大模型代码生成能力的研究

摘要

摘要 本文研究的是大模型的代码生成能力，在本文中所使用的数据集是 HumanEval 数据集，采用的研究方法包括设计 prompt 的格式对大模型的代码生成能力的影响，实验结果表明，prompt 的格式对代码生成的最终效果会有很大的影响，做代码生成有关的任务时，需要选择一个合适的 prompt 格式。

关键词： 大模型；代码生成；自然语言处理；

1 引言

大模型（LLM）在自然语言处理领域指的是基于深度学习算法训练的自然语言处理（NLP）模型，主要用于自然语言生成和自然语言理解等方面，随着大模型的发展，大模型现在也包括视觉 (CV) 大模型，多模态大模型和科学计算大模等。大模型是一种比较新型的技术，正在经历快速的发展和迭代过程，大模型的发展方面涵盖了各种参数规模、主流的技术架构、多种模态和各种各样的应用场景。从大模型的参数规模的发展来看，大模型的发展主要经历了预训练模型、大规模预训练模型、到超大规模预训练模型的三个阶段。每年，随着硬件技术的不断发展，大模型所能处理的数据规模也逐渐变大，参数规模至少提升 10 倍，实现了从亿级到百万亿级的巨大突破。在技术架构方面，Transformer 架构是当前大模型领域的主流算法基础，具体如图一所示。Transformer 架构主要用于解决序列到序列的问题 (Sequence- to-Sequence) 任务，该结构摒弃了传统的卷积神经网络结构 (CNN) 和循环神经网络 (RNN) 结构，而采用注意力机制进行代替，从而达到了减少计算量和提高并行效率的作用 [9]。大型语言模型通常通过在大规模文本数据上进行预训练来学习语言的表示，经过对应的预训练的过程，模型的能力更加突出。GPT 模型是一种自监督学习模型，通过在输入序列中掩盖一些标记，模型被要求预测这些被掩盖的标记 [7]。预训练之后，模型可以通过有监督学习在特定任务上进行微调。这有助于定制模型以执行更具体的任务，例如代码生成。这些模型的性能受益于大规模的训练数据。对于代码生成任务，模型通常在包含数百万到数十亿行代码的代码库上进行训练，以涵盖广泛的编码样式和模式。有些模型采用多任务学习的方法，通过同时在多个任务上进行训练，使得模型能够学到更通用的表示。这也有助于提高代码生成的性能，因为模型能够从不同的角度学习代码的表示。对于生成任务，解码策略是一个重要的考虑因素。通常使用的策略包括贪婪解码、束搜索（beam search）以及采样等，这些都可以影响生成结果的质量和多样性。尽管大型语言模型在很多生成任务上表现出色，但它们也有一些限制，例如可能生成不合理的结果或过于依赖训练数据的模式。一些方法正在研究中，以改进这些问题，例如使用对抗性训练、强化学习等。总的来说，大型语言模型的代码生成能力是通过采用先

进的模型架构、大规模数据集、预训练和微调等技术的结合来实现的。这些模型在多个任务上表现出色，包括自然语言处理和编程领域的代码生成。

2 相关工作

从下游任务的角度来看代码生成任务问题，代码生成任务可以对应为为类似于机器翻译的问题，即将自然语言描述对应的问题翻译为代码表示。而在机器翻译问题中，序列到序列模型（Sequence-to-Sequence Model）通常是最常用的模型，其核心思想是从训练数据中学习自然语言特征，然后利用这些特征生成相应的代码。为了实现这一过程，需要大量的自然语言-代码对以便模型能够学到双模态数据的对应关系。因此，这种方法通常被称为基于监督学习的代码生成方法。

2.1 基于代码特征的代码生成方法

基于代码特征的代码生成方法是一种利用源代码中的结构和语法信息来自动生成代码的技术。这种方法通常使用机器学习或其他自动化技术，通过学习输入代码的特征，生成与之类似的代码片段。以下是基于代码特征的代码生成方法的一般介绍：特征提取：该方法的核心是从源代码中提取关键的特征。这些特征可以包括语法树、抽象语法树（AST）、代码块、变量使用情况、函数调用关系等。特征的选择取决于具体的代码生成任务。数据集准备：在使用机器学习的情况下，需要一个训练数据集，其中包含输入代码和相应的目标生成代码。这些数据集需要经过预处理，将源代码转化为机器学习算法可以理解的格式，通常是将源代码表示为特征向量。模型选择：选择适当的机器学习或深度学习模型来学习源代码特征与目标代码之间的映射关系。常见的模型包括循环神经网络（RNN）、长短时记忆网络（LSTM）、变分自编码器（VAE）等，取决于任务的性质 [5]。模型训练：用准备好的训练数据集，对选定的模型进行训练。在这个过程中，模型会学习输入代码的特征与生成代码之间的关联。生成代码：训练完成后，模型可以用于生成新的代码。给定一个输入代码片段，模型将学到的映射关系应用于特征，并生成相应的目标代码。调整和优化：生成的代码可能需要进行一些调整和优化，以确保其符合语法规范、风格一致等。这可能需要额外的后处理步骤。基于代码特征的代码生成方法广泛应用于一些自动化编程任务，例如代码补全、函数生成、重构等。这种方法的优势在于能够利用源代码的结构信息，但也受到输入特征选择和模型训练质量的限制。在实际应用中，需要仔细选择特征、调整模型参数，以及考虑生成代码的质量和可维护性。尚未有研究发现预训练模型性能的瓶颈，采用这种方式的。与其他领域的任务一样，基于预训练的代码生成方法显著提高了代码生成任务的下限，并逐渐成为近几年研究代码生成问题的主要解决方案。

2.2 基于模板编排的方法

基于模板编排生成代码的过程是一种常见的代码生成技术，它通过使用预定义的模板和相应的数据模型来自动生成代码。这种方法的目标是提高开发效率，减少手动编码的工作量，并确保生成的代码符合特定的标准和约定。以下是基于模板编排生成代码的详细过程，定义数据模型：首先，需要定义数据模型，它表示要生成代码的目标结构和信息。数据模型可以是一个抽象的表示，包括类、属性、方法等，具体取决于生成的代码类型。选择或创建模板：

开发人员选择或创建适用于生成代码的模板。模板是包含占位符和逻辑的文本文件，其中的占位符表示将被实际数据替换的位置，而逻辑部分定义了生成代码的结构和流程。模板引擎：使用模板引擎来处理模板。模板引擎是一种工具，负责解析模板文件，识别占位符，并将其替换为相应的数据。常见的模板引擎包括 Jinja2、Velocity、FreeMarker 等。准备生成配置：为生成过程提供必要的配置信息，包括数据模型、模板文件、输出目录等。这些配置通常以配置文件或命令行参数的形式提供。执行代码生成：调用代码生成工具，该工具利用模板引擎和数据模型执行代码生成。在这个过程中，模板引擎根据模板和数据模型的组合生成最终的代码。代码输出：生成的代码被输出到指定的目录或文件中。这可能是一个或多个源代码文件，具体取决于模板和数据模型的结构和生成规则。代码质量和格式化：一些代码生成工具提供额外的功能，例如代码质量检查和格式化。这有助于确保生成的代码符合团队或项目的编码标准，并且是可读的。集成到开发流程：将代码生成过程集成到整个开发流程中。这可能包括构建系统、持续集成工具或自定义脚本，以便在需要时自动触发代码生成。维护和更新：随着需求的变化或新功能的添加，需要更新数据模型或模板。维护代码生成系统确保生成的代码保持最新，并反映了最新的设计和需求。

3 本文方法

随着大语言模型的不断发展，大模型在各种下游任务的表现也越来越好，这些大模型在对话任务 [6]、代码生成 [8]、总结话语 [10] 的分析等方面已经有了很好的表现。这些大模型 (LLM) 已经显示出显著的任务适应性和泛化能力，使得越来越多的语言任务得以解决。在众多下游任务中，代码生成的意义是根据自然语言需求自动生成源代码，代码理解的意义是指大模型能够理解代码所代表的具体含义，代码生成和代码理解是 large language 模增强代码质量和参与软件开发过程。但是，由于代码的复杂性和问题的多样性，大模型产生的代码还需要进一步完善，这使得大模型在编程题目的理解上还需要进一步的提升，使得这些任务仍然具有挑战性，与生成自然语言相比，生成对应的代码更加强调的是结构性和一致性。传统的验证大型语言模型的代码生成的能力的数据集有 HumanEval [2], MBPP [1], and APPS [3] 等多个数据集，这些数据集能够在一定层面检验出大语言模型在代码填充方面的能力。

3.1 本文方法概述

在本次对大模型的代码生成能力的探究选择的数据集是 Humaneval 数据集。首先介绍一下 Humaneval 数据集，是一个 Python 函数级代码生成基准，包含 164 个手写编程问题。每个编程问题由一个英文需求、一个函数名和几个测试用例组成，每个问题平均有 7.7 个测试用例，具体格式如下：有 task_id 表明题目的题号，比如 Humaneval/1,prompt 则用来表示输入的英文题目要求，输入的是 prompt 中的内容，要求大模型根据 prompt 中的问题和对应的提示来补全代码。所对应的评估方法是数据集是通过输入一个编程问题，让大模型对编程问题生成对应的代码解决方式，或者对缺失的代码进行填充。得到的代码解决方式对其进行数据处理后，筛选出只有代码的那部分，然后对其进行测试，如果能够通过对应的测试样例，则表示大模型产生的代码能够解决该编程问题，具体题目格式如图 1 所示。

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

图 1. Humaneval_ 数据集的具体内容

3.2 评价指标

在此次实验中选取的评价指标关于代码生成的评估指标,早期的评估指标通常使用 BLUE [4] 评估指标。在自然语言处理中，BLUE 通常用于评估生成的语句与原始数据集中的语句之间的相似性。同时，这些评估指标都是为自然语言生成设置的评估指标，但对生成的代码程序进行测量的效果相对较差。生成的代码的质量不能通过原始数据集的解决方案中代码之间的相似性来衡量。评估，所以在本文中，我们在实验中忽略了这些指标。根据最新的代码生成研究，我们使用在课堂上讲过的 pass@k [2] 作为我们的评价指标。具体来说，大型语言模型为每道题生成 k 个程序，并测试这 k 个程序是否能通过相应的测试样本。只要一个程序能够通过测试样本，结果就被证明是有效的，具体公式推导如图 2 所示。

$$\text{pass@}k := \mathop{\mathbb{E}}_{\text{Problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{1}$$

图 2. pass@k 的数学公式

可以针对每个问题生成 n 个代码 (n>k), 假设模型只能生成这 n 个代码，而且他们每一种被生成出来的概率是相等的，其中有 c 个可以通过测试。那么模型任意生成 n 个代码，全都不能通过测试的概率是：生成 k 个不能通过测试的代码的情况总和与生成 k 个代码的全部情况总和之比。pass@k 的无偏估计的证明如图 3 所示，2021 年 Chen 等人 [2] 指出上述方式计算 pass@k 方差比较大, 同时对该评测指标进行了改进: 对每个问题, 在预测时产生 n k 个样本, 统计能够通过单元测试的正确样本的数量 c n, 并且提供了修正后的结果。

```
def pass_at_k(n, c, k):
    """
    :param n: total number of samples
    :param c: number of correct samples
    :param k: k in pass@$k$
    """
    if n - c < k: return 1.0
    return 1.0 - np.prod(1.0 - k /
        np.arange(n - c + 1, n + 1))
```

Figure 3. A numerically stable script for calculating an unbiased estimate of pass@k.

图 3. pass@k 的无偏估计的证明

4 复现细节

4.1 与已有开源代码对比

首先这里使用了开源代码 https://github.com/bigcode-project/bigcode-evaluation-harness，该代码主要用来这是一个用于评估代码生成模型的框架。这项工作的灵感来自 EleutherAI/lm 评估工具，用于评估一般的语言模型，这里最主要的是利用了该代码对该 Humaneval 数据集进行评测，改进的地方主要是在数据集的 Prompt 格式上面，通过比较不同的 Prompt 格式，分析评测结果是否发生了改变，在这里我们测试了改进后的数据集的结果，工作量主要在数据集的格式的探究和针对不同的大模型做实验上。

4.2 实验环境搭建

环境安装的具体步骤如下：

1、将项目下载到服务器上 git clone https://github.com/bigcode-project/bigcode-evaluation-harness.git cd bigcode-evaluation-harness 2、安装特定的 Python 版本 3.10

3、安装 Pytorch 版本 torch==1.12.1

pip install torch==1.12.1+cu116 –extra-index-url https://download.pytorch.org/whl/cu116 https://pytorc started/previous-versions

4、安装其他依赖先跳转到项目文件夹, 再运行一下命令进行环境的包的安装

pip install -e .

5、将对应的 LLAMA 和 CodeLlama 对应的各个版本的模型下载到云服务器上

6、使用服务器硬件加速器 A100(GPU) 进行复现工作

4.3 界面分析与使用说明

项目的运行命令

accelerate launch main.py

–model <MODEL_NAME>

–tasks <TASK_NAME>

–limit <NUMBER_PROBLEMS>

–max_length_generation <MAX_LENGTH>

–temperature <TEMPERATURE>

–do_sample True –n_samples 100 –batch_size 10 –precision <PRECISION> –allow_code_execution –save_generations 各个参数的具体介绍 1、limit 表示要解决的问题数，如果未提供，则选择基准中的所有问题。

2、allow_code_execution 用于执行生成的代码：默认关闭，在调用它之前阅读显示的警告以启用执行。

3、–trust_remote_code–use_auth_token 某些在 HF 集线器上具有自定义代码的模型（如 SantaCoder）需要调用，对于私有模型，请添加.

4、save_generations 将后处理的生成保存在 JSON 文件中（默认情况下）。

5、ave_generations_pathgenerations.json 您还可以通过调用确定保存结果的路径。

6、max_length_generation 是生成的最大令牌长度，包括输入令牌长度。默认值为 512，但对于某些任务（如 GSM8K 和 GSM-Hard），带有 8 个镜头示例的完整提示（如 PAL 中使用）会占用令牌，因此该值应大于该值，建议的值为用于这些任务。

5 实验结果分析

在本次实验中选择了 LLAMA2 和 CodeLLAMA 等开源模型，并对每个模型的各个版本进行了实验，由于 CodeLLAMA 是在代码数据上经过训练过的模型，所以效果会比 LLAMA 的效果更好。

原论文研究了 temperature 参数对 pass@k 性能的影响。具体来说，我们使用 temperature 0.1,0.4,0.6,0.8 在 HumanEval 和 MBPP 上报告 pass@1, pass@10 和 pass@100。正如所料，随着温度的升高，pass@1 的分数越来越差，而 pass@10 和 pass@100 的分数却在提高所以在此次复现实验参数设计上，本文选择 temperature 的值为 0.8 时用来测模型的 pass@10 和 pass@100 的值，使用 temperature 的值为 0.2 时用来测模型的 pass@1 的值，而在这两种情况下 n_samples 的值为 100。

实验结果表明，当对原始数据集的 promp 进行处理时，以及在该 prompt 格式下，对于大模型理解代码生成任务还是有帮助的，根据实验结果可知，pass@1 实验结果变好了，但同时 pass@10 和 pass@100 的实验结果变差了，这表明虽然使用 prompt 方法可以在一定方面使得模型代码生成的能力变好，但是同时也会在一定方面使得计算复杂性高，prompt 的格式对结果也会有一定的影响。

Model	Pass@_1	Pass@_10	Pass@_100
LLAMA2_7b	0.137 Paper：0.122	0.263 ：0.252	0.451 ：0.444
LLAMA2_13b	0,21 Paper:0.20	0.32 :**0.348**	0.60 :**0.61**
CODE LLAMA_7b	0.304 Paper：**0.335**	0.585 ：**0.596**	0.847 ：0.859
CODE LLAMA _Python_7b	0.390 Paper:0.384	0.68 :**0.703**	0.872 :0.906
CODE LLAMA _Instruct_7b	0.329 Paper:0.348	0.614 :0.643	0.871 :**0.881**
CODE LLAMA_13b	0.378 Paper:0.360	0.667 :**0.694**	0.896 :**0.898**
CODE LLAMA _Python_13b	0.451 Paper:0.433	0.761 :**0.774**	0.926 :**0.941**
CODE LLAMA _Instruct_13b	0.402 Paper:0.427	0.702 :**0.716**	0.920 :0.916

图 4. 实验结果示意 Paper 对应的是原论文里面的实验结果，红色代表复现的没有原论文的结果好，黑色的代表比原论文复现结果好

6 总结与展望

本文首先对大模型进行了有关介绍，同时对序代码生成任务目前国内外最新进展进行了比较详尽的阐述与总结，介绍了代码生成有关的数据集以及代码生成评估指标发展，实验上对选择的 HumanEval 数据集进行了复现，同时设计了 prompt 格式对其进行了探究，结果表明虽然现在大模型的代码生成能力已经很强了，但是还是有比较大的进步空间，未来的工作可以往构造更加好的 prompt 格式对其进行研究，同时也要进一步利用高质量的代码生成类的数据集对其进行微调。

参考文献

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[3] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[4] Pierre Isabelle, Eugene Charniak, and Dekang Lin. Proceedings of the 40th annual meeting of the association for computational linguistics. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[5] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*, 2016.

[6] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[7] OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, and Ilge Akkaya. Gpt-4 technical report, 2023.

[8] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[10] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.