

# 大语言模型在软件工程领域的 技术洞察

---

报告人：文 成

西安电子科技大学

2023年04月12日



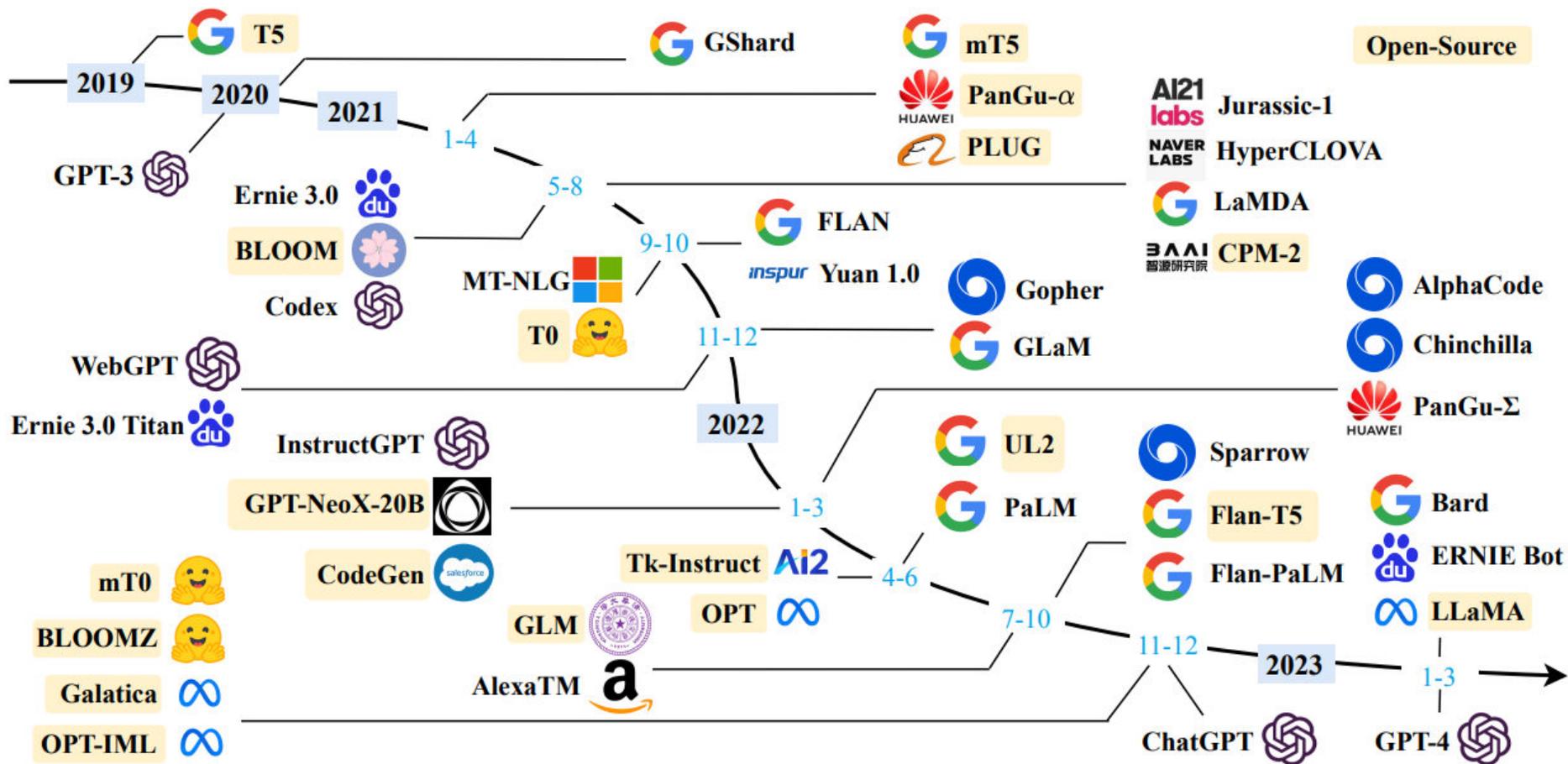
# 目 录

CONTENTS

- 01 **大语言模型：背景与介绍**
- 02 基于LLMs的近期研究工作
  - 软件测试
  - 缺陷复现
  - 缺陷修复
  - 面向大语言模型编程
- 03 总结与展望

# 大语言模型 ( Large Language Models , LLM )

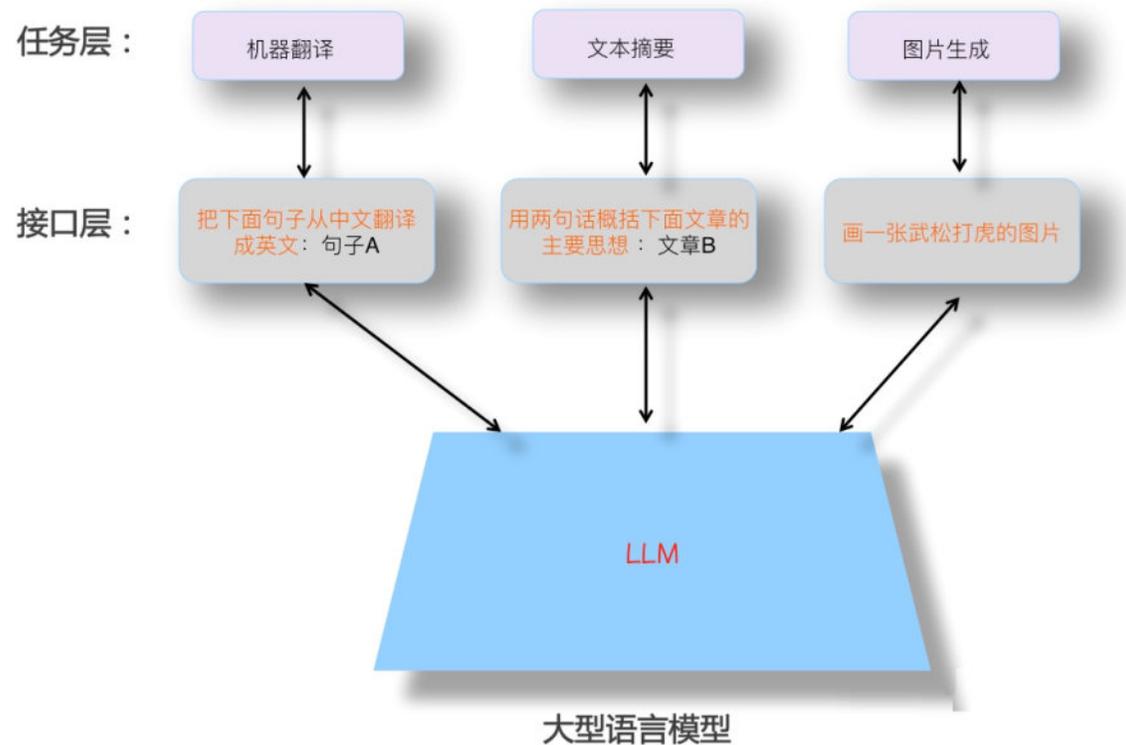
- 大型语言模型 ( LLM ) 是指包含数十亿 ( 或更多 ) 参数的语言模型 , 这些参数是在大量文本数据上训练的 , 它很大程度上扩展了模型大小、预训练数据和总计算量 ( 扩大倍数 ) 。
- LLM可以更好地理解 and 生成自然语言 , 根据给定的上下文 ( 例如 prompt ) 生成高质量的文本 , 并表现出一些小语言模型中不存在的涌现能力 , 例如上下文学习、指令遵循和循序渐进的推理。



# 通过预训练大语言模型和提示词完成下游任务

- 单一领域数据集；诸多数据集和诸多模型各成孤岛缺乏纵效；劳动密集型的数据标注。

- 超级海量数据；一个具有跨领域只是的“基础模型”，能执行五花八门的任务



大语言模型越来越受欢迎的主要原因：

- **性能提升**：大规模的模型能够学习复杂的语言模式，各种任务中表现出色，比以前的方法更准确、更流畅。
- **多功能性**：大语言模型可以做多种任务，不需要特定的架构或模型，可用于生成、翻译、总结等，适用于各种应用。
- **迁移学习**：大语言模型可以针对特定的任务进行微调，利用其语言理解，快速适应新的领域。这种迁移学习减少了对数据和训练时间的需求
- **高互动性**：大语言模型能够理解和产生类似人类的反应，与人工智能系统进行更自然和直观的互动，为人工智能工具和应用提供了新的可能性

# 大语言模型对于软件开发的辅助能力

- 通过与大模型对话完成需求分析、软件设计、代码实现、软件测试、代码重构、缺陷检查与修复等各种开发任务，甚至尝试用大模型实现端到端的软件应用开发。

WE 我们现在需要搭建一个基于Spring框架的教务管理系统，你能否帮我们尽可能完善地进行功能和非功能的需求设计

当然可以，我会帮助您列出一些关于基于Spring框架的教务管理系统的功能和非功能需求。这里是一个概要的需求列表，您可以根据实际情况进行调整和完善。

### 一、功能需求

1. 用户管理
  - 教师、学生、管理员的注册与登录
  - 个人信息管理（修改密码、更新资料等）
2. 课程管理
  - 课程的增加、删除、修改和查询
  - 课程分类与归档
  - 课程表生成和查看
3. 成绩管理
  - 学生成绩录入、修改和查询
  - 成绩统计与分析（平均分、最高分、最低分等）
  - 成绩导出（Excel、PDF等格式）
4. 教师管理
  - 教师信息录入、修改和查询
  - 教师课程分配
  - 教师排课安排
5. 学生管理
  - 学生信息录入、修改和查询
  - 学生选课、退课
  - 学生课程表查看
6. 通知公告
  - 发布、修改和删除通知公告
  - 按时间、关键词等方式筛选通知公告

需求生成、需求分析

WE 我们现在需要搭建一个基于Spring框架的教务管理系统，请写一个教务管理系统的html网页代码

以下是一个简单的基于Bootstrap的教务管理系统HTML网页代码。这个示例包括了导航栏、侧边栏和内容区域。您可以根据实际需求进一步完善和修改。

```
html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>教务管理系统</title>
  <!-- 引入Bootstrap CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <!-- 引入jQuery -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <!-- 引入Bootstrap JS -->
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>
  <!-- 导航栏 -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">教务管理系统</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" data-ks="true">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item active">
          <a class="nav-link" href="#">首页 <span class="sr-only">(current)</span>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">课程管理
        </li>
      </ul>
    </div>
  </nav>
  <div class="container">
    <h2>教务管理系统</h2>
  </div>
</body>
</html>
```

Stop generating

代码智能生成

WE 你现在是代码安全分析专家和网络安全专家。回答下列代码是否存在漏洞:

```
<?php
class File {
  public function flag() {
    $this->innocent();
  }
  public function innocent() {
    echo "Aquif no pasa nada :D\n";
  }
}
class GiveFlag extends File {
  public $offset = 23;
  public function innocent() {
    $stuff = fopen(flag.txt, "r");
    fseek($stuff $this->offset);
    print fread($stuff, filesize(flag.txt));
  }
}
class entry {
  public function destruct() {
    $this->awesome->flag();
  }
}
unserialize($argv[1]);
?>
```

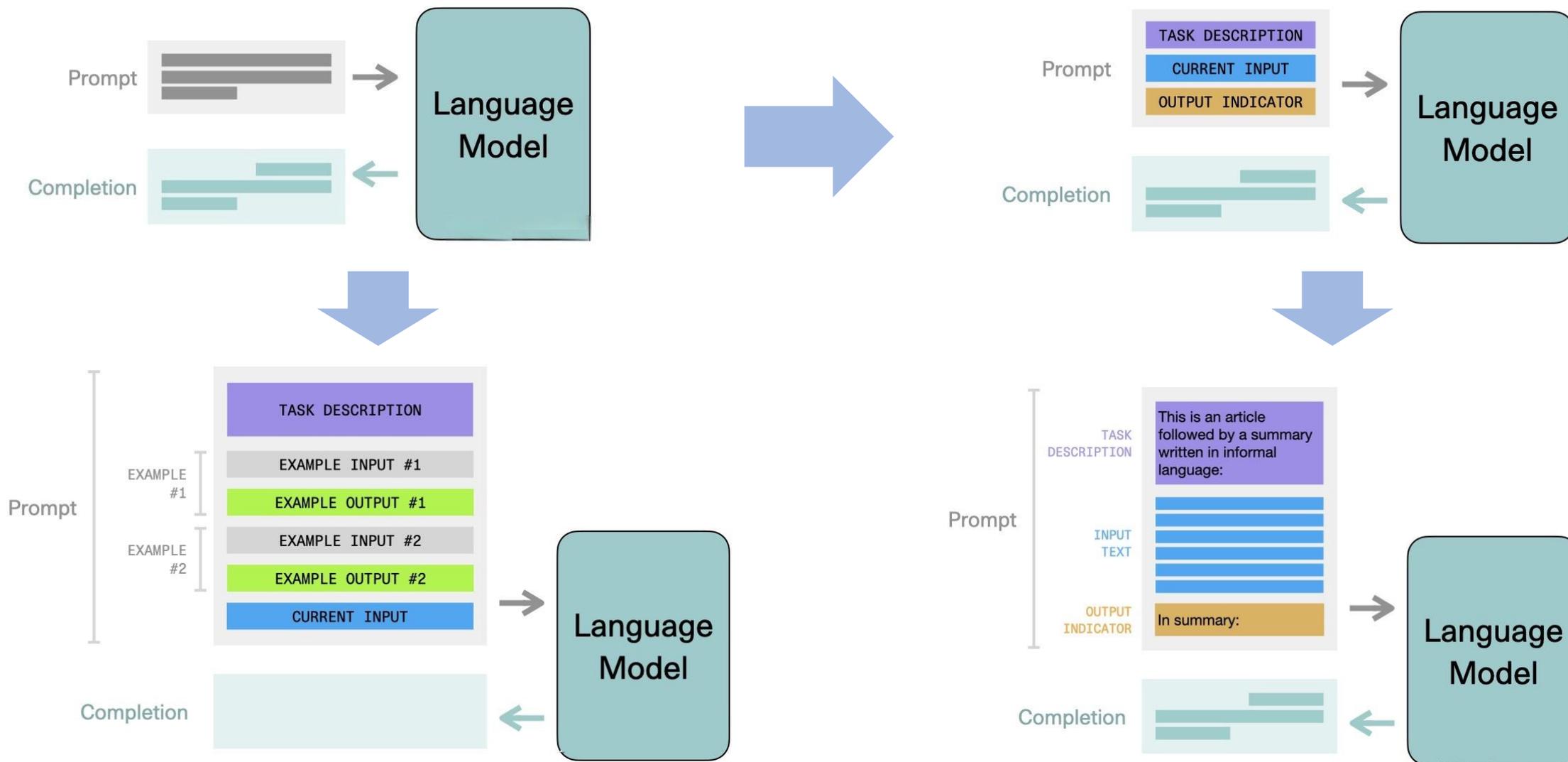
是的，这段PHP代码存在漏洞。以下是其中的几个安全问题：

1. 不安全的反序列化：使用`unserialize()`函数处理用户输入，例如`\$argv[1]`，可能导致远程代码执行或其他安全漏洞。在这种情况下，攻击者可以创建并提交精心构造的序列化对象来触发任意方法或代码执行。
2. 不安全的文件访问：`GiveFlag`类中的`innocent()`方法使用`fopen()`函数打开一个名为`flag.txt`的文件。这可能导致攻击者访问或泄漏文件中的敏感数据。在这种情况下，一个攻击者可能通过利用不安全的反序列化来触发这个方法，进而泄露`flag.txt`。

代码审计、缺陷检测

# 提示工程 ( Prompt Engineering )

- 提示工程是一种使用大型语言模型 ( LLM ) 执行各种任务的方法，通过提供问题、指令、示例等信息，引导模型生成更准确、可靠、符合预期的输出文本。提示工程的目的是优化输入提示，发挥LLM的优势和价值。

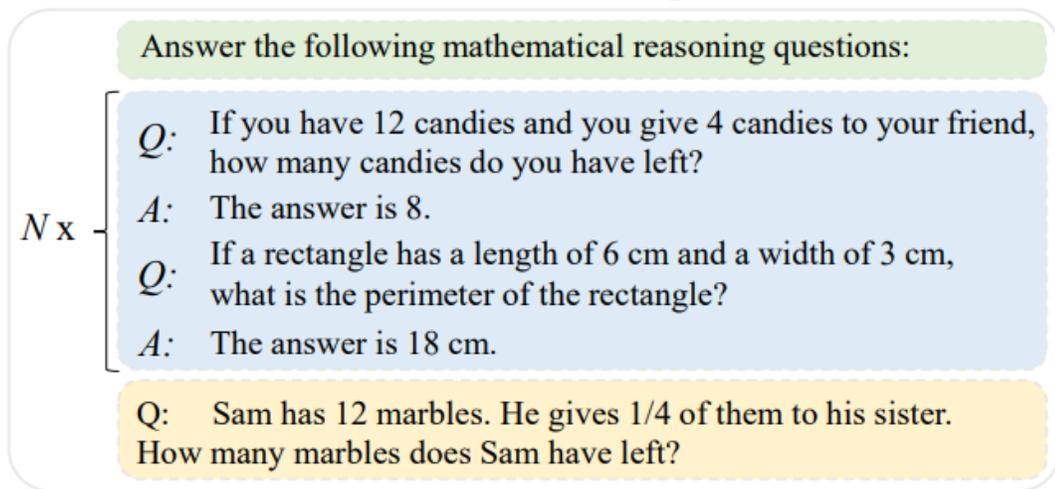


# 上下文学习 (ICL) 和思维链 (CoT)

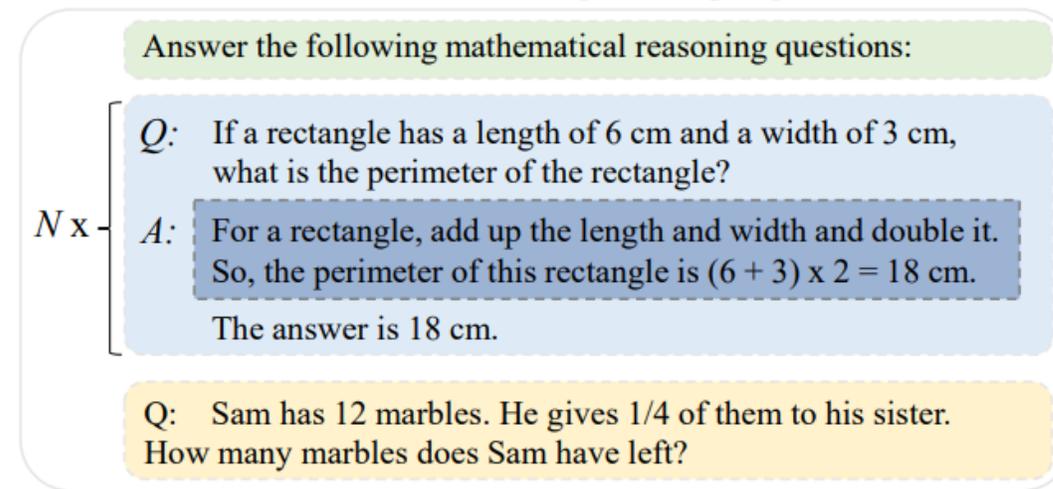
## 高级提示方法：

- Few-shot Prompting ([Brown et al. 2020](#))
- Zero-shot Prompting ([Wei et al. 2022](#))
- Chain-of-Thought Prompting ([Wei et al. 2022](#))
- Self-Consistency ([Wang et al. 2022](#))
- Generate Knowledge Prompting ([Liu et al. 2022](#))
- Automatic Prompt Engineer ([Zhou et al. 2022](#))
- Active-Prompt ([Diao et al. 2023](#))
- Directional Stimulus Prompting ([Li et al. 2023](#))
- ReAct ([Yao et al. 2022](#))

### In-Context Learning



### Chain-of-Thought Prompting



■ : Task description   ■ : Demonstration   ■ : Chain-of-Thought   ■ : Query



# 目 录

## CONTENTS

- 01 大语言模型：背景与介绍
- 02 **基于LLMs的近期研究工作**
  - 软件测试
  - 缺陷复现
  - 缺陷修复
  - 面向大语言模型编程
- 03 总结与展望

# 代码生成预训练模型概括

专门进行智能化代码生成任务的预训练模型：

预训练模型	研发机构	骨干模型	模型参数规模	训练数据量	训练语言
CodeT5	Salesforce Research 2021	T5	60M/223M/770M	8.35G	Ruby/JavaScript/Go/Python/Java/PHP/C/C#
CodeGEN	Salesforce Research 2022		350M/2.7B/6.1B/16.1B	82.5G	C/C++/Go/Java/JavaScript/Python/
InCoder	Facebook AI Research 2022		1.3B/6.7B	159G	Python and 28 other languages
<b>CodeX</b>	<b>OpenAI 2021</b>	<b>GPT-3</b>	<b>300M/2.5B/12B</b>	<b>159G</b>	<b>Python</b>
AlphaCode	DeepMind 2022		300M/1B/3B/9B/41B	715.1G	C++/C#/Java/JavaScript/Lua/PHP/Python/Ruby/Go/Rust/
PanGu-Coder	华为 2022	PanGu-alpha	317M/2.6B	147G	Python
CodeGeeX	清华大学 2022		13B		C++/Python/C/Java/JavaScript/Go/Html/PHP/Shell/CSS
aiXcoder L	aiXcoder 2020	GPT2	1.3B		Java
aiXcoder XL	aiXcoder 2022	自研	13B		Java

- 代码生成 ( Code Generation ) : 根据自然语言描述的需求, 机器生成特定编程语言的代码片段;
- 代码补全 ( Code Completion ) : 根据已编写代码的上文自动理解开发人员的编写意图并补全代码

# LLMs在等SE、PL、SEC领域的应用研究

## • 软件测试

- Large Language Models are Zero-Shot Fuzzers:Fuzzing Deep-Learning Libraries via Large Language Models (*ISSTA 2023*)
- CodaMOSA: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models (*ICSE 2023*)

## • 缺陷复现

- Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction (*ICSE 2023*)

## • 缺陷修复

- Examining Zero-Shot Vulnerability Repair with Large Language Models (*S&P 2023*)
- Automated Program Repair in the Era of Large Pre-trained Language Models (*ICSE 2023*)
- Automated Repair of Programs from Large Language Models (*ICSE 2023*)
- Keep the Conversation Going: Fixing 162 out of 337 bugs for \$0.42 each using ChatGPT (*arXiv*)

## • 程序设计语言

- Prompting Is Programming: A Query Language For Large Language Models (*PLDI 2023*)



# 目 录

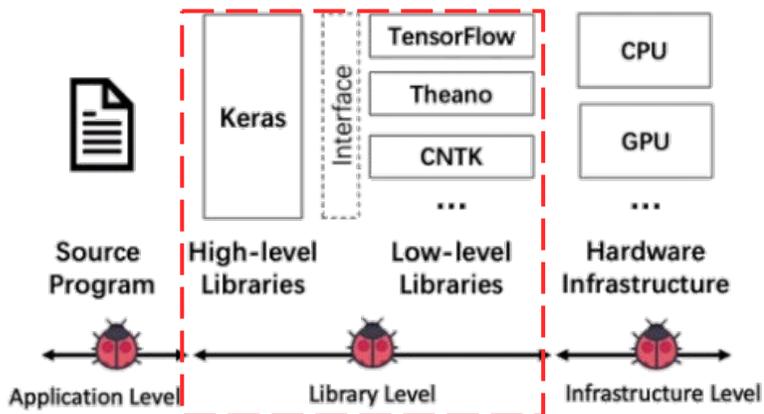
CONTENTS

- 01 大语言模型：背景与介绍
- 02 **基于LLMs的近期研究工作**
  - **软件测试**
  - 缺陷复现
  - 缺陷修复
  - 面向大语言模型编程
- 03 总结与展望

# TitanFuzz: 利用LLMs自动生成有效的输入程序以测试DL库 (ISSTA' 23)

## 背景介绍:

- 深度学习 (DL) 系统可以划分为应用、基础软件库和硬件设施三个层级。
- DL库 (如TensorFlow和PyTorch) 中是否存在缺陷对下游应用层的正确性和可靠性至关重要。



## 问题挑战:

- 测试DL库需要通过python编程调用库中的API
- 需提供API调用序列的多样性, 保证测试覆盖率
- 输入程序需要同时满足Python语言的语法/语义和DL库API的输入/类型约束

```
x = torch.randn(3, 3, dtype=torch.float32)
print("Intermediate: ", torch.log(x * 2 - 1)) # Intermediate contains NaN
output = torch.matrix_exp(torch.log(x * 2 - 1)) # on CPU: Contains NaN
x = x.cuda()
output = torch.matrix_exp(torch.log(x * 2 - 1)) # on GPU: Does not contain NaN
```

a)

```
x = [-0, 0, -0] # generated by previous statements
y = 1 / torch.clamp(x, min=0, max=1)
# on CPU y = 1 / [-0, 0, -0] = [-Inf, Inf, -Inf] wrong!
# on GPU y = 1 / [0, 0, 0] = [Inf, Inf, Inf] correct!
```

b)

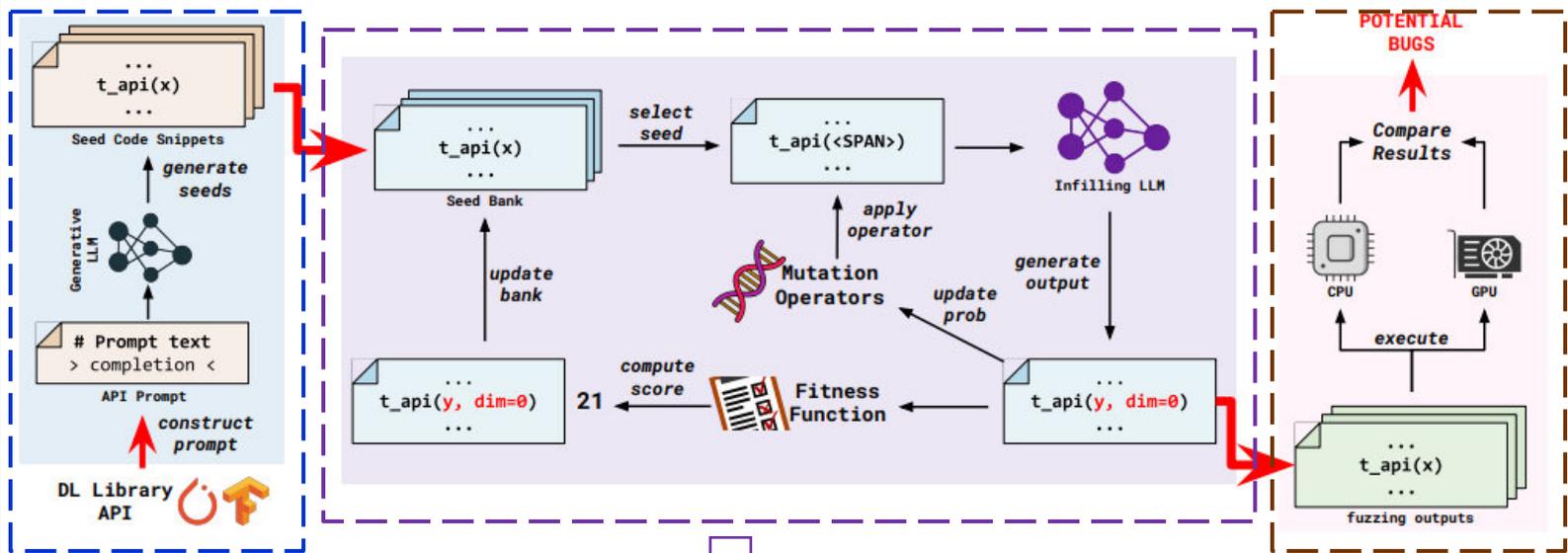
发现错误的输入程序示例

**关键技术:** 利用两种大型语言模型 (LLMs) 来生成有效的输入程序以对DL库进行模糊测试

- 生成型LLMs (例如Codex): 可以从头开始生成完整的DL程序
- 填充型LLMs (例如InCoder): 可以在给定一个部分DL程序的情况下, 填充缺失的代码片段

## 主要步骤:

1. 准备两类预训练好的生成型LLM和填充型的LLM (已从数十亿个代码片段中隐式地学习编程语言的语法/语义和复杂的API约束)



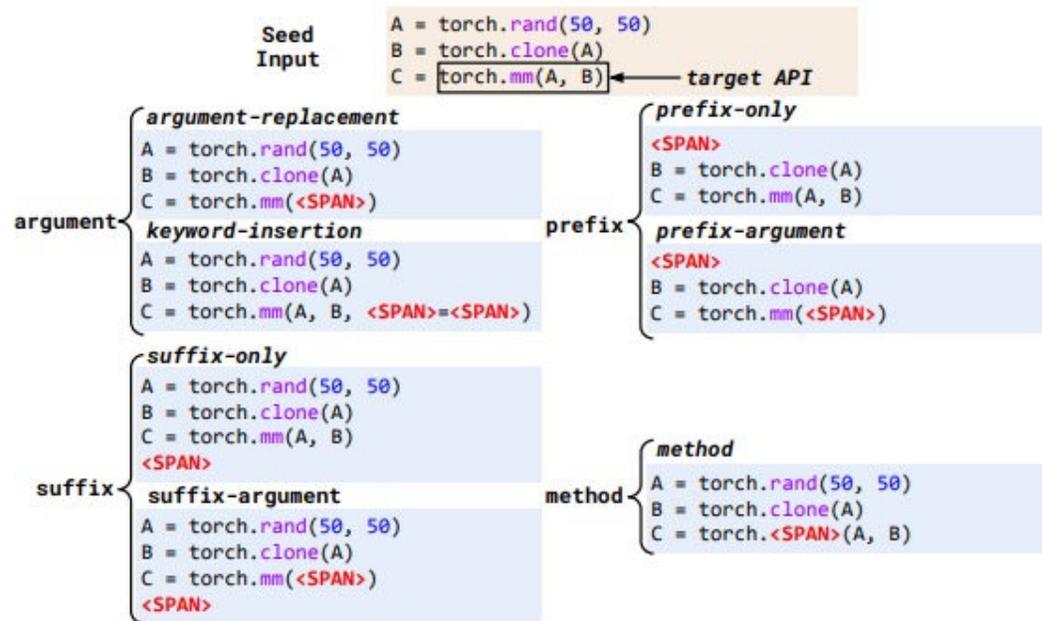
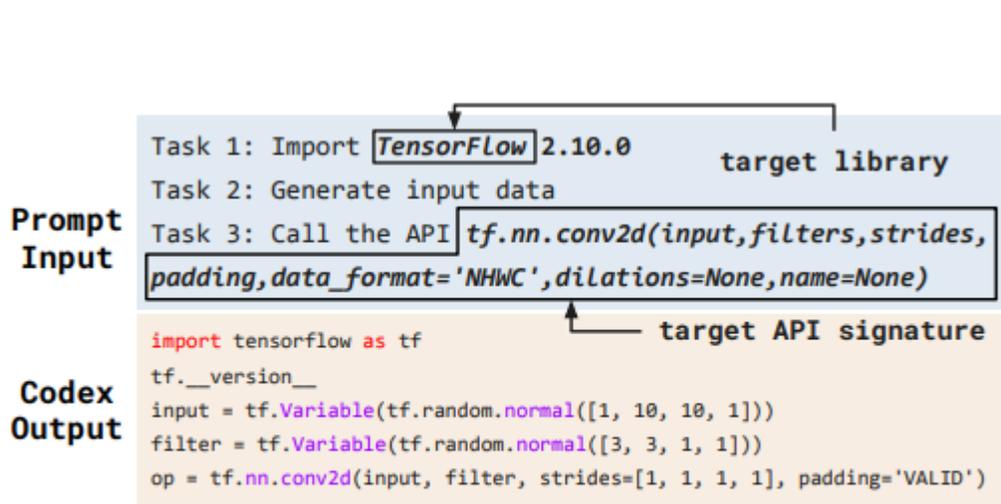
2. 利用生成型的LLM来生成有效的、高质量的输入程序作为初始种子集

3. 利用填充型的LLM对种子进行变异以生成更多样的输入程序对DL库进行测试

4. 输入程序在不同后端 (如CPU和GPU) 上运行结果的一致性可作为测试预言来判定潜在的错误

# TitanFuzz: 利用LLMs自动生成有效的输入程序以测试DL库 (ISSTA' 23)

- 生成型的LLM用于生成一语法/语义正确的输入程序，其中包含不同的API调用序列，这些输入程序可以作为模糊测试的初始种子集。
- 填充型的LLM用于在模糊测试过程中对已有的种子进行变异，以生成更多样的输入对DL程序进行模糊测试。



Mutation operators outputs (inputs for the model)

## 适应度函数

- 数据流图的深度：执行路径在数据流图中覆盖的边数量的最大值 (D)
- API调用的数量：唯一API调用的数量 (U)，以及相同参数的重复API调动的数量 (R)

$$\text{FitnessFunction}(C) = D + U - R$$

# TitanFuzz 实验评估

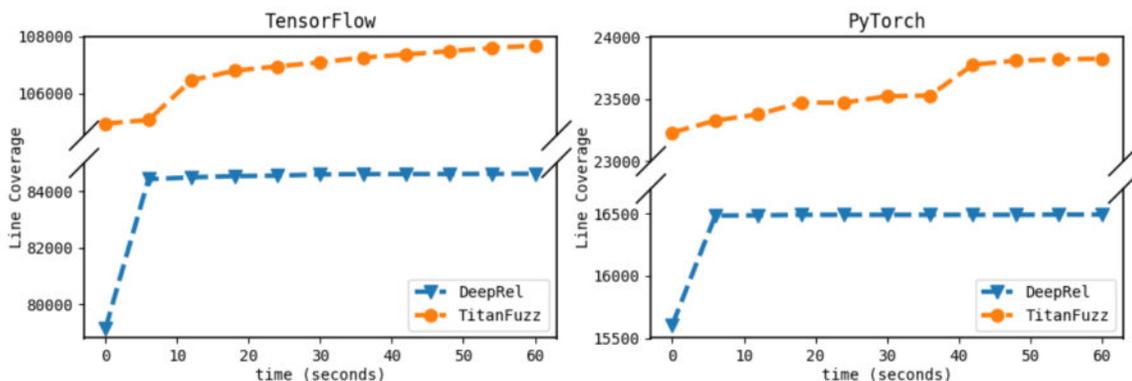
1. TitanFuzz在TensorFlow/PyTorch上的代码覆盖率比现有的技术高出30.38%/50.84%

	TITANFUZZ	DeepREL	FreeFuzz	Muffin	LEMON	Total
TensorFlow	2215	1159	581	79	35	3316
PyTorch	1329	1071	468	-	-	1593

覆盖的DL库APIs的数量对比

	PyTorch		TensorFlow	
	Coverage	Time	Coverage	Time
DeepREL	15794 (13.91%)	5.1h	82592 (30.65%)	9.3h
Muffin	-	-	79283 (29.42%)	6.8h
TITANFUZZ-seed-only (w/ DeepREL APIs)	18447 (16.25%)	3.4h	89048 (33.05%)	4.9h
TITANFUZZ-seed-only (w/ all APIs)	22584 (19.89%)	5.1h	103054 (38.35%)	11.9h
TITANFUZZ	23823 (20.98%)	9.9h	107685 (39.97%)	21.1h

与当前最先进技术的对比



2. 虽然Codex可以提供高质量的种子程序，但与较小的InCoder模型相比，它的运行速度相对较慢，填充型LLM对于功能强大但成本高昂的生成型LLM来说，起着重要的补充和增强作用

Library	Model	# Unique Prog. per API		Time per Prog. (s)	
		Valid	All	Valid	All
PyTorch	Codex	13.55	23.16	0.82	0.48
	INCODER	92.38	450.68	0.51	0.10
TensorFlow	Codex	6.85	22.26	1.69	0.52
	INCODER	67.17	358.06	0.67	0.13

Codex和InCoder生成测试用例的质量与效率对比

3. TitanFuzz新发现了65个缺陷，包括41个缺陷已确认的未知缺陷。

	Total	Confirmed	Unknown (Fixed)	Rejected
PyTorch	37	31	23 (6)	5
TensorFlow	28	22	18 (2)	3
Total	65	53	41 (8)	8

新发现的缺陷

# CodaMosa : 利用大语言模型在测试生成中突破覆盖率停滞 ( ICSE' 23 )

## 背景介绍 :

- 基于搜索的软件测试 ( SBST ) 通过结合测试用例的生成和变异, 为被测程序生成高覆盖的测试用例集。
- SBST的性能取决于是否能以合理的概率生成能够执行被测程序的核心逻辑的测试用例。

## 问题挑战 :

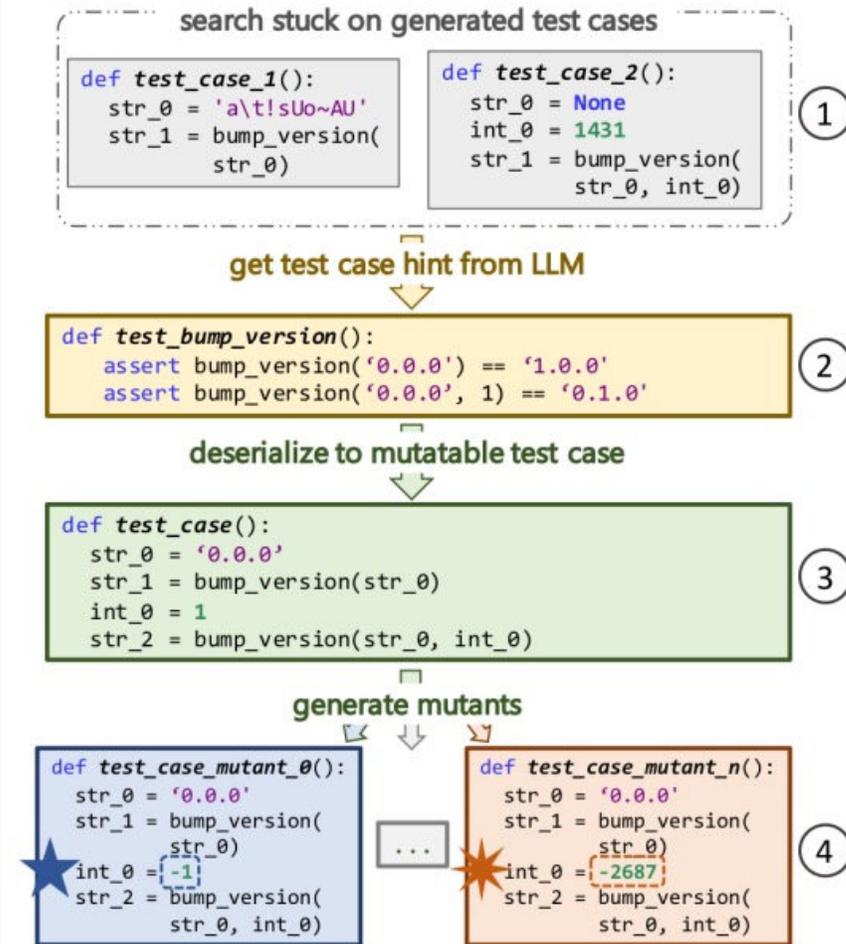
- 如果测试用例不能按照预期的方式调用程序被测的函数, 那么通过变异测试用例来提高覆盖率的概率就会非常低, 导致覆盖率停滞。

```
<...omitted code...>
Module under Test
def _build_version_bump_position(pos: int) -> int:
    pos_min = -3
    pos_max = 2
    if (pos_min <= pos <= pos_max) is False:
        raise ValueError("Invalid position")
    # Turn position into a positive number
    if pos < 0:
        pos_max += 1
        return pos_max + pos
    return pos
<...omitted code...>
def bump_version(version: str, pos: int = 2,
                 pre_release: Optional[str]= None) -> str:
    ver_info = _build_version_info(version)
    pos = _build_version_bump_position(pos)
    bump_type = _build_version_bump_type(pos, pre_release)
    <...omitted code...>
    return out
```

示例程序

技术手段 : 结合基于搜索的软件测试 ( SBST ) 和大语言模型 ( LLM ) 的优势

- 当SBST遇到覆盖率停滞时, 使用LLM生成针对低覆盖率函数的测试用例
- 将LLM生成的测试用例反序列化为SBST的内部表示, 并对其进行变异和评估



## 主要步骤 :

① 使用SBST技术生成一些随机测试用例, 通常很快会达到覆盖率停滞

② 通过查询LLM ( 即Codex ) 来生成一个目标函数的测试用例; 文章设计了一种提示生成的方法来控制LLM的输出

③ 将LLM生成的测试用例反序列化为SBST可以变异和评估的格式

④ 变异测试用例, 探索更多的程序行为; 成功突破覆盖率停滞

关键实验结论 : 在486个Python模块上, CodaMosa > SBST (即Mosa) ≈ LLM-only (即Codex)



# 目 录

## CONTENTS

- 01 大语言模型：背景与介绍
- 02 **基于LLMs的近期研究工作**
  - 软件测试
  - **缺陷复现**
  - 缺陷修复
  - 面向大语言模型编程
- 03 总结与展望

# LIBRO: 利用LLMs复现软件Bug ( ICSE' 23 )

**背景介绍：**开发者通常需要根据Bug报告来复现Bug，以帮助快速定位和修复Bug；但根据Bug报告来构造一个能复现Bug的测试用例是十分繁琐且困难的。

Issue No.	MATH-370 <sup>1</sup>
Title	NaN in “equals” methods
Description	In “MathUtils”, some “equals” methods will return true if both argument are NaN. Unless I’m mistaken, this contradicts the IEEE standard. If nobody objects, I’m going to make the changes.

Bug报告示例

**问题挑战：**如何从一般性的Bug报告中自动化地生成能够复现Bug的测试用例？

```
1 public void testEquals() {
2     assertFalse(MathUtils.equals(Double.NaN, Double.NaN));
3     assertFalse(MathUtils.equals(Float.NaN, Float.NaN));
4 }
```

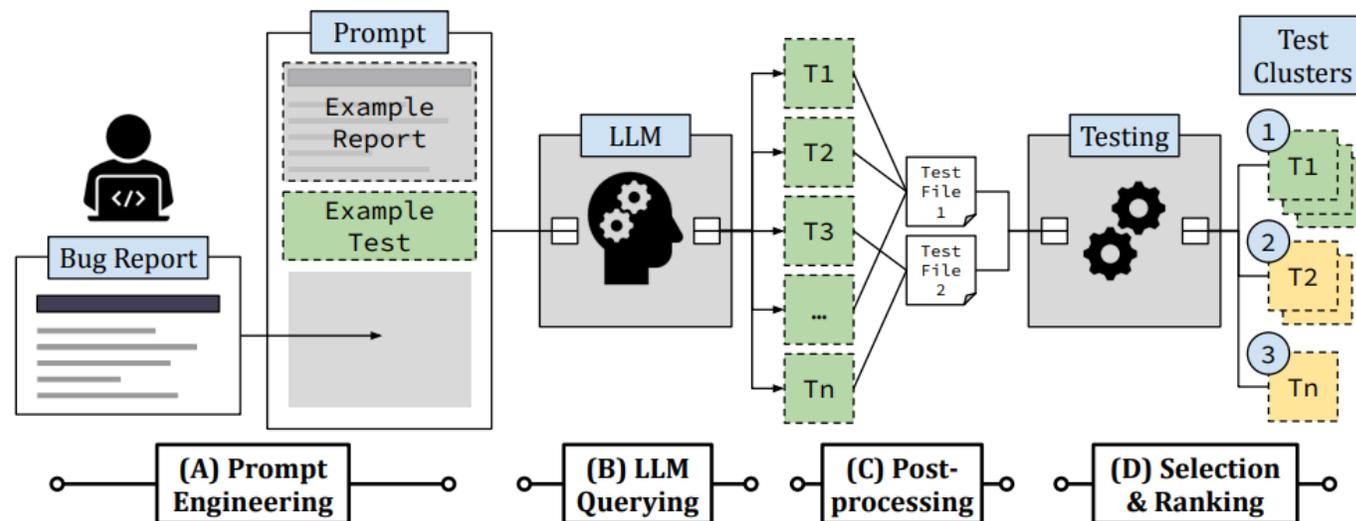
LLM生成的测试用例

```
1 public final class MathUtilsTest extends TestCase {
2     ...
3     public void testEquals() {
4         assertFalse(MathUtils.equals(Double.NaN, Double.NaN));
5         assertFalse(MathUtils.equals(Float.NaN, Float.NaN));
6     }
7     ...
```

经过后处理能复现Bug的测试用例

**主要贡献：**

- 首次尝试使用LLM来从一般的Bug报告中生成测试用例，而不仅限于崩溃Bug
- 提出了一套有效的后处理和排序方法，使得LLM生成的测试用例能够在实际场景中帮助开发者定位和修复Bug。



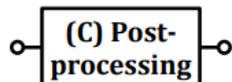
**方法流程：**

- 使用OpenAI Codex作为LLM，利用Markdown文档（Bug报告+样例）作为提示词
- 查询LLM，生成多个测试用例的候选
- 结合项目源代码和测试套件，使用后处理步骤来解决依赖问题
- 最后，根据一些启发式规则来对测试用例进行评估和排序

# LIBRO: 利用LLMs复现软件Bug ( ICSE' 23 )

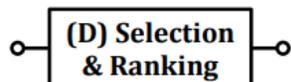
Issue No.	MATH-370 <sup>1</sup>
Title	NaN in "equals" methods
Description	In "MathUtils", some "equals" methods will return true if both argument are NaN. Unless I'm mistaken, this contradicts the IEEE standard. If nobody objects, I'm going to make the changes.

Bug报告示例



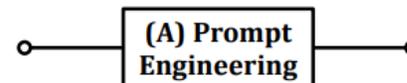
```
1 public final class MathUtilsTest extends TestCase {
2     ...
3     public void testArrayEquals() {
4         assertFalse(MathUtils.equals(new double[] { 1d }, null));
5         assertTrue(MathUtils.equals(new double[] {
6             Double.NaN, Double.POSITIVE_INFINITY,
7             ...
8         }));
9     }
10 }
```

通过后处理搜索匹配的测试用例类，并解决项目依赖问题



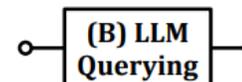
保留语法上唯一的测试用例，并通过启发式规则对测试用例进行优选：

- 测试失败的消息中显示了报告中提到的行为或异常输出值；
- 查看输出集群的大小来检查测试用例之间的一致性；
- 根据测试长度进行优先级排序（更短的测试用例更容易理解）。



```
1 # NaN in "equals" methods
2 ## Description
3 In "MathUtils", some "equals" methods will return true if both argument
4 are NaN.
5 Unless I'm mistaken, this contradicts the IEEE standard.
6 If nobody objects, I'm going to make the changes.
7
8 ## Reproduction
9 >Provide a self-contained example that reproduces this issue.
10 ```
11 public void test
```

不包含样例的提示词



```
1 public void testEquals() {
2     assertFalse(MathUtils.equals(Double.NaN, Double.NaN));
3     assertFalse(MathUtils.equals(Float.NaN, Float.NaN));
4 }
```

LLM生成的测试用例

# LIBRO: 实验评估

评估数据集：Defect4J version 2.0（17个Java项目，共750个bug）

大语言模型：Codex, July 2021

## 1. 如何判断生成的测试用例是否能复现Bug？



2. 大量Bug（251/750）可以自动化地被复现；随着提示词中的样例数量和LLM生成测试用例的尝试次数的增加，整体效果也更好。

Setting	reproduced	FIB
No Example (n=10)	124	440
One Example (n=10)	166	417
One Example from Source Project (n=10)	152	455
One Example with Constructor Info (n=10)	167	430
Two Examples (n=10, 5th percentile)	161	386
Two Examples (n=10, median)	173	409
Two Examples (n=10, 95th percentile)	184	429
Two Examples (n=50)	<b>251</b>	<b>570</b>
One Example, Crash Bugs (n=10)	69	153
One Example with Stack, Crash Bugs (n=10)	84	155

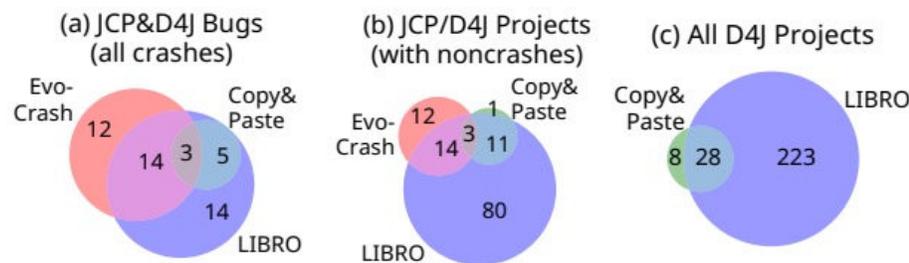
不同提示词下的性能表现

## 3. 不同项目中复现的Bug的比例存在较大偏差

Project	rep/total	Project	rep/total	Project	rep/total
Chart	5/7	Csv	6/16	JXPath	3/19
Cli	14/29	Gson	7/11	Lang	46/63
Closure	2/172	JacksonCore	8/24	Math	43/104
Codec	10/18	JacksonDatabind	30/107	Mockito	1/13
Collections	1/4	JacksonXml	2/6	Time	13/19
Compress	4/46	Jsoup	56/92	<b>Total</b>	<b>251/750</b>

Bug reproduction per project in Defects4J

## 4. 相对于之前的研究工作，LIBRO能够复现一组数量较多且独特的Bug



## 5. GHRB数据集评估：LIBRO能够处理近期的Bug报告（约30%复现率）

Project	rep/total	Project	rep/total	Project	rep/total
AssertJ	3/5	Jackson	0/2	Gson	4/7
checkstyle	0/13	Jsoup	2/2	sslcontext	1/2

## 6. 时间开销评估：LIBRO消耗的时间开销并不比其它方法要大



# 目 录

## CONTENTS

- 01 大语言模型：背景与介绍
- 02 **基于LLMs的近期研究工作**
  - 软件测试
  - 缺陷复现
  - **缺陷修复**
  - 面向大语言模型编程
- 03 总结与展望

# ChatRepair: 通过与ChatGPT对话修复Bug(162/337) ( arXiv' 23 )

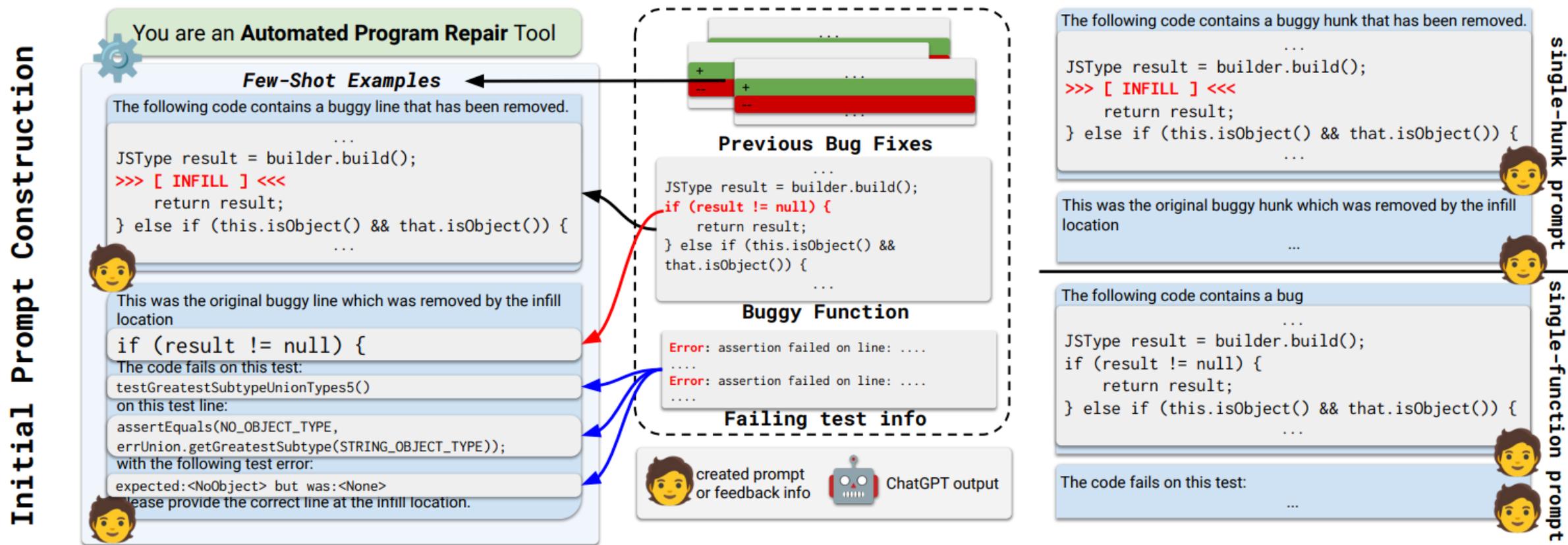
**背景介绍：**自动缺陷修复 ( APR ) 是一种自动化生成软件缺陷的补丁的方法，可以节省程序调试的时间和成本，提高软件的质量和安全性。

**问题挑战：**如何高效地生成能通过测试的补丁，并避免生成无效或低质量的补丁；如何保证生成的补丁的正确性和可靠性。

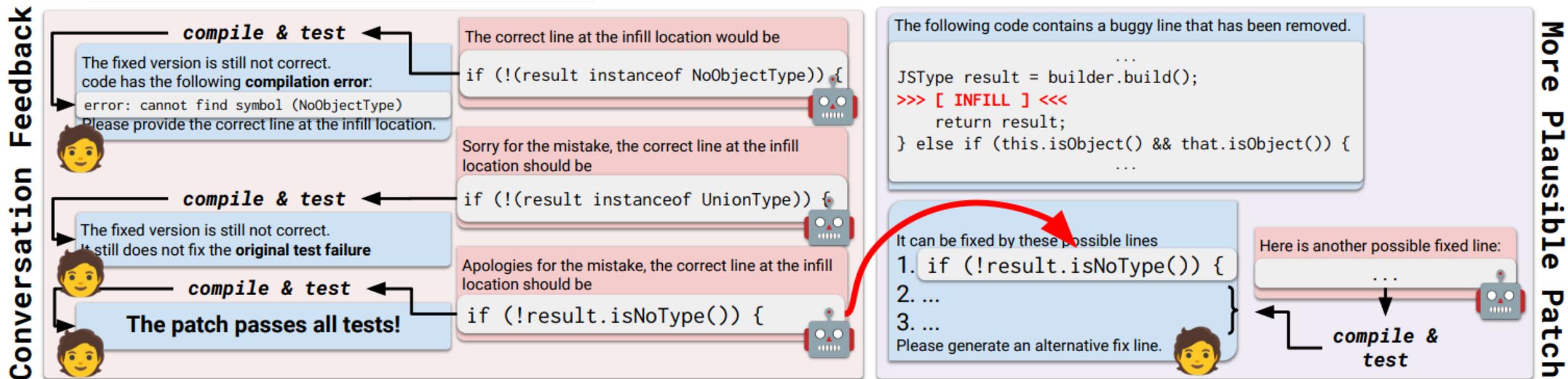
**技术创新：**（1）尝试使用最新的基于对话的LLM—ChatGPT，它能够更好地理解输入提示和指令，并跟踪和引用之前的对话。

（2）利用了测试失败信息和之前的补丁尝试的丰富语义信息，来提示LLM生成更正确的补丁。

（3）提出了一种基于对话的缺陷修复范式，将补丁生成和即时反馈相结合，不断提示LLM生成质量更高的补丁。



# ChatRepair: 通过与ChatGPT对话修复Bug(162/337) ( arXiv' 23 )



## 实验评估

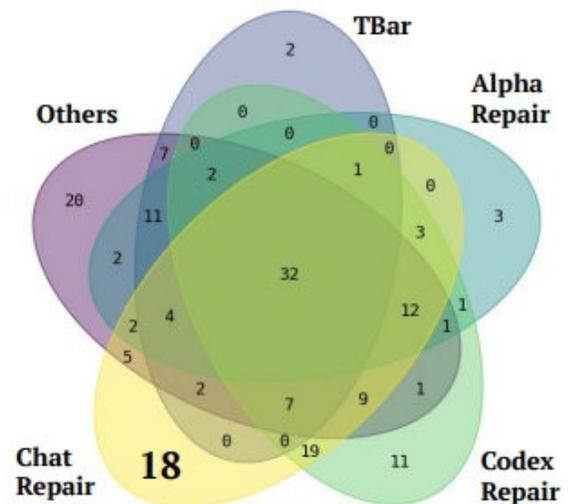
评估数据集：Defect4J v1.2, v2.0 ; QuixBugs

对比的技术：Codex Repair ; BaseChatGPT ; 基于LLMs的APR技术 ; 传统APR技术

研究问题：

- RQ1 : ChatRepair对比当前最先进的APR技术 ( 包括基于LLMs的APR技术和传统APR技术 ) 表现如何？
- RQ2 : ChatRepair在不同缺陷修复场景 ( 包括单行修复、单代码块修复、单函数修复 ) 下的表现如何？
- RQ3 : ChatRepair中的不同组件对缺陷修复的性能影响如何？

缺陷修复成本：仅计算了使用ChatGPT API的成本，可以以平均每个缺陷0.42美元的价格修复162/337个错误。



Bug fix Venn diagram on Defects4j 1.2



# 目 录

## CONTENTS

- 01 大语言模型：背景与介绍
- 02 **基于LLMs的近期研究工作**
  - 软件测试
  - 缺陷复现
  - 缺陷修复
  - **面向大语言模型编程**
- 03 总结与展望

# 提示即编程: 一种面向大语言模型的查询语言 ( PLDI' 23 )

**问题挑战**: 大语言模型在使用时存在诸多局限

- **交互性**: 许多提示技术依赖LLM与用户之间的多次交互, 需要大量手动工作。
- **约束**: 由于LLM是基于tokens的, 因此很难将解码过程限制在一组约束 ( 合法单词或短语 ) 上
- **效率和成本**: 由于LLM一次只能生成一个token, 完成一次查询可能需要多次调用。随着前缀、提示和已生成文本的增加, 查询成本越来越高。

**主要贡献**: 文章提出了一种**面向大语言模型的查询语言LMQL**。

- LMQL将提示词从纯文本提示推广到文本提示和脚本的组合, 提供了高级API所不具备的精确约束、脚本和高效解码功能,
- 用户可在不需要了解语言模型内部的情况下使用, 并且有效降低查询大语言模型的成本。

**LMQL包含五个部分**:

- **Decoder**: 一个将输入转换为输出的函数
- **Query**: 用于生成查询的程序, 由字符串和python语言组成
- **Model**: 选择需要查询的大语言模型
- **Constraints**: 允许指定对语言模型输出的约束条件。这使得适应许多任务变得容易
- **Distribution**: 一个用于生成输出分布的函数

(a) Manual Prompt

```
What is the circumference of the earth?  
I believe the best person to answer this question is _____  
Indeed, _____ addressed this question:  
  
Prompt 1           LM completion           Prompt 2
```

(c) LMQL query

```
What is the circumference of the earth? I believe  
the best person to answer this question is [EXPERT]  
Indeed, {EXPERT} addressed this question: [ANSWER]
```

(d) LMQL constraint

```
len(words(EXPERT)) <= 3 and stop_at(EXPERT, ".")
```

(b) GPT-2 completions after Prompt 1 :

- a physicist
- an astronomer
- a geologist
- Neal deGrasse Tyson
- William O'Malley, who has a PhD in Geodesy and is a professor at Colorado State University.
- the person having the knowledge and answer will probably have to refer to the relevant geophysics book and equations derived from that theory.
- a physicist, like Thomas Kugler at UC Irvine or one of the other physicists working with NASA ...
- a man named David
- actually Mother Earth herself?

启发性示例: 查询地球的周长

```
LMQL Program  
  
<decoder> <query>  
from <model>  
[where <cond>]  
[distribute <dist>]
```

```
<decoder> ::= argmax | beam(n=<int>) | sample(n=<int>)  
<query> ::= <python_statement>+  
<cond> ::= <cond> and <cond> | <cond> or <cond> | not <cond> | <cond_term>  
           | <cond_term> <cond_op> <cond_term>  
<cond_term> ::= <python_expression>  
<cond_op> ::= < | > | = | in  
<dist> ::= <var> over <python_expression>
```

LMQL的语法

# 提示即编程: 一种面向大语言模型的查询语言 ( PLDI' 23 )

**更多LMQL的示例程序**：展示的核心功能包括：脚本提示、急切的输出约束和验证 和验证，以及带有控制流的提示。

```
beam(n=3)
"A list of good dad jokes. A indicates the "
"punchline \n"
"Q: How does a penguin build its house? \n"
"A: Igloos it together. END \n"
"Q: Which knight invented King Arthur's Round"
"Table? \n"
"A: Sir Cumference. END \n"
"Q: [JOKE] \n"
"A: [PUNCHLINE] \n"
from "gpt2-medium"
where
  STOPS_AT(JOKE, "?") and STOPS_AT(PUNCHLINE, "END")
  and len(words(JOKE)) < 20
  and len(characters(PUNCHLINE)) > 10
```

(a) LMQL query to generate a joke.

```
1 argmax
2 "A list of things not to forget when "
3 "travelling:\n"
4 things = []
5 for i in range(2):
6   "- [THING]\n"
7   things.append(THING)
8 "The most important of these is [IMPORTANT]."
9 from "EleutherAI/gpt-j-6B"
10 where
11   THING in ["passport",
12             "phone",
13             "keys", ...] // a longer list
14   and len(words(THING)) <= 2
```

(b) LMQL query utilizing a python list.

两个LMQL示例程序

## 约束解码：

- 基于回溯的方法：先生成序列，再检查序列是否满足约束（**Naïve方法、成本高**）
- Eager Validation：在每个新token上执行约束检查，而不是等待整个序列的生成（**检查的成本相比查询LLM的成本可以忽略不计**）

```
A list of things not to forget when travelling:
- sun screen
- beach towel
The most important of these is sun screen.
```

(a) With **argmax** decoding.

```
A list of things not to forget when travelling:
- keys
- passport
The most important of these is sun screen.
```

```
A list of things not to forget when travelling:
- watch
- hat
The most important of these is keys.
```

(b) With **sample(n=2)** decoding.

不同解码方法下的查询结果

**distribute** IMPORTANT over things

```
A list of things not to forget when travelling:
- sun screen
- beach towel
The most important of these is { sun screen 65%
                                beach towel 35%
```

增加distribute元素后的查询结果

# LMQL实验评估一：Chain-of-Thought Prompting

argmax

```
"Pick the odd word out: skirt, dress, pen, jacket.\n"
"skirt is clothing, dress is clothing, pen is an object, jacket is clothing.\n"
"So the odd one is pen.\n\n"
"Pick the odd word out: Spain, France, German, England, Singapore.\n"
"Spain is a country, France is a country, German is a language, ....\n"
"So the odd one is German.\n\n"
"Pick the odd word out: {OPTIONS}\n"
"[REASONING]"
"[RESULT]"
```

from "EleutherAI/gpt-j-6B"

where

```
not "\n" in REASONING and not "Pick" in REASONING and
stops_at(REASONING, "Pick the odd word") and stops_at(REASONING, "\n") and
stops_at(REASONING, "So the odd one") and stops_at(REASONING, ".") and len(WORDS(REASONING)) < 40
```

distribute

```
RESULT over OPTIONS.split(", ")
```

## 评价指标：

- generate() calls：一个新的解码过程被启动的次数，这个指标主要说使用LLM时的API请求或延时方面的成本
- Model Queries：模型查询次数，为预测下一个token而查询模型的次数，这个指标直接衡量LLM的计算成本
- Billable Tokens：模拟API-gated模型的按token计费方式
- LOC：每个实现的案例研究的代码行数

## 研究问题：

- RQ1，Expressiveness：能否用简单和简洁的查询逻辑来轻松实现常见和先进的提示技术，特别是在交互式提示的情况下？
- RQ2，Performance：LMQL能否有效地降低所需的模型查询次数，从而降低使用LM所涉及的计算或API相关的成本？
- RQ3，Accuracy：约束解码能否提高LM在标准基准测试上的准确性？

场景一：思维链提示旨在解决LLM对于推理思路不清晰、所得到的答案可能显得很随意的问题。

	Standard Decoding	LMQL (constrained)	Δ	Cost Savings
<hr/>				
<i>Odd One Out</i>				
Accuracy	33.00%	33.00%	0.00%	
generate() calls	6.95	5.95	-14.38%	
Model Queries	52.98	40.85	-22.89%	
Billable Tokens	993.41	849.65	-14.47%	0.29¢/query
LOC	34	9	-73.53%	
<hr/>				
<i>Date Understanding</i>				
Accuracy	17.00%	23.00%	6.00%	
generate() calls	7.84	6.84	-12.75%	
Model Queries	63.37	57.27	-9.63%	
Billable Tokens	3291.87	2843.80	-13.61%	0.9¢/query
LOC	38	13	-65.79%	

LMQL与标准解码相比，在思维链提示任务上平均性能统计

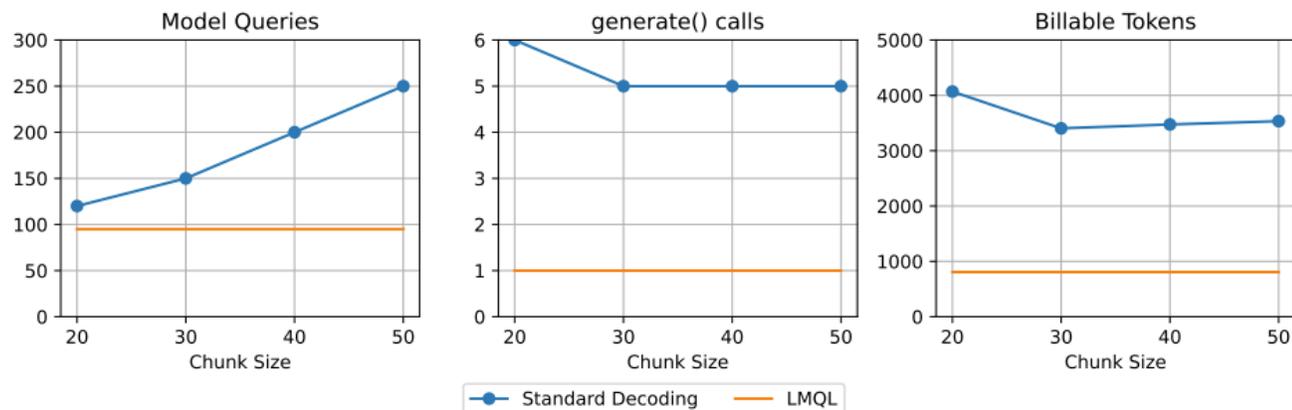
# LMQL实验评估二：Interactive Prompting

```
import wikipedia_utils
sample(no_repeat_ngram_size=3)
"What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?"
"Tho 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado ...\n"
"Act 2: Search 'Colorado orogeny'\n"
"Obs 2: The Colorado orogeny was an episode of mountain building (an orogeny) ...\n"
"Tho 3: It does not mention the eastern sector. So I need to look up eastern sector.\n"
...
"Tho 4: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft."
"Act 5: Finish '1,800 to 7,000 ft'"
"Where is Apple Computers headquartered?\n"
for i in range(1024):
    "[MODE] {i}:"
    if MODE == "Tho":
        "[THOUGHT] "
    elif MODE == "Act":
        " [ACTION] '[SUBJECT]\n"
        if ACTION == "Search":
            result = wikipedia_utils.search(SUBJECT[:-1]) # cutting of the consumed '
            "Obs {i}: {result}\n"
        else:
            break # action must be FINISH
from "gpt2-xl"
where
MODE in ["Tho", "Act"] and stops_at(THOUGHT, "\n") and
ACTION in ["Search", "Finish"] and len(words(THOUGHT)) > 2 and
stops_at(SUBJECT, "'") and not "Tho" in THOUGHT
```

**场景二：**由于LMQL在其提示子句中支持循环、分支和函数调用，它很适合于实现这些类型的[交互式提示场景](#)。

	Standard Decoding	LMQL (constrained)	$\Delta$	Cost Savings
<i>ReAct (Case Study 2)</i>				
generate() calls	5	<b>1</b>	-80%	
Model Queries	150	<b>95</b>	-36.67%	
Billable Tokens	3,404	<b>807</b>	-76.29%	5.2¢/query
LOC	59	<b>22</b>	-62.71%	

LMQL与标准解码相比，在交互式提示场景上的平均性能统计



针对交互式ReAct提示，比较了使用基线实现的不同分块大小与不需要分块解码的LMQL之间的差异

# LMQL实验评估三：Arithmetic Reasoning

```
argmax(distribution_batch_size=1, max_length=2048)
"<few-shot examples>"
"Q: {QUESTION}\n"
"A: Let's think step by step.\n"
for i in range(1024):
    "[REASON_OR_CALC]"
    if REASON_OR_CALC.endswith("<<"):
        "[EXPR]"
        result = calculator.run(EXPR)
        "{result} >> "
    elif REASON_OR_CALC.endswith("So the answer"):
        break
" is [RESULT]"
from "EleutherAI/gpt-j-6B"
where
    int(RESULT) and
    stops_at(REASON_OR_CALC, "<<") and
    stops_at(EXPR, "=") and
    stops_at(REASON_OR_CALC, "So the answer")
```

(a) LMQL query for arithmetic reasoning.

Q: Noah is a painter. He paints pictures and sells them at the park. He charges \$60 for a large painting and \$30 for a small painting. Last month he sold eight large paintings and four small paintings. If he sold twice as much this month, how much is his sales for this month?  
A: Let's think step by step.

```
He sold 8 large paintings and 4 small
paintings last month.
He sold twice as many this month.
8 large paintings x $60 = << 8*60= 480 >> 480
4 small paintings x $30 = << 4*30= 120 >> 120
So the answer is 480
```

(b) Interaction Trace.

## 实验结论：

(1) LMQL具有**很强的表达能力**，目前大多最先进的 Prompting方法都可以直接用简单的脚本风格的语言进行编码，并且该编码所需的代码行数比基于python的相应实现少得多。

(2) LMQL大大**减少了模型查询的数量**，从而降低了效率和运行时间。这是因为LMQL支持tokens级别的验证，这使得我们能够即时执行约束，而不是通过分块解码和回溯。

(3) LMQL不会影响模型的准确性。事实上，在某些情况下，强制约束甚至可以**提高准确性**。

(4) 在付费的API-gated模型中使用时，LMQL将能够大大**节省需要支付的tokens资源**。

**场景三：** 现有的研究表明，大语言模型在正确地**计算算术表达式**方面可能会遇到困难。即使推理步骤是正确的，但在具体的算术计算中出现错误也会导致结果不正确。

	Standard Decoding	LMQL (constrained)	Δ	Cost Savings
<i>Arithmetic Evaluation (Case Study 3)</i>				
generate() calls	7	<b>1</b>	-85.71%	
Model Queries	210	<b>73</b>	-65.24%	
Billable Tokens	3,649	<b>541</b>	-85.17%	6.2¢/query
LOC	78	<b>18</b>	-76.92%	

LMQL与标准解码相比，在交互式提示场景上的平均性能统计



# 目 录

## CONTENTS

- 01 大语言模型：背景与介绍
- 02 基于LLMs的近期研究工作
  - 软件测试
  - 缺陷复现
  - 缺陷修复
  - 面向大语言模型编程
- 03 **总结与展望**

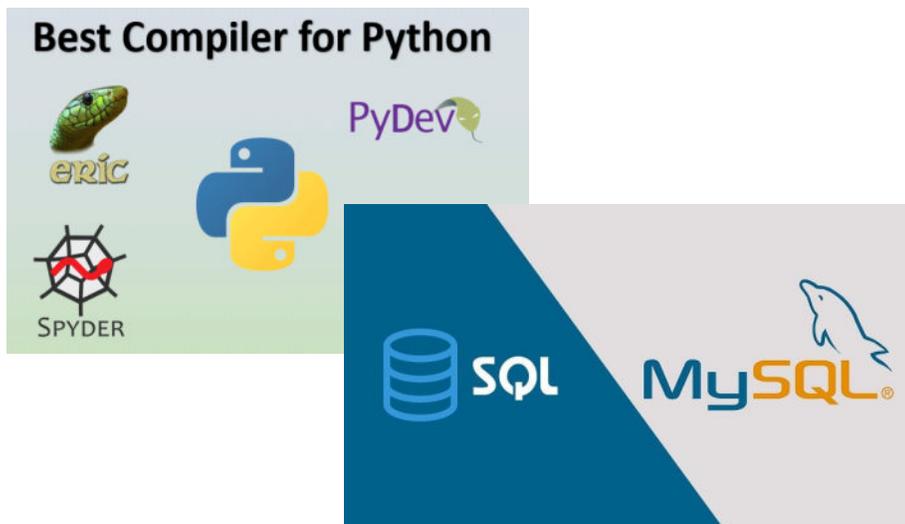
# 大语言模型在SE、PL、SEC等领域的应用：挑战与机遇

## 大语言模型 (LLMs) 在应用上的局限性：

- **不同任务上的表现不同**：在某些单点任务上的表现较为突出，但在一些复杂的整体性任务上的表现仍然较弱；
- **输入长度的限制**：ChatGPT接收的提示词长度最大为4096 tokens，GPT-4为32k tokens；
- **适应不同编程语言的能力不同**：对于高级编程语言而言，LLMs适应Java和Python的能力更好，但适应C/C++、Rust和LLVM IR上的能力仍然不足。

## 大语言模型的潜力挖掘，及其它可能的应用场景：

1. 用LLMs生成语法/语义正确的输入来**测试编译器、数据库、SMT求解器等**基础应用；
2. 对于**依赖人工标注的静态分析**，可以用LLMs自动对代码进行标注。
3. 形式化验证领域可用LLMs生成**不变式、断言、规约等**性质。



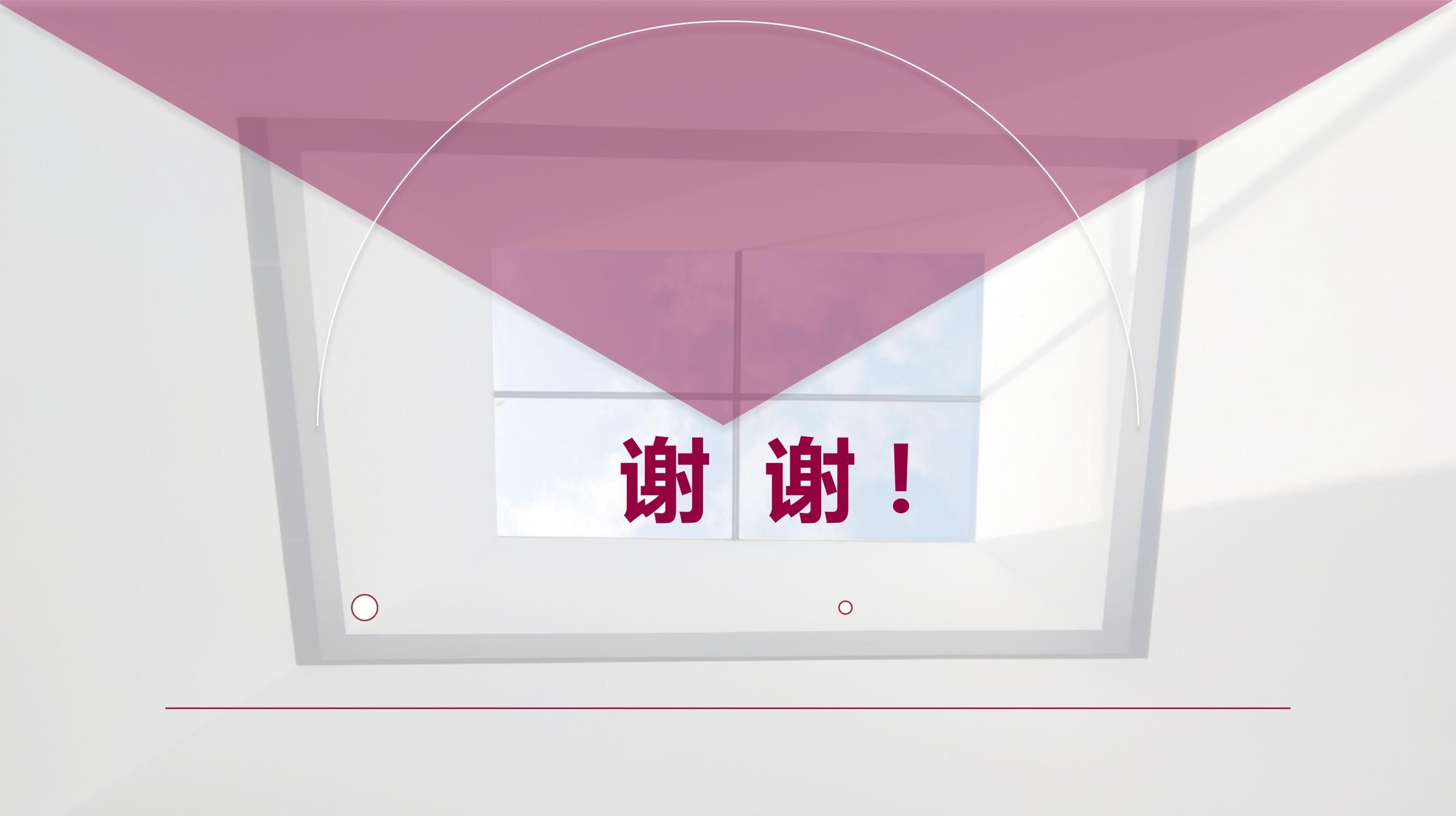
```
Mutex mu1, mu2;
int a GUARDED_BY(mu1);
int b GUARDED_BY(mu2);

void foo() REQUIRES(mu1, mu2) {
    a = 0;
    b = 0;
}

void test() {
    mu1.Lock();
    foo();           // Warning! Requires mu2.
    mu1.Unlock();
}
```

Clang的线程安全分析静态工具

```
1: int sum(int array[], int size)
2: {
3:     int k = 0;
4:     int s = 0;
5:     /* M(0) */
6:     while (k < size) {
7:         /* M(k) */
8:         s = s + array[k];
9:         /* M(k+1) */
10:        k = k + 1;
11:        /* M(k) */
```



**谢谢!**