生成式预训练语言模型在代码管理上的性能分析

汇报人：刘一尘

目录

知识增强的日志理解预训练语言模型

一. 研究问题

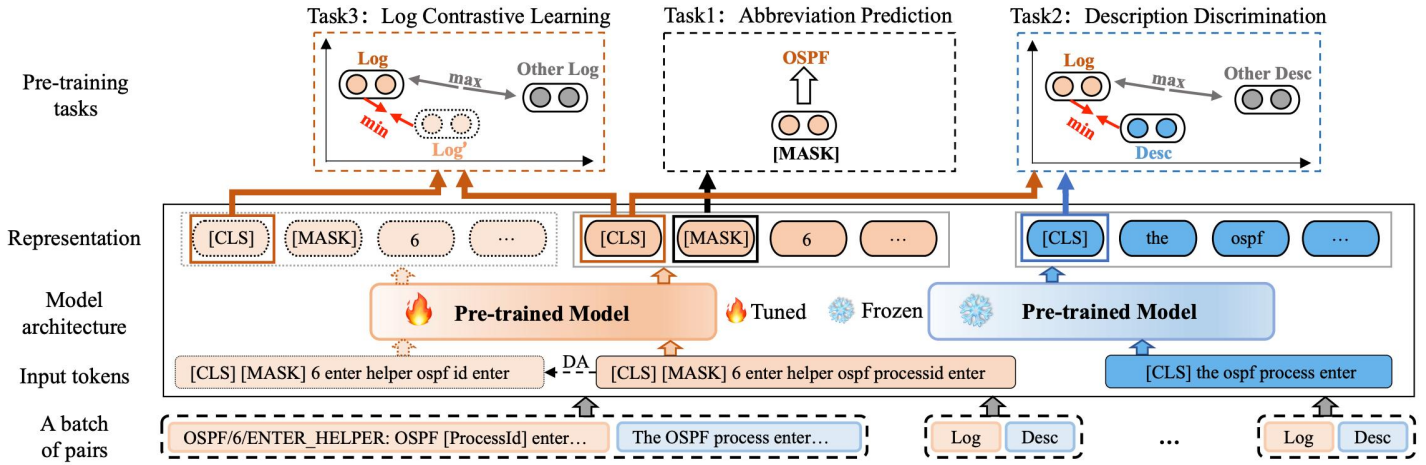
- 提出了知识增强的日志理解预训练语言模型KnowLog，以及三个新的预训练任务，有效地利用领域知识来增强模型效果，使日志的表示更具通用性，其包括缩写预测任务、描述判别任务、日志对比学习任务。
- 在知识增强的预训练之后，KnowLog在六个不同的下游任务上进行了微调。与其他没有知识增强的预训练模型相比，KnowLog达到了最先进的性能。

二. 研究现状

- HitAnomaly、SwissLog利用预训练模型进行基于日志的异常检测；
- NuLog、Logstamp被提出用于对预训练模型进行微调以实现日志解析；
- Translog和LogEncoder也基于日志异常检测进行了微调。
- 然而，现有的方法仍然存在三个弱点：首先，这些模型不能理解特定于领域的术语，尤其是缩写；其次，这些模型难以充分捕获完整的日志上下文信息；第三，这些模型难以获得同一日志的不同样式的通用表示。

研究方法

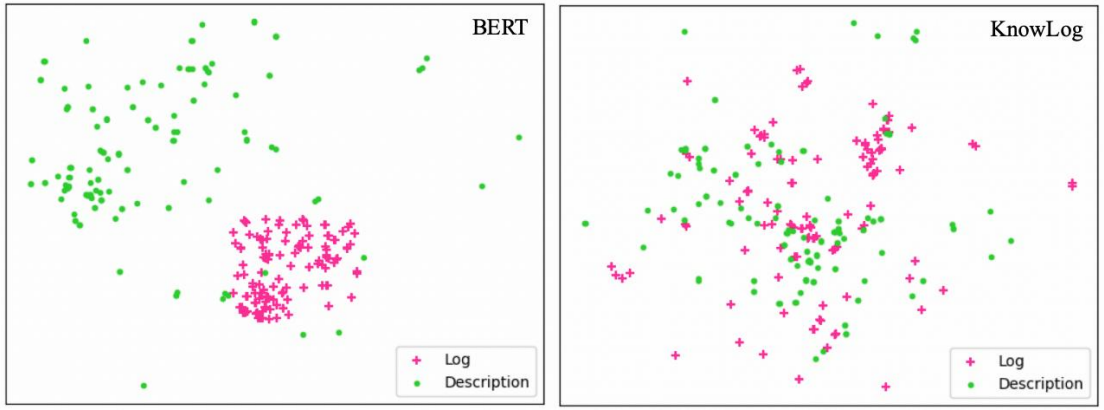
- 使用日志作为预训练语料库，并将日志的缩写和自然语言描述分别作为局部知识和全局知识，设计了三个新的预训练任务来有效地利用领域知识来提高对日志的理解。
- 提出了缩写预测任务，利用缩写信息作为局部知识，使模型能够准确理解这些领域缩写；
- 提出了描述判别任务，通过利用文档中的自然语言描述知识作为全局知识来补充日志的底层知识并充分捕获完整的日志上下文信息；
- 提出了日志对比学习任务，以捕获日志中的共性，并使日志的表示更加通用。



评估

- KnowLog能够更好地工作有两个原因。
- 首先，KnowLog对日志进行了预先训练，并对不同的任务进行了微调。
- 其次，与现有的预训练模型相比，KnowLog分别利用局部和全局知识来增强预训练。
- 相比之下，KnowLog能够将日志和相应的描述拉得更近，使模型能够全面理解日志。

任务	作用	数据集	指标
模块分类	用于识别日志所属的模块。该任务的输入是一条模块名被屏蔽的日志，输出是对应的模块名。	使用日志模板作为原始数据，日志中的模块名称作为ground truth，然后将模块名称替换为[MASK]	Accuracy和Weighted F1-score
风险日志识别	以确定日志是否对系统有影响为目标，一种二分类任务	从文档中解析“处理步骤”这个字段，并根据它是否包含明确的操作步骤来构造基本事实。具体来说，带有操作步骤的日志为True，没有操作步骤的日志为False。	Precision、Recall和F1-score
故障现象识别	识别日志所属的故障现象，该任务基于实际数据。	从实际场景中收集了602个华为交换机的日志实例作为数据集，这些日志经过专家标注，共涵盖43种故障现象。	Hamming-score
日志和描述的语义匹配	二分类问题，用于度量日志的语义l是否与相应的自然语言描述相匹配d	从文档中获得日志的描述，然后将日志与其对应的描述标记为True，并随机选择另一个日志标记为False的描述，其中True与False的比例为1:1。	Accuracy and Weighed F1-score
日志和可能原因排序	是一种日志对类型的排序任务，用于根据给定日志的语义上下文确定其最可能的潜在原因。	文档中的每条日志都包含对可能原因的介绍，我们将此字段解析为基础事实。将ground truth和随机选择的5个可能原因的其他日志作为候选集，并要求模型对候选集进行排序。	Precision@1和平均倒数秩(MRR)，Precision@1表示第一名的精度
厂商间模块匹配	一种日志对类型的二进制任务，用于跨供应商对齐相同的模块，它可以度量日志的语义表示是否更通用。	根据模块名称是否相同来构建不同厂商日志之间的ground truth，并随机抽取不同模块的日志作为负样本，其中True与False的比值为1:1。并且将我们将日志中的模块名称替换为[MASK]令牌，以避免训练过程中的标签泄漏。	Accuracyand and Weighed F1-score



使用生成预训练模型生成实用代码的测试

一. 研究问题

- 现有的代码评估基准与实际的代码生成场景之间存在差距。一方面，这些基准测试主要关注独立函数。另一方面，非独立函数在实际的代码生成场景中普遍存在。
- 为了弥补上述问题，本文作者提出了CoderEval，一个上下文感知的基准测试，它可以用于评估在实用代码生成方面的代码生成模型。
- 同时，作者通过在该测试模型上对现有生成式预训练语言模型依照在大型语言模型生成正确上下文信息的能力与生成正确代码的能力（Pass@k）标准进行定量分析测试，得出了有建设性的结论。

二. 研究现状

- 随着Codex的发布，HumanEval成为了一个用于评估代码生成模型生成的Python程序功能正确性的基准测试。
- HumanEval包含164个手写问题，每个问题包括一个函数签名、一个文档字符串、一个标准的参考函数以及多个单元测试。
- DS-1000被提出用于评估代码生成模型在生成依赖第三方数据科学库的代码方面的有效性。
- 除了Python，还有针对Java和其他编程语言的基准测试。

研究方法

- CoderEval的构建包括三个步骤：数据集收集、数据集探索和评估过程。
- 抓取GitHub上所有项目的标签来选择候选项目，挑选出14个标签中出现频率最高且星级较高的项目。
- 提取所选项目中的所有函数，只保留不属于测试、接口或已废弃函数的函数，这些函数要满足一系列要求；其次，通过人工筛选从候选函数中选出高质量函数。
- 对于行覆盖率或分支覆盖率未达到100%的测试，作者手动编写了额外的测试用例，以实现函数更高的行覆盖率和分支覆盖率。
- 要求13名开发人员评判的在实际开发场景中经常使用的函数。开发人员偏好函数的多样性可以消除潜在的偏差。

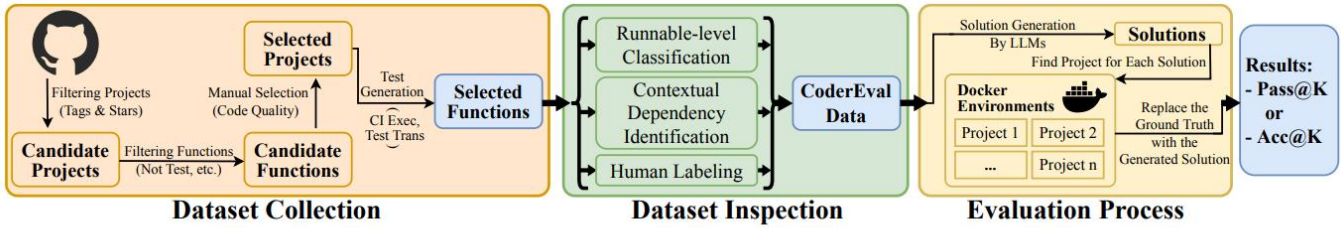


Figure 1: Overview of CoderEval construction process

评估

- 在CoderEval for Python和CoderEval for Java上，三种模型生成独立函数的效率都大大高于生成非独立函数的效率。不同的模型在生成代码方面有其独特的能力。

- 大型语言模型生成正确上下文信息的能力（Acc@k）与生成正确代码的能力（Pass@k）高度相关。大语言模型生成不同类型上下文标记的能力因语言而异。

Table 3: Overall effectiveness of three models on two benchmarks

Benchmark	Model	Python			Java		
		Pass@1	Pass@5	Pass@10	Pass@1	Pass@5	Pass@10
CoderEval	CodeGen[1]	9.48%	19.58%	23.48%	13.91%	27.34%	33.48%
	PanGu-Coder[2]	11.83%	20.93%	27.39%	25.43%	37.39%	43.04%
	ChatGPT[3]	21.04%	27.31%	30.00%	35.39%	42.77%	46.09%
HumanEval	CodeGen[1]	10.2%	19.79%	22.8%	5.78%	9.0%	9.45%
	PanGu-Coder[2]	13.42%	21.48%	22.73%	8.21%	15.88%	18.63%
	ChatGPT[3]	39.21%	64.09%	72.96%	38.21%	59.35%	67.23%

[1] We use the 350M CodeGen-Mono model with the default settings for Python. Since CodeGen does not have a monolingual version of Java, on CoderEval for Java, we use the CodeGen-Multi model instead.
[2] PanGu-Coder has two different models for Python and Java, we use the 300M models with the default settings for Python and Java.
[3] We use the "gpt-3.5-turbo" for ChatGPT in our experiments.

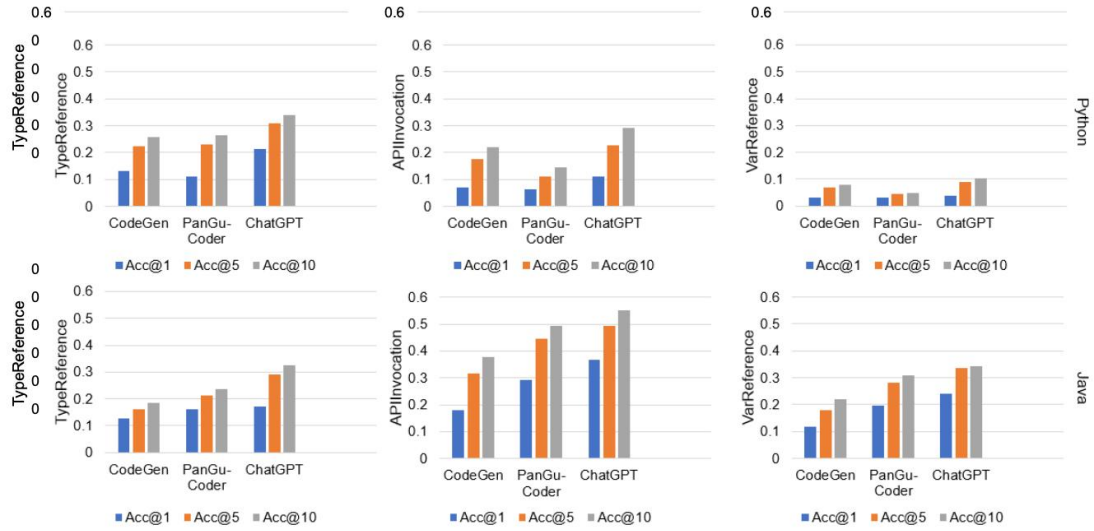


Figure 4: The accuracy of three models that they can correctly generate the contextual tokens

代码生成基准数据集研究

一. 研究问题

- 提出了一个基准数据集，用于评估代码生成模型。
- 以自然语言为输入，以代码为输出。基准数据集包括175个样本用于自动化评估，161个样本用于手动评估。
- 同时提出了一个新的自动化评估生成代码正确性的指标，以及一套用于手动评估生成代码整体质量的标准。

二. 研究现状

- 现有指标局限性：传统指标如精确匹配、BLEU或困惑度不适用于方法级代码生成的正确性评估。
- 本文对比了HumanEval、APPS、PandasEval等数据集，并指出了它们的局限性：

（1）HumanEval数据集是纯Python的；

（2）主要是关于纯算法和字符串操作，它们只是现实世界问题的一小部分。

（3）HumanEval中的提示在“文本代码”交互场景中与人类编写的有较大误差等。

- 可以发现目前缺乏包含Java代码，且自然语言描述部分包含英语和中文的自动化测试数据集和NL任务描述数据集。

研究方法

- 提出了一个用于自动评估代码生成模型生成代码正确性和手动评估整体质量的基准数据集。
- 为代码制定了正确性、代码质量、可维护性三大测试标准


```
{
    "raw_nl": "Close Reader. If object is null it is ignored",
    "signature": "public static void close(Reader reader)"
}
{
    "raw_nl": "max() that works on three integers",
    "signature": "public static float max(float a, float b, float c) "
}
{
    "raw_nl": "将 Date 类型转为时间字符串，格式为 format",
    "signature": "public static String date2String(final Date date, final DateFormat format)"
}
{
    "raw_nl": "获取类上具有指定注解的接口的名称，如果有多个，则以第一个为准 找不到符合条件的接口则返回 clazz 类的名称",
    "signature": "public static String getInterfaceName(Class<?> clazz, Class<? extends Annotation> annotation)"
}
```

```
{
    "raw_nl": "return the last day of the date's month of specified string value in format: yyyy-MM"
}
{
    "raw_nl": "transform a string to a valid classname string"
}
{
    "raw_nl": "从 http 服务拉取并解析 Properties 文件"
}
{
    "raw_nl": "创建简单颜色选择板"
}
```

评估

- 使用pass@1和AvgPassRatio来评估代码生成模型。

- 对aiXcoder XL和GitHub Copilot进行了评估，并比较了它们在自动化测试和手动评估上的表现。

Metrics	aiXcoder XL	Copilot
pass@1	86	81
	(49.14%)	(46.29%)
AvgPassRatio	120.1979	121.7152
	(68.68%)	(69.55%)

Table 2: Automatic Comparison on Correctness over 175 samples

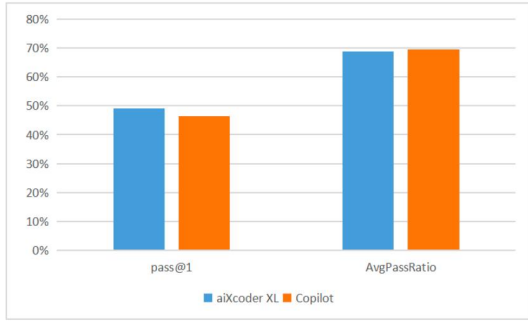


Figure 3: Automatic Comparison on Correctness

Metrics	aiXcoder XL	Copilot
Correctness	2.9503	2.9875
Code Quality	2.2049	2.1739
Mantainability	2.9937	3.0931

Table 3: Manual Comparison between aiXcoder XL and Copilot on NL Task Description Dataset

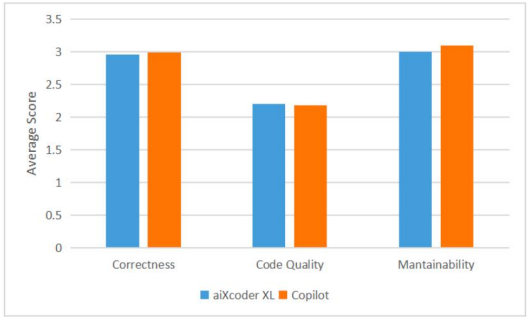


Figure 4: Manual Comparison between aiXcoder XL and Copilot on NL Task Description Dataset

感谢指导