AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

AI驱动软件研发
全面进入数字化时代

中国·深圳 11.24-25

AI+
software
Development
Digital
summit

代码生成遇上大模型：进展与挑战

王焱林 中山大学助理教授

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

KEYLINK ing

K+峰会

K+峰会 深圳站

K+ 全球软件研发行业创新峰会

会议时间：2024.05.24-25

K+峰会 上海站

K+ 全球软件研发行业创新峰会

会议时间：2024.09.20-21

AiDD 峰会

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2023.11.24-25

AiDD峰会 北京站

AI+ 软件研发数字峰会

会议时间：2024.07.19-20

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2024.11.15-16

演讲嘉宾

王焱林

中山大学助理教授，入选“百人计划”

中山大学软件工程学院助理教授、硕士生导师，博士毕业于香港大学，曾任微软亚洲研究院主管研究员。主要研究领域为智能软件工程，尤其是与大模型结合的代码智能，包括代码搜索、代码摘要、代码生成等。已在ICSE、ISSTA、AAAI、ACL等软件工程及人工智能领域的高质量会议和期刊上发表三十余篇学术论文。在多个国际学术会议如ICSE，ISSTA，FSE industry等担任程序委员会委员，是TOSEM，TSE，JSS，EMSE等国际期刊的审稿人。

目录
CONTENTS

PART 01
代码生成简介

代码生成简介

自动代码生成是指根据给定的自然语言描述或部分代码片段，自动生成满足需求的代码。

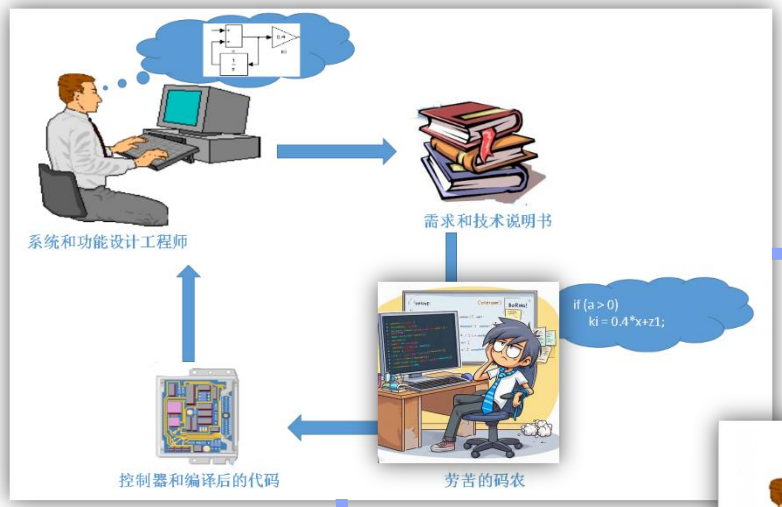
```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```

Copilot

https://github.com/features/copilot

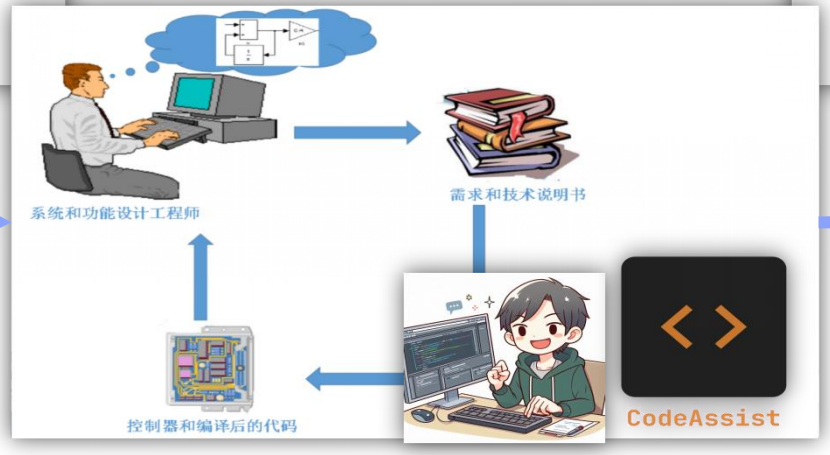
重要性

自动代码生成可以大大提高软件开发和维护效率、简化开发流程、节省大量的人力物力。



现状，时间紧，任务重，
开发者身心俱疲。

自动代码生成技术应用后的开发愿景



有了自动生成代码工具，大大减少开发负担。

最终，机器可以自动完成编码任务。

图片来源 https://www.zhihu.com/question/52838341

代码生成任务概览

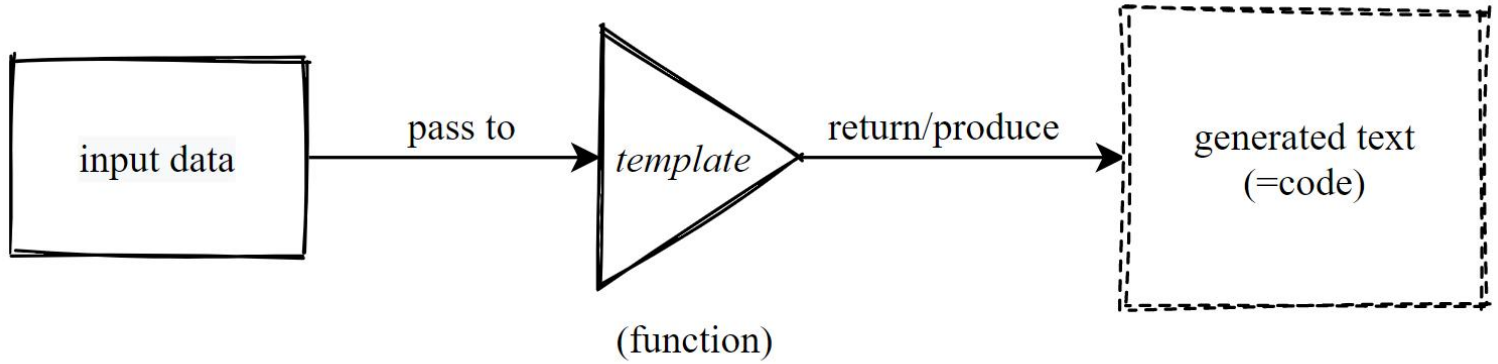
分类	相关工作	输入	输出	评价指标
语句级别代码生成 (statement-level)	[1][2]	自然语言需求、部分代码片段	一行代码	BLEU、EM CodeBLEU、Pass@k
函数级别代码生成 (function-level)	[3][4]		一个函数	
类级别代码生成 (class-level)	[5]		一个类	
仓库级别代码生成 (repo-level)	[6]		整个项目	

[1] Sun, Zeyu, et al. "Treegen: A tree-based transformer architecture for code generation." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. No. 05. 2020.
[2] **Wang, Yanlin**, and Hui Li. "Code completion by modeling flattened abstract syntax trees as graphs." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. No. 16. 2021.
[3] Jiang, Xue, et al. "Self-planning code generation with large language model." arXiv preprint arXiv:2303.06689 (2023).
[4] Huang, Baizhou, et al. "Enhancing Large Language Models in Coding Through Multi-Perspective Self-Consistency." *arXiv preprint arXiv:2309.17272* (2023).
[5] Du, Xueying, et al. "Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation." *arXiv preprint arXiv:2308.01861* (2023).
[6] Liu, Tianyang, Canwen Xu, and Julian McAuley. "RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems." arXiv preprint arXiv:2306.03091 (2023).

PART 02
代码生成技术的历史与现状

经典模型与方法

基于模板的代码生成



简介

- 在代码生成研究早期，主流的技术是基于模板的代码生成方法。这类方法通常基于人工总结的规则和模范构建代码。

优点

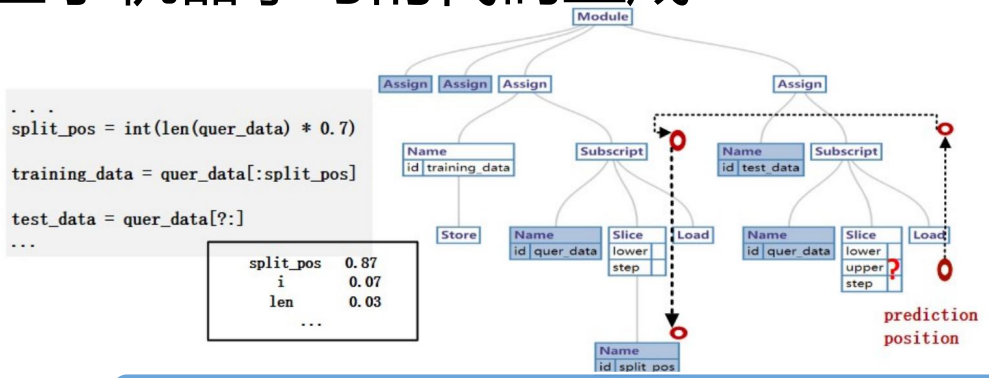
- 此类方法解释性高、生成速度快、对计算资源的要求小。

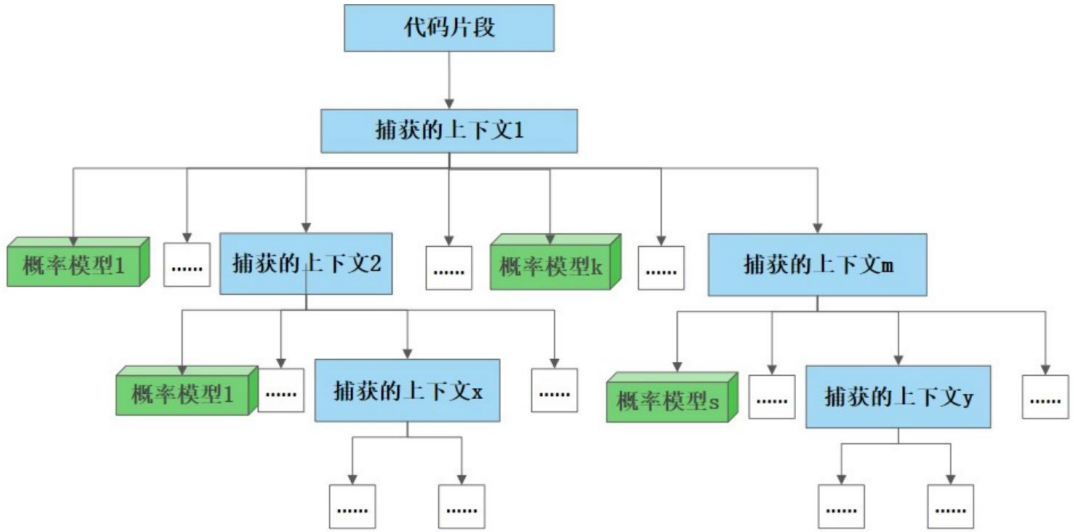
缺点

- 需要大量的人工维护，且生成代码逻辑固定，难以满足实际开发场景中复杂的、快速变化的开发需求。

经典模型与方法

基于机器学习的代码生成





简介

- 结合代码特性，设计算法，从数据中提取特征， 使用统计学或者概率学的手段建模特性、做出预测和做出决策。比较典型的机器学习方法如决策树、支持向量机。

优点

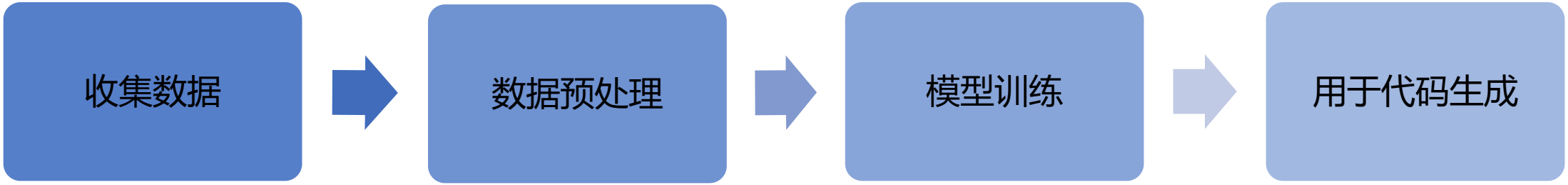
- 可解性高，数据量要求少。

缺点

- 需要进行大量的特征工程、建模的多为浅层特征。

经典模型与方法

基于深度学习的代码生成



简介

- 收集大量的自然语言与代码的数据对，利用神经网络技术自动学习自然语言和代码之间的对应关系，达成代码生成的目的。

优点

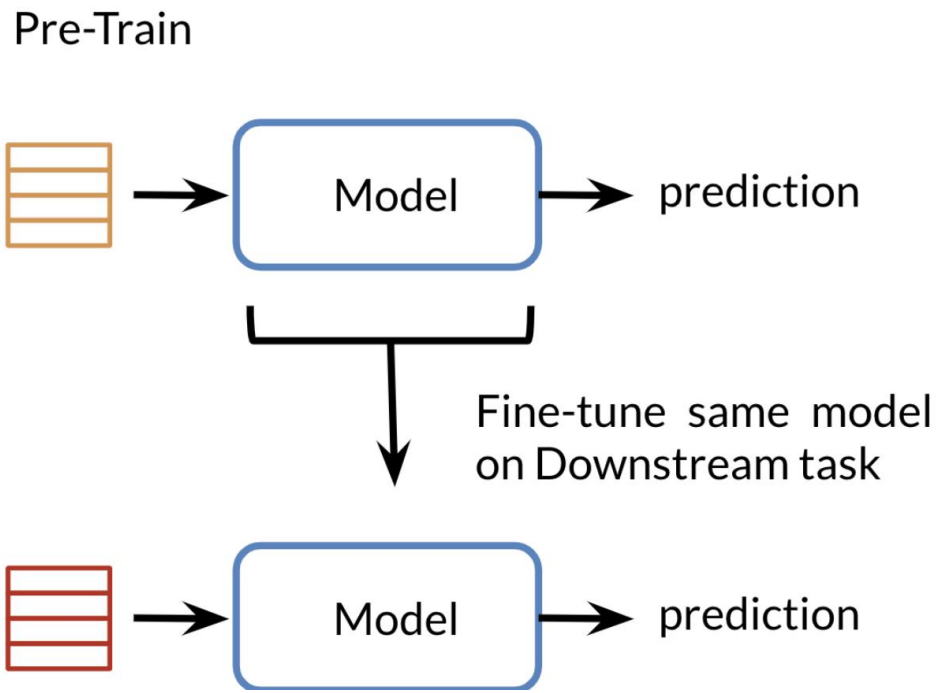
- 可以从海量的数据里自动提取特征。强大的拟合能力，可以逼近任何复杂的函数。

缺点

- 计算量大、模型设计复杂、性能表现易受数据的影响。

经典模型与方法

基于预训练模型的代码生成



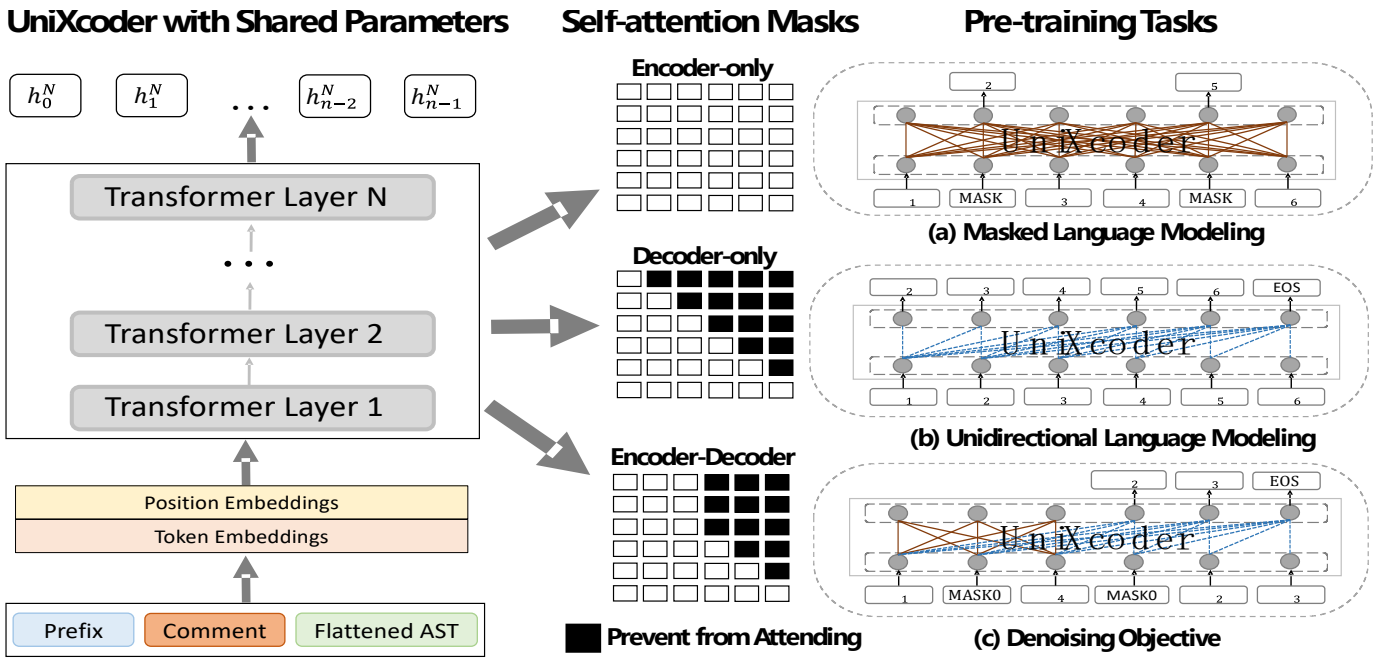
模型名称	模型架构	参数量
CodeBERT[1]	Encoder	125M
GraphCodeBERT[2]	Encoder	125M
CodeGPT[3]	Decoder	124M
CodeT5[4]	Encoder-Decoder	60/223/770M
UniXcoder[5]	Unified	125M
…	…	…

[1] Feng, Zhangyin, et al. "CodeBERT: A Pre-Trained Model for Programming and Natural Languages." Findings of the Association for Computational Linguistics: EMNLP 2020. 2020.
[2] Guo, Daya, et al. "Graphcodebert: Pre-training code representations with data flow." arXiv preprint arXiv:2009.08366 (2020).
[3] Lu, Shuai, et al. "Codexglue: A machine learning benchmark dataset for code understanding and generation." arXiv preprint arXiv:2102.04664 (2021).
[4] Wang, Yue, et al. "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation." arXiv preprint arXiv:2109.00859 (2021).
[5] Guo, Daya, et al. "Unixcoder: Unified cross-modal pre-training for code representation." arXiv preprint arXiv:2203.03850 (2022).

基于预训练模型的代码生成

Unixcoder： 基于跨模态预训练的统一代码表示 [ACL’22]

- 将代码注释和扁平化 AST 作为输入。通过自注意力掩码和前缀支持各种任务。
- 在代码生成数据上微调预训练参数可支持代码生成任务。



D. Guo, S. Lu, N. Duan, **Y. Wang*** et al., “Unixcoder: Unified cross-modal pre-training for code representation”, ACL, 2022.

大模型时代下的代码生成

代码生成的范式的转变

专家系统时代

- 人工设计大量的模板
- 模板设计耗时耗力
- 模板泛化能力低

机器学习时代

- 代码特性分析
- 特征提取
- 浅层模型

深度学习时代

- 大量的代码生成数据
- 不用模态的代码表征
- 更深的神经网络

预训练模型时代

- 海量的数据预训练一个深度模型
- 预训练的基础模型学习基本的代码知识
- 在少量代码生成数据上进行微调以适配此任务

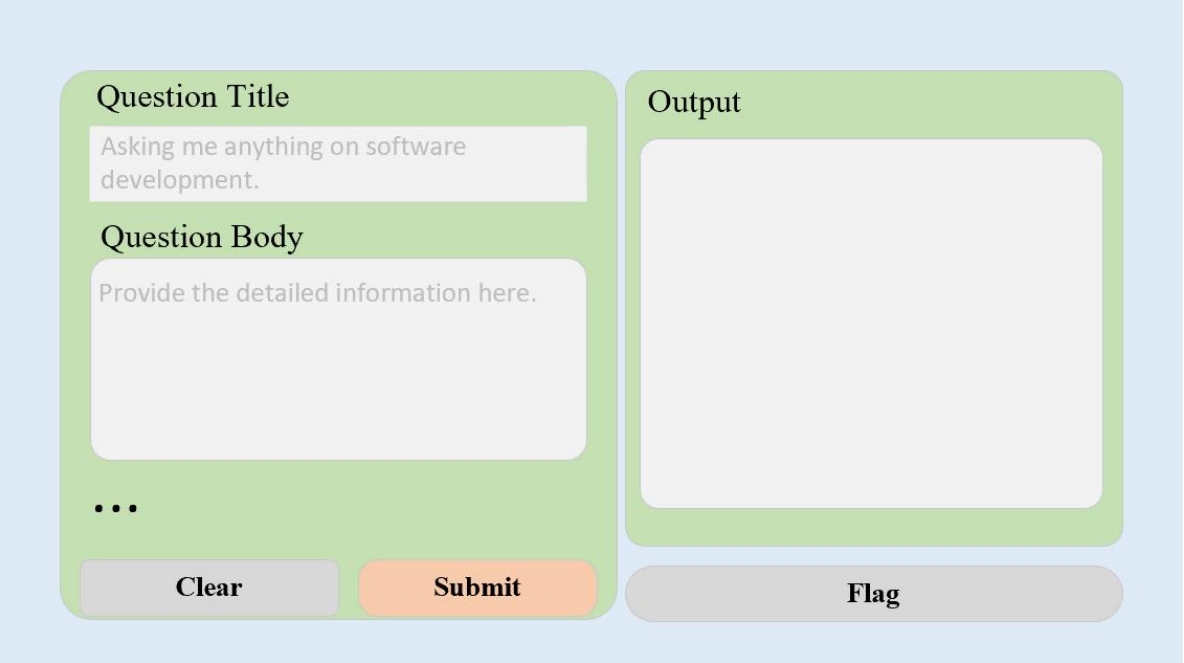
大模型时代

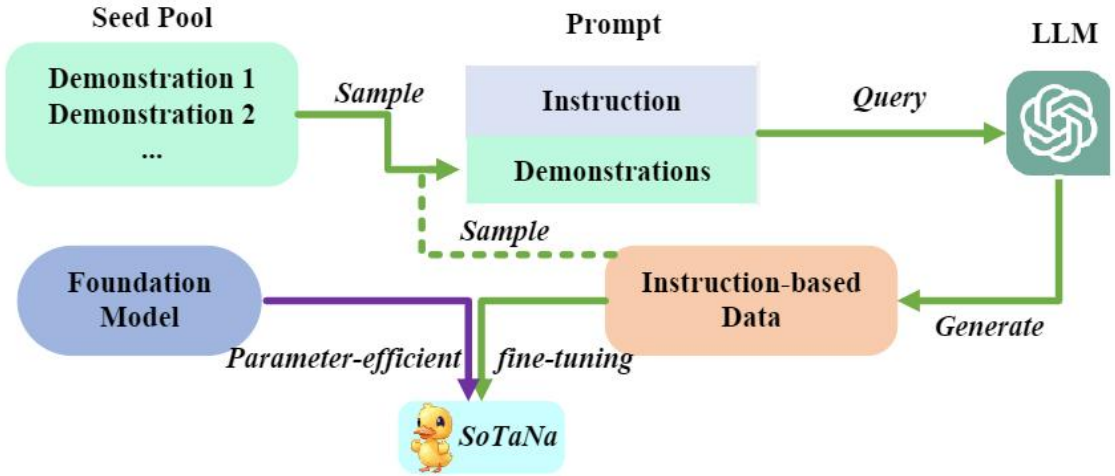
- 海量的数据，百/千亿参数的基座模型
- 学习世界知识和与人交互
- 模型可以直接生成或者补全代码
- 可以根据极少的样本的提示工程可以直接调用基座模型

大模型时代下的代码生成

SoTaNa: 基于大模型的开源编程问答助手

- 使用大模型（ ChatGPT ）生成大量软件工程领域相关的指令数据集。
- 基于开源基座大模型，用参数高效方式做编程领域数据微调训练。
- **实现**了一个开源编程问答助手SoTaNa，可在常见编程问题（如代码生成）上取得较好效果。

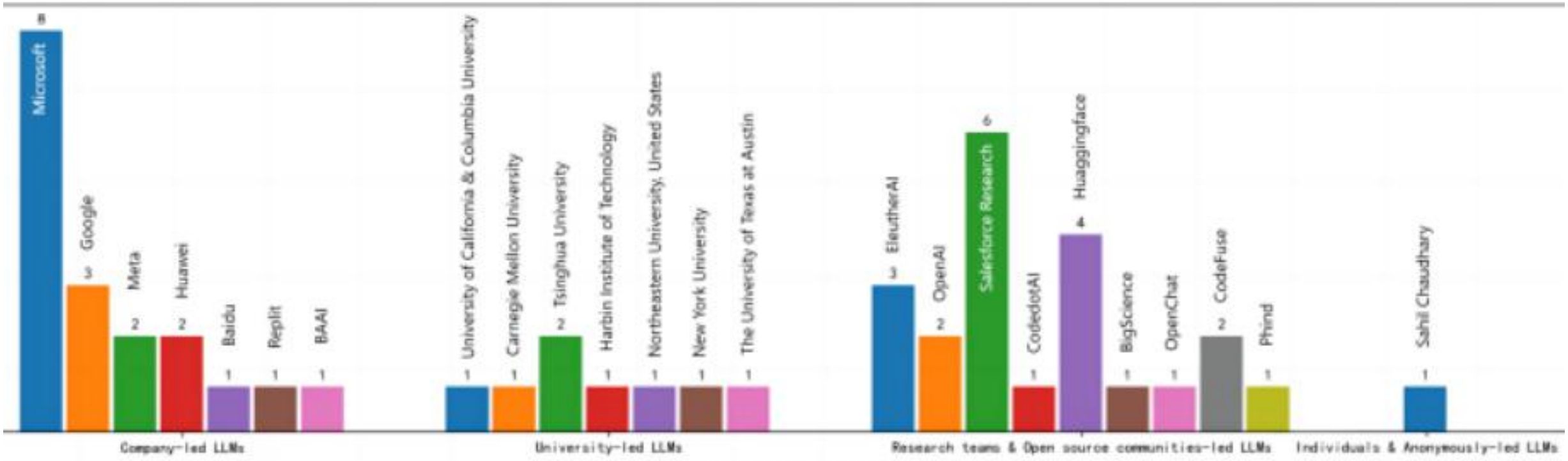




Shi, E, Zhang, F, **Wang, Y***, Chen, B, Du, L., Zhang, H., ... & Sun, H. (2023). SoTaNa: The Open-Source Software Development Assistant. arXiv preprint arXiv:2308.13416.

基于大模型的代码生成

代码大模型：发布机构分布



[1] Zheng Z, Ning K, Chen J, **Wang Y**, et al. (2023) Towards an Understanding of Large Language Models in Software Engineering Tasks. arXiv.
[2] Zheng Z, Ning K, **Wang, Y***, et al. (2023). From General to Specific: The Evolution and Benchmarking of Code LLMs. arXiv.

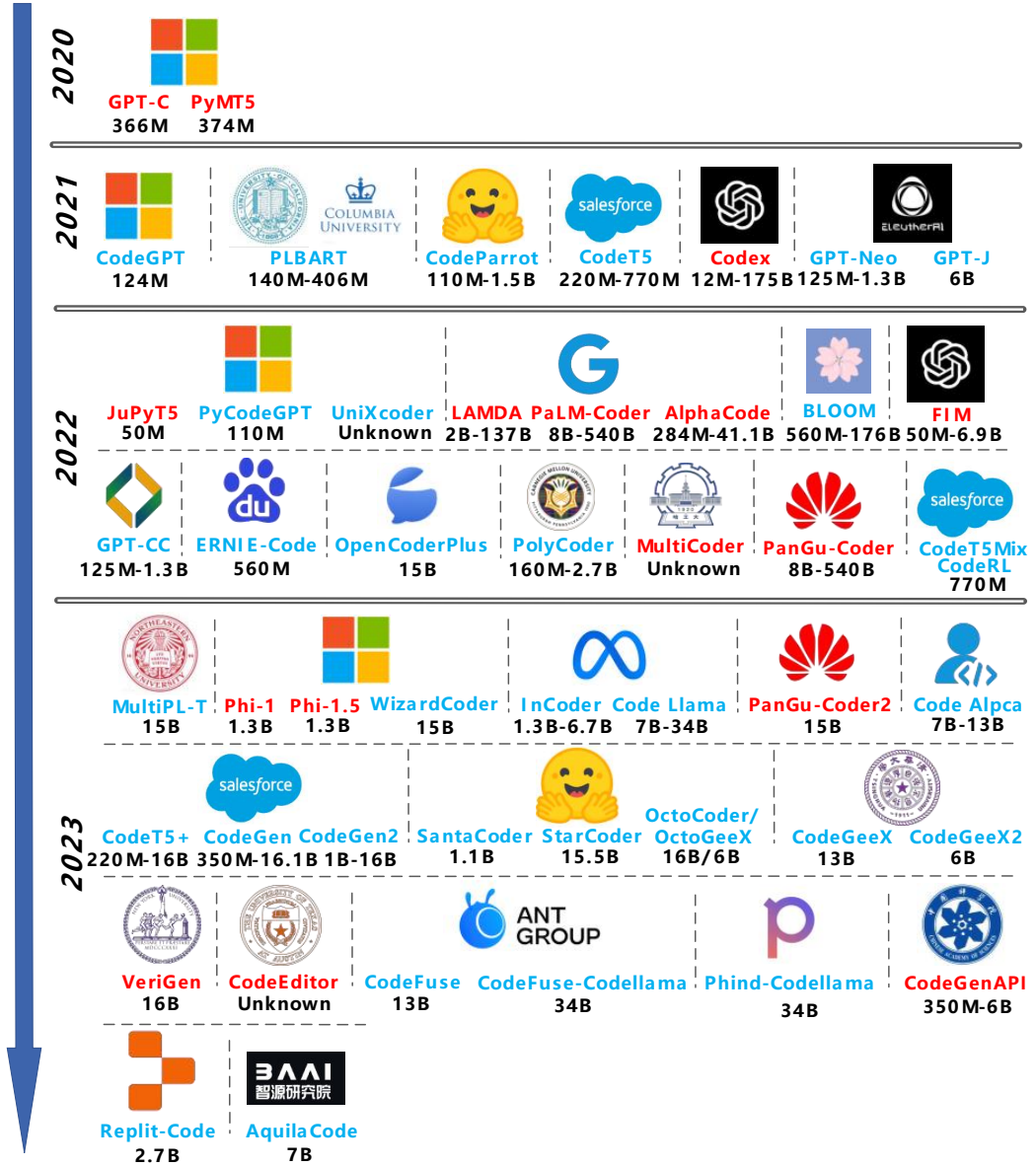
基于大模型的代码生成

代码大模型：演进历史

- 模型大小：越来越大
- 模型数量：越来越多

模型越大越好吗？

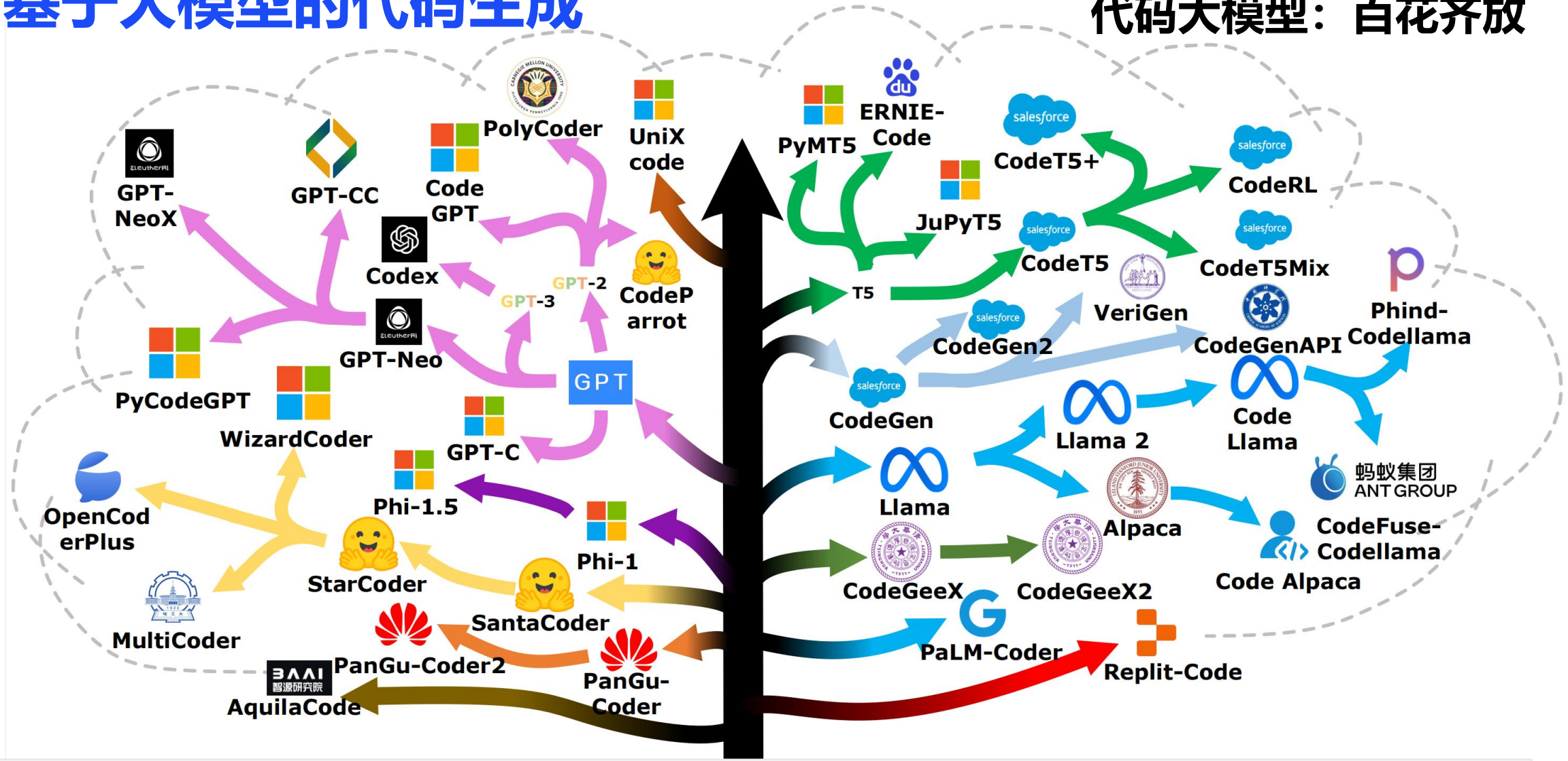
- CodeGen-Mono: 2.7B, 6.1B, 16.1B
- Codex: 2.5B, 12B



[1] Zheng Z, Ning K, Chen J, **Wang Y**, et al. (2023) Towards an Understanding of Large Language Models in Software Engineering Tasks. arXiv.
[2] Zheng Z, Ning K, **Wang, Y***, et al. (2023). From General to Specific: The Evolution and Benchmarking of Code LLMs. arXiv.

基于大模型的代码生成

代码大模型：百花齐放



[1] Zheng Z, Ning K, Chen J, **Wang Y**, et al. (2023) Towards an Understanding of Large Language Models in Software Engineering Tasks. arXiv.
[2] Zheng Z, Ning K, **Wang, Y***, et al. (2023). From General to Specific: The Evolution and Benchmarking of Code LLMs. arXiv.

基于大模型的代码生成

LLMs	HumanEval		
	Pass@1	Pass@10	Pass@100
WizardCoder-Python-34B	**73.2**	-	-
WizardCoder-15B	57.3	73.2	90.46
Unnatural-Code-LLaMA-34B	62.2	**85.2**	**95.4**
StarCoder-Python-15B	33.6	-	-
StarCoder-Prompted-15B	40.8	-	-
StarCoderBase-15B	30.4	-	-
StarCoder-15B	33.60	45.78	79.82
SantaCoder-1.1B	18	29	49
Replit-Finetuned-2.7B	30.5	-	-
Replit-3B	21.9	-	-
Replit-2.7B	21.9	-	-
PyCodeGPT-110M	8.33	13.36	19.13
PolyCoder-400M	2.96	5.29	11.59
PolyCoder-2.7B	5.59	9.84	17.68
PolyCoder-160M	2.13	3.35	4.88
Phind-CodeLlama-Python-34B-v1	69.5	-	-
Phind-CodeLlama-34B-v2	**73.8**	-	-
Phind-CodeLlama-34B-v1	67.6	-	-
Phi-1-1.3B	50.6	-	-
PanGu-Coder-317M	17.07	24.05	34.55
PanGu-Coder2-15B	61.64	**79.55**	91.75
PanGu-Coder-2.6B	23.78	35.36	51.24
PaLM-Coder-540B	36.0	-	88.4
PaLM-8B	3.6	-	18.7
PaLM-62B	15.9	-	46.3
PaLM-540B	26.2	-	76.2
PaLM-2-S	37.6	-	88.4
OctoCoder	46.2	-	-
MIM-2.7B	30.7	48.22	69.6
MIM-1.3B	22.4	41.7	53.8
LLaMA-7B	10.5	-	36.5

LLMs	HumanEval		
	Pass@1	Pass@10	Pass@100
Codex-25M	3.21	7.1	12.89
Codex-2.5B	21.36	35.42	59.50
Codex-12M	2.00	3.62	8.58
Codex-12B	28.81	46.81	72.31
CodeT5Plus-16B-mono	30.9	51.6	76.7
CodeT5-770M	12.09	19.24	30.93
CodeT5+-770M	15.5	27.2	42.7
CodeT5+-6B	28.0	47.2	69.8
CodeT5+-2B	24.2	38.2	57.8
CodeT5+-220M	12.0	20.7	31.6
CodeT5+-16B	30.9	51.6	76.7
CodeParrot-small-110M	3.58	8.03	14.96
CodeParrot-multi-1.5B	4.0	8.7	17.9
CodeParrot-110M	3.80	6.57	12.78
CodeParrot-1.5B	3.8	6.57	12.78
CodeParrot-1.5B	3.99	8.69	17.88
Code-LLaMA-Python-7B	38.4	70.3	90.6
Code-LLaMA-Python-34B	53.7	**82.8**	**94.7**
Code-LLaMA-Python-13B	43.3	**77.4**	**94.1**
Code-LLaMA-Instruct-7B	34.8	64.3	88.1
Code-LLaMA-Instruct-34B	41.5	**77.2**	93.5
Code-LLaMA-Instruct-13B	42.7	71.6	91.6
Code-LLaMA-7B	33.5	59.6	85.9
Code-LLaMA-34B	48.8	76.8	**93.0**
Code-LLaMA-13B	36.0	69.4	89.8
CodeGen-nl-6.1B	10.43	18.36	29.85.
CodeGen-nl-350M	2.12	4.10	7.38
CodeGen-nl-2.7B	6.70	14.15	22.84
CodeGen-nl-16.1B	14.24	23.46	38.33
CodeGen-multi-6.1B	18.16	27.81	44.85
CodeGen-multi-350M	6.67	10.61	16.84

LLaMA-65B	23.7	-	79.3
LLaMA-33B	21.7	-	70.7
LLaMA2-7B	12.2	25.2	44.4
LLaMA2-70B	30.5	59.4	87.0
LLaMA2-34B	22.6	47.0	79.5
LLaMA2-13B	20.1	34.8	61.2
LLaMA-13B	15.8	-	52.5
LaMDA-137B	14.0	-	47.3
JuPyT5-300M	5.40	15.46	25.60
InstructCodeT5+-16B	35.0	54.5	77.9
InCoder-multi-6.7B	15.2	27.8	47.0
InCoder-6B	15.2	27.8	47.0
InCoder-6.7B	15.2	27.8	47.0
InCoder-1.3B	8.9	16.7	25.6
GPT-NeoX-20B	15.4	25.6	41.2
GPT-NEO-350M	0.85	2.55	5.95
GPT-NEO-2.7B	6.41	11.27	21.37
GPT-Neo-125M	0.75	1.88	2.97
GPT-Neo-1.3B(1.5B)	4.79	7.47	16.30
GPT-Neo-1.3B	4.79	7.47	16.3
GPT-J-6B	11.62	15.74	27.74
GPT-4(OpenAI)	67	-	-
GPT-4(*)	**82**	-	-
GPT-3.5(OpenAI)	48.1	-	-
GPT-3.5(*)	**68.9**	-	-
Codex-85M	8.22	12.81	22.40
Codex-679M	16.22	25.70	40.95
Codex-42M	5.06	8.8	15.55
Codex-300M	13.17	20.37	36.27
code-davinci-002	47.0	74.9	**92.1**
code-davinci-001	39.0	60.6	84.1
code-cushman-001	33.5	54.3	77.4

CodeGen-multi-2.7B	14.51	24.67	38.56
CodeGen-multi-16B	18.3	-	-
CodeGen-multi-16.1B	19.22	34.64	55.17
CodeGen-mono-6B	26.1	42.3	65.8
CodeGen-Mono-6.1B	26.13	42.29	65.82
CodeGen-Mono-350M	12.76	23.11	35.19
CodeGen-mono-350M	12.76	23.11	35.19
CodeGen-mono-2B	23.7	36.6	57
CodeGen-Mono-2.7B	23.70	36.64	57.01
CodeGen-mono-16B(16.1B)	29.28	49.86	75.00
CodeGen2-7B	18.83	31.78	50.41
CodeGen2-16B	20.46	36.5	56.71
CodeGen2.5-7B-mono	33.4	58.4	82.7
CodeGen2.5-7B	28.36	47.46	75.15
CodeGen-16B-multi	19.2	34.6	55.2
CodeGen-16.1B	34.6	-	-
CodeGeeX2-6B	35.9	62.6	88.3
CodeGeeX-13B	22.89	39.57	60.92
CodeFuse-CodeLlama-34B	**74.4**	-	-
BLOOM-7.1B	7.73	17.38	29.47
BLOOM-560M	0.82	3.02	5.91
BLOOM-3B	6.48	11.35	20.43
BLOOM-176B	15.52	32.20	55.45
BLOOM-1.7B	4.03	7.45	12.75
BLOOM-1.1B	2.48	5.93	9.62
AlphaCode(dec)-89M	4.3	12.2	20.0
AlphaCode(dec)-685M	14.2	24.4	38.8
AlphaCode(dec)-55M	4.2	8.2	16.9
AlphaCode(dec)-302M	11.6	18.8	31.8
AlphaCode(dec)-29M	3.4	5.8	11.2
AlphaCode(dec)-13M	1.5	3.6	8.6
AlphaCode(dec)-1.1B	17.1	28.2	45.3

在Pass@1上，性能最好的是**GPT-4**，在Pass@10和Pass@100上，性能最好的是**Unnatural-Code-LLaMA-34B**.

Zheng Z, Ning K, **Wang, Y***, et al. (2023). From General to Specific: The Evolution and Benchmarking of Code LLMs. arXiv.

PART 03
代码生成面临的核心挑战

低计算资源场景

如今的大模型通常是百亿参数规模的，这使得它们在代码生成利于取得了显著的成就。然而，确实需要大量的计算资源来训练和部署这些大型模型。低计算资源场景下可能无法充分支持大模型的使用。

模型	模型参数	数据量/token	训练时间 /gpu hours
GPT-4	未知	未知	未知
GPT-3.5	未知	未知	未知
LLama	7/13/33/65B	1.0/1.0/1.4/1.4T	1.76M
Llama2	7/13/34/70B	2T	3.36M
codellama	7/13/34B	2T+500B	400K
starcoder	15.5B	1T	320K

低计算资源场景

参数高效微调技术在最大限度地提高模型性能，同时最大限度地减少所需的计算资源。

降低了计算成本

更少的GPU资源

更快的训练/推理时间

更快地完成推理

更低的硬件要求

更小的GPU个数和内存要求

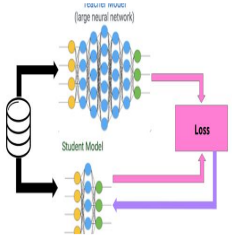
更好的建模性能

减少过拟合

更少的存储空间

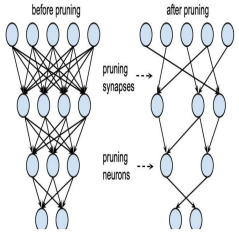
基座权重共享

参数高效微调技术



知识蒸馏

- 将知识从大型、性能良好的模型（称为教师模型）转移到较小模型（称为学生模型）的技术。使学生模型能够从教师模型的知识中受益并更好地泛化。
- 知识蒸馏不仅可以减小模型大小，还有助于规范学生模型。可以有效提高性能，同时保持效率。



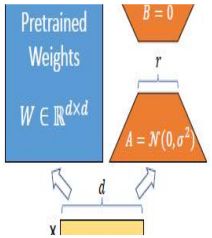
参数剪枝

- 修剪是一种涉及从预训练模型中删除不必要的权重或连接的技术。
- 通过识别和消除冗余或不太重要的参数，可以显着降低模型的大小和计算要求。实现高效的微调，而不牺牲性能。

模型量化

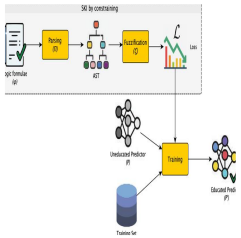
- 量化是一种降低模型参数精度以降低内存和计算要求的技术。在传统的深度学习模型中
- 量化可以通过训练后量化（其中预训练模型在训练后进行量化）或通过量化感知训练（其中模型在训练时考虑到量化）来执行。两种方法都提供了有效的微调选项，同时保留了模型性能。

参数高效微调技术



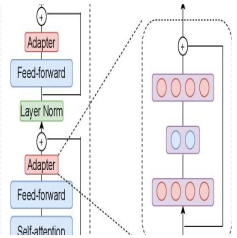
低阶因式分解

- 低秩分解是一种用低秩矩阵近似预训练模型的权重矩阵的技术。通过将权重矩阵分解为低秩分量，减少了模型的参数数量和计算复杂度。
- 低秩分解方法旨在找到保留当前任务最关键信息的低秩近似。该技术提供了参数高效的微调选项，同时实现了合理的性能。



知识注入

- 知识注入是一种通过注入特定于任务的信息而不修改原始参数来增强预训练模型在特定任务上的性能的技术。
- 知识注入可以实现高效的微调，因为它可以最大限度地减少大量再训练的需要，同时利用预训练模型中已有的知识。它在需要对原始模型进行最小修改的场景中特别有用。



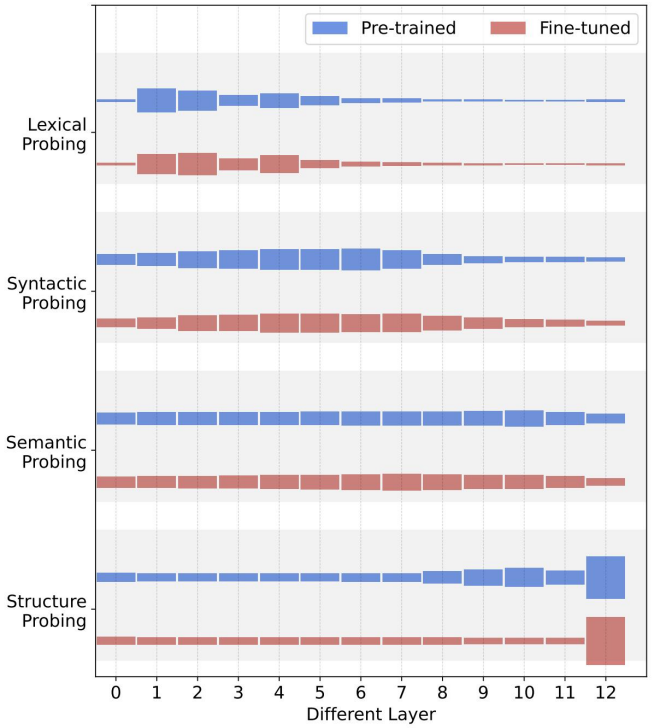
•适配器模块

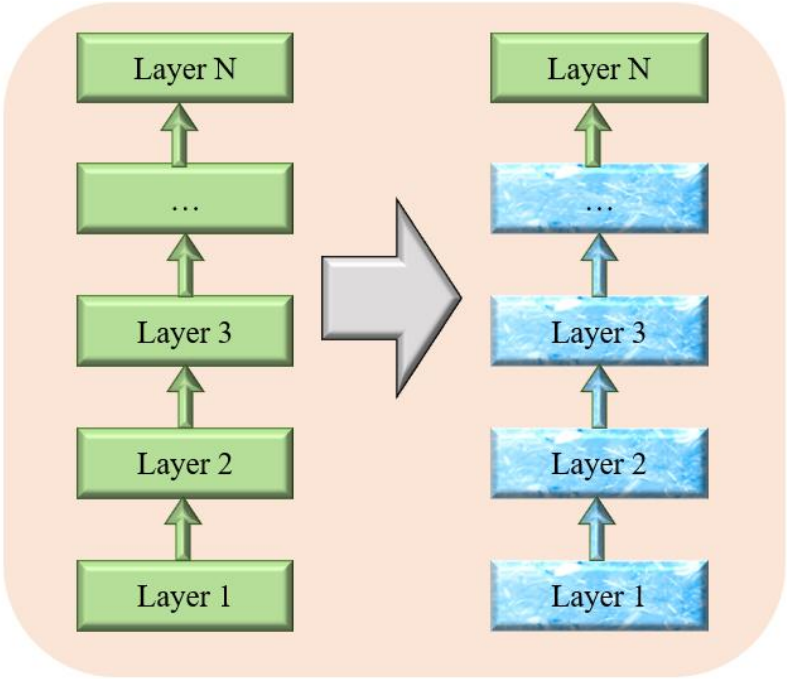
- 适配器模块是添加到预训练模型中以执行特定任务的轻量级模块，无需修改原始参数。这些模块是特定于任务的，可以插入到预训练模型架构的不同层中。
- 适配器模块使模型能够学习特定于任务的信息，同时保持大部分预训练参数不变，从而实现高效的微调。这种方法提供了灵活性和效率，因为可以针对不同的任务轻松添加或删除适配器，而无需重新训练整个模型。

参数高效微调技术

Telly: 高效地微调代码预训练模的实证研究 [ISSTA’ 23]

- 实证研究发现预训练模型的不同层编码不同的代码信息：词法、语法、语义、结构。
- 提出基于层冷冻的高效微调技术、仅需微调某些层的参数，就可以达到相近效果。





E. Shi, Y. Wang* et al., “Towards Efficient Fine-tuning of Pre-trained Code Models: An Experimental Study and Beyond”, ISSTA, 2023.

数据匮乏场景

如今的大模型的训练和优化都需要海量数据，而高质量的数据难以获得。匮乏数据匮乏场景，如何训练、应用和部署大模型是我们面临的一大挑战。

模型	训练数据量/token
GPT-4	未知
GPT-3.5	未知
LLama	1.0/1.0/1.4/1.4T
Llama-2	2T
codellama	2T+500B
starcoder	1T

数据匮乏场景

解决思路(一)： 数据增强

数据增强通过对现有数据进行改动、变换等来生成新数据，以增加训练数据样本多样性和数量。常见的针对代码的数据增强包括重写变量名、重写函数名、插入死代码等。

```
def get_tri_area(pts):
    a, b, c = pts[0], pts[1], pts[2]
    v1 = np.array(b) - np.array(a)
    v2 = np.array(c) - np.array(a)
    area_tri = abs(sp.linalg.norm(sp.cross(v1, v2
        )) / 2)
    return area_tri
```

(a) Original code

```
def calculate_triangle_area(pts):
    a, b, c = pts[0], pts[1], pts[2]
    v1 = np.array(b) - np.array(a)
    v2 = np.array(c) - np.array(a)
    area_tri = abs(sp.linalg.norm(sp.cross(v1, v2
        )) / 2)
    return area_tri
```

(b) Rewrite method name

```
def get_tri_area( points ):
    a, b, c = points [0], points [1], points [2]
    vector1 = np.array(b) - np.array(a)
    vector2 = np.array(c) - np.array(a)
    area_triangle = abs(sp.linalg.norm(sp.cross(
        vector1,vector2))/2)
    return area_triangle
```

(c) Rewrite variables

```
def get_tri_area(pts):
    a, b, c = pts[0], pts[1], pts[2]
    v1 = [b[i] - a[i] for i in range(len(a))]
    v2 = [c[i] - a[i] for i in range(len(a))]
    area_tri = math.sqrt(pow((v1[0] * v2[1] - v1
        [1] * v2[0]), 2)) / 2
    return area_tri
```

(d) Use different library functions

Wang, Y*, **Guo**, L, **Shi, E, et al** (2023). You Augment Me: Exploring ChatGPT-based Data Augmentation for Semantic Code Search ICSME 2023

数据匮乏场景

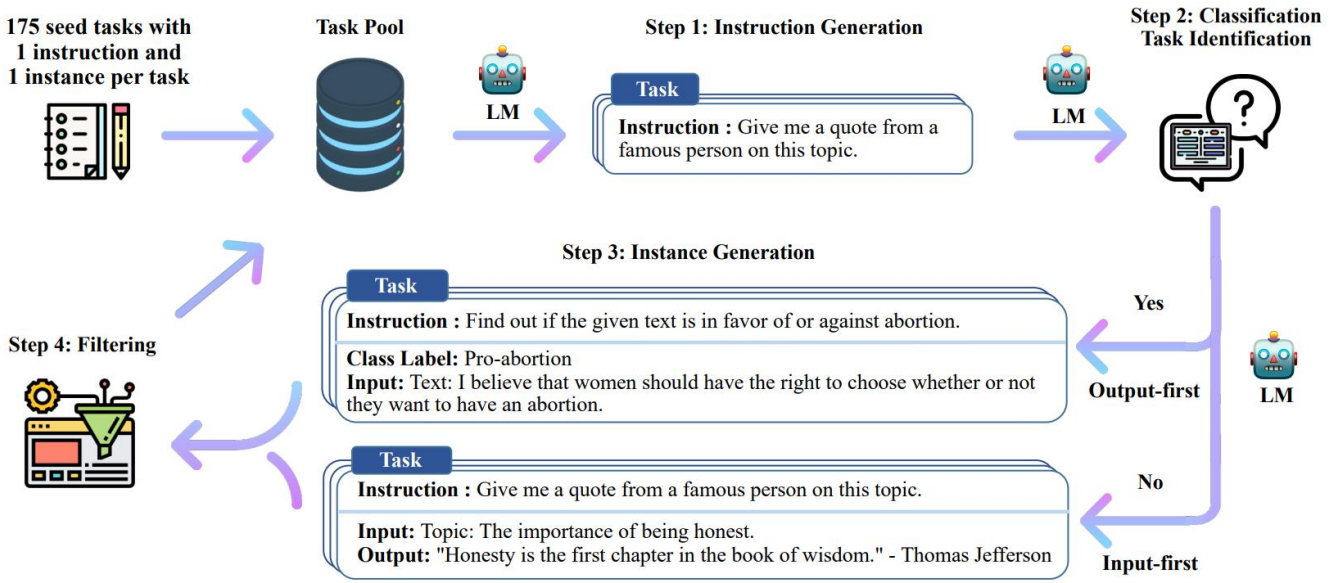
解决思路(二)： 数据生成

数据生成技术是在不使用原始数据集的情况下使用模型(GPT-3, ChatGPT，GPT-4，Llama 2等)生成大量高质量的期望数据。

You are asked to come up with a set of 20 diverse software engineering-related task instructions...

Requirements:
1. Include diverse types of software engineering tasks such as coding, debugging, testing, documentation, etc.
…
9. The instructions should be 1 to 2 sentences long. Either an imperative sentence or a question is permitted.

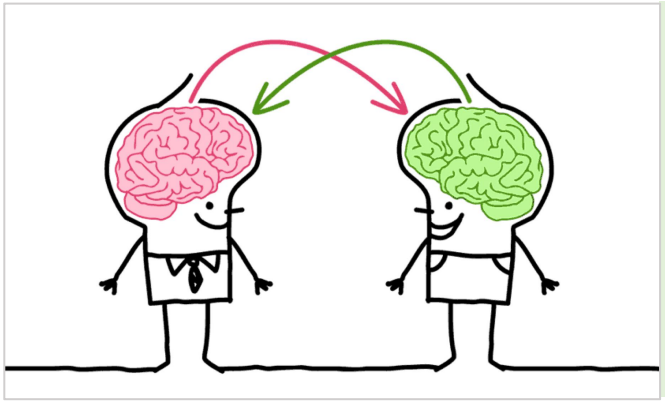
Example 1
Instruction: Explain the concept of a stack in computer science.
Input:
<noinput>
Output:
A stack is a data structure in which elements can be added or removed only from the top
…
Example 6
Instruction:



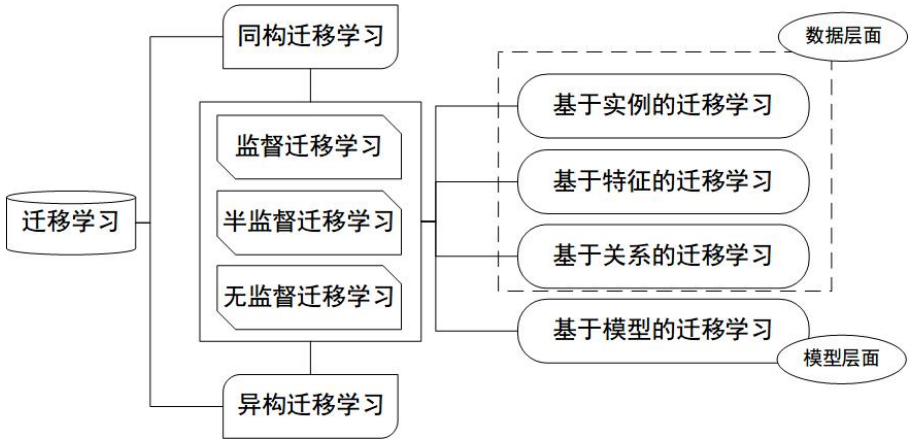
【1】**Shi, E,** Zhang, F, **Wang, Y***, Chen, B, Du, L., Zhang, H., ... & Sun, H. (2023). SoTaNa: The Open-Source Software Development Assistant. arXiv preprint arXiv:2308.13416.
【2】Self-Instruct: Aligning Language Models with Self-Generated Instructions

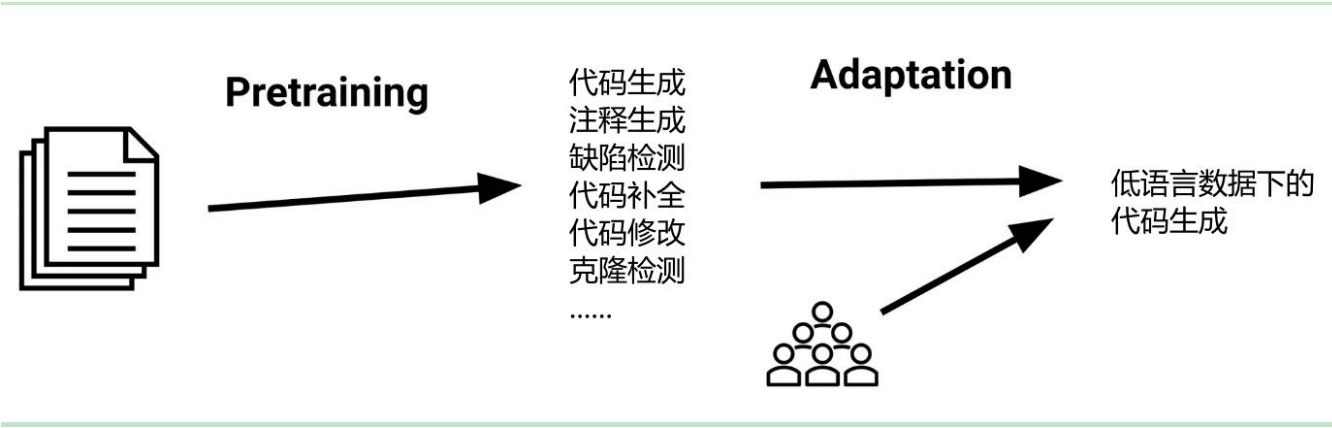
数据匮乏场景

解决思路(三)： 迁移学习



迁移学习旨在利用少量的任务数据，将从模型从其他任务学到的知识迁移到另一个相关任务上，以提高该任务的性能。它的主要思想是利用已有的数据和模型，以加速新任务的学习过程，尤其是在新任务的数据有限或难以获得时。





特定需求场景

场景一： 代码风格要求严格

合理的变量命名

- ✓ 减少无意义的名字
- ✓ 避免抽象的名字
- ✓ 减少太长的名字
- ✓ 消除有歧义的命名

必要的注释

- ✓ 当代码本身无法清晰地阐述开发者的意图。
- ✓ 逻辑比较复杂、晦涩难懂

避免深层嵌套

- ✓ 谨防回调地狱
- ✓ 避免if 嵌套很深
- ✓ 减少函数调用嵌套

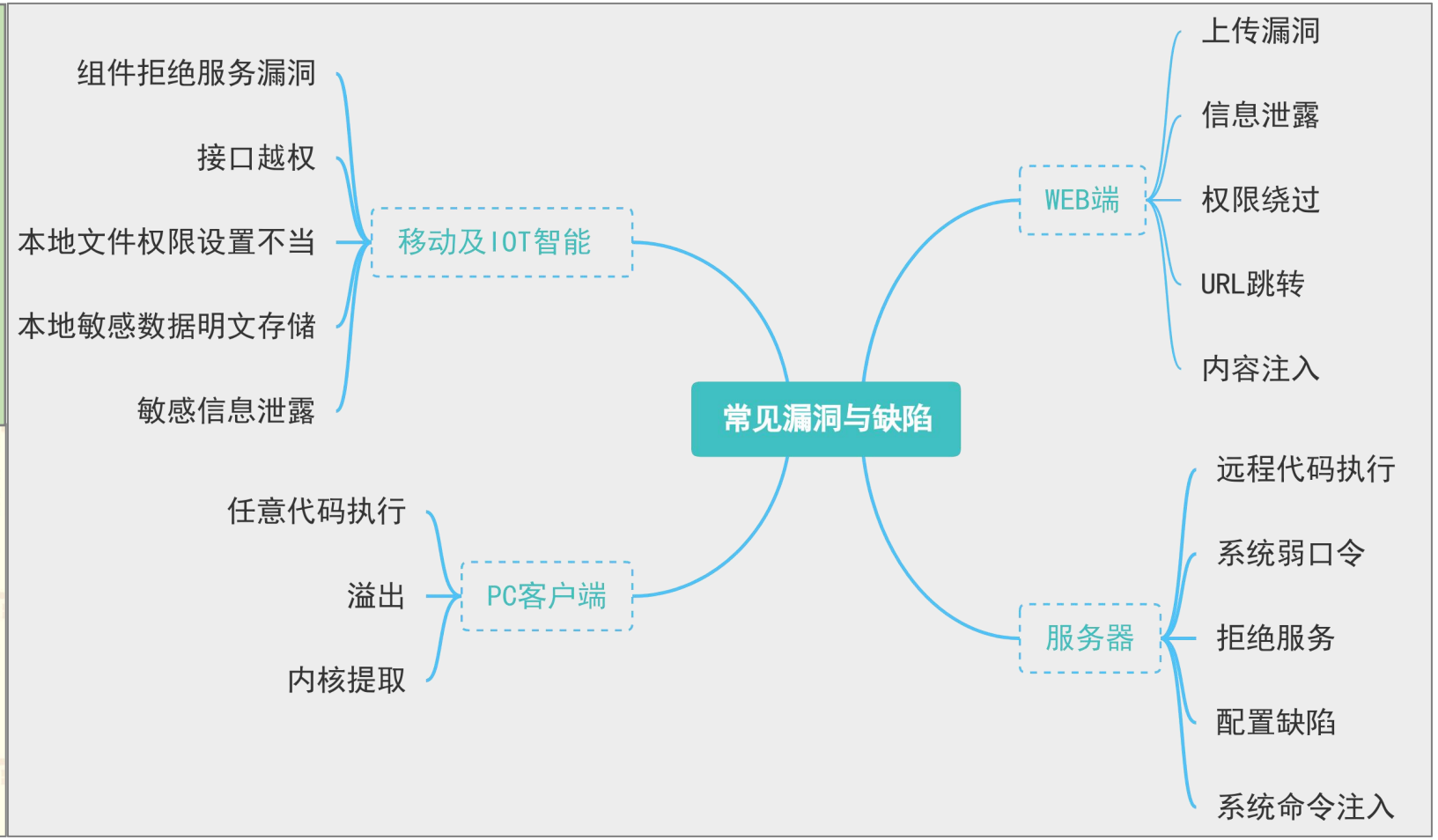
其他要求

- ✓ 限制函数的参数数量。
- ✓ 限制函数的圈复杂度
- ✓ 执行时间要求
- ✓ 占用显存要求

特定需求场景

场景二： 代码安全性要求高

代码安全性至关重要，安全漏洞会导致数据泄漏、系统崩溃和未经授权的访问等问题，造成大量的金钱损失，甚至引发灾难性事故。



特定需求场景

解决思路： 定制化与个性化模型

定制化大模型是指根据特定的需求和应用场景来创建、训练或调整大型人工智能模型，以满足定制化的要求。

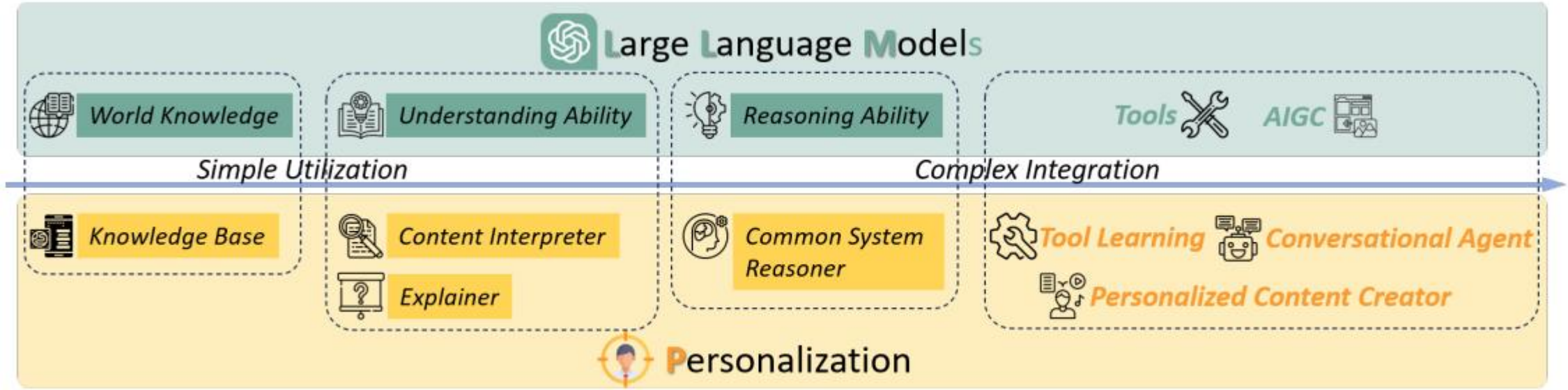


Fig. 1. The Overview of LLM for Personalization

Chen, Jin, et al. "When large language models meet personalization: Perspectives of challenges and opportunities." arXiv preprint arXiv:2307.16376 (2023).

PART 04
大模型与代码生成的未来展望

软件工程3.0时代下的代码生成

➢ 大语言模型重新定义软件开发

开发人员从主要“写代码，执行调试” 到“给出自然语言提示（Prompt ）”

大模型理解开发人员用自然语言表述的需求，完成“自动生成UI，自动生成产品代码，自动生成测试脚本，自动完成代码优化”

基于大模型自反馈的代码生成

- 现有工作[1]表示自动生成代码依旧存在错误率高、低效率、低安全性等问题
- 丰富的外部反馈信息[2][3][4]有利于大模型提高生成代码质量。

Supplementary Comment
Line 3, `print n + k`
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(n + k)? Fix the bug.

- 程序报错信息作为反馈信息

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

- 程序性能分析信息作为反馈信息

Solution . java :12: ForLoopCanBeForeach : This for loop can be replaced by a foreach loop

- 程序静态分析信息作为反馈信息

[1] Arefin, Sayed Erfan, et al. "Unmasking the giant: A comprehensive evaluation of ChatGPT's proficiency in coding algorithms and data structures." arXiv preprint arXiv:2307.05360 (2023).
[2] Zhang, Kechi, et al. "Self-Edit: Fault-Aware Code Editor for Code Generation." *arXiv preprint arXiv:2305.04087* (2023).
[3] Madaan, Aman, et al. "Self-refine: Iterative refinement with self-feedback." *arXiv preprint arXiv:2303.17651* (2023).
[4] Liu, Yue, et al. "Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues." *arXiv preprint arXiv:2307.12596* (2023).

基于大模型自反馈的代码生成

反馈信息

- 简单的优化提示

  "The generated code has quality issues. Please provide a better code implementation as expected by the task description."

- 利用外部工具信息，例如编译信息、运行时错误信息等

- 大模型的自我评价信息

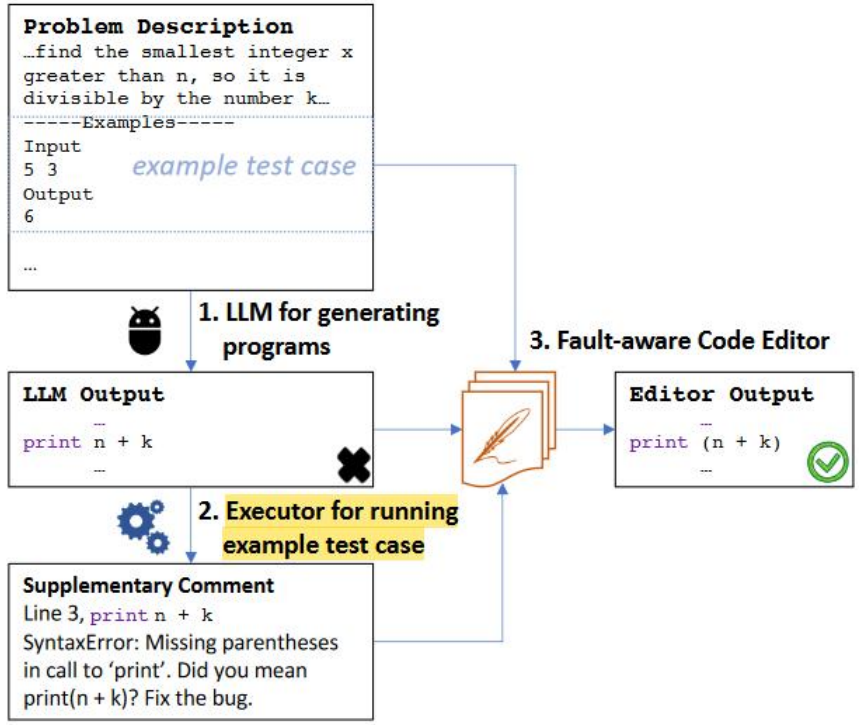


Figure 2: Pipeline of our self-edit approach.

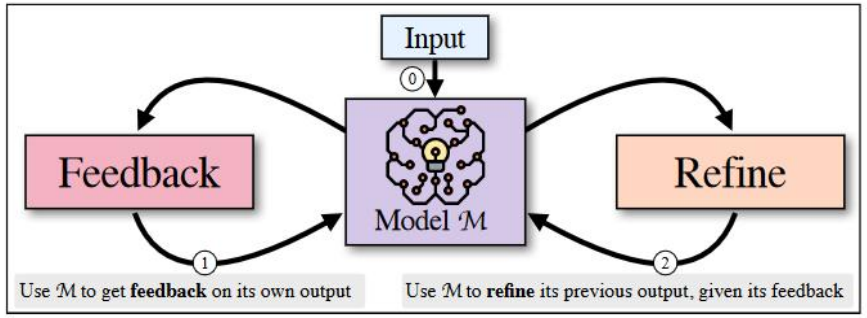


Figure 1: Given an input (⓪), SELF-REFINE starts by generating an output and passing it back to the same model $\mathcal{M}$ to get feedback (①). The feedback is passed back to $\mathcal{M}$, which refines the previously generated output (②). Steps (①) and (②) iterate until a stopping condition is met. SELF-REFINE is instantiated with a language model such as GPT-3.5 and does not involve human assistance.

基于大模型自反馈的代码生成

- 工作[1]发现更加精确复杂的反馈信息优化效果比简单的反馈信息更好。
- 已有工作存在缺陷：
  - 一句简单的提示作为反馈信息过于简单、优化方向单一
  - 将大模型对代码的评价信息作为反馈信息，内容质量没有保证。
- 如何构造高质量反馈信息？

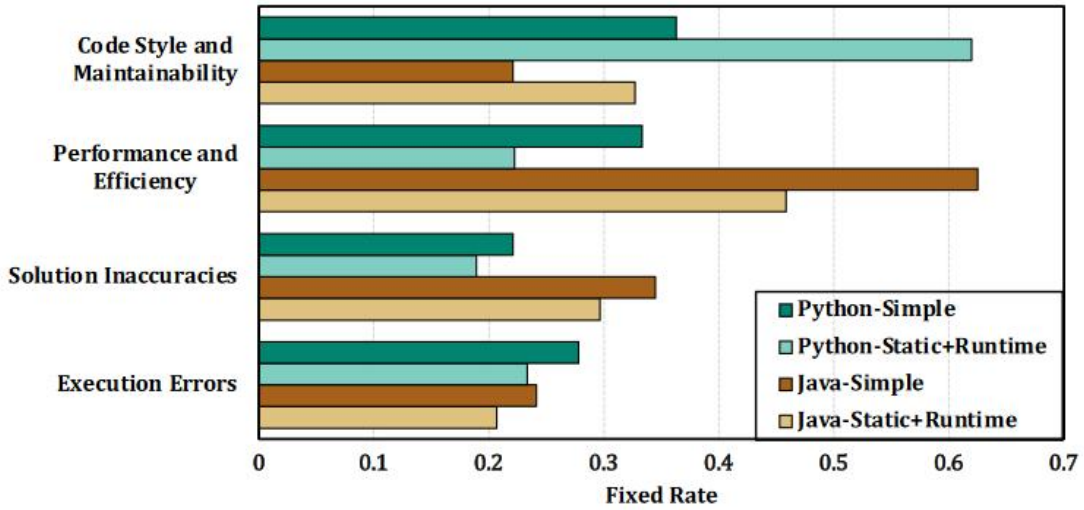
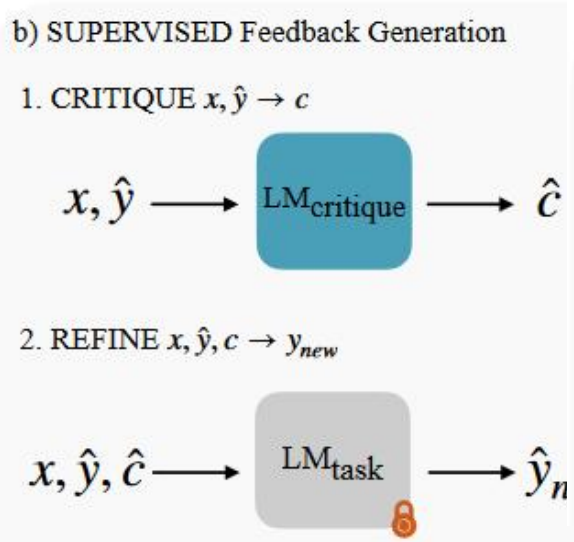
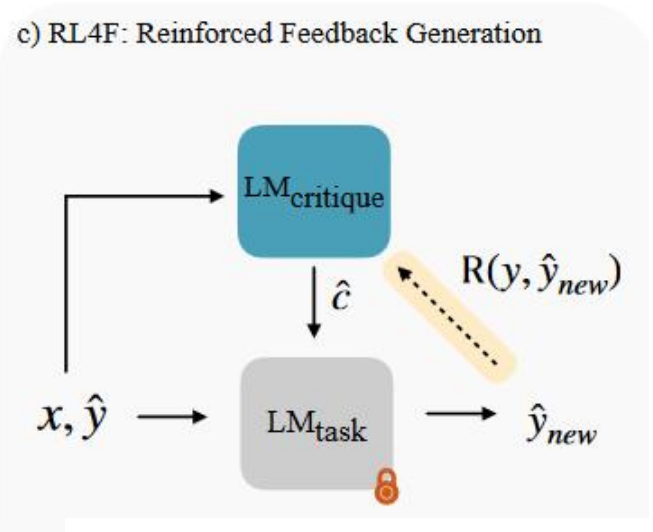


Fig. 4. Comparison of Fix Rates for Different Feedback Types and Code Quality Issues





[1] Liu, Yue, et al. "Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues." arXiv preprint arXiv:2307.12596 (2023).

基于大模型的repo-level的代码生成

已有工作

- 仅有约30%的函数是独立函数，剩余70%的函数都依赖于仓库中已有的某些信息。

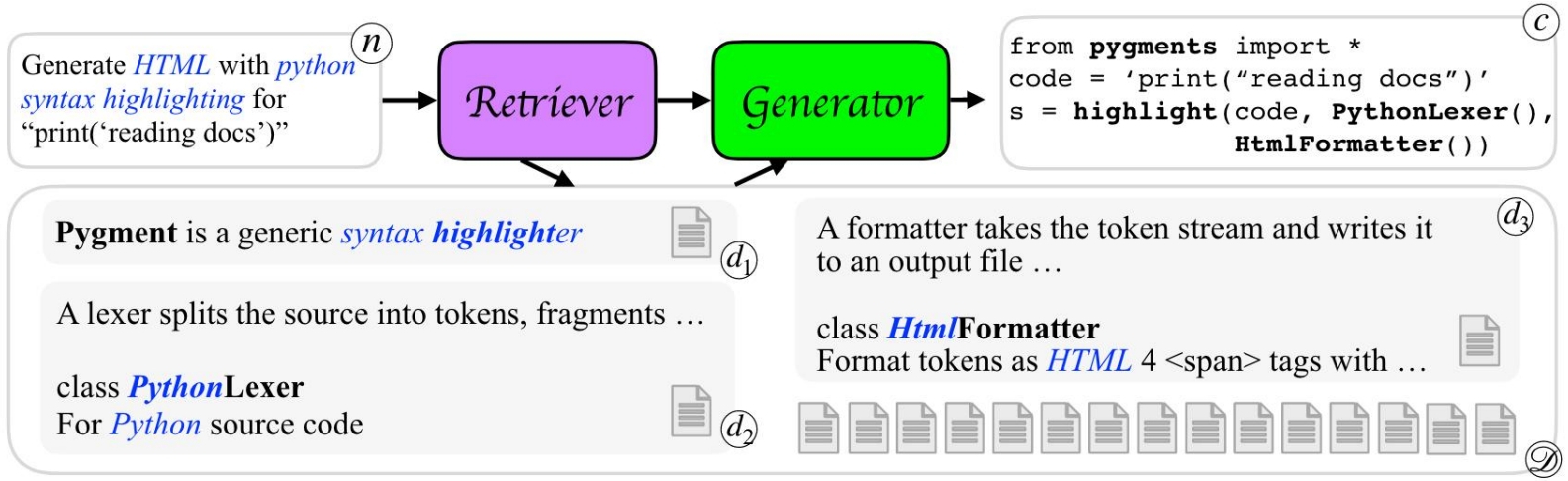
- 已有一些工作开始探索使用仓库中已有的信息来辅助生成目标代码。[2][3][4][5]针对实际开发场景下，生成调用特定API的目标代码进行了探索，通过相似性检索从仓库中的源代码或API文档中检索出目标代码所需调用的API的详细信息来辅助大模型完成代码生成任务。

[1] Yu, Hao, et al. "CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models." arXiv preprint arXiv:2302.00288 (2023).
[2] Zan, Daoguang, et al. "When language model meets private library." arXiv preprint arXiv:2210.17236 (2022)
[3] Zan, Daoguang, et al. "Private-library-oriented code generation with large language models." arXiv preprint arXiv:2307.15370 (2023).
[4] Liu, Mingwei, et al. "CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation.
[5] Zhou, Shuyan, et al. "Docprompting: Generating code by retrieving the docs." The Eleventh International Conference on Learning Representations. 2022.

基于大模型的repo-level的代码生成

已有工作

- 已有的工作目前主要分为两个阶段：检索阶段和生成阶段。
  - 检索阶段：根据目标函数的自然语言描述，在仓库中检索出高相似性的文档或源代码段。
  - 生成阶段：将检索阶段的结果与目标函数的自然语言描述联合构造成prompt，让模型生成目标函数。



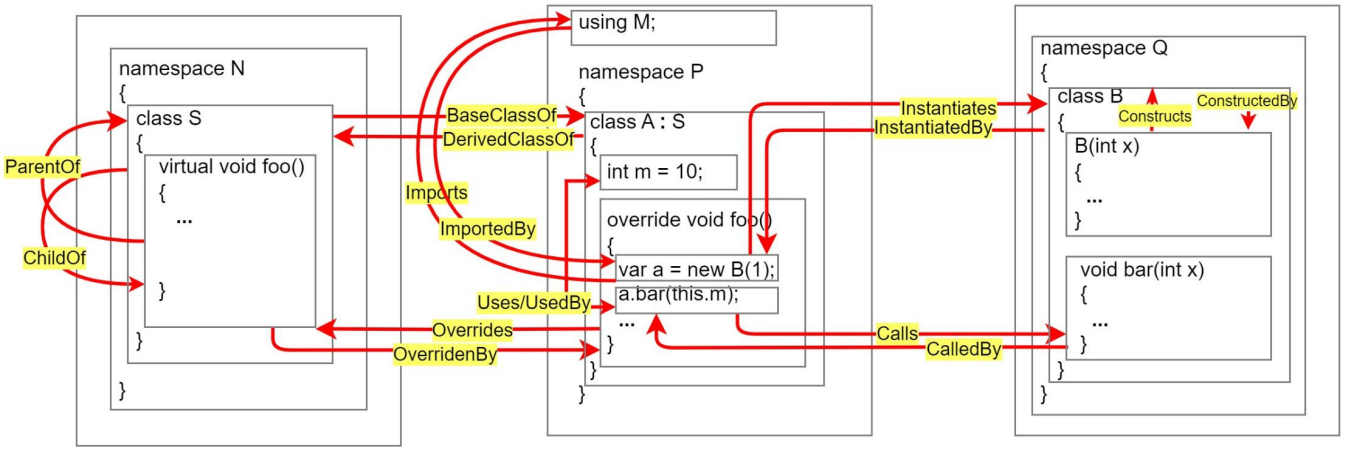
[1] Zhou, Shuyan, et al. "Docprompting: Generating code by retrieving the docs." The Eleventh International Conference on Learning Representations. 2022.

基于大模型的repo-level的代码生成

已有工作的局限

- 主要围绕着目标函数的API调用展开，缺乏对整体代码仓库的关注。例如，仓库整体依赖关系，与目标代码较为相似的代码段等

- 目前代码生成的粒度较小，代码生成的目标是仓库中某个函数。然而考虑实际开发场景下，函数的数量一般都比较多，因此应考虑粒度更大的生成目标，如一个类、一个文件、甚至一个package。



[1] Bairi, Ramakrishna, et al. "CodePlan: Repository-level Coding using LLMs and Planning." arXiv preprint arXiv:2309.12499 (2023).

基于大模型的repo-level的代码生成

目前正在开展的工作

- Repo-level的代码生成本质上是面向lib的代码生成任务，仅仅考虑了如何让生成的代码准确地调用API，而真实场景下仓库级别的代码生成应该考虑更为通用的场景，应该覆盖仓库中所有的函数。

- 由于仓库级别的函数存在着复杂的依赖调用关系和代码语境，为了让大模型可以更好地理解仓库中的这些复杂信息，我们对仓库中与目标代码有关仓库上下文信息进行提取，分别通过相似性检索和相关性分析来获取仓库上下文。最后，将检索和分析的结果与目标函数的自然语言描述一起构造成提示词，让模型在考虑仓库中复杂依赖调用关系的同时，深入仓库中的代码语境，生成精确的代码。

特定领域代码生成

基于大模型的智能合约代码生成与补全

模型选择：llama2-chat-13b

数据来源：以太坊mainnet链（截至2023年3月）

筛选标准：交易数量不少于10

预处理：代码去重、删除注释

```
301  \n \n pragma solidity ^0.8.10;\n \n import "./ERC721A.sol";\n import "@openzeppelin/contracts/
302  pragma solidity 0.5.17;\n \n \n import './interfaces/IPublicLock.sol';\n import '@openzeppelir
303  pragma solidity ^0.4.18;\n \n library SafeMath {\n   function mul(uint256 a, uint256 b) interr
304  pragma solidity ^0.6.0;\n \n abstract contract ILendingPool {\n     function flashLoan( addres
305  \n pragma solidity ^0.8.0;\n \n contract FeeDistributor {\n \n     uint256 public immutable sh
306  pragma solidity ^0.8.0;\n \n import "./DolaFlashMinter.sol";\n \n contract MainnetDolaFlashMir
307  pragma solidity ^0.6.0;\n \n abstract contract CompoundOracleInterface {\n     function getUnc
308  \n \n \n \n pragma solidity >=0.6.0 <0.8.0;\n \n interface IERC20 {\n     \n     function tota
309  \n                                                \n pragma solidity 0.8.11;\n \n abstrac
310  pragma solidity >=0.8.10 >=0.8.0 <0.9.0;\n \n import "@openzeppelin/contracts/token/ERC20/ERC2
311  pragma solidity ^0.6.0;\n pragma experimental ABIEncoderV2;\n \n import "../../utils/FlashLoar
312  \n \n pragma solidity ^0.8.9;\n \n abstract contract Context {\n     function _msgSender() int
313  pragma solidity ^0.6.0;\n \n import "../../DS/DSMath.sol";\n import "../../DS/DSProxy.sol";\n
```

图1 预训练数据集（部分）

```
--validation_split_percentage 0.001 \
--per_device_train_batch_size ${per_device_train_batch_size} \
--do_train \
--seed $RANDOM \
--fp16 \
--num_train_epochs 1 \
--lr_scheduler_type cosine \
--learning_rate 2e-4 \
--warmup_ratio 0.05 \
--weight_decay 0.01 \
--logging_strategy steps \
--logging_steps 10 \
--save_strategy steps \
--save_total_limit 3 \
--save_steps 200 \
```

图2 关键参数设置

基于大模型的智能合约代码补全

数据来源：以太坊mainnet链上（截至2023年3月）交易数量大于1000的合约

数据集构建方法：

- 编译该合约，获取抽象语法树(AST)，从AST中读取该合约中所有子合约的**依赖信息**
- 对于每份子合约，将其中的函数片段作为**input**，该子合约**依赖**的其他子合约以及自身作为**output**


```
    {
        "instruction": "The following is incomplete Solidity code for a smart contract. I need you to finish writ
        "input": "function totalSupply(){\n    function safeAdd(uint a, uint b) public pure returns (uint c) {\n
        "output": "\n\npragma solidity ^0.5.0;\n\ncontract ERC20Interface {\n    function totalSupply() public vi
    },
    {
        "instruction": "Please take this Solidity smart contract code snippet and expand it out into a finished,
        "input": "function balanceOf(address owner){\n        return balances[owner];\n    }\nfunction transfer(a
        "output": "\npragma solidity ^0.8.2;\n\ncontract Token {\n    mapping(address => uint) public balances;\r
    },
    {
        "instruction": "Please expand on this snippet of a smart contract written in Solidity to make it a comple
        "input": "function Put(uint _unlockTime){\n        var acc = Acc[msg.sender];\n        acc.balance += msg
        "output": "\ncontract ALFA_Bank\n{\n    function Put(uint _unlockTime)\n    public\n    payable\n    {\n
    },
    {
        "instruction": "I have a part of a smart contract's code in Solidity. Could you please complete the code
        "input": "function balanceOf(address account){\n    function deploy(address stakingToken, uint256 rewardA
        "output": "pragma solidity ^0.6.2;\n\ninterface IStakingRewards {\n    function balanceOf(address account
    },
```


图1 微调数据集（部分）

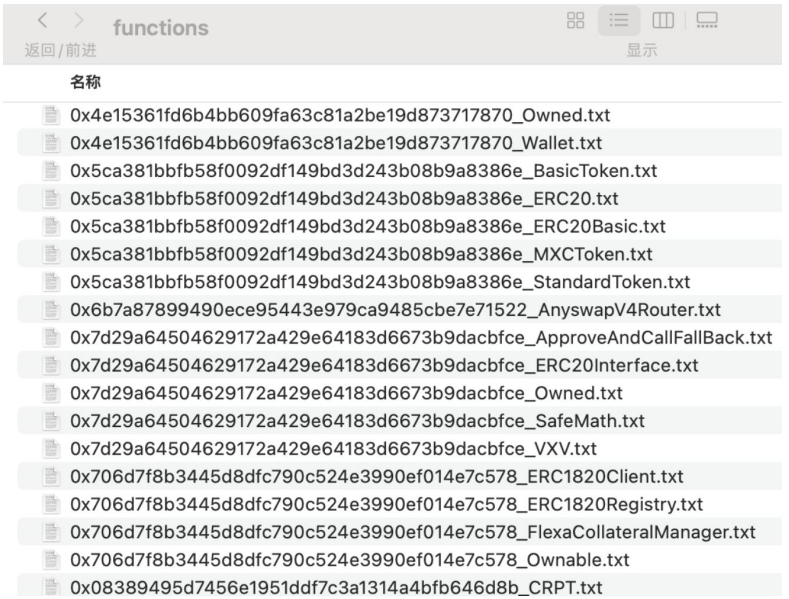


图2 子合约拆分过程

基于大模型的智能合约代码补全

输入：以Solidity编写的智能合约的代码片段

输出：包含该片段的完整智能合约

```
function Ownable(){
    owner = msg.sender;
}

function transferOwnership(address newOwner){
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
```

图1 输入的代码片段

```
pragma solidity ^0.4.26;

contract Ownable {

  address public owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  construct Ownable() public {
    owner = msg.sender;
  }

  modifier onlyOwner() {
    require(owner == msg.sender);
    _;
  }

  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

  function () external payable {}
}
```

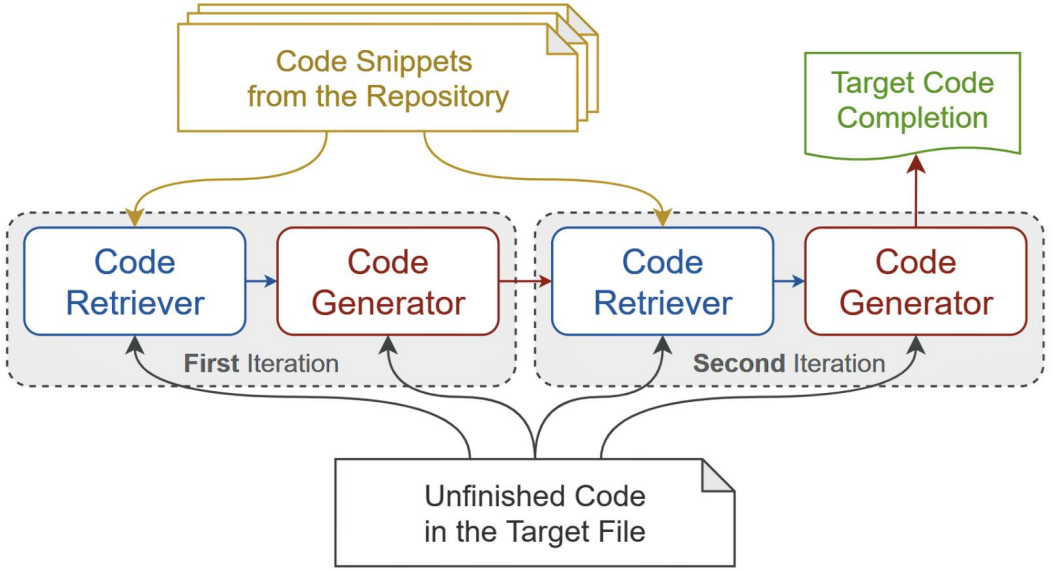
图2 大模型补全的完整代码

如何将大模型更好地融入软件开发

大模型显示了强大的代码、文本的理解生成能力。如何将为大模型找到合适的应用场景，融入实际的软件开发中是当前大模型落地的关键。

➢ 面向仓库级别的代码生成

代码仓库是代码项目的载体。如何使用代码大模型理解、维护代码项目，并根据相关需求进行代码开发是一个值得研究的问题

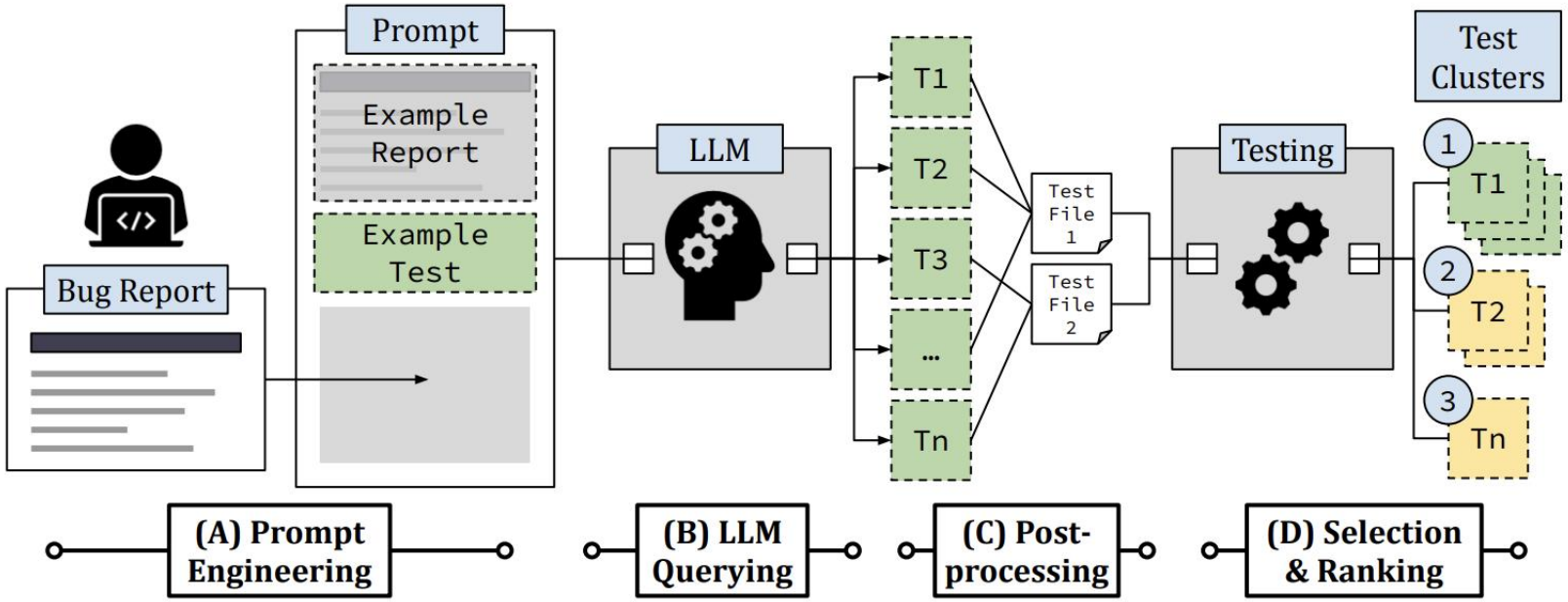


[1] Zhang, Fengji, et al. "Repocoder: Repository-level code completion through iterative retrieval and generation." arXiv preprint arXiv:2303.12570 (2023).

如何将大模型更好地融入软件开发

∅ 融合大模型的自动化代码测试

将代码大模型融入自动化测试流程中，使用大模型完成测试样例生成、测试脚本的编写，可以有效跳高代码开发的效率



[1] Kang, Sungmin, Juyeon Yoon, and Shin Yoo. “Large language models are few-shot testers: Exploring llm-based general bug reproduction”, ICSE, 2023.

如何将大模型更好地融入软件开发

∅ 基于大模型的自动代码文档编写

将代码大模型融入自动化测试流程中，使用大模型完成测试样例生成、测试脚本的编写，可以有效跳高代码开发的效率

∅ 基于大模型的提示消息生成

在实际代码开发中，频繁的项目更迭往往需要多次向仓库提交代码。为了更好的理解提交代码的意图，提示消息十分重要。如何自动化为代码编写提示消息是一个值得研究问题。

期望的突破

- 大模型的函数级别代码生成能力达到甚至超越人类程序员
- 大模型在复杂开发场景如仓库级别中，代码理解、生成能力接近、达到人类程序员水平
- 代码大模型的数据收集、构造成本不断降低，代码大模型的训练成本降低
- 大模型生成的代码在安全性、可靠性、风格统一性上得到优化
- 大模型在软件工程应用中无“延迟感”
- more?

AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

THANKS