# Paper shredding: InternML2

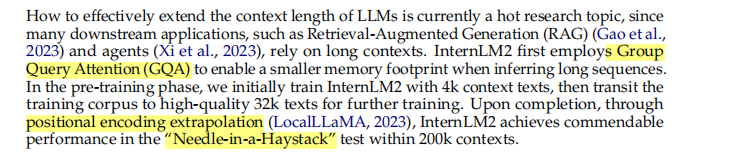
### 关于本轮存在的疑惑

1. 本文档Page 21 关于COOLRLHF为了防止过拟合只训练一个周期是否不妥？
2. 本文档Page 23 这里的lambda 指的是损失函数的lambda参数？也就是几乎所有权重都由惩罚函数提供，那么前者排序损失函数的意义岂不是变得很小？

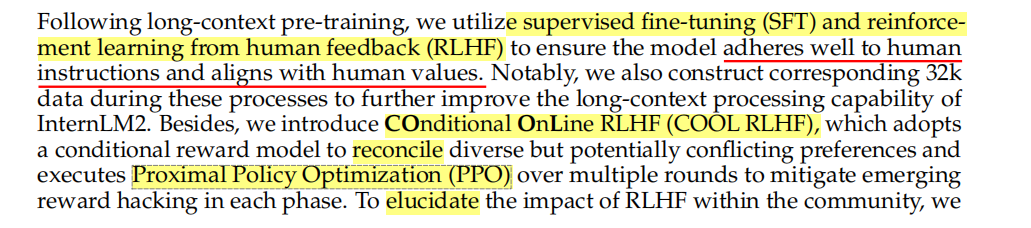
# Introduction

InternLM2 extensively details how it prepares text, code, and long-context data for pre-training.

InternML2数据预处理主要包含三部分： 文本，代码，长文本（包含强位置关联性）的数据。

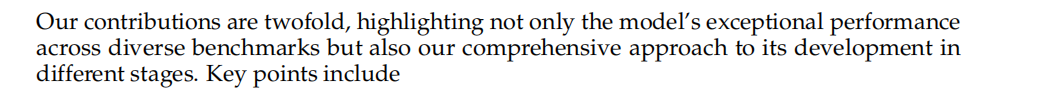


延长文本长度：使用GQA（分组队列注意力机制，一种注意力机制的变体）用以长序列推断，并采用位置信息外扩法进行位置信息的建模和调参，最终用以实现“大海捞针”，即长文本中的语义理解和信息关联问题。

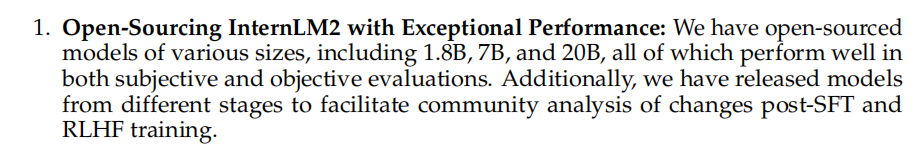


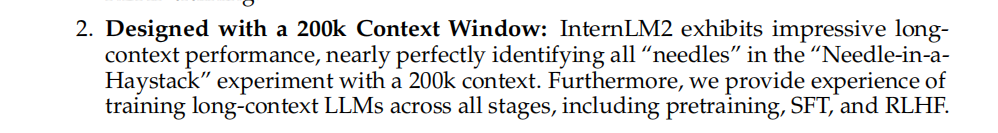
使用监督调优（SFT）和人类回馈强化学习（RLHF）帮助模型更好地与人类所给出的instructions和价值观念相符。

# Contributions

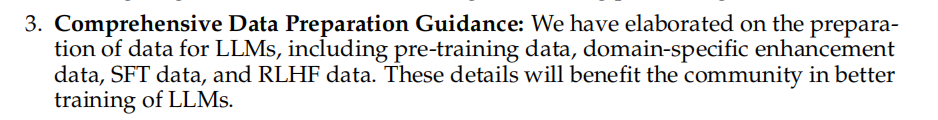


本文的两方面成就：提出新的模型，以及阐释不同层级的实现方法，其主要包含四个方面：

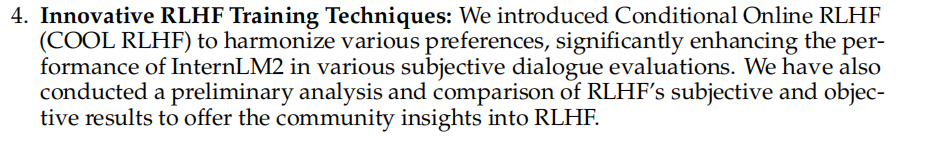




200K长度token中的语义窗口。



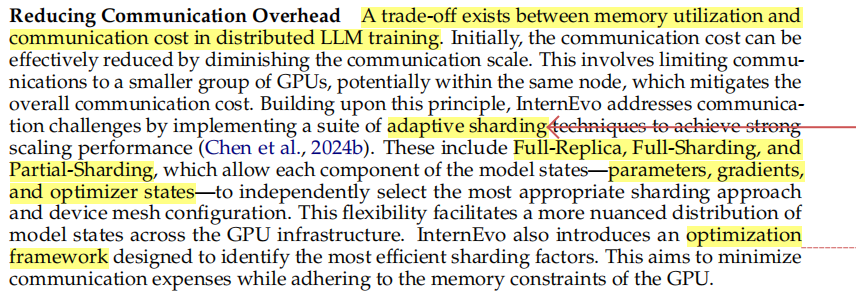
数据清洗和预处理方法。



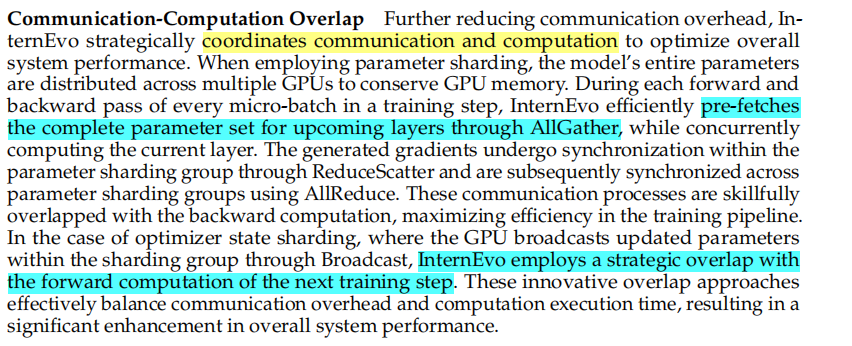
COOLRLHF：在线条件强化学习，用以在不同场景下泛化模型表现。

## Infrastructure

### InternEvo

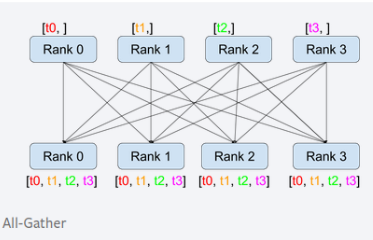
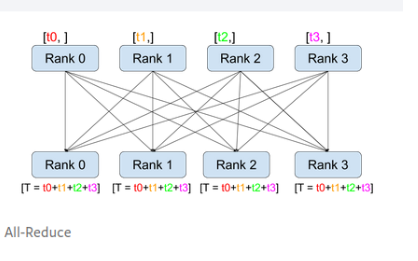


减少通信负载：通常需要调整通信规模，即划分为同一节点内组更小的集群GPU。InternEvo采用自适应分层方法，包含三种分层模式，将包含参数，梯度，优化器状态在内的模型参数独立进行最优化分层策略。同时也提出了最优化框架用以设计和调整分层参数。

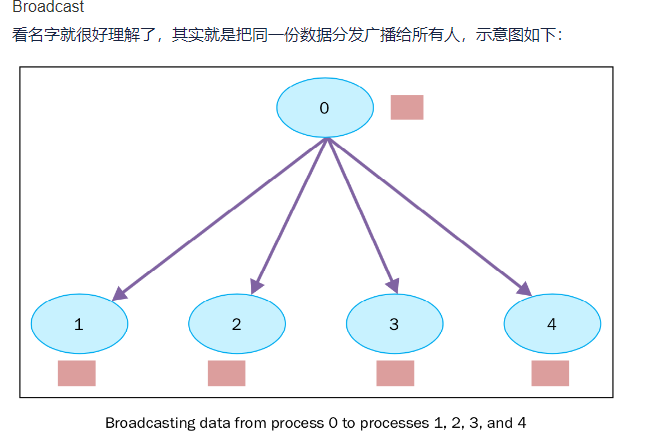


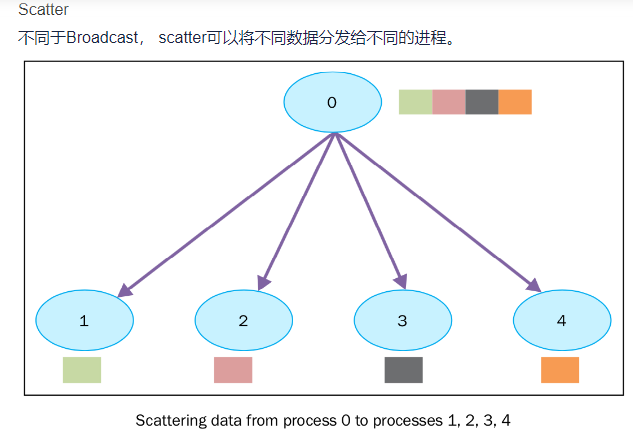
通信-计算重叠：

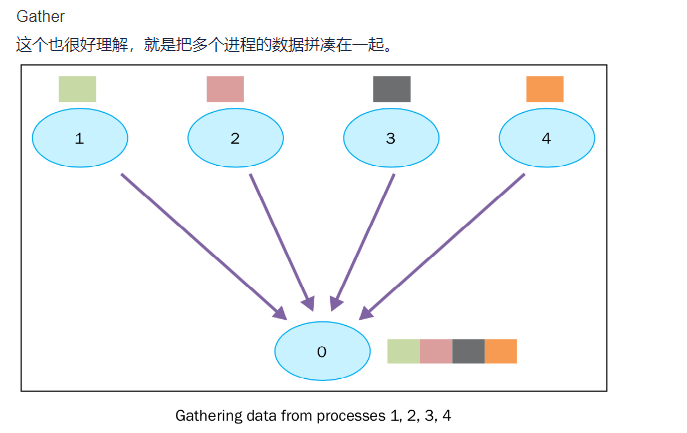
当使用参数分片时，模型的整个参数分布在多个GPU上，以节省GPU内存。在训练步骤中每个微批的每次正向和反向传递中，inmenEvo通过AllGather有效地预获取即将到来的层的完整参数集，同时计算当前层。生成的梯度通过ReduceScatter在参数分片组内进行同步，随后使用AllReduce在参数分片组之间进行同步。这些通信过程与反向计算巧妙地重叠，使训练管道中的效率最大化。

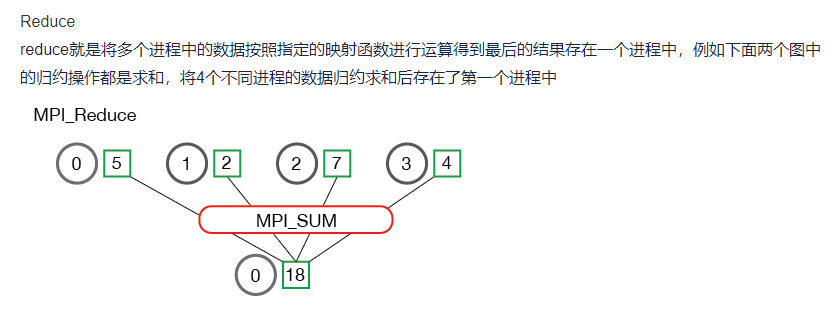


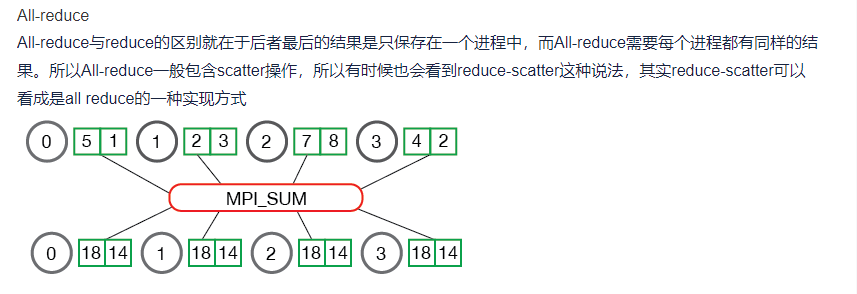
两者都是分布式数据的通讯方式。

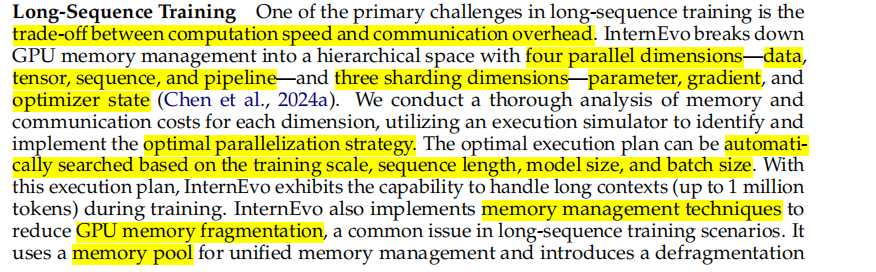




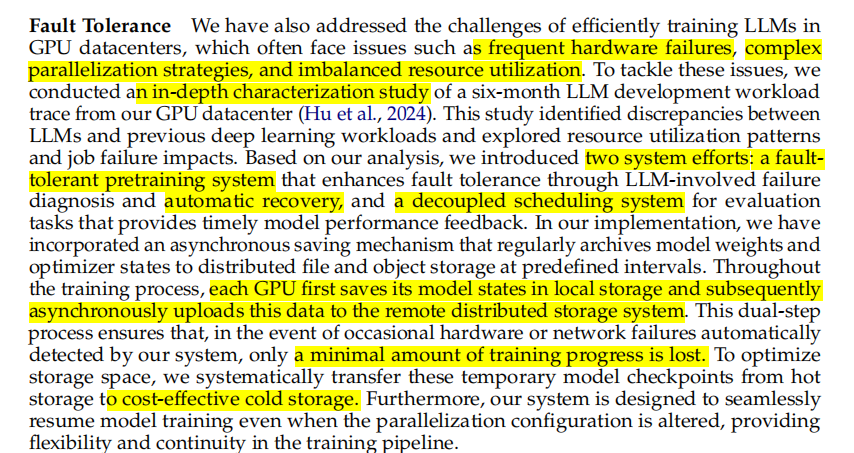




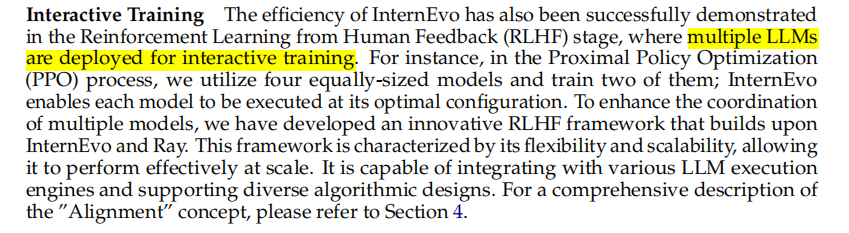




长序列训练涉及到计算速度和通讯成本之间的权衡，InternEvo将GPU内存分为四个平行维度：数据，张量，序列和流水线。分块维度分成三个：参数，梯度和优化器状态。通过分析总体内存和通讯成本构建最优化平行策略，用来解决长文本训练时的内存溢出问题。

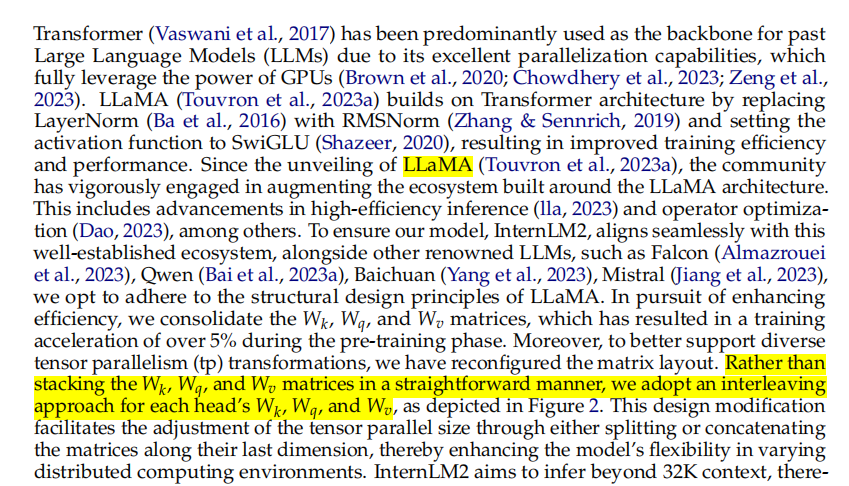


报错恢复：针对可能出现的硬件问题和资源使用不平衡进行的改善。通过演剧主要有两个成果：错误容忍和恢复系统，以及分析性能系统。类似联邦学习，GPU分别训练并存储模型，并异步分发给相邻的数据节点，通过这种方式可以最小化数据通讯量，并减小出现错误时造成的数据损失。为了优化存储空间，使用冷存储（离线存储，访问频率低但需要硬性记录的数据）进行节约数据。

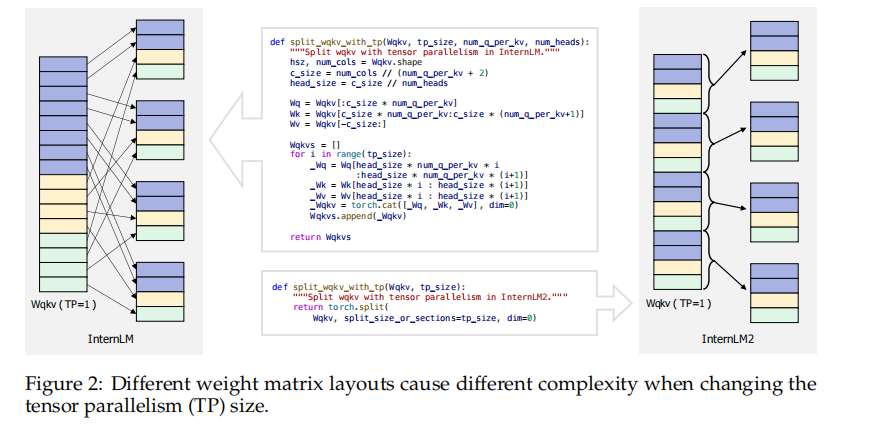


交互训练：融合封装多种LLM进行训练用于强化学习。

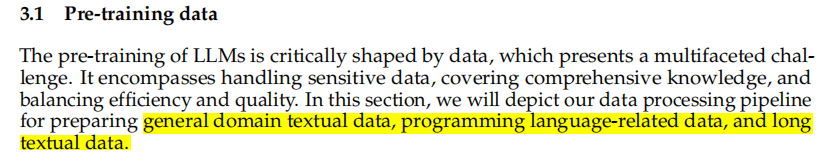
## Model Structure



基于LLaMa模型架构，对k,q,v矩阵进行分插来提高运算性能。

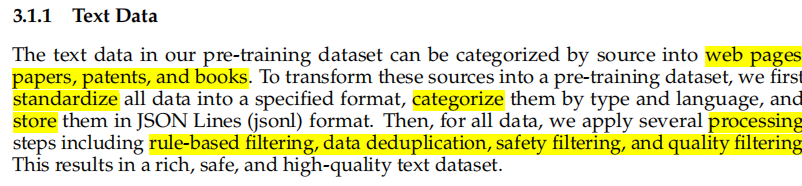


# Pre-train

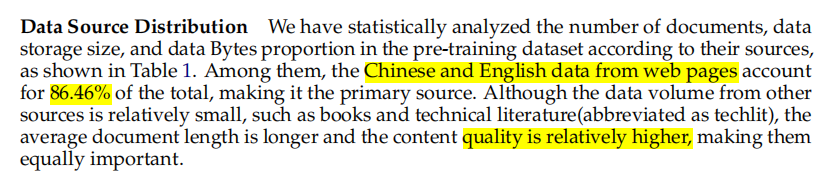


预训练数据集主要分为三部分，分别为普通文本，代码，以及长文本数据。.

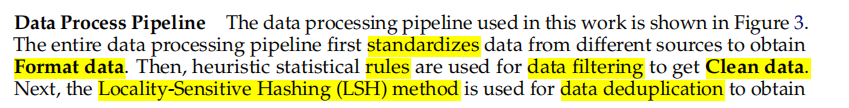
### Text Data



文本数据预处理流程可以简化为获取数据、标准化数据、分类数据、存储数据、处理数据。

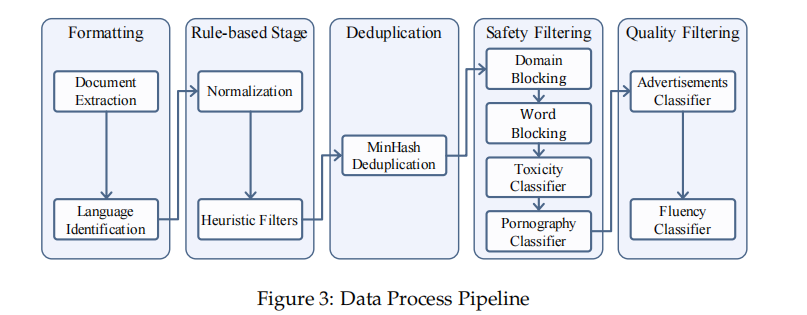


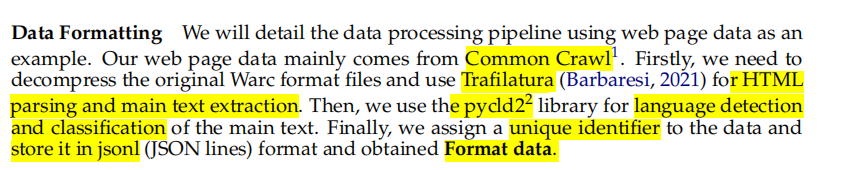
数据分布：主要来源于中英文网页，其他书籍、杂志、报纸等占比较小，但相对质量较高。



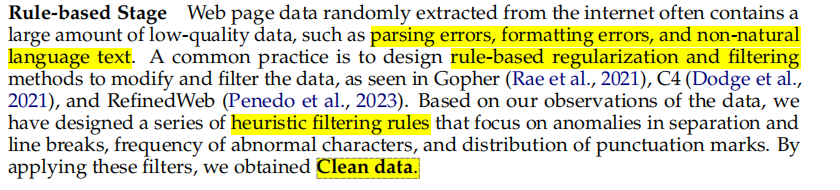


数据处理流程：标准化为标准格式数据、统计学规则清洗为干净数据、位置信息清洗为无冗余数据、安全策略检测为安全数据、质量评估检测为高质量数据。

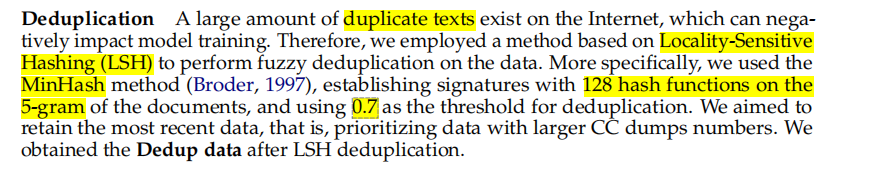




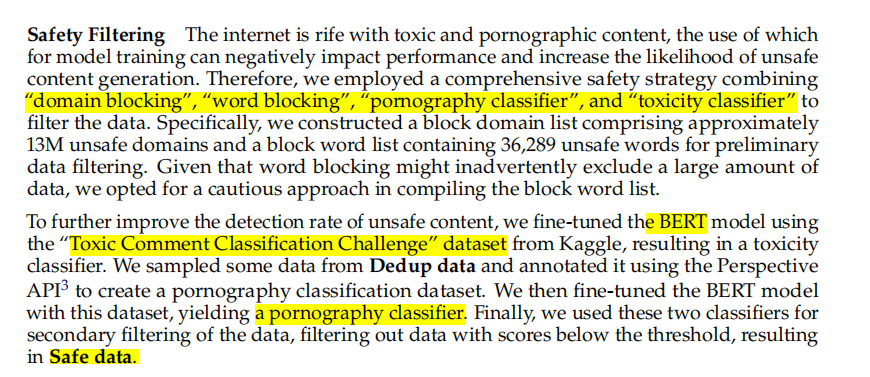
调库干活，没啥好说的



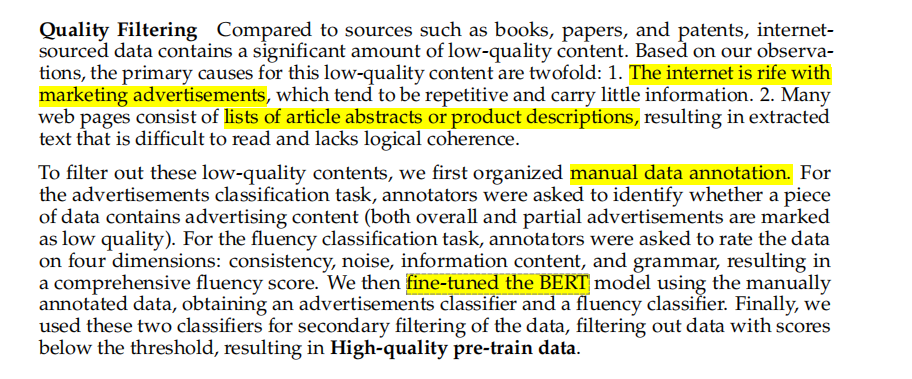
大部分网页数据质量很低或带有格式错误，需要进行过滤



数据去重：位置敏感哈希方法。

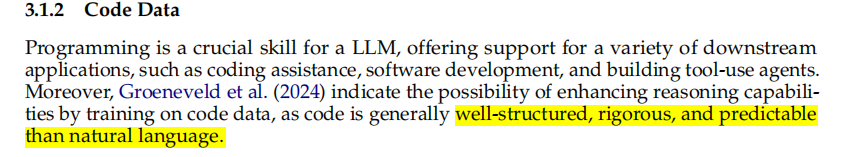


安全性：过滤词+BERT调参

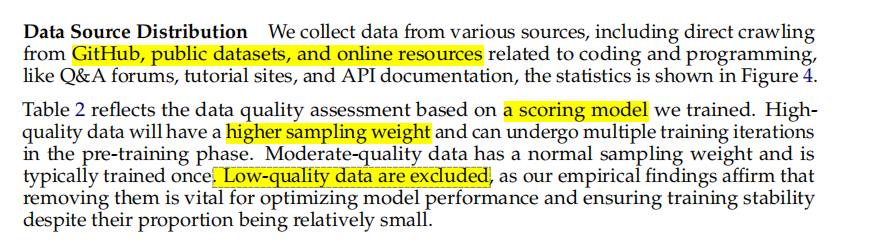


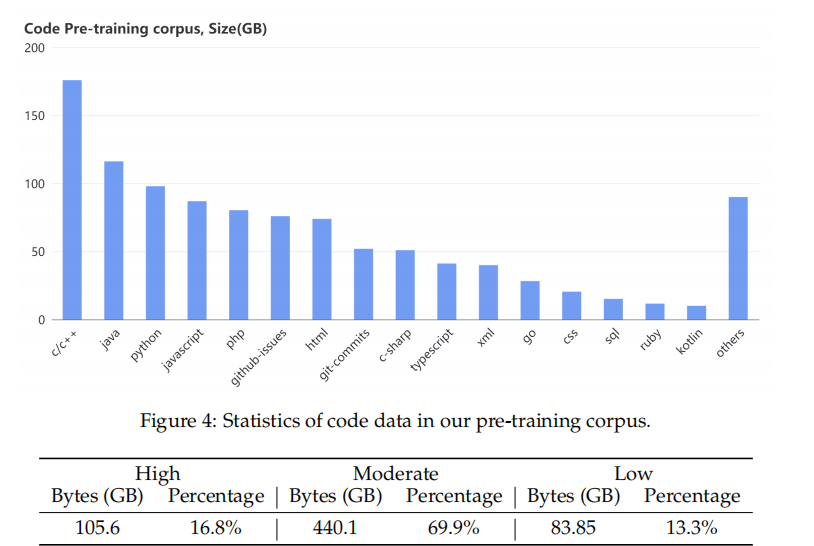
质量过滤：手动标注+BERT调参

### Code Data

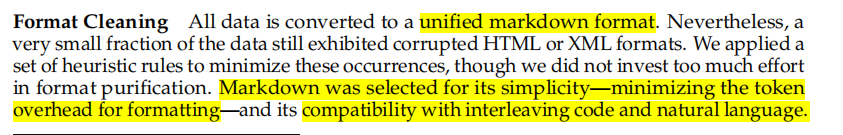


代码数据通常比自然语言更结构清晰，易于建模和预测（通常来说越接近汇编语言逻辑性和规则性越强，越易于建模）

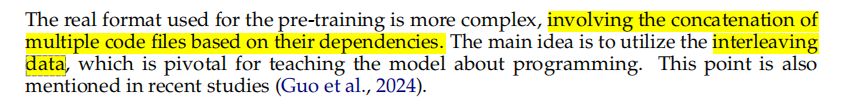




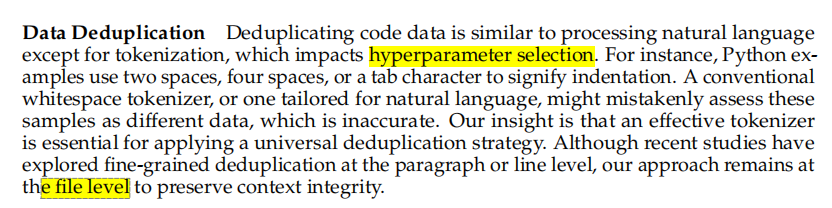
代码数据集：根据代码质量进行评分，高质量的代码数据采样权重更高，低质量的数据将会被排除。



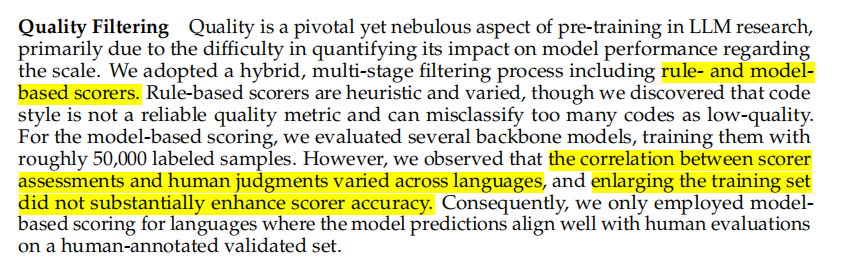
数据清洗：统一简化格式



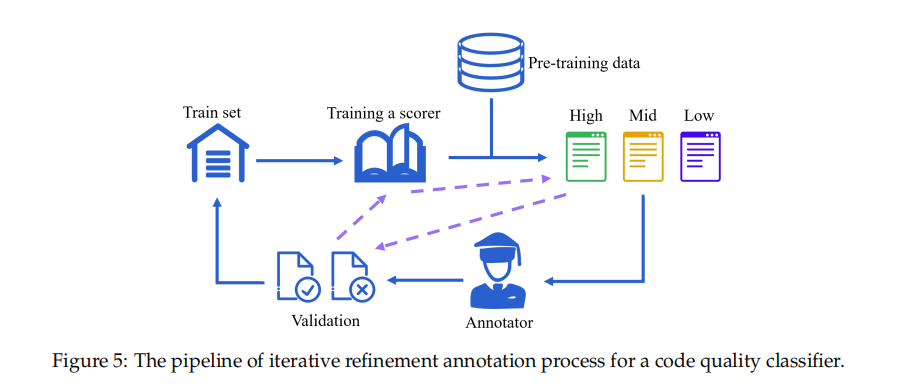
需要注意真正的数据包含多个代码文件的结合，通过使模型理解他们之间的依赖关系来理解分散的数据，从而增强代码建模能力。

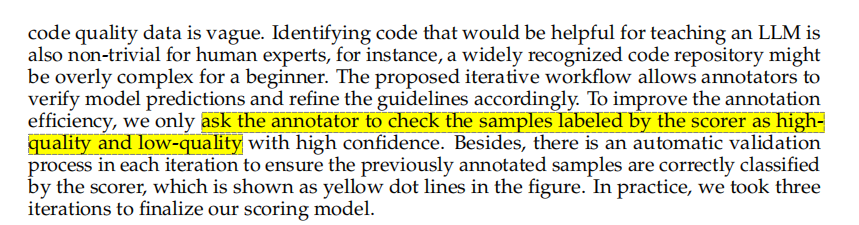


去重：与文本数据类似，涉及部分编程语言的特定格式转换。

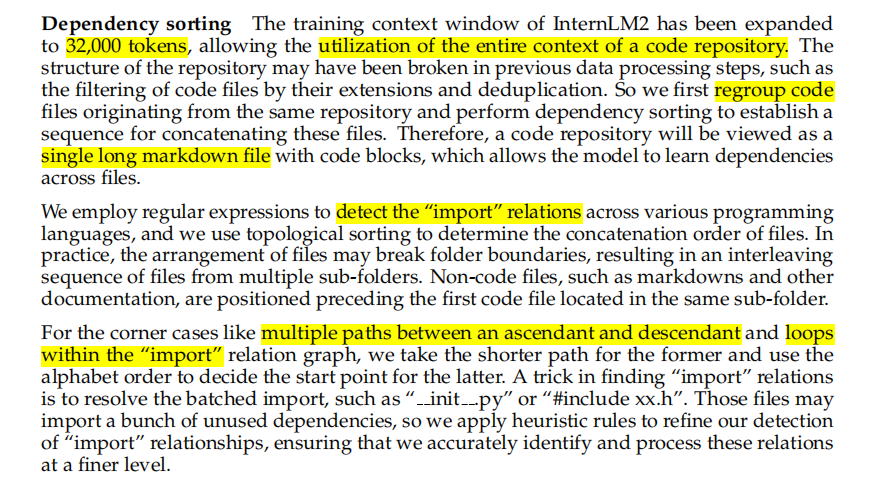


质量评估：评分器与人类结果的相关性随着编程语言不同有较大差异，且不随数据集增大有所改善。



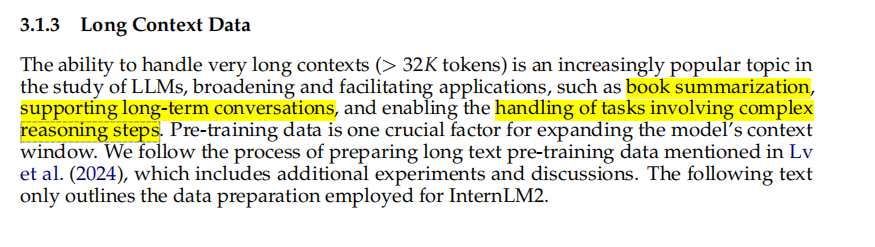


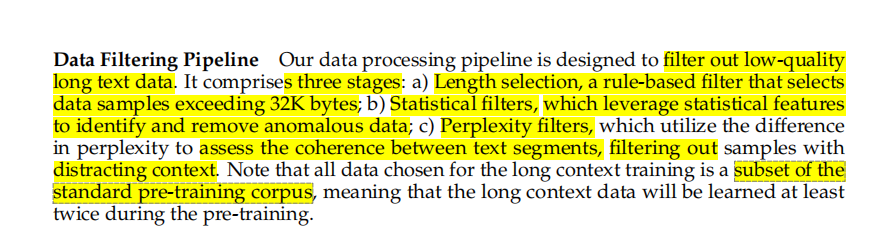
通过循环标注判断代码质量为低/高



依赖判断：通过检测import引入判断依赖关系

### Long Text Data

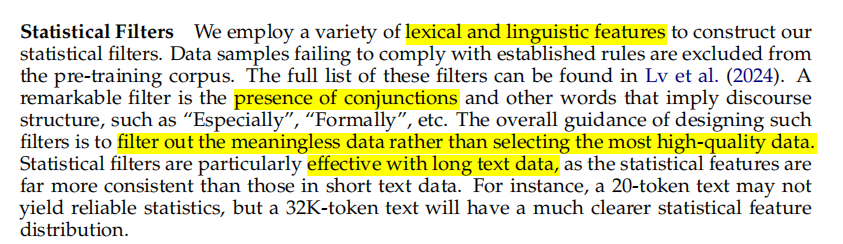




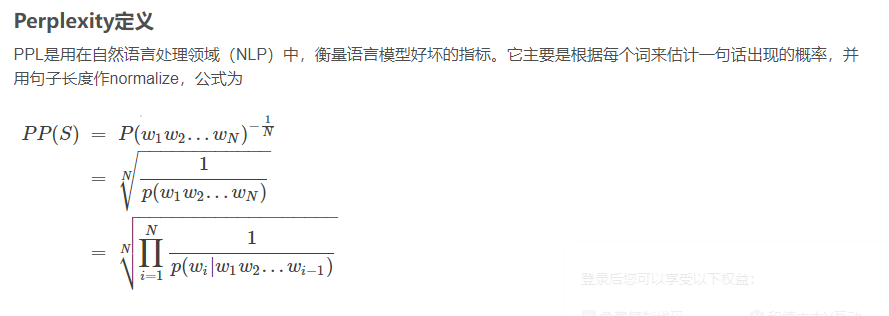
长文本处理：三个阶段：

1. 长度选择，选择大于32Kbytes的数据
2. 统计学过滤：从统计学特征分辨不规则数据
3. 复杂数据过滤：通过计算perplexity分析文本关系

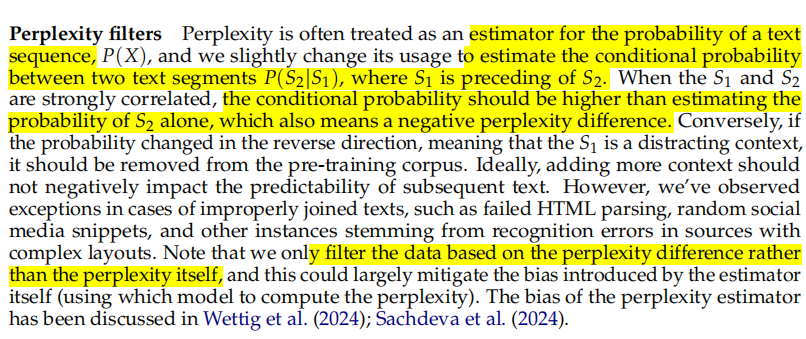
所有的长文本数据样本都来源于基础的语料库，已经被预处理过一遍。



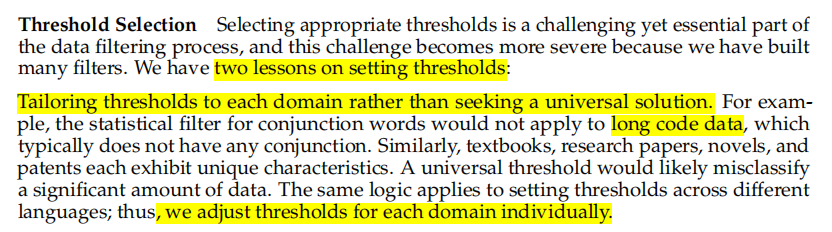
使用词义和语义学特征进行统计分析。关联词往往指代上下文结构。目的是去除无意义的数据而不是选取高质量数据。在长文本中统计学特征更有效。





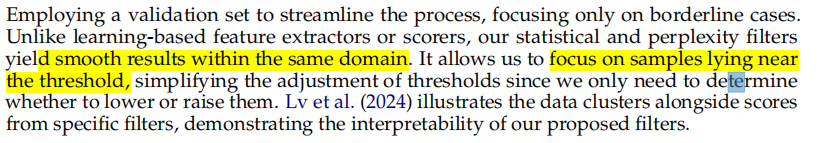


使用Perplexity的差值估计前后文间是否存在关联。关联性强的前后文应在条件概率上相比原始概率有较大增长，意味着Perplexity负差值。



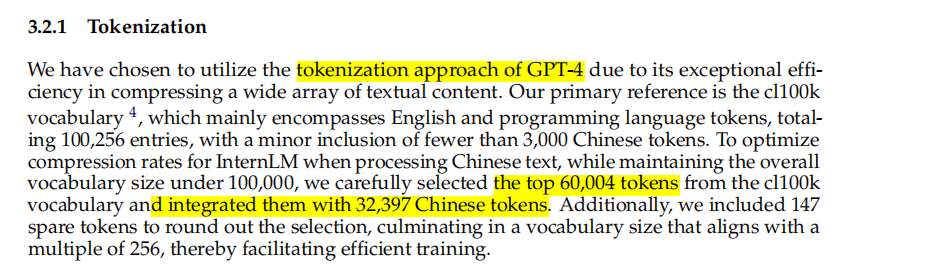
阈值选取：

1.针对不同的语料类型分别调整阈值



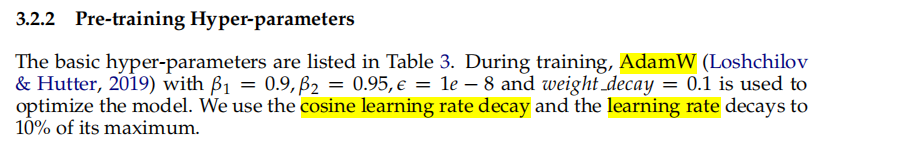
2.只关注边界样例。统计学和perplexity在界内的表现平滑，因此只需关注边界值的情况来决定阈值调整。

### 3.2.1. Tokenization

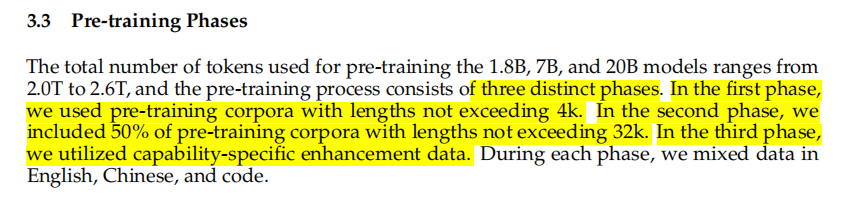


采用GPT4的方式进行分词。

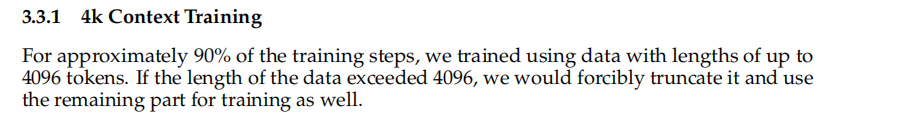
### 3.2.2. Pre-training Hyper-parameters



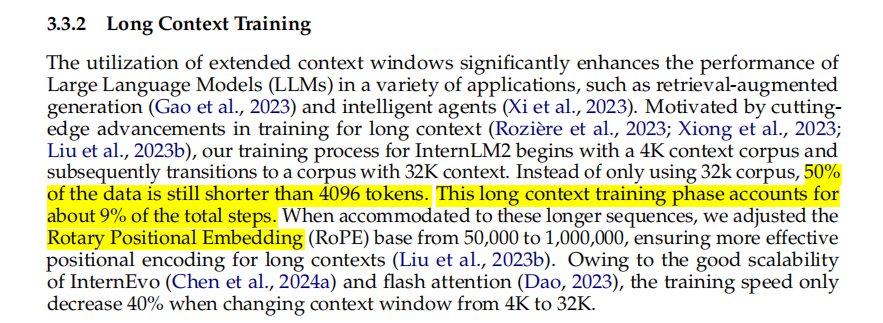
超参数：经典Adam优化器，学习率。



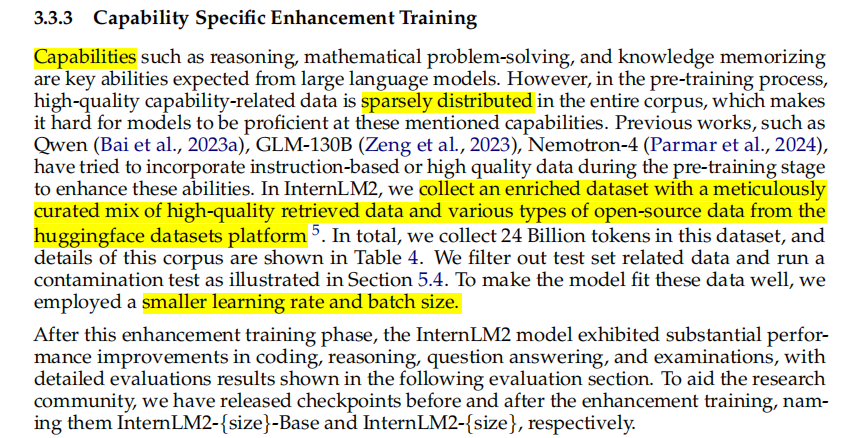
根据文本长度不同在三次预训练过程中分别训练语料库



大多数情况下使用短语料进行训练（先从较短语料提取特征，然后推广到长语料情形）

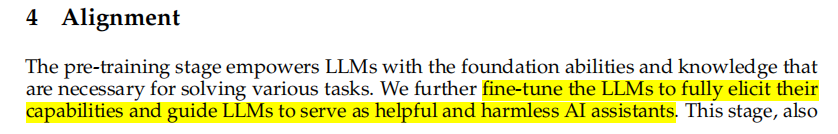


长语料训练情形约占总共的9%。调整位置信息编码方式更好地获取位置信息从而加强长语料建模能力。

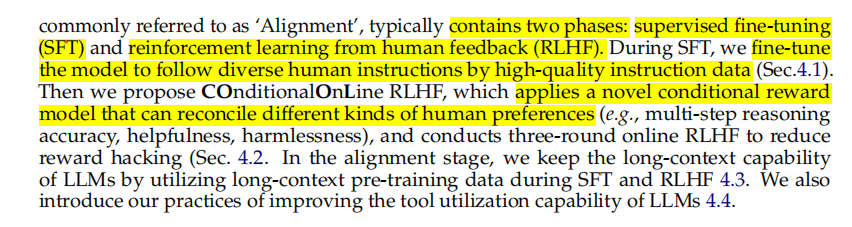


特定能力增强训练：针对特定能力（原因推理归纳，数学问题等），专门建立高质量语料库进行训练，同时使用较小的学习率和块尺寸更好拟合。

# Alignment



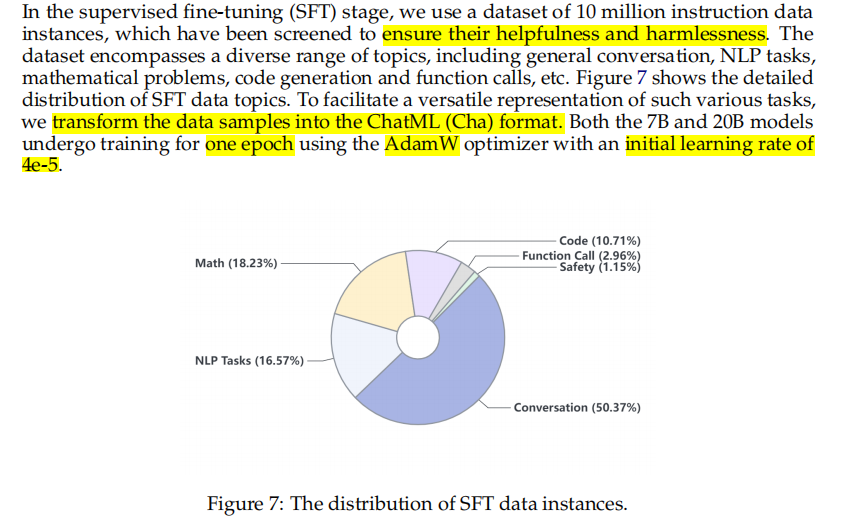
Alignment的原因：更好地满足人类需求，与人类价值观相符，遵循人类instruction生成结果。



包含两个方面：

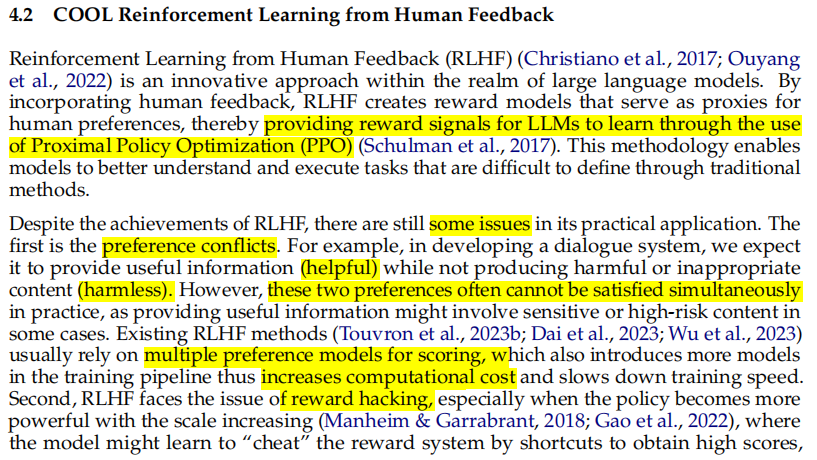
1. SFT，监督调参，用于遵循人类instructions生成结果
2. COOLRLHF，条件-在线-基于人类反馈的强化学习，用于满足不同人类需求（每一步的置信度，无害程度等）

## Supervised Fine-Tuning

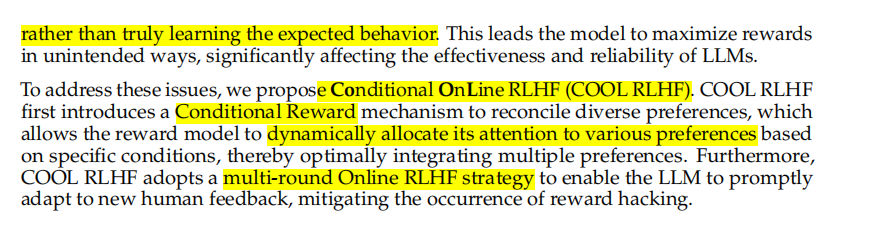


使用不同instructions进行训练，主要为对话

## 4.2 COOL Reinforcement Learning from Human Feedback

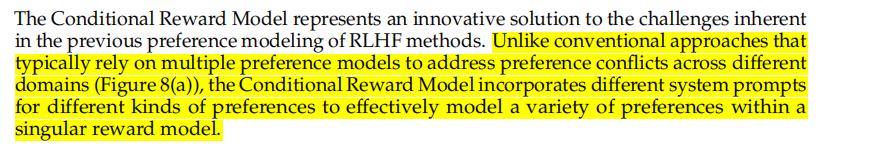


通过近端优化策略的强化学习过程提高性能。同时存在一些问题：偏好冲突，指某些需求很难同时实现（例如提供准确信息的同时不泄露隐私等）。另一个问题是奖励函数“骇入”，即模型通过欺骗奖励函数获得高评分，而不是真正进行学习。

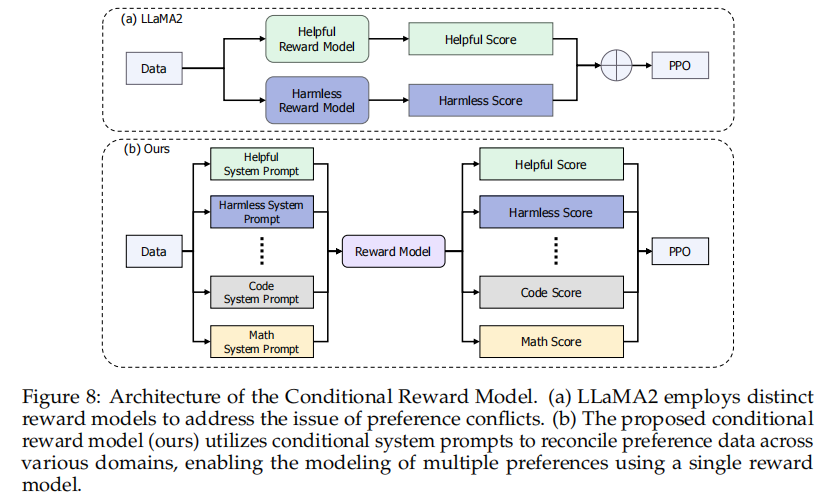


针对这两点问题提出了条件-在线-强化学习机制，即通过“条件”奖励函数在不同的场景下动态调整侧重，从而满足多个奖励函数的需要。通过“在线”强化学习策略及时更新人类反馈，以防止奖励函数骇入的问题。

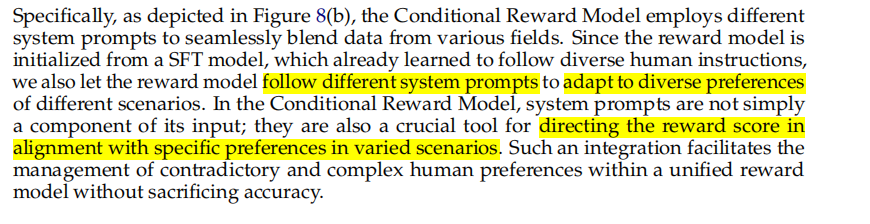
### 4.2.1. Conditional Reward Model



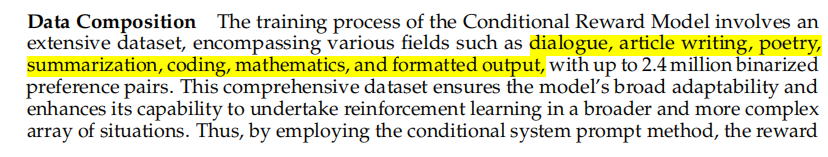
不同于传统方式，条件奖励模型将不同系统的偏好结合到一个模型中。



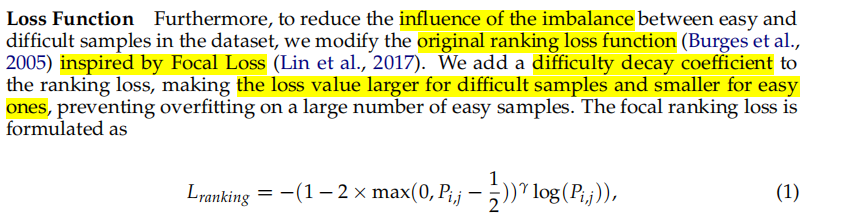
用同一个奖励模型，当时使用不同的prompt进行评估（应该根据需要进行加权），最终算出评估分数。



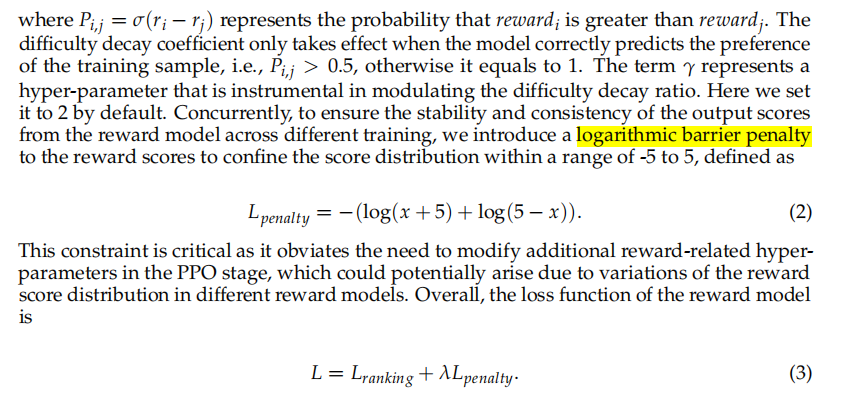
借由不同的prompts对不同场景下的偏好进行校准。



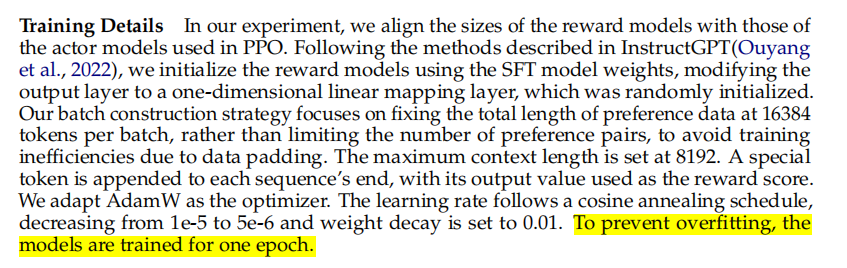
强化学习的数据分布



损失函数：基于排序损失函数，增加难度衰减系数，问题越难对应的损失函数值越大

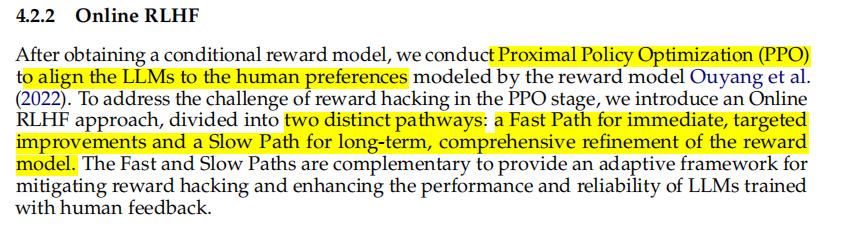


还有一个惩罚函数，最终的损失函数是两者加权和

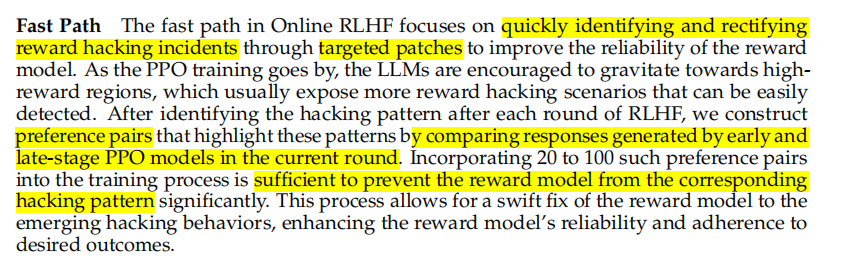


通过只训练一个周期防止过拟合？那要是没有收敛怎么办？

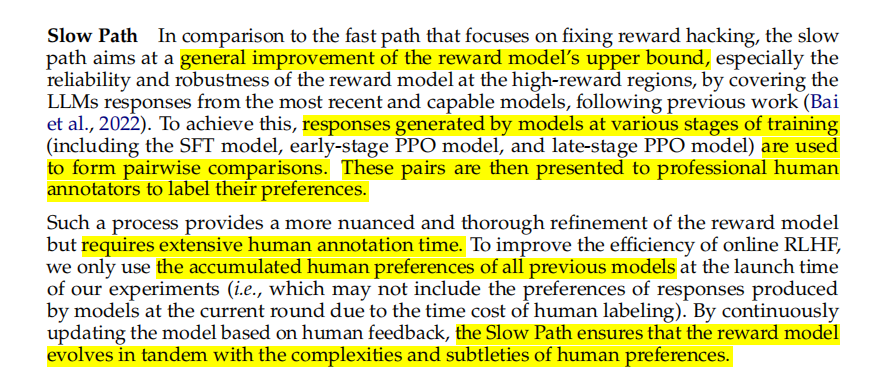
### 4.2.2. Online RLHF



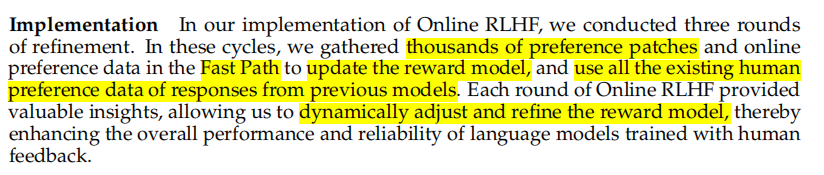
一个快速的、有针对性的改进的快速路径和一个对奖励模型的长期的、全面的改进的缓慢路径。快速路径和慢路径是互补的，提供了一个自适应的框架，以减少奖励黑客攻击和提高由人工反馈训练的llm的性能和可靠性。



快速路径：出现奖励骇入的模型会对某些提供较高奖励的区域偏移，因此可以采用偏好对的方法比较模型在学习早期和后期对这一区域的偏好，以确定是否存在某种程度的奖励骇入现象。

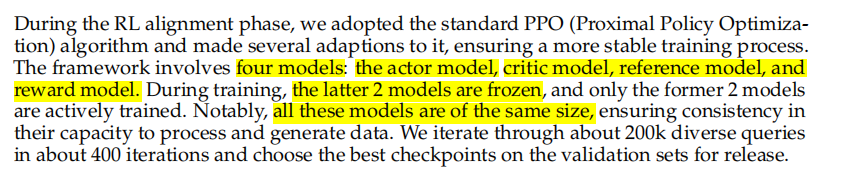


慢速路径：更偏好于提高奖励模型的上限。对不同阶段的模型生成的响应进行比较，由人类进行标注，来确定偏移程度和及时更正。他耗费更多的时间但可以达到更好的效果，在模型复杂性和人类偏好上取得不错的折中效果。

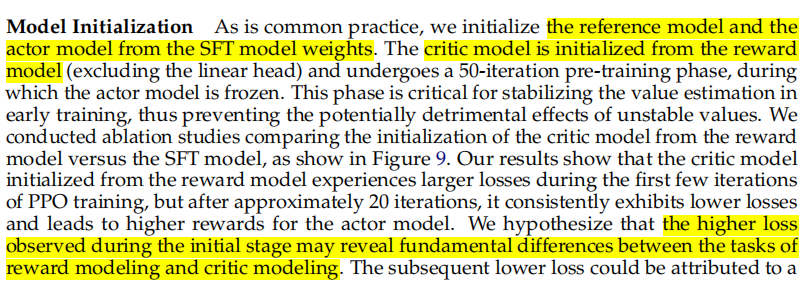


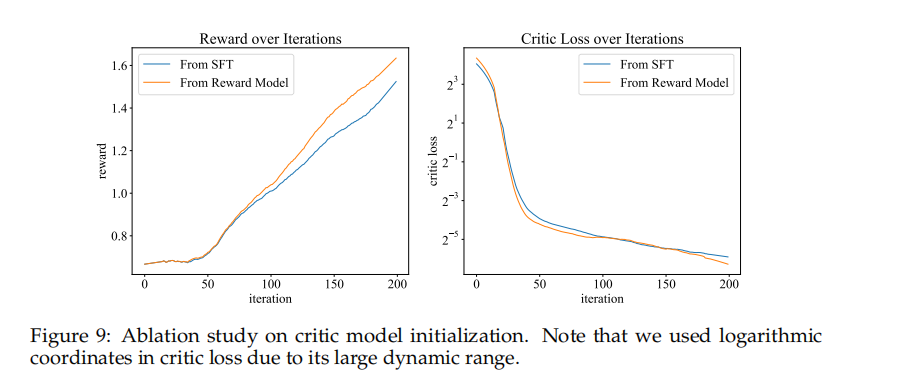
在三个轮次中优化强化学习模型。采用快速路径的偏好对和慢速路径的人类偏好数据进行调整。

### 4.2.3. PPO Training Details

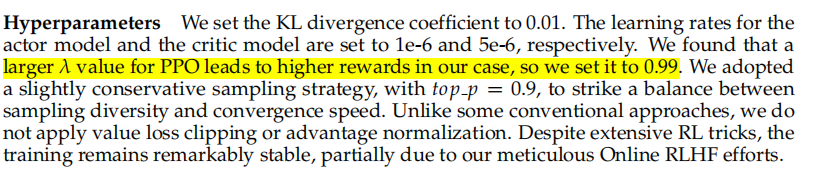


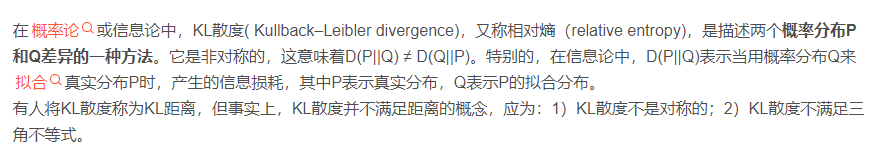
4 models，前两个在训练过程中被训练，所有模型的大小相同。





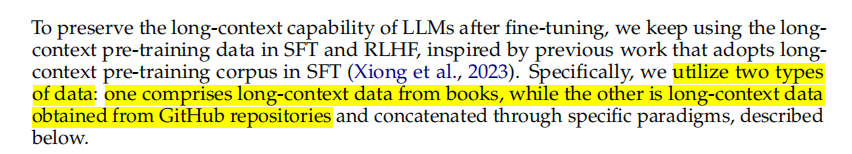
使用critic loss时初始的损失函数值都很高，可能与奖励建模和批评建模的区别有关。



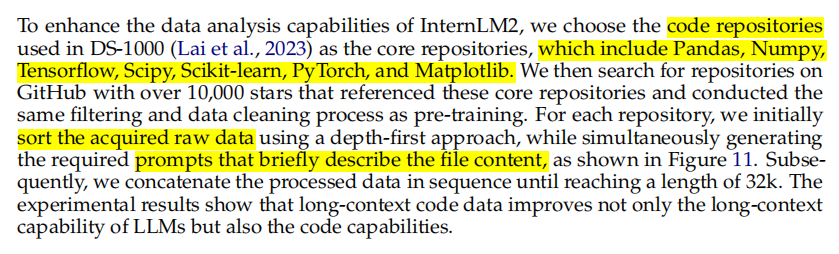


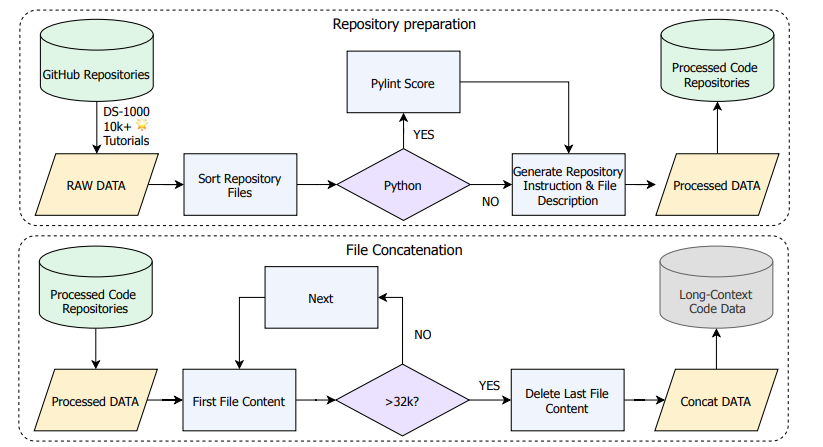
这里的lambda是损失函数里那个？那不是相当于加权和都加在了惩罚函数，而没有排序损失算法的权重上？

## 4.3 Long-Context Finetuning

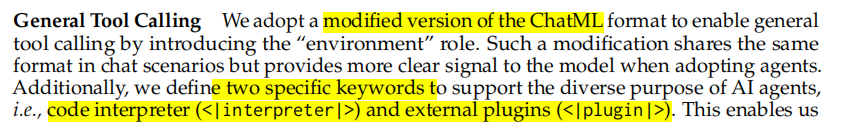


具体来说，我们使用了两种类型的数据：一种包含来自书籍的长上下文数据，而另一种是从GitHub存储库中获得的长上下文数据，并通过特定的范式进行连接，如下所述。

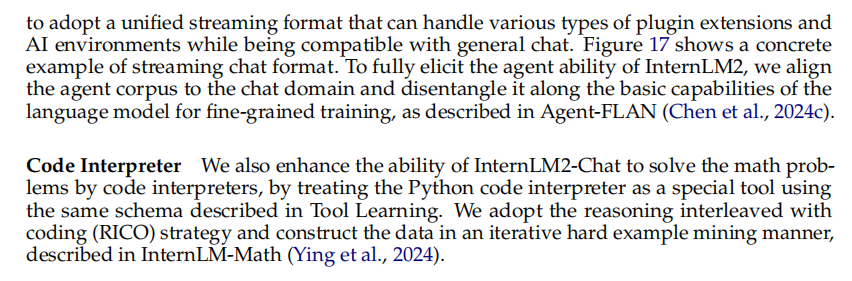


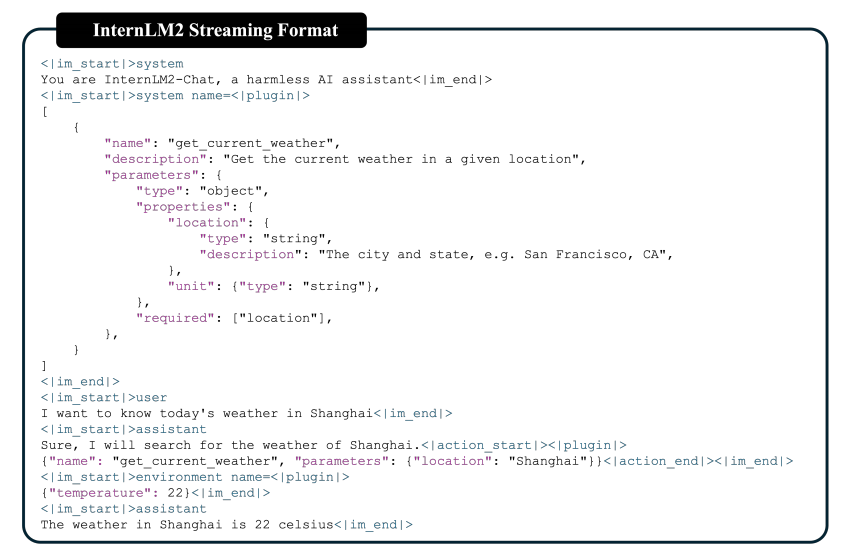


## 4.4 Tool-Augmented LLMs



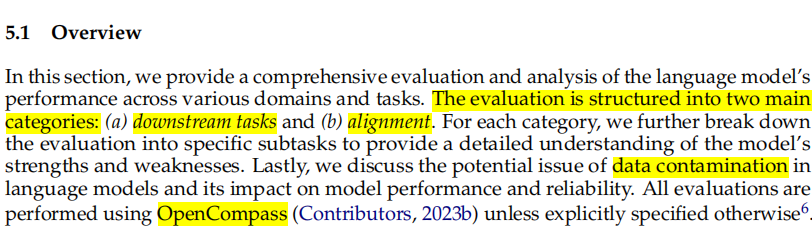
采用两个关键词：代码编译器和外部插件。





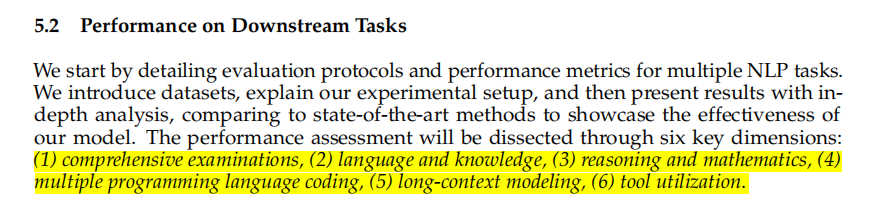
# 5 Evaluation and Analysis

## 5.1 Overview

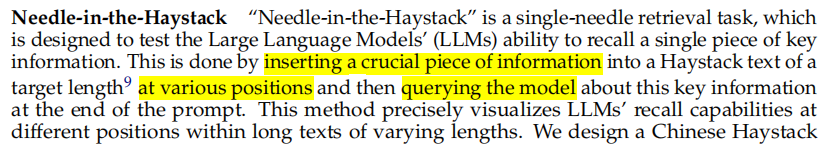


模型评估分为两个种类：下游任务和对齐。存在潜在的数据污染问题。

## 5.2 Performance on Downstream Tasks



总的来说表现评估围绕六个维度，基本都是不同方向的关键能力。



大海捞针测试。将关键信息安插到语料的不同位置中，然后进行搜索和队列

## 5.3 Performance on Alignment

主要是模型校准能力，满足人类需求的能力

# 6 Conclusion

