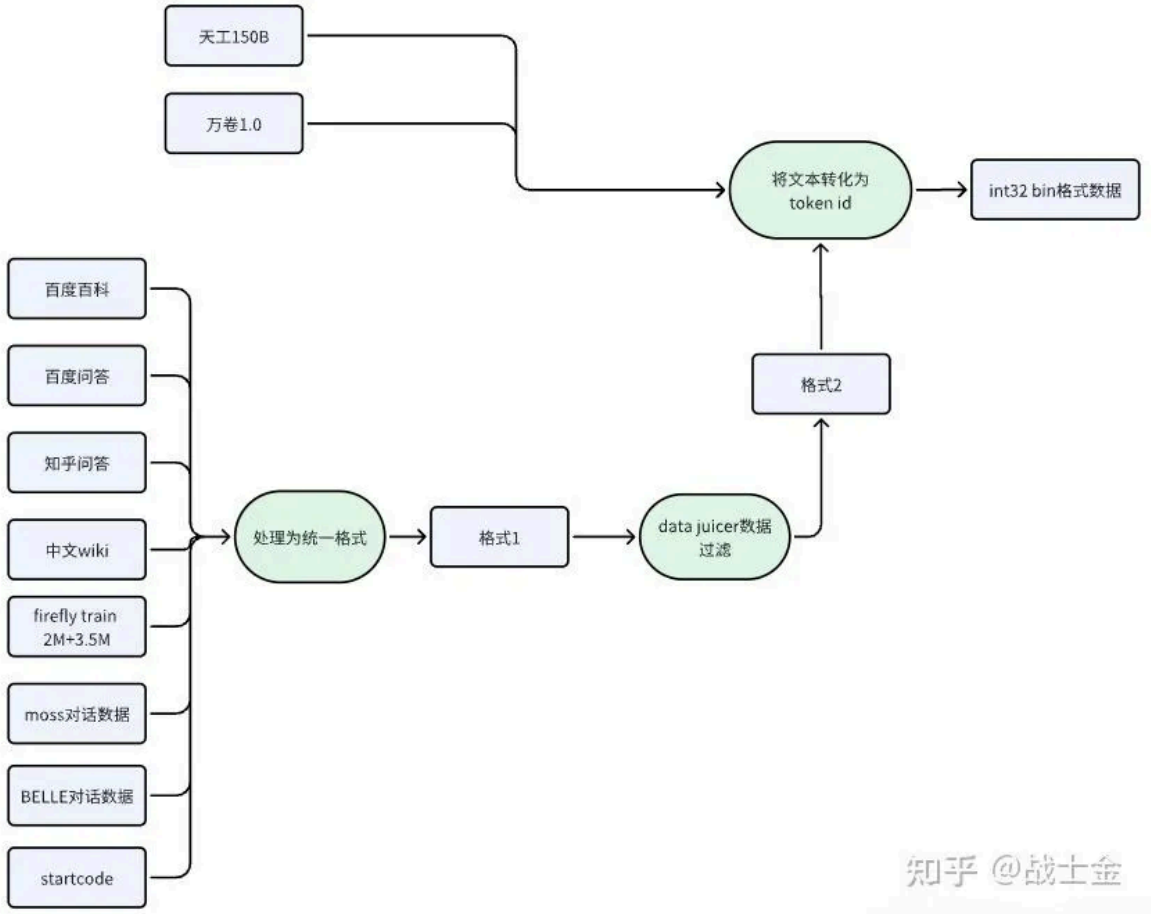
(1)数据预处理：语言检测筛选英文和中文数据。随后对语料库的编码进行了标准化，并将所有繁体中文文本转换为简体中文文本。在此之后，消除了无意义的标记，例如不可见的控制字符、特殊符号、表情符号等。
(2)语言质量评分：使用中文维基百科和英文维基百科训练KenLM模型，预测语料PPL，判断语料质量好坏。
(3)基于规则过滤：在文档、段落和句子三个层级进行过滤。在文档级别，规则是围绕内容长度和格式设计的。在段落和句子层面，规则的重点转向毒性。
(4)重复文本删除：使用 Bloomfilter 和 Simhash来删除重复文本。

4.总结

(1)目标语言过滤：一些LLM只关注特定语言，可通过语言试别模型筛选。
(2)个人隐私内容：个人隐私相关信息不能让LLM学到，可通过正则表达式检测。
(3)有害内容：种族歧视、zz不正确以及色情信息等，通过域名黑名单、词黑名单，有害性检测分类器检测。
(4)格式正确性：是否包含HTML标记、符号或标点是否使用正确等，通常通过人工定义规则和正则表达式解决。
(5)重复数据：重复数据不仅会浪费计算资源，还会对模型性能产生负面影响。目前的重复数据删除方法主要分为三类：基于URL的匹配、基于字符串的匹配和模糊匹配。
(6)内容的质量：指数据集中的信息能否有效提升模型的性能。可以在小模型上进行训练和评估来判断训练数据的质量。或者，可以使用已知的高质量数据集（如wiki数据）作为正样本，然后训练语言模型或分类器进行评分和过滤。也可规定一些内容质量相关的规则。

3.2 Steel-LLM数据预处理

完全复制Steel-LLM的数据处理流程大概需要4T左右硬盘，如果想在几个小时内将文本转换为token id的需要开32进程，200GB左右的内存。Steel-LLM的训练数据处理流程如下所示：



天工150B（600GB+）和万卷1.0（1TB+）从技术报告来看相对干净并且数据量比较大，就不再进一步数据过滤，直接转化为token id保存，给训练程序使用。万卷1.0数据解压前有500g左右，单进程解压需要2小时，解压后大小为1T。万卷1.0数据为jsonl格式，"content"字段存储文本内容。但是有一个文件比较特殊（CN/WebText-cn/part-010669-a894b46e.jsonl）为选择题数据，如下格式所示，需要单独处理，在github项目Steel-LLM/pretrain_modify_from_TinyLlama/scripts/prepare_steel_llm_data.py文件的process_jsonl_file函数中也有所体现。

```
1 {"id":"BkQQU-7xK3YAJdm0cWMc","q_type":"单选题","q_main":"下列属于同种物质的是（）
```

其他8种数据首先统一成统一的格式，便于后续数据过滤。Steel-LLM使用qwen1.5的tokenizer，因此我们也选择遵循qwen的对话模式，需要将moss、BELLE等对话数据转化为qwen对话格式，如下所示。这里有个细节，在处理对话格式数据时，{回答}后边的<|im_end|>在此处是不用加的，因为在后续将文本转换为token id时，会统一在每段完整文本后边加一个<|im_end|>(qwen1.5的终止符(eos_token)，为什么要这么做后文会介绍)

```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
{问题}<|im_end|>
<|im_start|>assistant
{回答}<|im_end|>
```

数据过滤使用的是阿里开源的data-juicer工具，是一款一站式的数据处理工具。每个数据处理步骤在data-juicer中抽象为一个算子，用户可以方便的配置yaml文件实现自定义的数据处理流程，其主要提供的算子有如下5大类，用户也可以自定义数据处理的算子。data-juicer处理数据的实现比较暴力，很多算子是用python原生的for循环遍句子中的字，效率不是很高。

类型	数量	描述
Formatter	7	发现、加载、规范化原始数据
Mapper	43	对数据样本进行编辑和转换
Filter	41	过滤低质量样本
Deduplicator	5	识别、删除重复样本
Selector	2	基于排序选取高质量样本

图2中的格式1和格式2，均为jsonl格式，“text” 字段存储文本。如下所示：

```
{"text":"电子荒原"}
```

我们的数据预处理流程还存在一些不足，比如没有做预训练语料中不同领域文本的配比。

3.3 SteelLLM数据与现有训练框架兼容

在数据量较小的情况下，跑通一个预训练流程很简单，将文本加载到内存中，并将文本变为token id后供训练使用（例如LLaMA-Factory项目），训练时使用Transformers库的Trainer类或是使用pytorch起个分布式任务都不难。但是当数据大到T量级时，将数据完全加载到内存中就比较有压力了。因此，需要找一些能够支持大数据量训练的开源项目进行修改然后进行SteelLLM的训练。经过调研，可以使用TinyLlama项目和OLMo项目提供的训练流程。OLMo项目的代码结构比较重，最终选择代码更加简洁易懂的TinyLlama项目进行魔改。

数据处理的最后一个流程是将文本转换为token id并保存为TinyLlama训练代码能够读取的格式。很多LLM（例如ziya2）在预训练时都会在每段完整的文本后边加入“停止符（eos_token）”，我们使用的tokenizer来自Qwen1.5项目的chat模型，因此对应的停止符

是"<|im_end|>"。Qwen1和Qwen1.5的词表大小不一样，base模型和chat模型的停止符也是不一样的，base模型的停止符是"<|endoftext|>",不要用错。

生成最终的训练数据时，先预定义好一块二维空间，形状为(n, seq_len+1)（seq _len之所以要+1是因为训练方式为预测next token，句子需要移一个位置当作标签），先用用pad_token_id （qwen1.5的pad_token是"<|endoftext|>"）预填充,然后再往近塞文本token id。每一行存放训练时的一段文本对应的token id，文本如果太长就直接截断，放到下一行。最后一个part的二维空间基本不会被完全填满，有很多行只有pad_token_id，如下图所示。不过不要紧，训练时计算loss忽略掉pad_token就好。

321	222	1345	4704	666	377
377	eos_token_id	2235	7427	618	866
73	52	33	eos_token_id	pad_token_id	pad_token_id
pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id
pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id
pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id	pad_token_id

将文本转化token id并保存的这个过程不是很快，621GB的数据，使用32进程并行处理，大概需要6个小时，最终生成578GB的数据。

点个关注再走吧~