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

AI+
software
Development
Digital
summit

大语言模型是软件工程的银弹吗？

夏鑫　华为

科技生态圈峰会+深度研习 ——1000+技术团队的共同选择

K+峰会

2023K+
全球软件研发行业创新峰会
上海站
会议时间 | 06.09-10

2023K+
全球软件研发行业创新峰会
北京站
会议时间 | 07.21-22

2024K+
全球软件研发行业创新峰会
深圳站
会议时间 | 05.17-18

K+峰会详情

AiDD峰会

会议时间 | 08.18-19
AiDD AI+软件研发数字峰会
北京站

会议时间 | 11.17-18
AiDD AI+软件研发数字峰会
深圳站

演讲嘉宾

夏鑫　华为软件工程应用技术实验室主任

夏鑫的研究方向是智能化软件工程、软件仓库挖掘和经验软件工程。夏鑫至今发表了290多篇论文，其中包括120多篇CCF A类期刊和会议长文， 谷歌学术引用1.2万多次，H-index 62。夏鑫获得了2022年ACM SIGSOFT Early Career Researcher Award（亚太地区第一位），部分论文获得国际会议最佳/杰出论文奖项，包括6篇ACM SIGSOFT 杰出论文奖（连续四年获得软工顶会ASE 2018-2021的杰出论文奖）。此外他担任MSR、SANER、PROMISE等会议的Steering Committee，多个国际会议的PC（ICSE，ESEC/FSE，ASE等），多个期刊的编委（TOSEM、EMSE、ASEJ、JSEP等），以及参与组织了多个国际会议（ICSE 2023和2024， ASE 2016、2020和2021等）。

目录

CONTENTS

PART 01

背景

构建软件工程的未来

背景 CMU软件工程研究所正在引领社区构建多年的研发愿景和路线图用于设计下一代依赖软件的系统。

重点研究领域和目标（未来10-15年）

1. 先进的开发范式

□**AI 增强的软件开发（ AI-Augmented Software Development ）**:为实现这一目标，需要通过增加AI和自动化工具对开发人员的支持重新设想整个软件开发过程，需要确保利用整个生命周期中生成的数据。重点：AI 增强的软件开发意味着 AI 将在软件开发过程和持续演化过程的每个阶段起特别的作用。

□**确保持续演化的系统（Assuring Continuously Evolving Systems）**

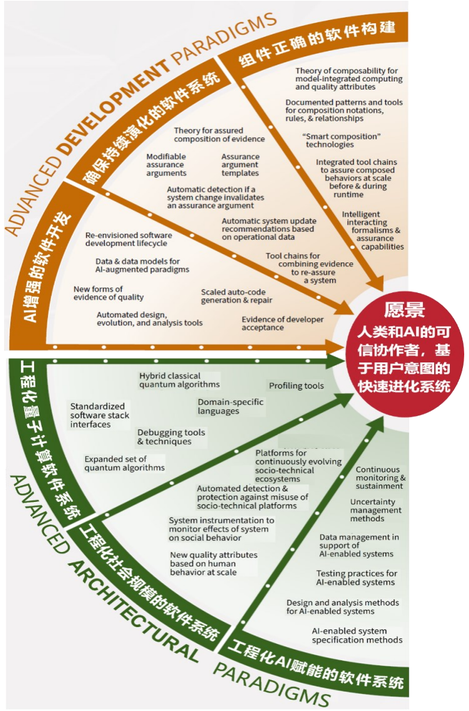
□**组件正确的软件构建（Software Construction through Compositional Correctness）**

2. 先进的架构范式

□**工程化社会规模的软件系统（Engineering Societal-Scale Software Systems）**

□**工程化 AI 赋能的软件系统（Engineering AI-enabled Software Systems）**

□**工程化量子计算软件系统（Engineering Quantum Computing Software Systems）**

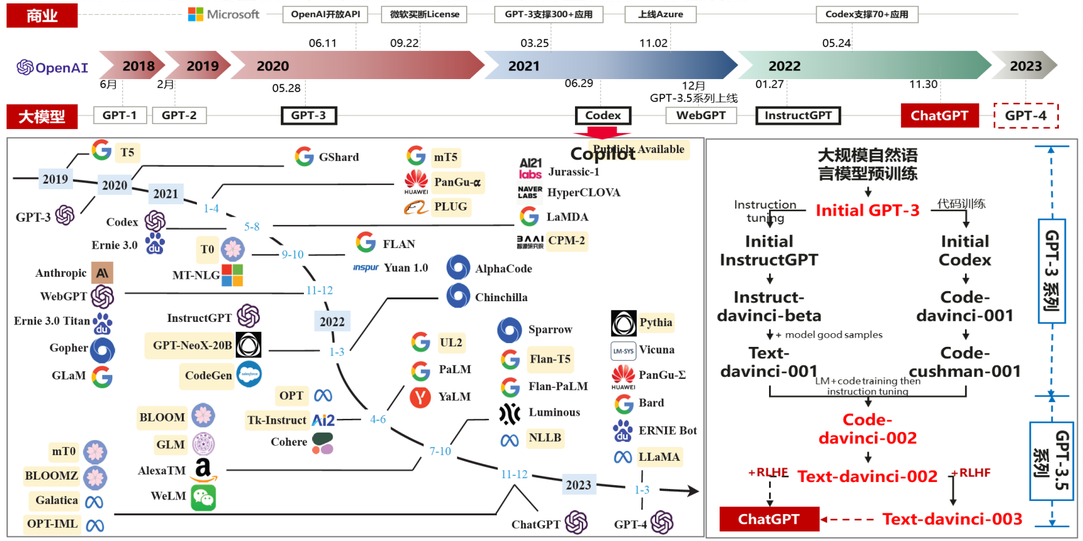


ChatGPT大模型带来智能化软件工程技术新机遇

2020年12月，Gartner报告：**人工智能增强软件开发**的新兴技术有可能与脚本化自动化技术相结合，软件工程师手动完成的**70%工作**将实现自动化；

2022年11月，OpenAI发布**ChatGPT**，充分利用了微软GitHub代码库数据，基于RLHF强化学习技术，**在编程辅助的代码样例生成方面取得一定进展**。

从代码表征模型到大语言模型，深度学习对智能化软件工程的影响在增强



大模型的快速发展历程

ChatGPT的快速演进

- **OpenAI ChatGPT：在辅助编程领域取得一定进展，呈现人机协同提升研发效能的发展趋势。**

代码解释
Explain code
Explain a complicated piece of code.

程序文档生成
Write a Python docstring
An example of how to create a docstring for ...

程序代码翻译
Translate programming languages
Translate from one programming language ...

代码压缩
JavaScript one line function
Turn a JavaScript function into a one liner.

Copilot X：
AI 结对编程工具

Ghostwriter：
从谷歌云上托管和提供 Replit 的代码编辑软件

StarCoder：
代码大模型工具

Cursor：
GPT-4代码编辑器

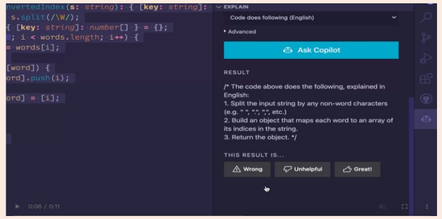
基于大模型的智能化软件工程应用开始涌现。

基于大模型的智能化结对编程助手：Copilot X

除了基础的代码生成功能外，目前Copilot集成了最新的一些基于LLM的功能，包括代码解释、测试生成、代码翻译等，同时还提供了一些代码质量保障的工具，例如，bug修复，调试等。

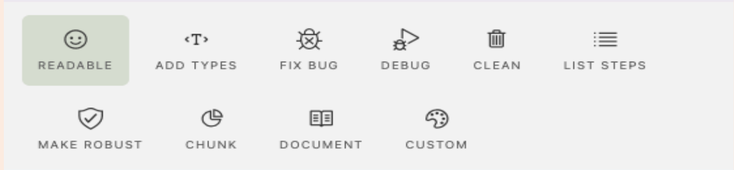
Code Explanation

根据选中的代码生成可理解的自然语言描述，
Prompt:
explain what a particular block of code does



Brushes

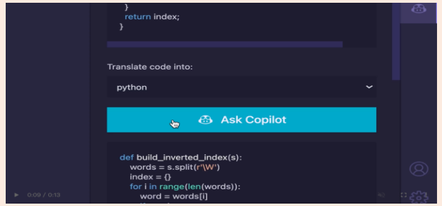
针对代码质量，提供了代码格式化、bug修复、调试、注释生成等功能。



Code Translation

实现不同编程语言之间的翻译，当前几乎覆盖了所有主流编程语言。目前这个功能处于试用过程中，还在收集用户反馈。

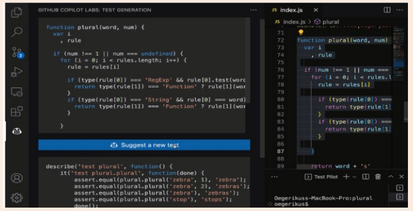
https://githubnext.com/projects/copilot-labs



Test Generation

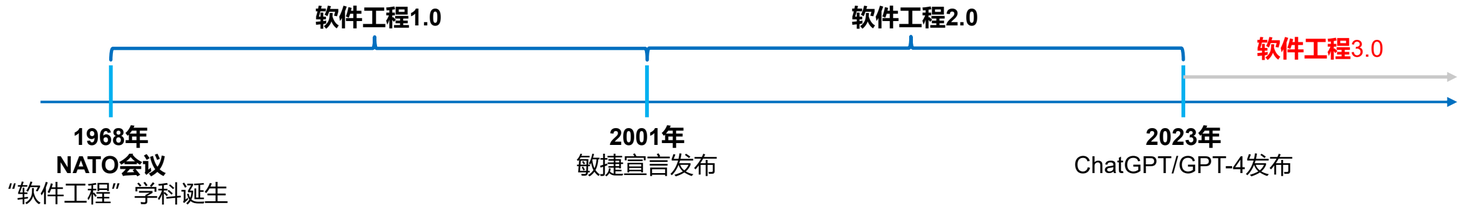
集成TestPilot，实现从代码到完整测试方法的生成。TestPilot 将扫描repo中相关的文档注释和代码示例，然后根据这些信息为该函数生成一个单元测试。

Adaptive Test Generation Using a Large Language Model

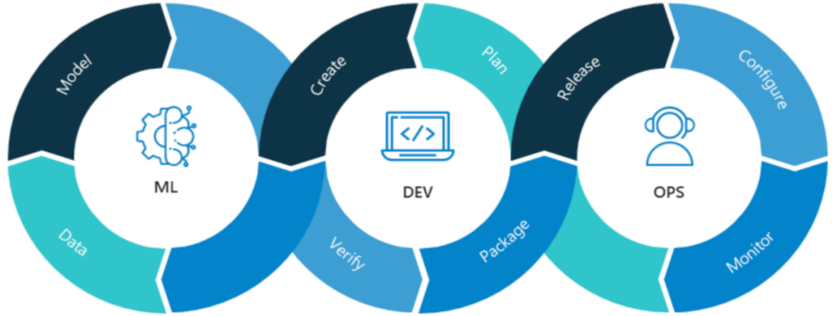


- **基于GPT4为软件开发工作流提供AI交互接口与辅助能力，支撑包括对话辅助、拉取请求 、命令行执行 、文档辅助的相关活动**

GPT-4 开启 “软件工程3.0” 全新时代



研发团队的主要任务不是写代码、执行测试，而是训练模型、参数调优、围绕业务主题提问或给提示（prompt）。



- **数字化**：软件研发平台开始能够理解需求、设计、代码等，软件研发从过去的信息化进入数字化时代；
- **AIGC**：生成软件（software）的各种ware：验收标准、测试用例、UI、代码、测试脚本等；
- **极致的持续交付**：虽然软件工程2.0开始面向CI/CD，但还存在许多障碍，而在软件工程3.0，得益于设计、代码、测试脚本等生成，可以真正实现持续交付，即及时响应客户需求，交付客户所需的功能特性；
- **人机交互智能**：软件研发过程就是人与计算机的交互过程；
- **以模型和数据为本**：研发人员服务于大模型和大数据平台，包括模型创建、训练、调优、使用等；

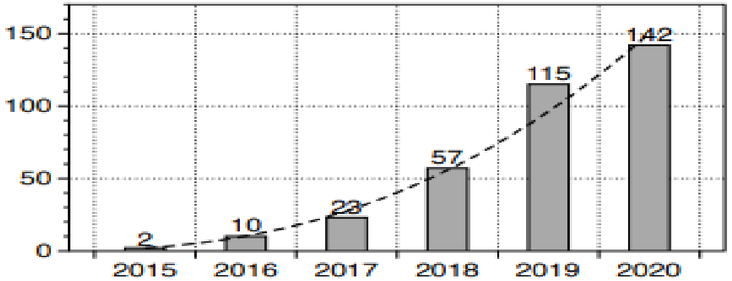
PART 02

基于大语言模型的软件工程

前期工作：基于深度学习的软件工程技术受到学术界广泛的关注

背景：

近年来随着深度学习的快速兴起，深度学习在软件工程领域也取得了广泛应用和突破性进展。仅2020年一年，就有超过100篇相关论文发表在软件工程高水平期刊和会议上。



通用深度学习模型构建流程

数据采集 → 数据预处理 → 输入构建 → 模型训练 → 模型测试 → 模型应用

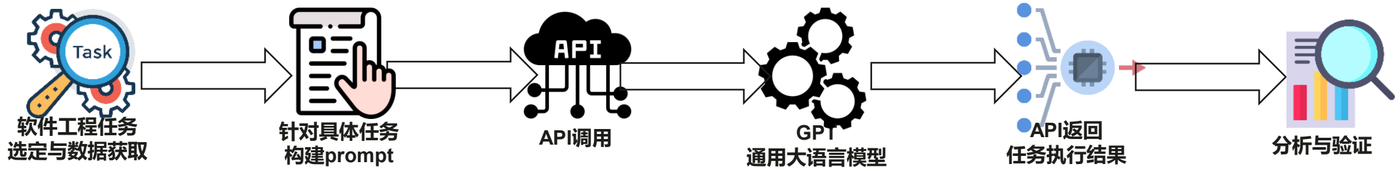
深度学习模型在软件任务中的应用

软件需求	软件设计	软件实现	软件测试和缺陷	软件维护	软件管理
NER深度模型	CNN Autoencoder	DeepCS	代码搜索	SDAE模型	缺陷预测	RNN-based	克隆检测	CRNN	任务评估
软件需求分析	软件设计模式识别	Tree-LSTM	API推荐	DBN 模型	语义缺陷检测	FNN-based	注释生成	Answerbot	SO挖掘
软件需求验证	RNN / CNN / KNN	Graph-GNN	软件定位	SRLA 模型	漏洞检测	CoCoNut	程序修复	DeepTip	Github挖掘
	GUI 建模	Attention	代码补全	BGRU 模型	CVE检测	Tree CNN	程序表征	DeepTip	Youtube挖掘
				CNNFL 模型	缺陷定位	CNN-based	SATD		
				DeepSQLi 模型	测试用例生成	Deep Review	代码评审		

A Survey on Deep Learning for Software Engineering. CSUR

前期工作：基于GPT-3.5系大模型的智能化软件工程技术研究

总览 以ChatGPT为主的GPT系模型代表着当前业界大语言模型的最前沿涌现与泛化能力。本项工作主要为通过GPT-3.5系大语言模型，探索**大语言模型时代下的软件工程技术构建模式**，包括如何有效构建各类软件工程任务的Prompt数据，使大语言模型具备高性能的代码生成、代码语言转换、代码文档生成、测试用例生成、缺陷自动修复等能力。



- **整体方案：选取软件工程各个周期（开发，维护，测试，质量保证）的代表性任务，通过ChatGPT研究最前沿大语言模型在各类任务上的性能及可能的改进方向**

软件开发任务

代码生成：
- 任务描述：根据代码功能的自然描述和方法的定义信息，生成完整的代码。
- 数据集与评价指标：HumanEval-X数据集；指标为Pass@K (K值取1和10)。

代码语言转换：
- 任务描述：将给定代码片段转换至另一种编程语言。
- 数据集与评价指标：CodeXGlue数据集；CodeBLEU分数和BLEU-4分数。

软件维护任务

代码文档生成：
- 任务描述：根据给定的代码片段，自动生成符合该段代码功能描述的自然语言 (如Python方法的docstring)。
- 数据集与评价指标：数据集使用CodeSearchNet；评价指标为BLEU-4分数，衡量生成的结果与数据集中标准答案的相似度。

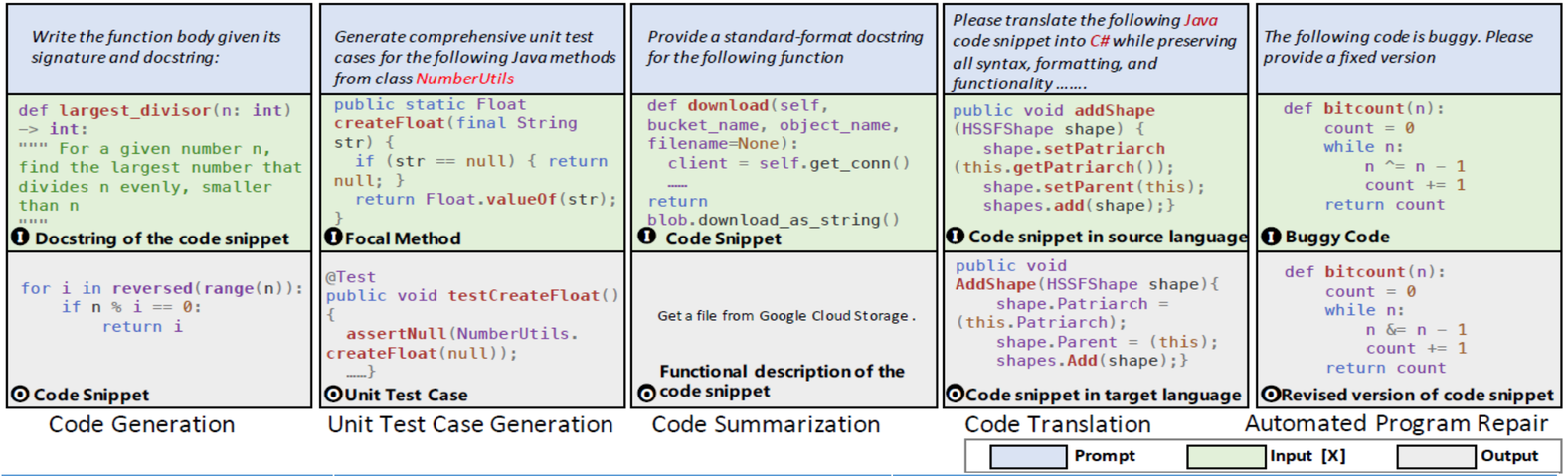
软件测试

测试用例生成：
- 任务描述：根据给定的代码，自动生成与其对应的单元测试用例 (Unit Test Cases)。
- 数据集与评价指标：数据集使用Defects4J；评价指标为测试用例的行覆盖率和条件覆盖率，其中行覆盖率衡量测试用例能够覆盖到的代码行数，条件覆盖率衡量条件语句中每个条件可能取值的覆盖情况。

软件质量保证

缺陷自动修复：
- 任务描述：给定一段包含缺陷的代码，模型返回对应的已经修复好缺陷的代码。
- 数据集与评价指标：使用QuixBugs数据集，包含Python和Java两种语言的缺陷代码和测试用例；评价指标为修复后的代码片段能通过所有测试用例的数量和比例。

前期工作：各类软件工程任务的输入与输出



任务名称	输入[I]	输出[O]
代码生成	包含代码功能描述的docstring	相应代码片段
测试用例生成	焦点方法 (Focal Method)	测试用例
代码文档生成	代码片段	代码片段的功能描述
代码语言转换	源语言代码片段	目标语言代码片段
缺陷自动修复	含缺陷的代码片段	修复好的代码片段

前期工作：基于ChatGPT的代码生成

ChatGPT在代码生成上取得了**远强于其他非GPT系大模型**的效果，相比于GPT-3.5系的其他大模型提升也十分显著。

Models	Python		C++	
	P@1 (%)	*P@10 (%)*	*P@1 (%)*	*P@10 (%)*
GPT-J	11.1	18.7	7.5	13.8
GPT-NeoX	13.8	22.7	9.9	19.0
InCoder	16.4	26.6	9.5	19.3
CodeGen	19.2	34.6	18.1	30.8
CodeGeeX	22.9	39.6	17.1	32.2
Text-davinci-002	20.4	63.4	18.7	63.4
Text-davinci-003	56.9	84.8	45.6	80.5
GPT-3.5-turbo	**69.9**	**90.2**	**53.1**	**81.1**

Listing 1 A code generation example .

```
#Input to this function is a string represented multiple groups for
#nested parentheses separated by spaces.For each of the group,
#output the deepest level of nesting of parentheses.

# Generated Python Program (correct)
def parse_nested_parens(paren_string: str) -> List[int]:
    depths = []
    max_depth = 0
    curr_depth = 0
    for char in paren_string:
        if char == "(":
            curr_depth += 1
            if curr_depth > max_depth: max_depth = curr_depth
        elif char == ")": curr_depth -= 1
        elif char == " ":
            depths.append(max_depth)
            max_depth = 0
            curr_depth = 0
    depths.append(max_depth)
    return depths
```

生成的Python代码可以通过该样例所有的测试用例

前期工作：基于ChatGPT的代码生成

ChatGPT在测试用例生成上取得了**强于其他SOTA方法的效果，** 提升了现有测试用例生成的**行覆盖率**和**条件覆盖率**

生成测试用例主要包含三个方面：

- 有效值的测试
- 空值的测试
- 不合法值的测试

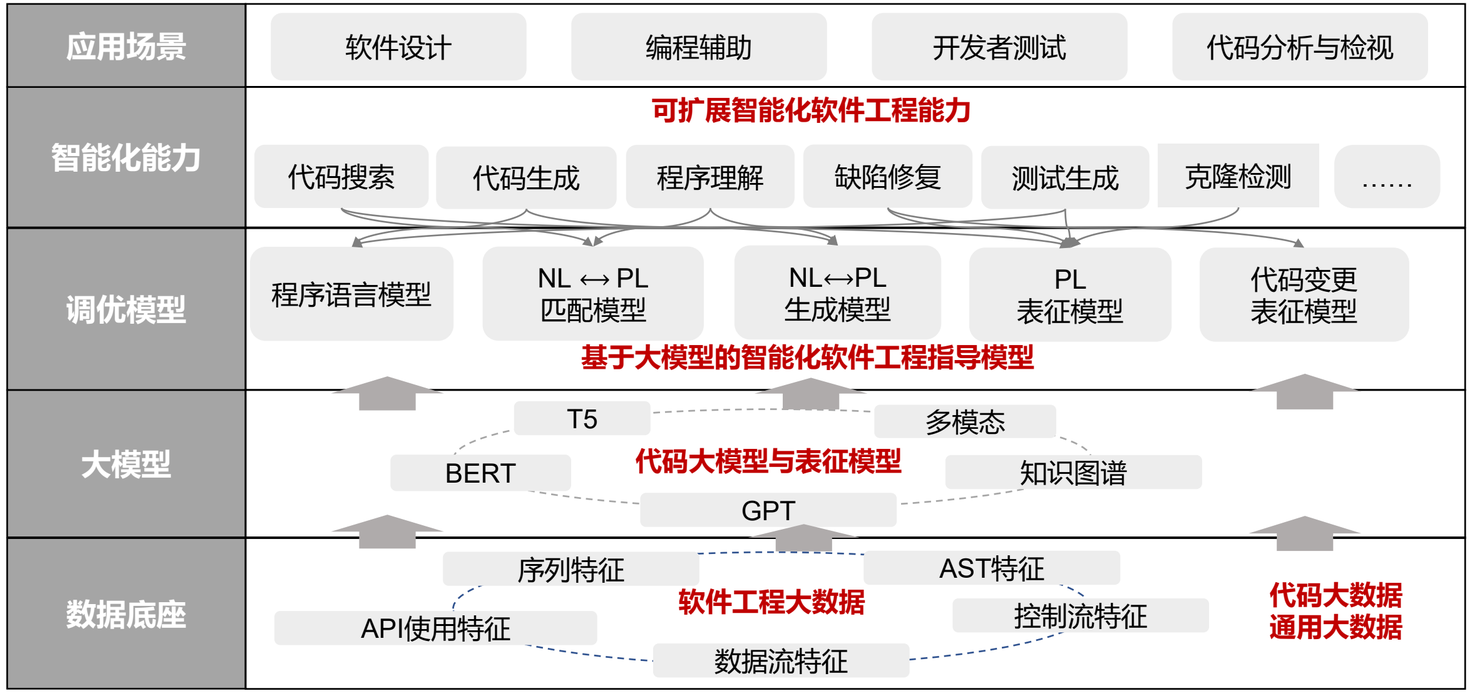
ChatGPT生成的测试用例具有**高可读性、完备的测试Oracle**

Models	Line Coverage	Condition Coverage
EvoSuite	23.1	3.8
AthenaTest	23.2	4.1
Text-davinci-002	25.3	6.1
Text-davinci-003	25.5	6.0
GPT-3.5-turbo	**26.2**	**6.7**

Listing 2 A test example generated by gpt-3.5-turbo.

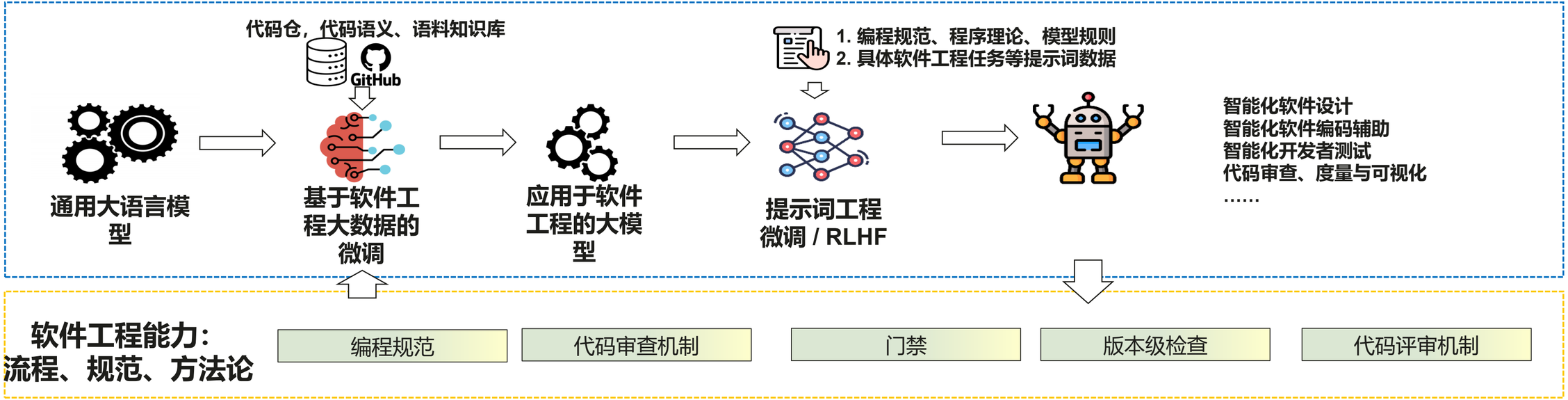
```
// Focal method from Defects4J
public static int toInt(final String str, final int defaultValue){
    if(str == null) { return defaultValue; }
    try {
        return Integer.parseInt(str);
    } catch (final NumberFormatException nfe) {
        return defaultValue;
    }
}
// Test generated by gpt-3.5-turbo
@Test public void testToInt() {
    // Test case 1: Valid input string
    String input1 = "123";
    int expectedOutput1 = 123;
    int actualOutput1 = NumberUtils.toInt(input1, 0);
    assertEquals(expectedOutput1, actualOutput1);

    // Test case 2: Null input string
    String input2 = null;
    int defaultValue2 = 0;
    int expectedOutput2 = defaultValue2;
    int actualOutput2 = NumberUtils.toInt(input2, defaultValue2);
    assertEquals(expectedOutput2, actualOutput2);

    // Test case 3: Invalid input string
    String input3 = "abc";
    int defaultValue3 = 0;
    int expectedOutput3 = defaultValue3;
    int actualOutput3 = NumberUtils.toInt(input3, defaultValue3);
    assertEquals(expectedOutput3, actualOutput3);
}
```

基于大语言模型的智能化软件工程体系规划蓝图



以通用大模型为基础，融入编程规范、程序理论、模型规则，构建软件工程专用大语言模型

充分结合编程规范、模型规则、程序分析、软件资产、社区知识，构筑智能化软件工程技术服务能力，提升编码效率与质量，达成人机协同开发软件



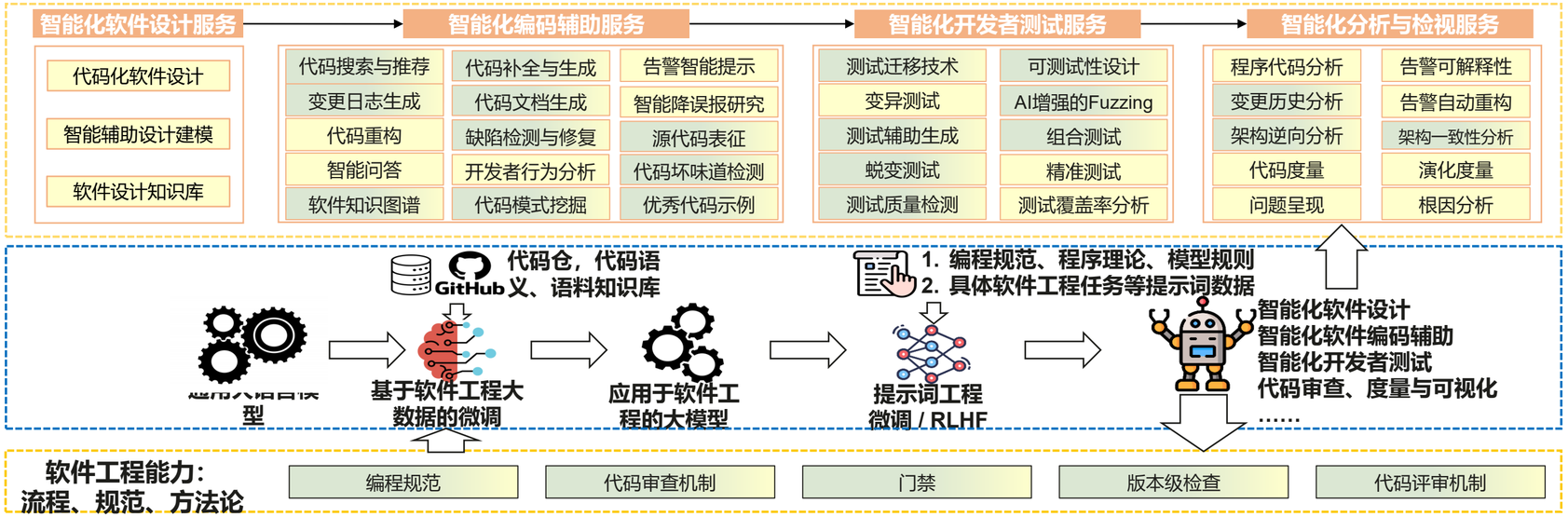
以通用大模型为基础，融入编程规范、程序理论、模型规则，构建软件工程专用大语言模型

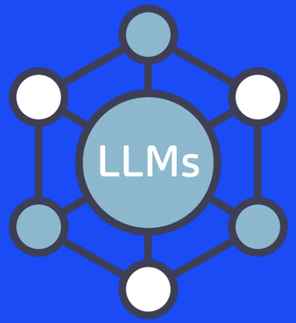
挑战

- ChatGPT在智能化辅助编程方向的取得可观进展，急需跟进国际先进技术
- OpenAI和微软拥有GitHub超大代码库资源，在数据上有一定差距
- 软件工程领域知识如何与大语言模型深度融合并改变现有软件开发范式，当前研究相对空白，具有很大技术挑战

目标

- 以通用大语言模型为基础，结合软件工程大数据，构建应用于软件工程的大模型
- 充分结合编程规范、程序理论、模型规则、程序分析、软件资产、社区知识，构筑智能化软件工程技术服务能力，提升编码效率与质量，达成人机协同开发软件





PART 03

数据底座

大模型数据底座--数字化软件资产

- **项目总目标：**通过收集全量开源软件资产数据、软件和AI物料清单、软件过程元数据，软件资产全量有序管理，可视可追溯、主动风险削减/防腐化，持续识别/挖掘推荐高价值资产，支持软件工程大语言模型的训练和调优

软件工程领域大语言模型

漏洞E2E工程能力

软件风控和合规

全量开源软件资产数据收集

- 基于Git的代码仓数据（WoC）
- 软件开发活动数据：邮件列表、Issue/PR
- 基于包管理器的开源生态元数据
- 其他开源数据：社交网络、漏洞、SO等

SBOM和AIBOM的构建

- SBOM验证
- AIBOM验证
- SBOM生成
- AIBOM生成

软件过程元数据

- 软件开发全过程元数据服务
- 软件开发全过程元数据采集

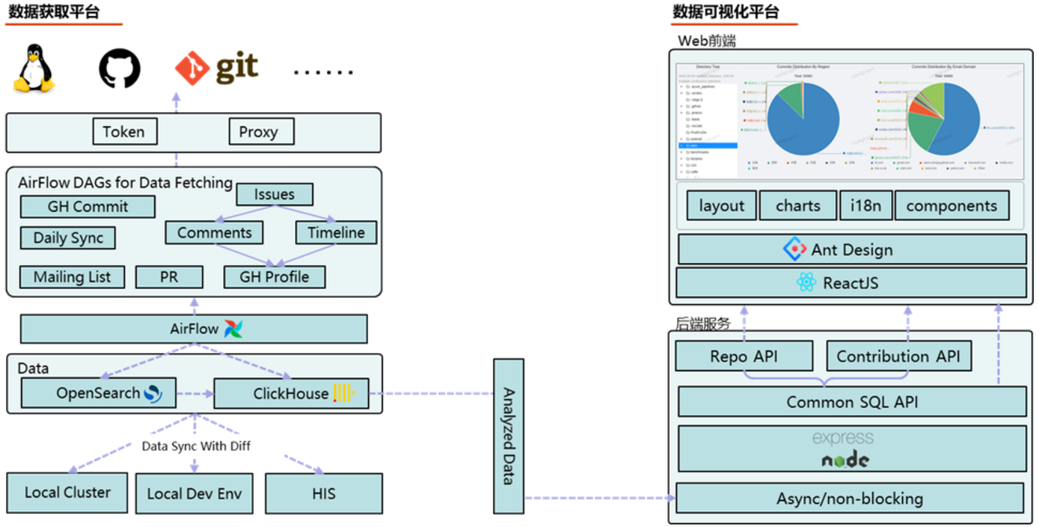
开源软件知识图谱

全量开源软件大数据采集与分析基础设施

全量开源代码数据集

完成了多源异构的开源数据收集收集技术，初步打通了“华为云-华为云对象存储服务（obs）-浙大”的数据链路，收集了Top 1000万的Github数据、7个主流开源生态元数据、GHArchive等数据集。

全量开源数据收集框架



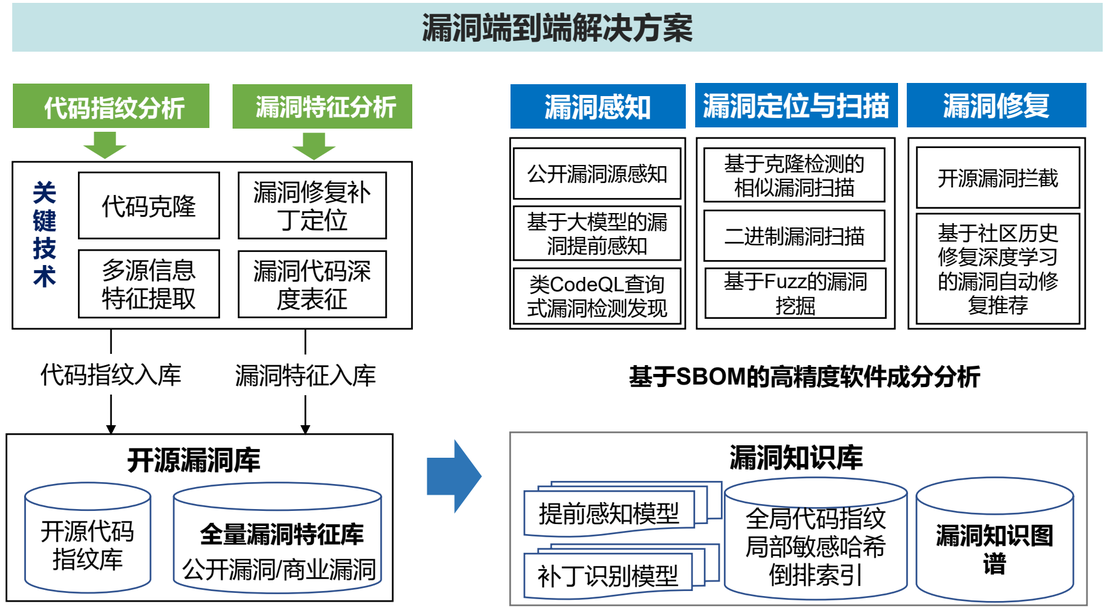
开源软件合规分析

系统分析了PyPI、Maven、NPM、RubyGems、和Cargo等多个主流包管理中的开源许可证使用情况，分析了**4000多万**开源软件包

	Maven	NPM	PyPI	RubyGems	Cargo
软件包数量	10,288,841	29,750,552	4,413,827	1,427,793	709,755
标准许可证使用率	28.71%	85.73%	71.16%	64.34%	97.27%
许可证变更次数	62,863	217,481	58,896	26,271	8,563
许可证兼容率	7.33%	4.14%	3.29%	7.17%	7.81%

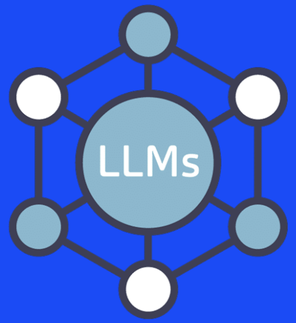
漏洞工程技术

构建漏洞端到端解决方案，利用代码指纹分析和漏洞特征分析技术，构建漏洞知识库，实现基于SBOM的高精度软件成分分析，支持漏洞感知、漏洞定位与扫描、漏洞修复等任务



- **漏洞感知**：提出面向代码变更和缺陷报告的漏洞提前感知技术，达成60%高危漏洞提前感知战略目标。
- **漏洞定位与扫描**：提出基于面向漏洞的SZZ算法，即时追溯定位引入漏洞的软件版本；构建基于克隆检测的高性能漏洞扫描技术，一千万行代码扫描时间5分钟（比业界领先工具SourcerCC快一倍）。
- **漏洞修复**：提出基于代码图结构的代码表示模型，识别代码中是否包含漏洞，从而实现函数级代码的漏洞识别技术。

更多信息，请参加周六下午胡星老师的“基于大语言模型的漏洞提前感知、检测和定位”



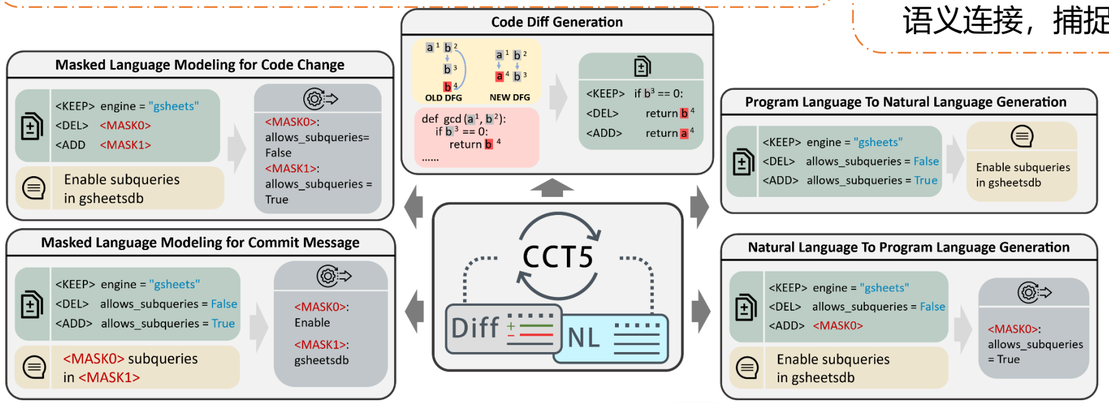
PART 04

软件工程大语言模型研究进展

基础理论与模型：面向代码变更的预训练模型

◆ 问题：
- 大量重要软件工程任务与代码变更的表征与生成相关。
- 是否能够构建面向代码变更的预训练模型以提升此类任务？

◆ 关键技术
- 采用Transformer Encoder-Decoder架构，为代码变更特别设计了五种预训练任务，能够帮助模型构建代码变更与自然语言之间的语义连接，捕捉代码变更中的语义信息。



设计的五种预训练任务：
- MLM for Code Change
- MLM for Commit Message
- Code Diff Generation
- PL to NL Generation
- NL to PL Generation

Dataset	NNGen	CodeT5	CodeReviewer	CCT5
$MCMD_{java}$	17.81	17.64	18.47	**20.80**
$MCMD_{C\#}$	22.92	18.76	20.36	**25.53**
$MCMD_{C++}$	13.69	14.41	15.94	**17.64**
$MCMD_{python}$	16.64	18.23	17.65	**21.37**
$MCMD_{js}$	18.03	21.53	19.84	**24.94**
Average	17.82	18.11	18.45	**22.06**

代码变更日志生成

Approach	Accuracy	GLEU
Toper	30.1	62.5
CoditT5	32.3	67.1
CodeReviewer	23.8	63.5
CodeT5	22.8	64.0
CCT5	**36.1**	**70.7**

即时注释更新

Approach	F1	AUC
JITLine	0.261	0.802
JITFine	0.431	0.881
CodeBERT (JITFine - EF)	0.375	0.856
CodeReviewer	0.341	0.802
CCT5	0.451	0.871
CCT5 + EF	**0.472**	**0.882**

即时缺陷预测

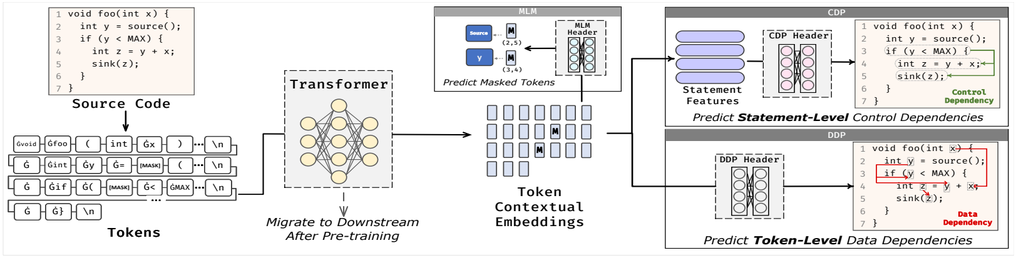
基础理论与模型：程序语义感知的预训练代码模型

◆ 问题：

- 程序的语义结构（例如代码中的控制依赖、数据依赖）对于分析和理解代码至关重要。但现有预训练代码模型都忽视了对于程序语义结构的学习。
- 如何让大模型学习和吸收关于程序语义结构的知识？

◆ 关键技术

- 提出两种新颖的预训练任务CDP和DDP，在预训练时基于输入代码片段分别预测代码中的控制依赖关系以及符号级数据依赖关系。
- 预训练了一种代码模型PDBERT，能够直接被用于分析代码中的依赖关系，也能提升程序语义敏感的下游任务的性能。



PDBERT的预训练策略

Dependency	Precision	Recall	F1-Score
Control	99.32	99.44	99.38
Data	89.24	97.60	93.23
Overall	92.66	98.26	95.37

程序依赖分析

CodeBERT	51.91	57.37	0.5030
GraphCodeBERT	54.74	59.82	0.5331
PDBERT			
MLM	54.49	60.42	0.5372
MLM+CDP	56.51	59.93	0.5325
MLM+DDP	56.93	62.07	0.5614
MLM+CDP+DDP	**57.96**	**62.60**	**0.5644**

漏洞CWE类别预测

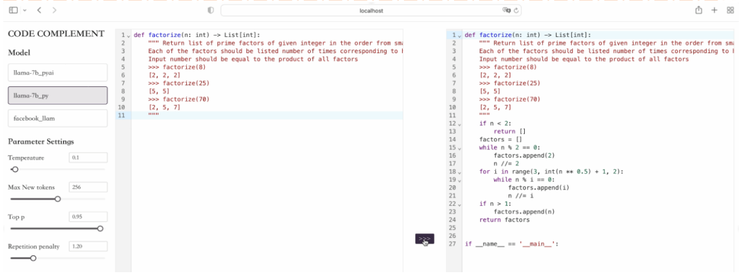
Model	Dataset		
	ReVeal (F1-Score)	Big-Vul (F1-Score)	Devign (Accuracy)
Bi-LSTM	34.25	33.11	61.24
Transformer	40.91	34.90	60.51
VulDeePecker	15.74	13.03	45.68
Devign	26.43	13.77	52.27
ReVeal	32.24	23.93	54.55
CodeBERT	44.27	54.48	62.08
GraphCodeBERT	45.03	54.06	64.02
DISCO†	46.4*	-	63.8
PDBERT			
MLM	44.65	55.99	65.52
MLM+CDP	45.93	56.28	66.29
MLM+DDP	47.42	58.51	65.85
MLM+CDP+DDP	**48.38**	**59.41**	**67.61**

漏洞检测

基于大语言模型的代码生成

◆ 基于自然语言大模型训练代码模型：

- **基础模型**：LLaMA 7B
- **数据集**：
  - 小规模Python AI代码数据集，5.8G
  - 大规模Python开源代码数据集，**173G**，约60B Tokens
- **数据处理**：
  - 高质量项目，过滤，去重，**删除评估数据集中的数据**
- **训练方式**：全量微调
- **训练精度**：混合精度，bf16
- **并行策略**：Fully Sharded Data Parallel （FSDP）



- LLaMA-7B-py达到了LLaMA-33B的Pass@1性能
- 中小规模模型利用代码数据微调后可以实现可观的表现。

◆ 评估数据集及方法：

- **30个AI相关问题**：基于初学者的PyTorch使用经验和StackOverFlow中PyTorch相关的热门问题构建。人工撰写相应的prompt。
  - **评估方法**：人工判断正确、部分正确、错误
- **HumanEval**，**MBPP**：广泛使用的Benchmark
  - **评估指标**：Pass@k

◆ 初步评估结果：

模型	正确	部分正确	错误
LLaMA-7B	15	2	13
LLaMA-7B-AI	**23**	**5**	**2**

数据集	模型	Pass@1
HumanEval	LLaMA-7B	10.5
	LLaMA-13B	15.8
	LLaMA-33B	21.7
	LLaMA-7B-py	**24.63**
MBPP	LLaMA-7B	17.7
	LLaMA-13B	22.0
	LLaMA-33B	**30.2**
	LLaMA-7B-py	29.62

基于代码大模型的单元测试样例生成

- **基础模型**：StarCoder
- **数据集**：Method2Test
  - 78万条 Focal method/ Unit test 对
  - Focal method 包含其所在类的上下文
- **训练方式**：LoRA微调
- **并行策略**：Distribute Data Parallel （DDP）
- **初步验证**：
  - 生成测试样例无语法错误
  - 部分生成测试样例与实际样例差别较大
  - 部分生成样例与实际样例几乎一致，简单修改后可直接编译运行

```
Generated:
@Test
public void testFindProcess() throws Exception {
    VirtualMachineHandler handler = new VirtualMachineHandler(new OptionsAndArgs(new String[] { "-pid", "1234" }));
    ProcessDescription desc = handler.findProcess(Pattern.compile(".*"));
    assertEquals("1234", desc.getId()); }<|endoftext|>


target_output: TestCase:
@Test
public void findProcess() throws Exception, NoSuchMethodException, IllegalAccessException {
    List<ProcessDescription> procs = filterOwnProcess(vmHandler.listProcesses());
    for (ProcessDescription desc : procs) {
        try { if (desc.getDisplay() != null && desc.getDisplay().length() > 0) {
            Pattern singleHitPattern = Pattern.compile("^" + Pattern.quote(desc.getDisplay()) + "$");
            assertTrue(tryAttach(singleHitPattern.pattern())); break; } }
        catch (Exception exp) {
            System.out.println("Cannot attach to " + desc.getDisplay() + " (" + desc.getId() + "): " + exp);
            exp.printStackTrace(); } }
        assertFalse(tryAttach("RobertMakClaudioPizarro",".*No.*process.*"));
        if (procs.size() >= 2) {
            assertFalse(tryAttach(".",procs.get(0).getId())); }
    }
```

（1）生成测试样例与实际样例差别较大

```
// generated
@Test(expected = IllegalArgumentException.class)
public void testSetObjectValue() throws Exception {
  EnumExtractor extractor = new EnumExtractor();
  extractor.setObjectValue(null, null, null, null); }


 // target_output:
@Test(expectedExceptions = IllegalArgumentException.class)
public void setValue() throws InvocationTargetException, IllegalAccessException {
  enumExtractor.setObjectValue(null,null,null,null); }
```

（2）生成测试样例与实际样例一致

```
@Test
public void testAngle() {
    final Angle angle = Angle.fromDeg(180);
    assertEquals(180, angle.getAsDouble(), 0.000001);
}
```

(1)生成样例调用了不存在的API

```
@Test
public void testAngle() {
    final Angle angle = Angle.Deg.of(180);
    assertEquals(180, angle.getAsDouble(), 0.000001);
}
```

(2)修改为正确API即可运行

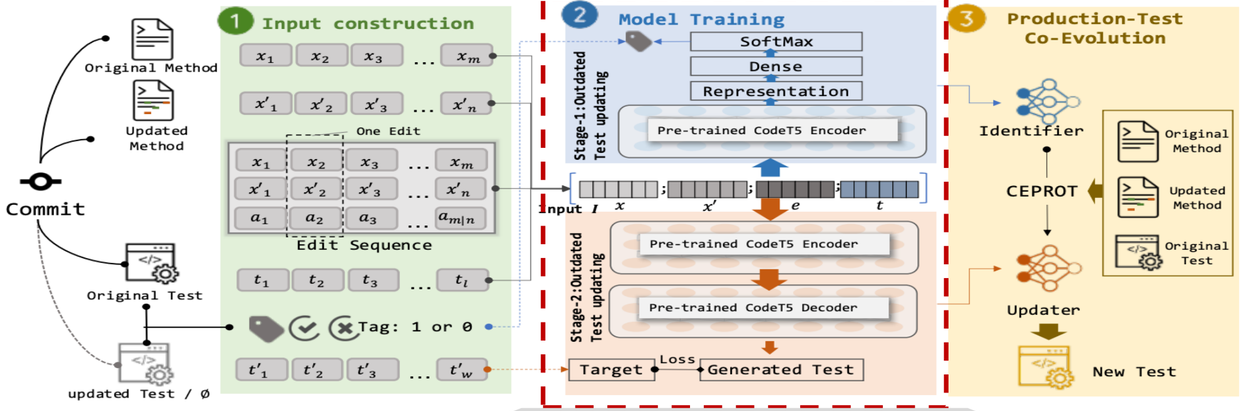
```
[INFO] Running org.apache.commons.numbers.angle.AngleTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 s - in org.apache.commons.numbers.angle.AngleTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.151 s
[INFO] Finished at: 2023-08-09T09:36:20+08:00
```

（3）在Apache 开源项目Commons-numbers中生成测试样例，经简单修改后即可编译运行

基于大规模预训练技术的测试代码智能演化技术

◆问题：。

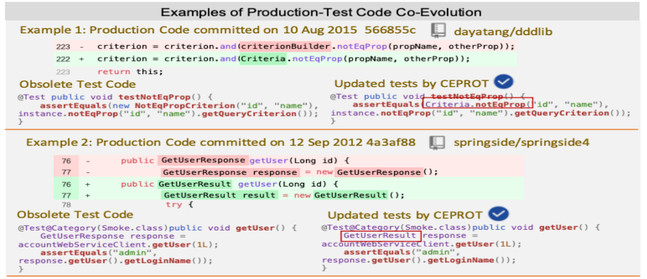
开发人员在进行生产代码更新时，经常忽略测试代码的更新与维护，为软件带来了极大的安全隐患。现有的测试生成工具大多是为测试代码生成新的测试案例，而无法对已有的测试案例进行维护



大规模预训练模型可以有效的学习生产代码变更，更好的实现测试代码的维护

	Obsolete Test Identification			Obsolete Test Updating	
Approaches	Precision	Recall	F1	CodeBLEU	Accuracy
KNN	83.4%	72.3%	77.5%	37.6%	3.9%
SITAR	78.3%	38.9%	52.0%	-	-
NMT	96.3%	89.2%	92.6%	32.3%	5.0%
CEPROT	**98.3%**	**90.0%**	**94.0%**	**63.1%**	**12.3%**

Approach	Compilability	Cov.	Quality	Co-Evolvability
Obsolete Test	40%	18.1%	4.53	6%
KNN	44%	6.5%	3.11	4%
NMT	30%	22.2%	2.22	22%
CEPROT	**48%**	**34.2%**	**3.81**	**38%**



基于大模型的测试代码演化模型就可以有效的检测过时的测试案例，以及大幅度提升了现有测试案例的覆盖率以及演化性。

PART 05

结论和展望

未来已来：以大语言模型为底座，构建人机物融合的下一代软件工程能力

主要挑战

1. 如何训练软件工程大语言模型，达到与ChatGPT类似甚至超越ChatGPT的能力？

2. 如何将软件工程大语言模型融合到开发过程中，针对项目定制和微调，从而生成适配项目的代码和测试用例？

3. 软件工程大语言模型如何与传统软件工程技术（如程序分析等）融合，更好地进行代码分析与检视？

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

THANKS