

大模型时代软件研发正确打开方式

朱少民 | 同济大学

科技生态圈峰会 + 深度研习

—1000+ 技术团队的选择



敦煌站

K+思考周®研习社

时间: 2025.08.29-30



上海站

K+金融专场

时间: 2025.10.17-18



香港站

K+思考周®研习社

时间: 2025.11.25-26



K+峰会详情



上海站

AI+研发数字峰会

时间: 2025.05.17-18



北京站

AI+研发数字峰会

时间: 2025.08.08-09



深圳站

AI+研发数字峰会

时间: 2025.11.28-29



AIDD峰会详情



朱少民

“软件工程” 3.0定义者/CCF杰出会员、质量工程SIG主席

同济大学特聘教授、CCF杰出会员、质量工程SIG主席、软件绿色联盟标准评测组组长。

近三十年来一直从事软件测试、质量管理等工作，先后获得多项省、部级科技进步奖，已出版了二十多部著作和4本译作，代表作主要有《软件测试方法和技术》、《全程软件测试》、《敏捷测试》等，并经常在国内外学术会议或技术大会上发表演讲。

曾任思科（中国）软件有限公司QA高级总监、IEEE ICST2019工业论坛主席、IEEE ICST、QRS、DAS等国际学术会议的程序委员、《软件学报》审稿人等。

目录

CONTENTS

- I. 国内LLM落地软件研发的现状
- II. LLM更适合编程工作吗?
- III. 代码生成的正确打开方式
- IV. 软件研发终极打开方式
- V. 未来展望

01

国内LLM落地软件研发的现状

发布 《2024软件研发应用大模型国内现状调研报告》



目录

- 01 调查获得的一些结论
- 02 参与调查的基本情况
- 03 国内使用大模型的概况
- 04 在软件研发生命周期使用LLM
- 05 未来展望

附录
特别鸣谢

国内应用整体状况

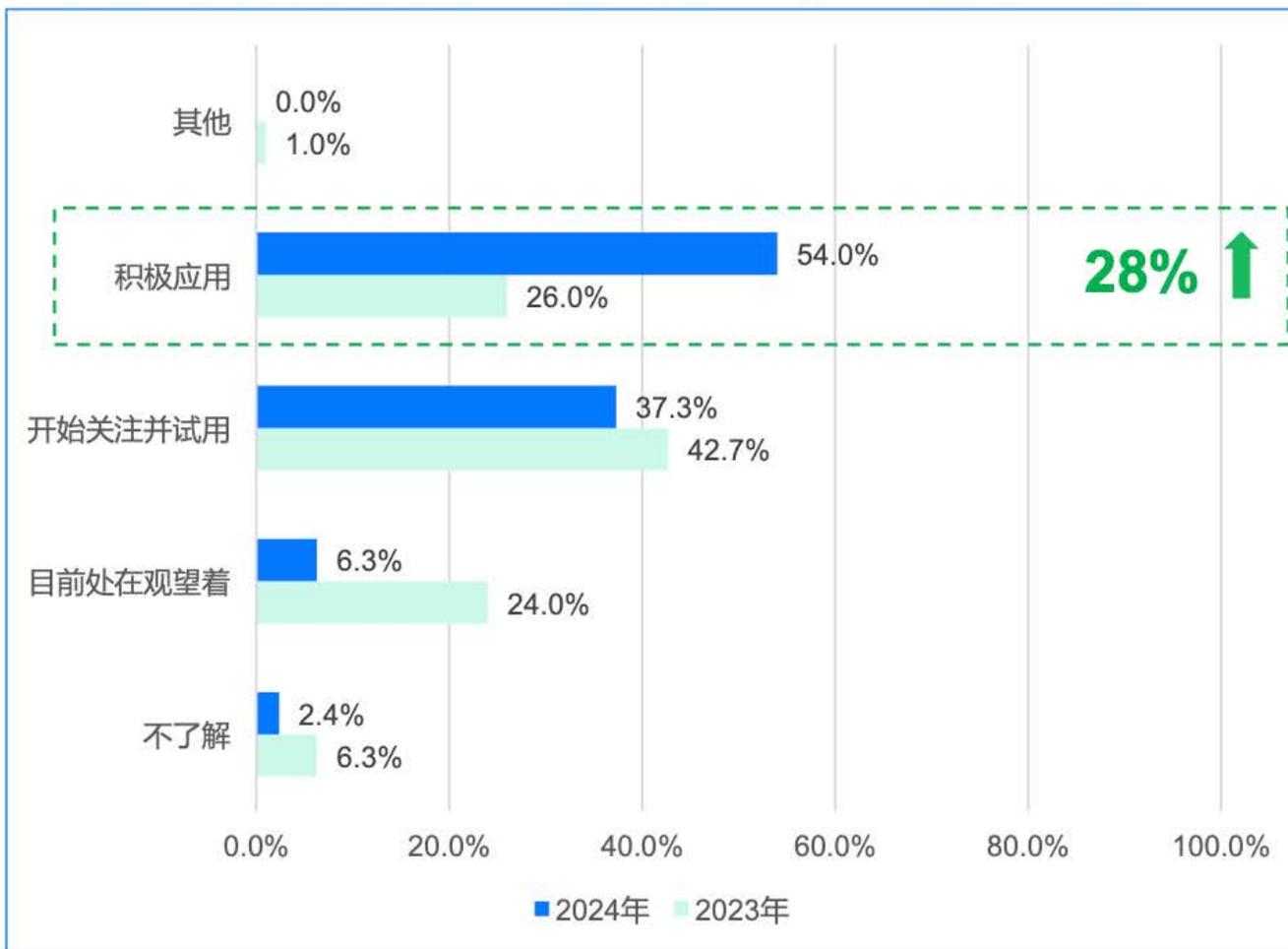
我们先看一下国内软件研发团队对大模型（LLM）的态度，绝大多数（**91.3%**）是拥抱LLM，关注、开始使用或积极使用LLM的。

积极应用大模型的团队，占比超过了一半，达到**54%**，比去年增多了**28%**，变化非常明显。

有6.3%的团队或公司处在观望中，比去年下降**17.7%**，越来越多的团队开始行动起来。

如我们所预料，不了解LLM的团队越来越少，比去年下降了**3.9%**。

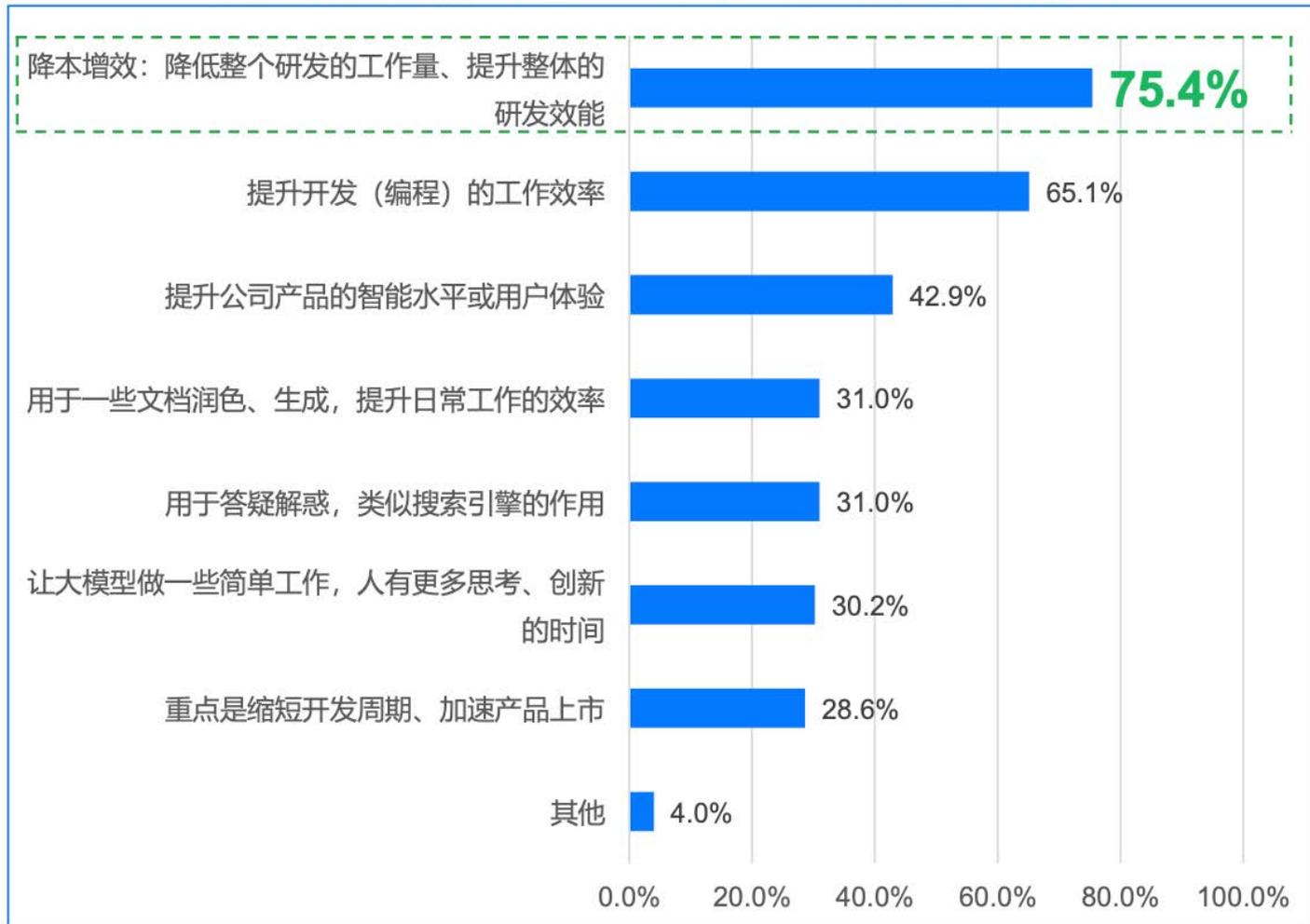
国内软件研发团队对大模型（LLM）的态度



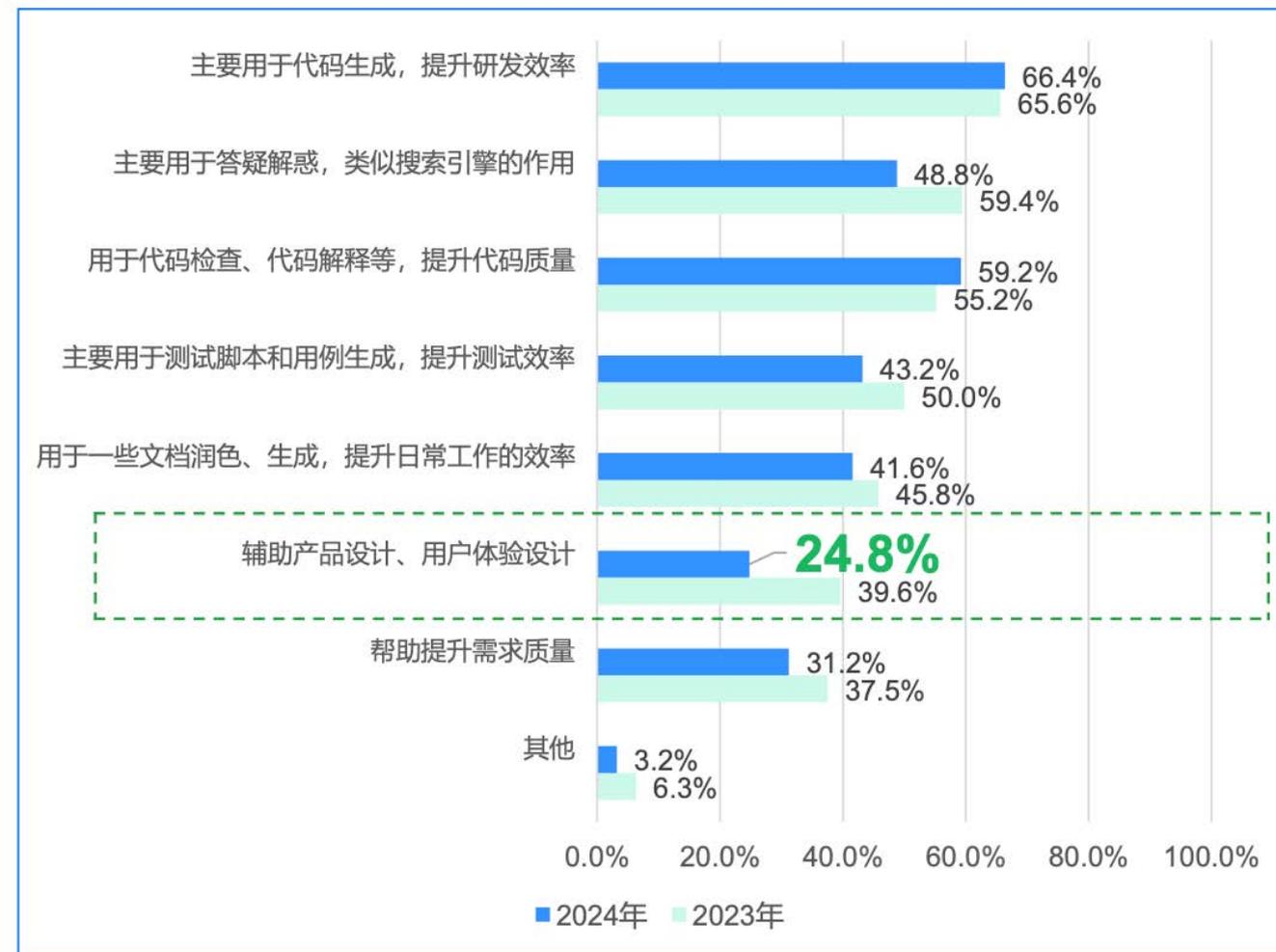
重视LLM且能够投入较多人力和资源的团队占比已超过一半（**59.5%**），比去年提升了**12.9%**

国内应用LLM的动机和效益

国内团队期望让LLM发挥哪些方面的价值

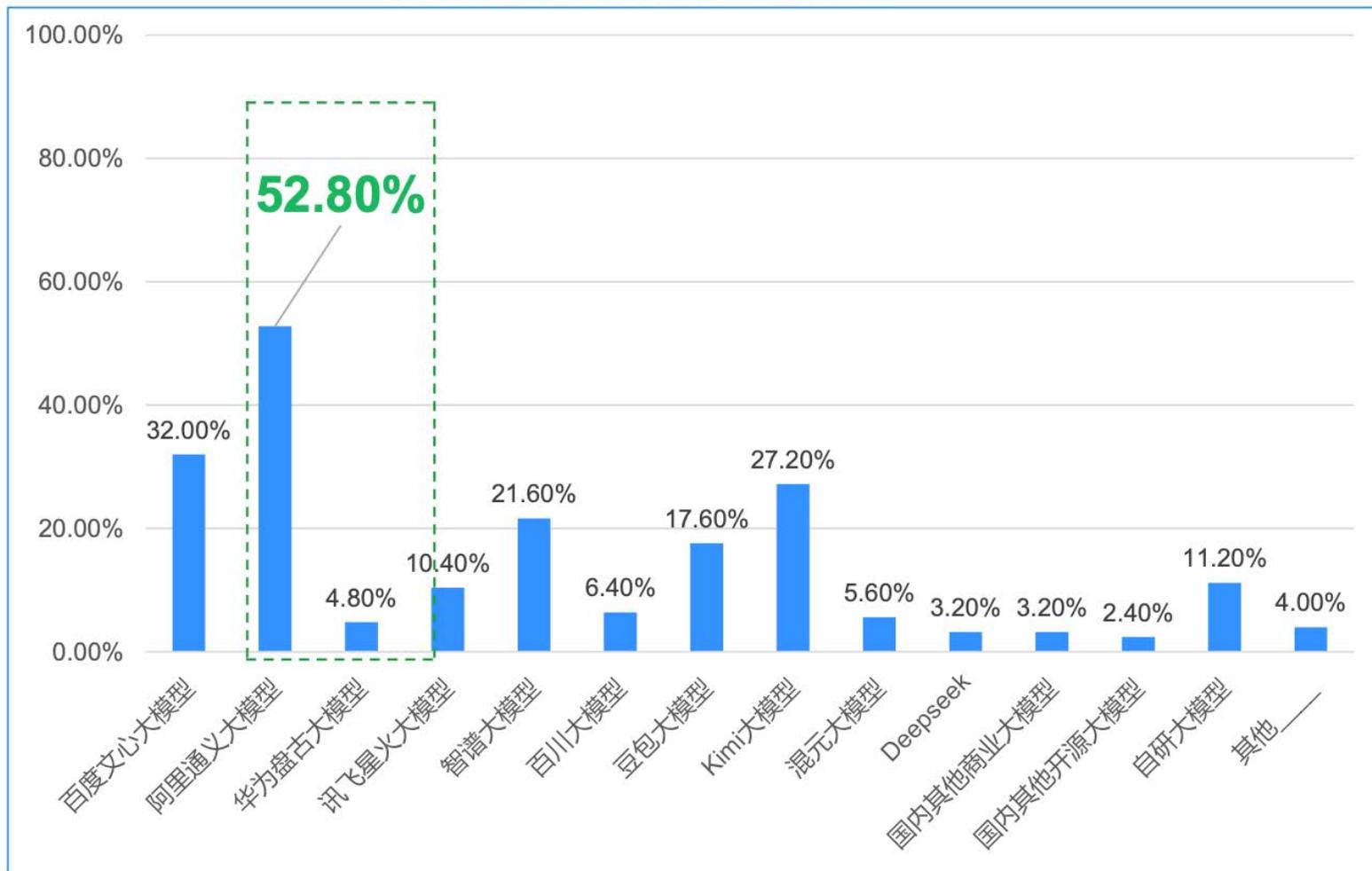


国内团队应用LLM实际发挥了什么作用



使用的LLM情况和面临的困难

国内团队使用哪些国内的LLM



接近1/3的
软件开发
团队是采用成本最低的方式：
直接使用
第三方
LLM API



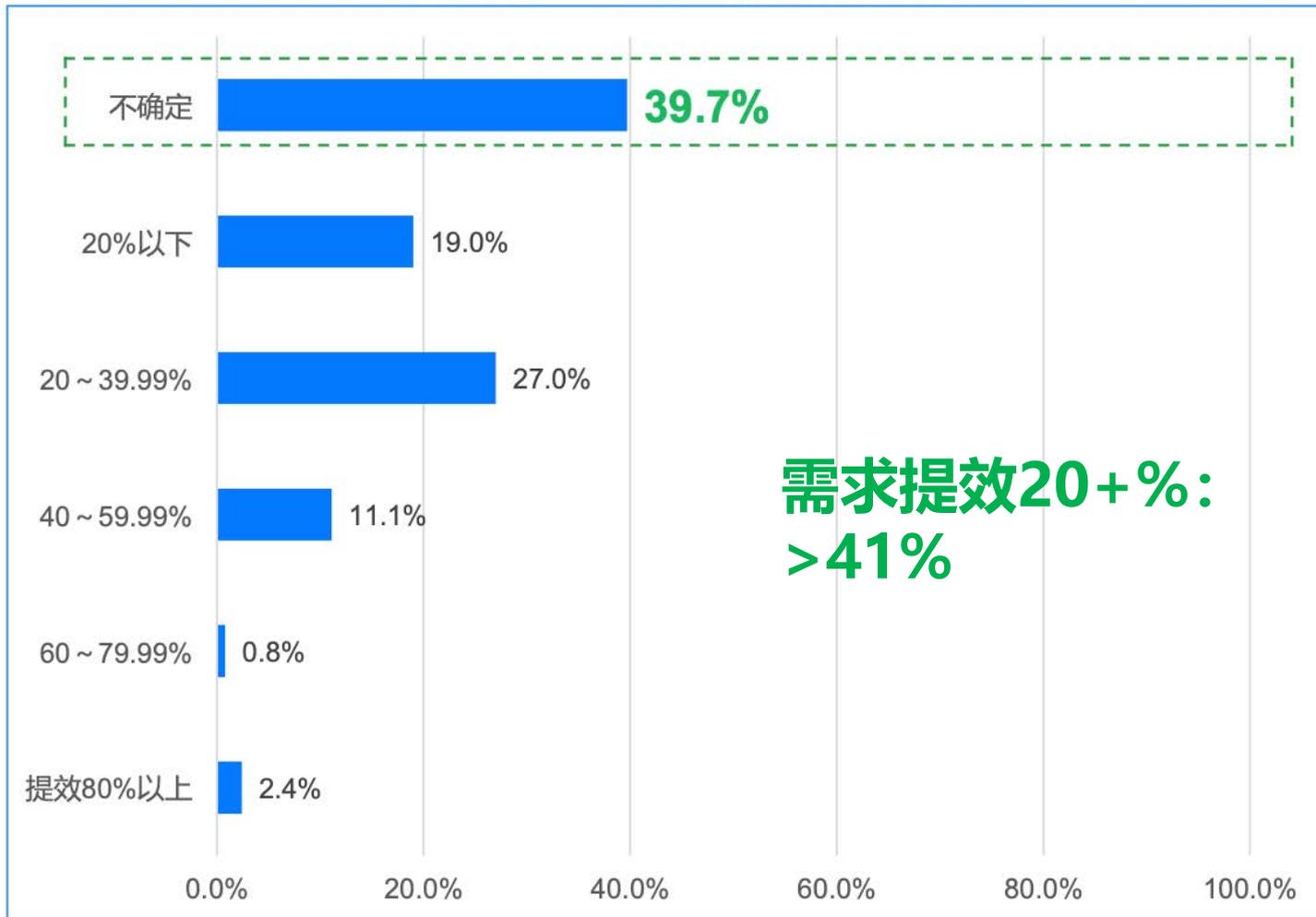
应用面临的困难

任何事情都不是一帆风顺的，特别是对新兴的LLM应用领域，遇到的问题还是比较多的。从调查结果看，LLM应用主要有七大困难（从高到低排列）：

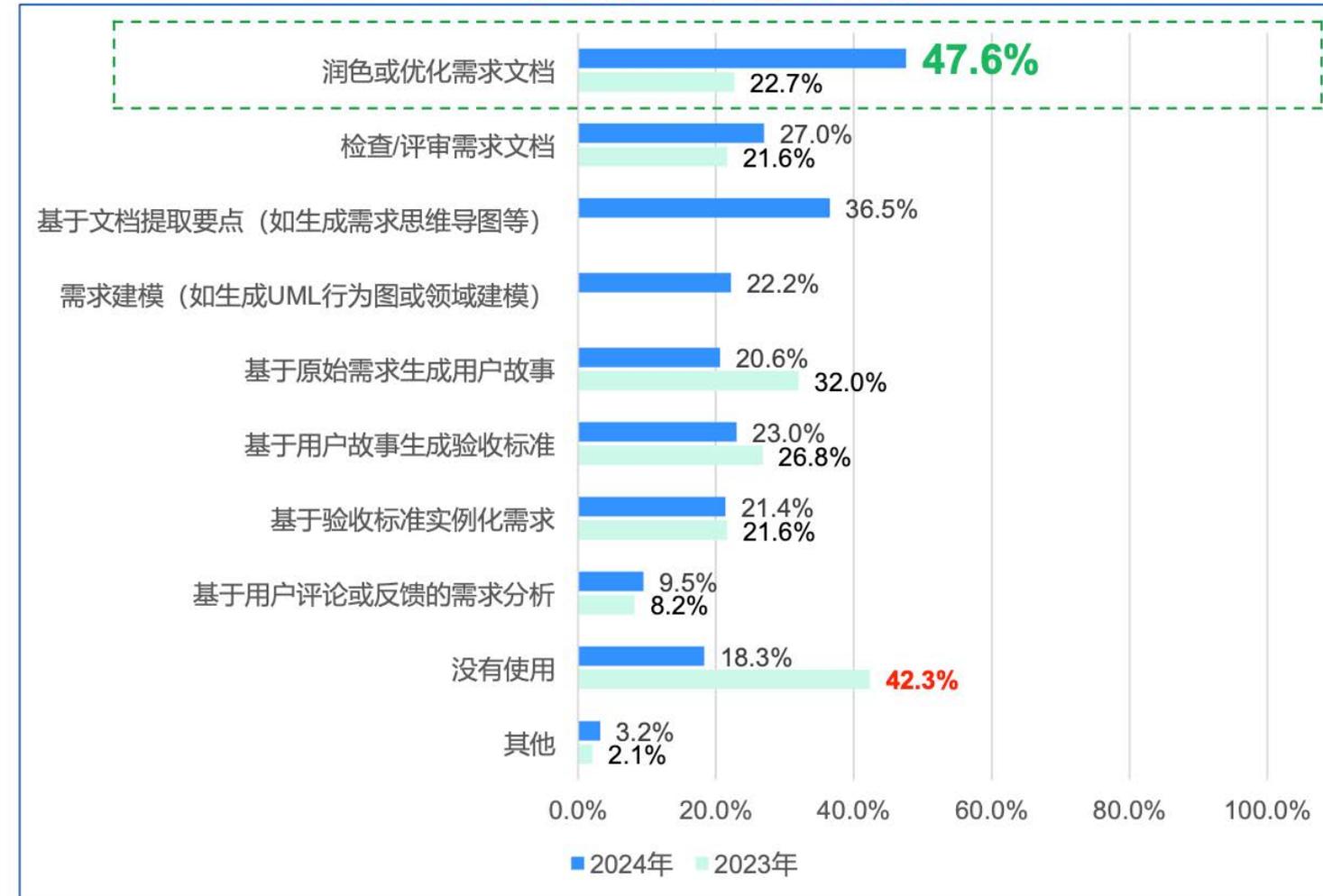
- 1) 缺少高质量的（业务/研发）训练数据
- 2) 缺少机器学习和大模型训练人才
- 3) 缺少算力或云计算平台
- 4) 目前研发人员的应用水平不行
- 5) 安全、隐私担忧
- 6) (业务/研发) 数据数量不足
- 7) 缺少大模型部署的能力

LLM驱动软件产品创新、需求定义

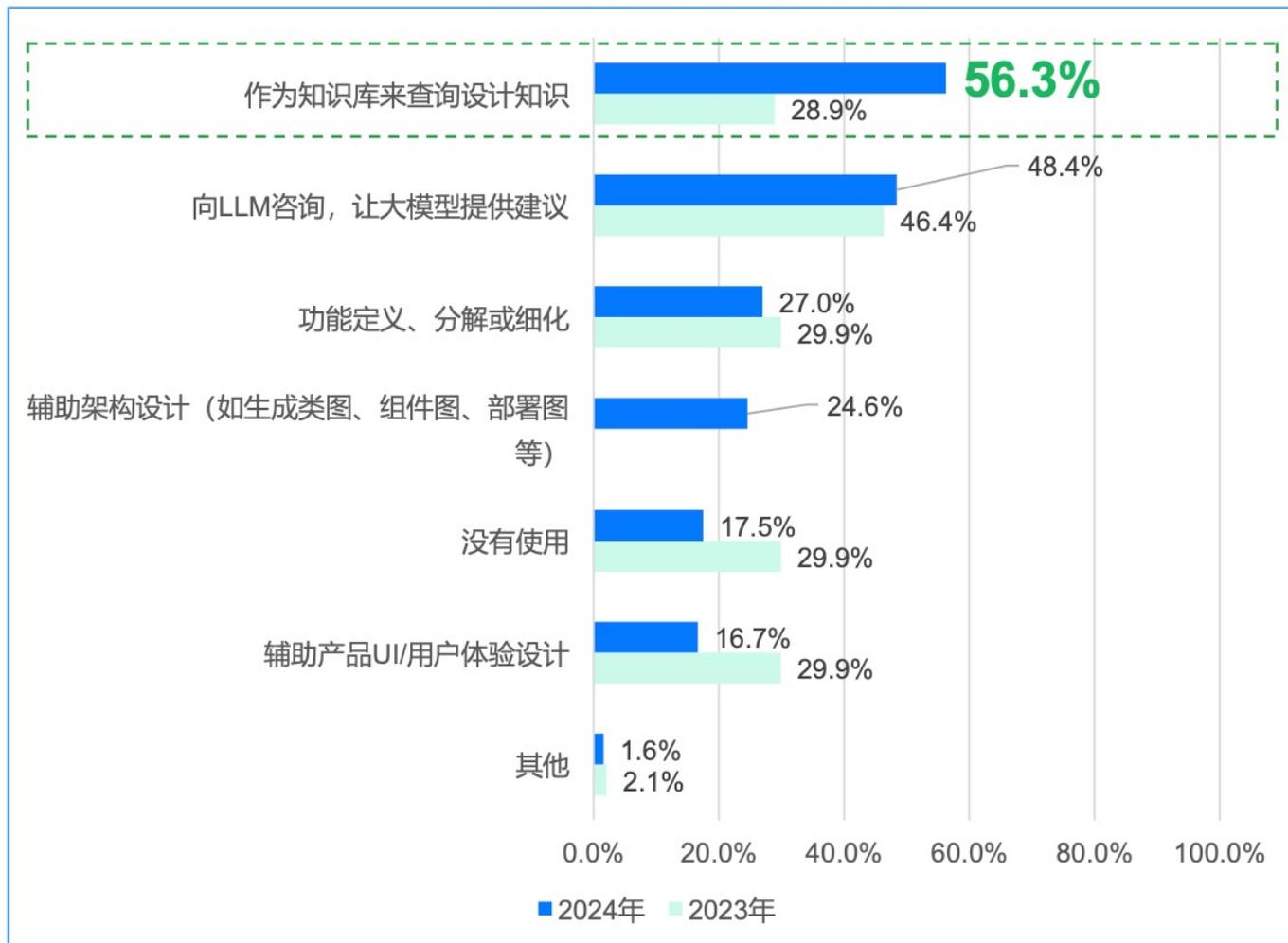
LLM在需求上应用后实际提效多少



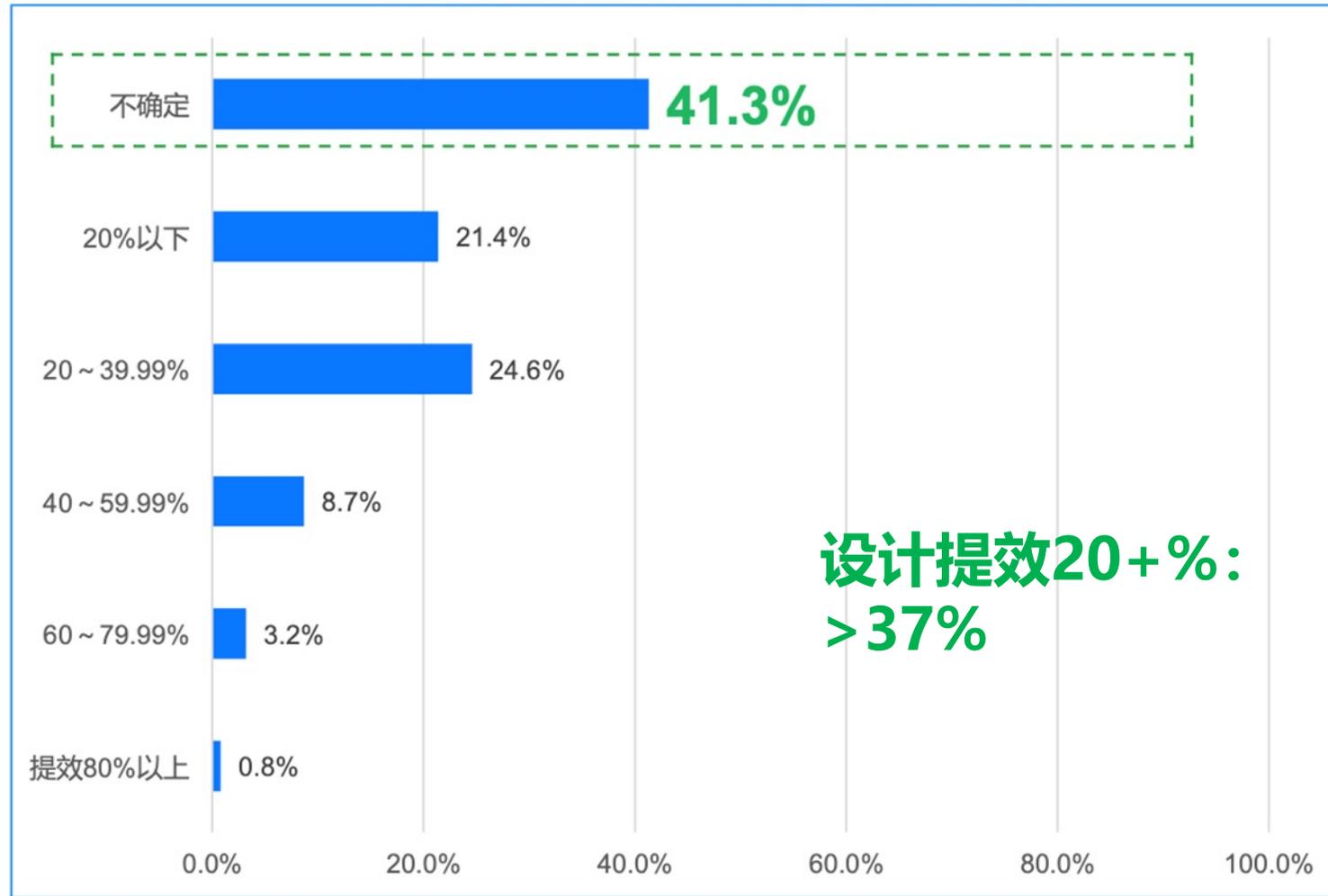
LLM在需求上应用的状况



LLM在设计上应用的状况

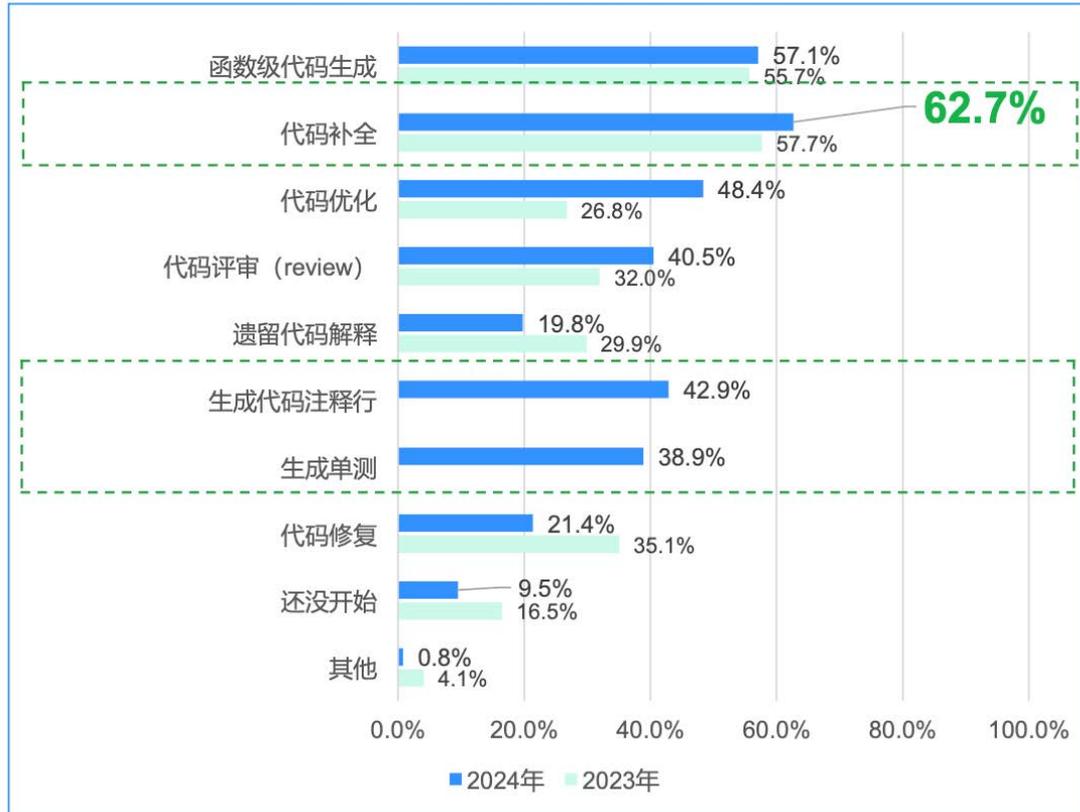


LLM在设计上应用后实际提效多少

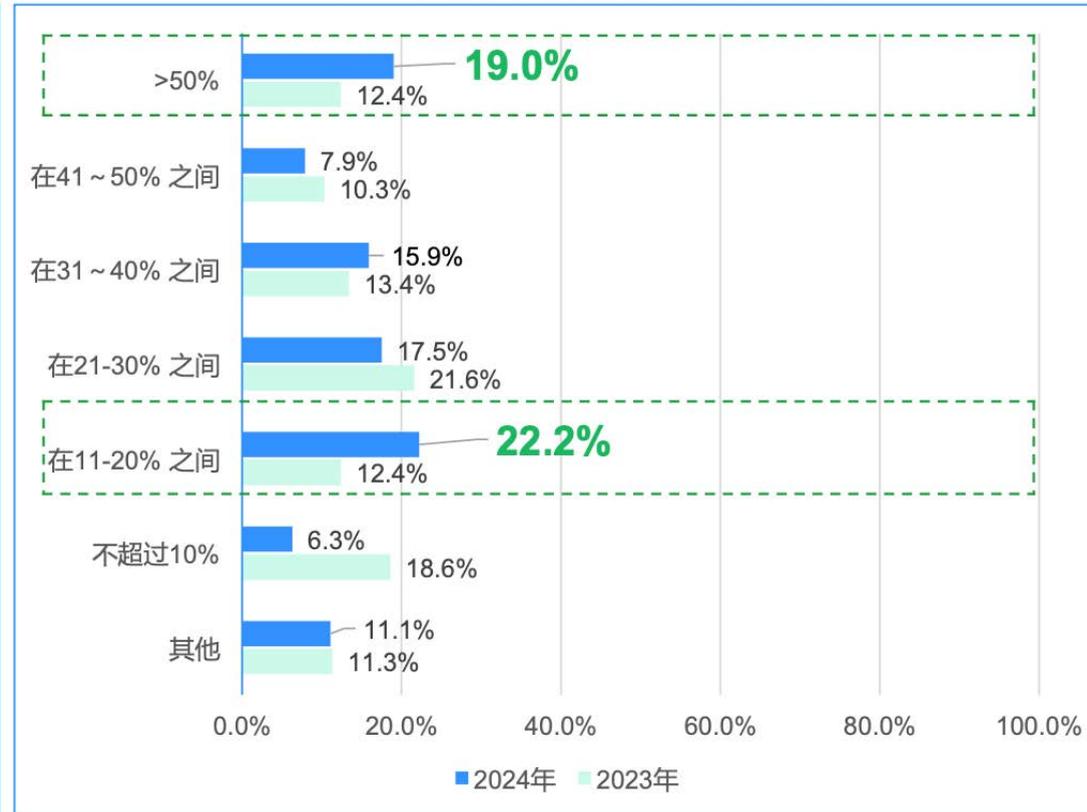


设计提效20+%:
>37%

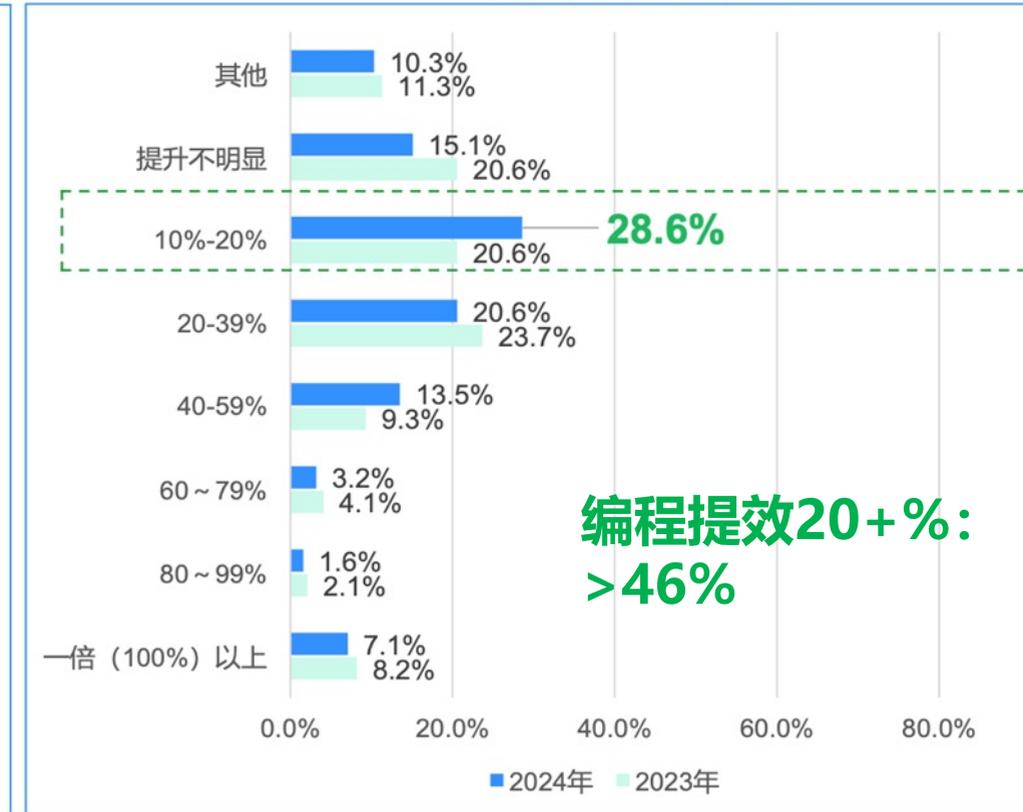
LLM在编程上应用的状况



LLM生成代码的采纳率状况

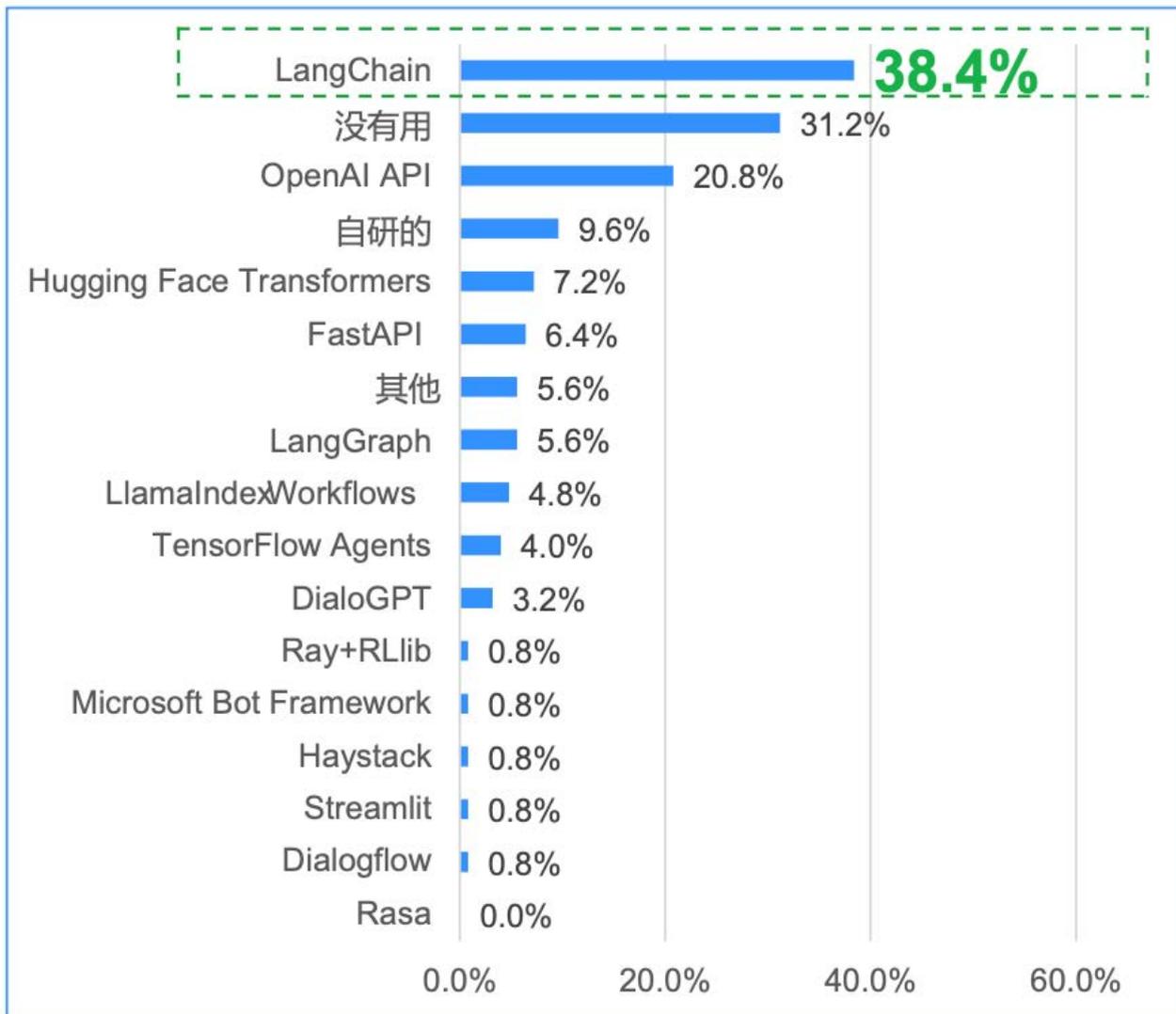
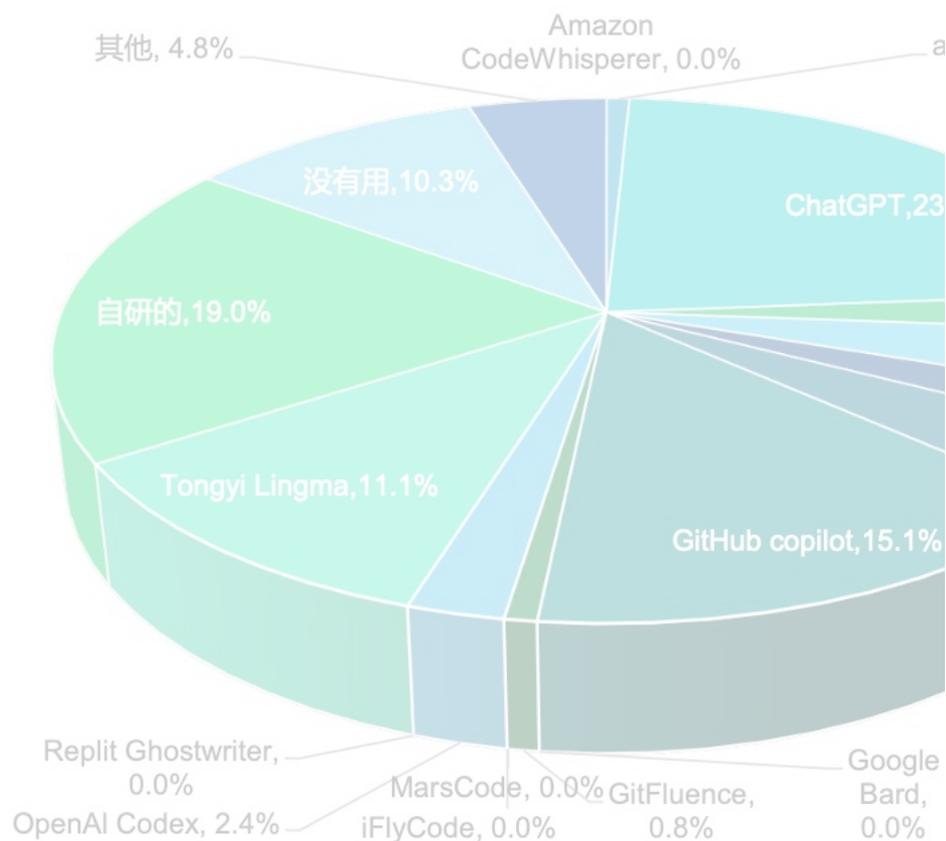


LLM提升编程的效率

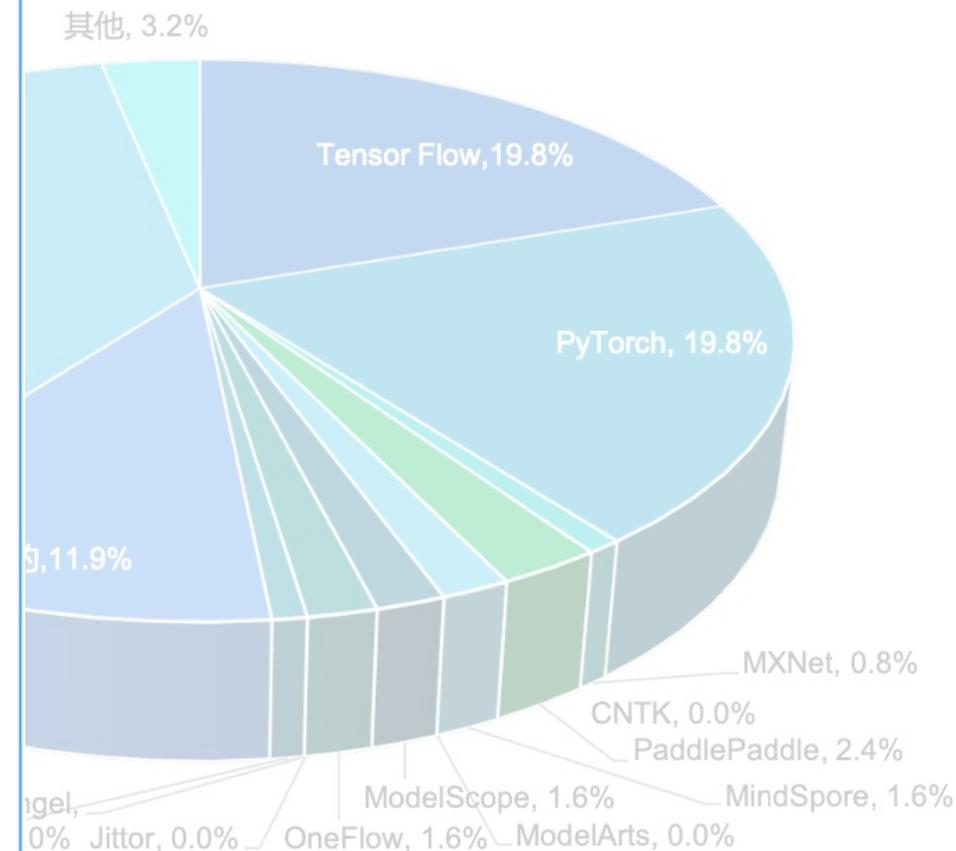


▶ 哪家AI编程工具、框架强?

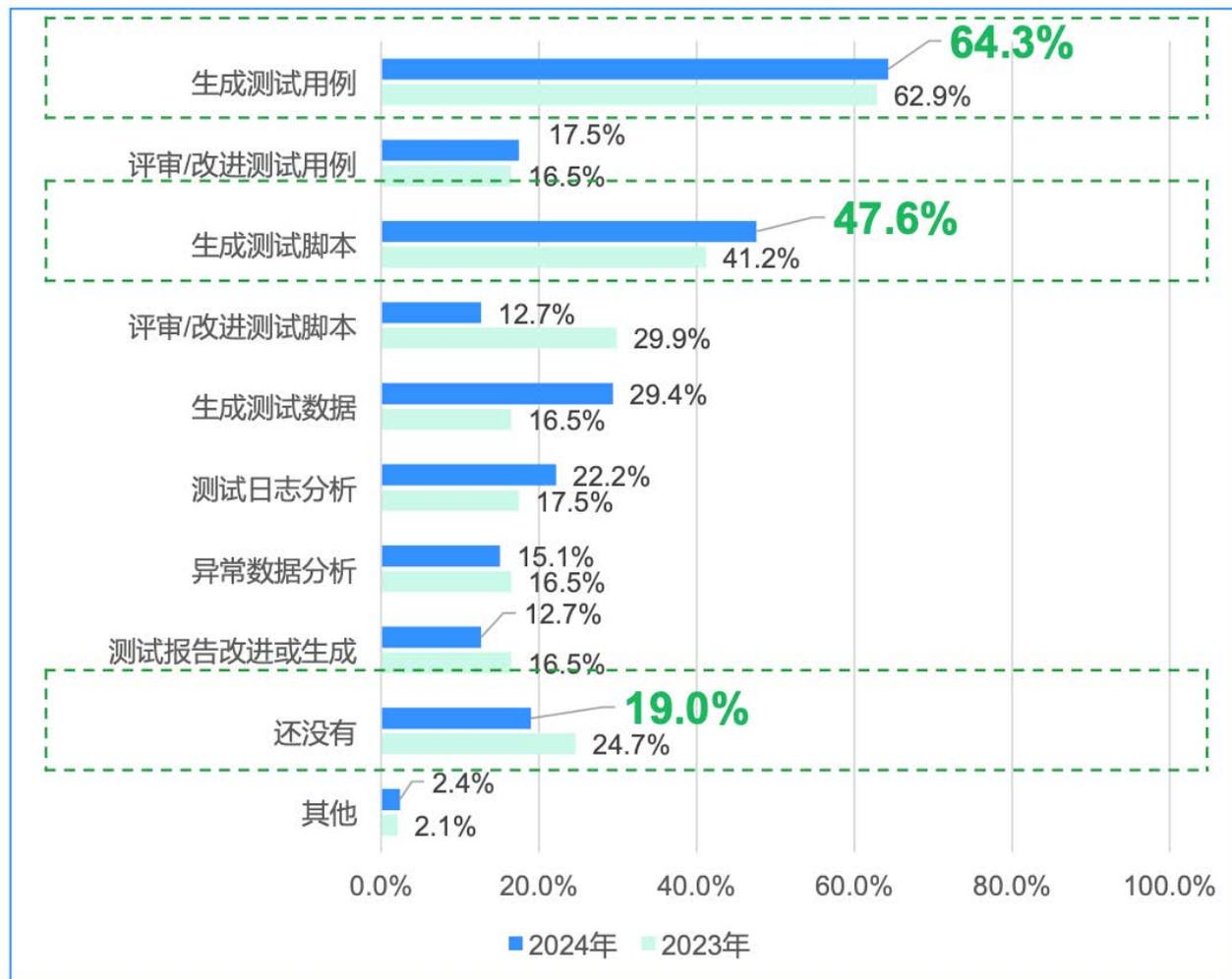
基于LLM的编程助手被使用的统计



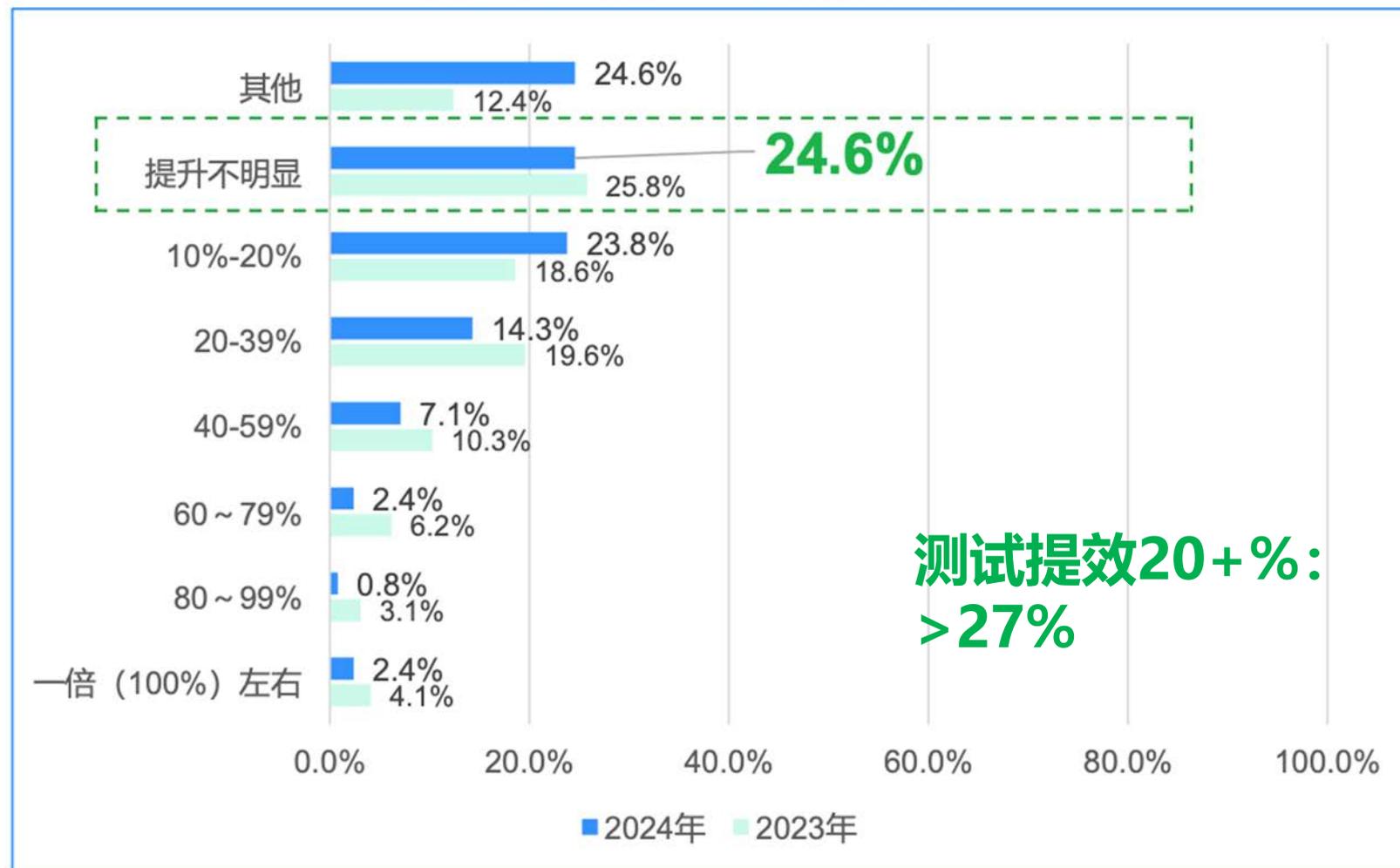
AI框架被使用的统计结果



LLM在测试上应用现状

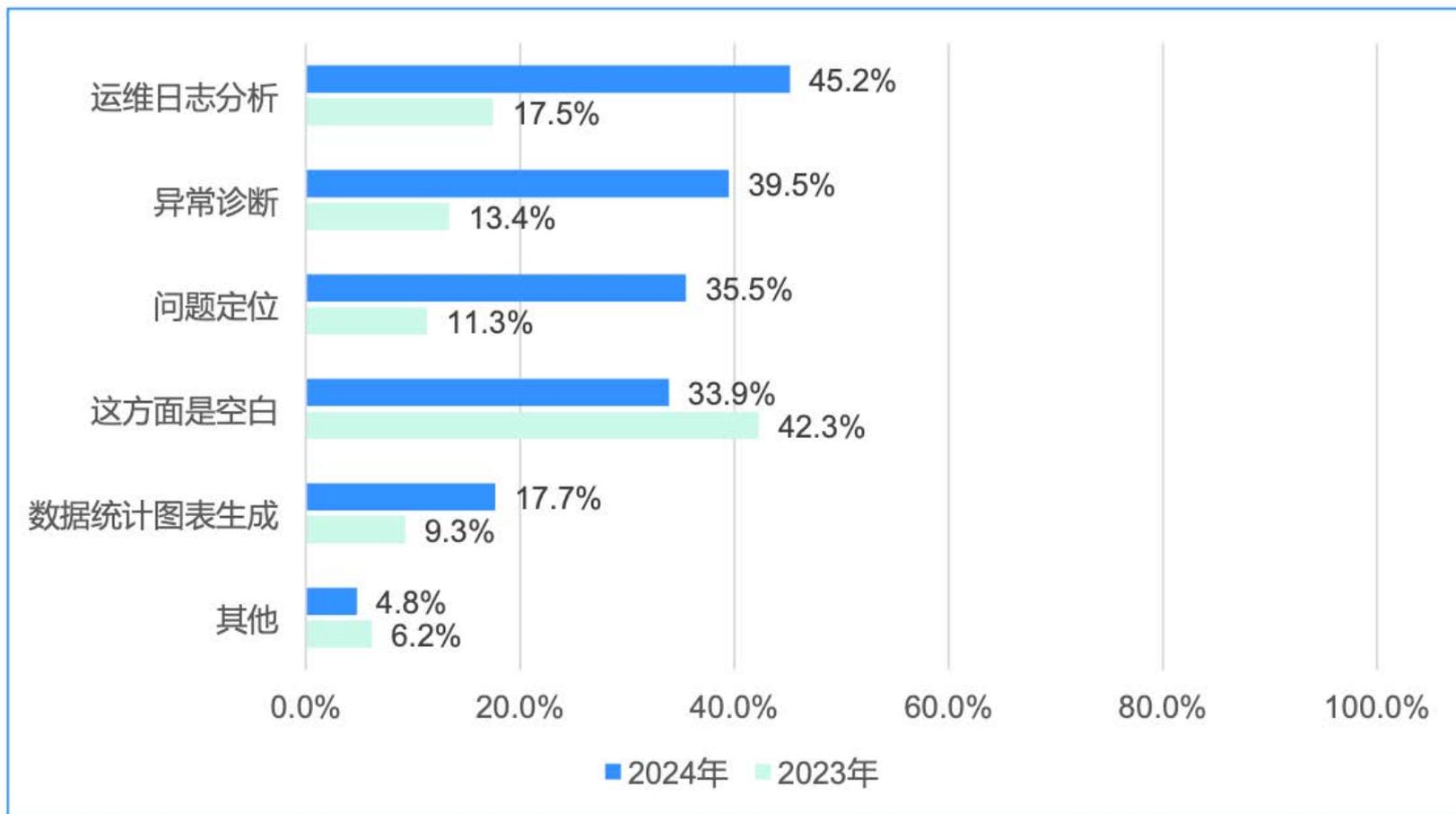


LLM提升测试效率的统计情况

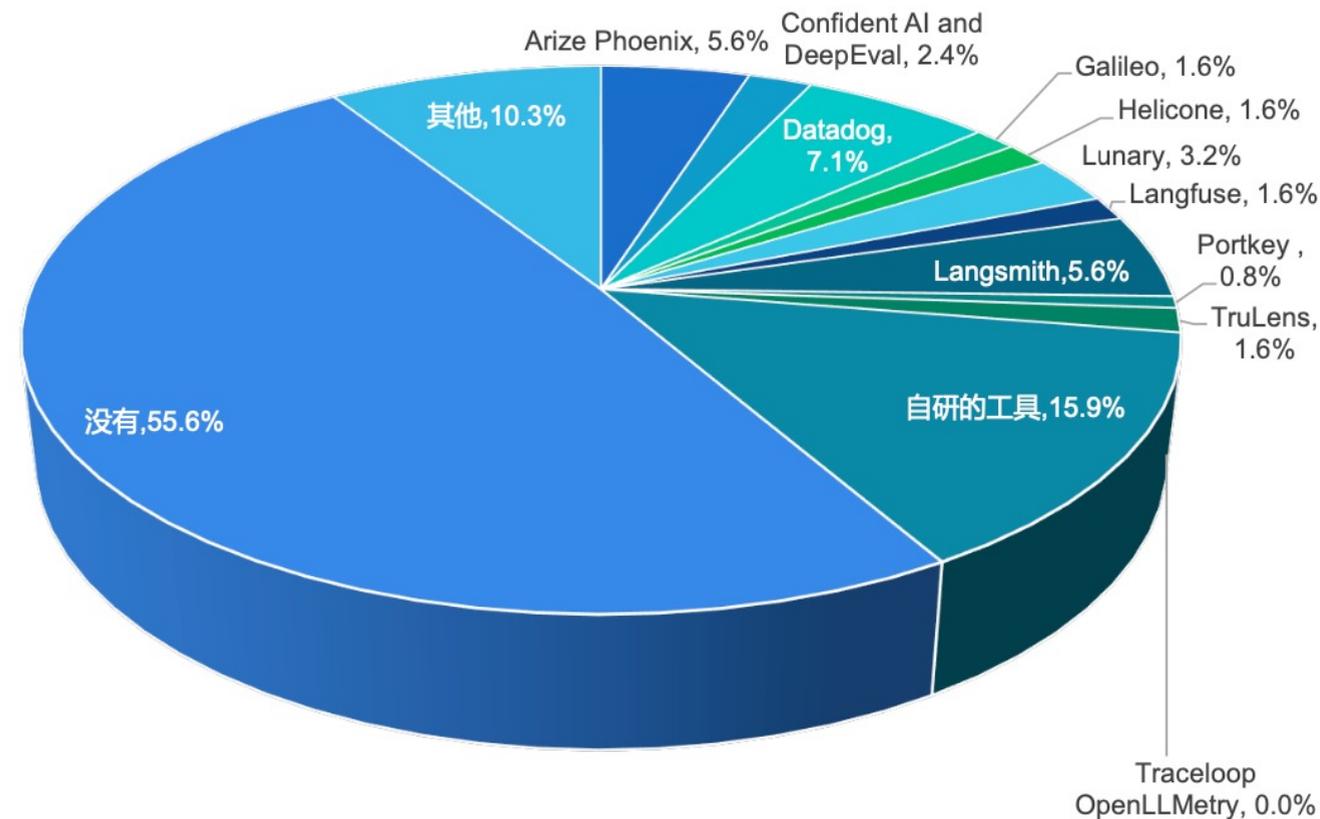


测试提效20+%：
>27%

LLM在运维上应用情况

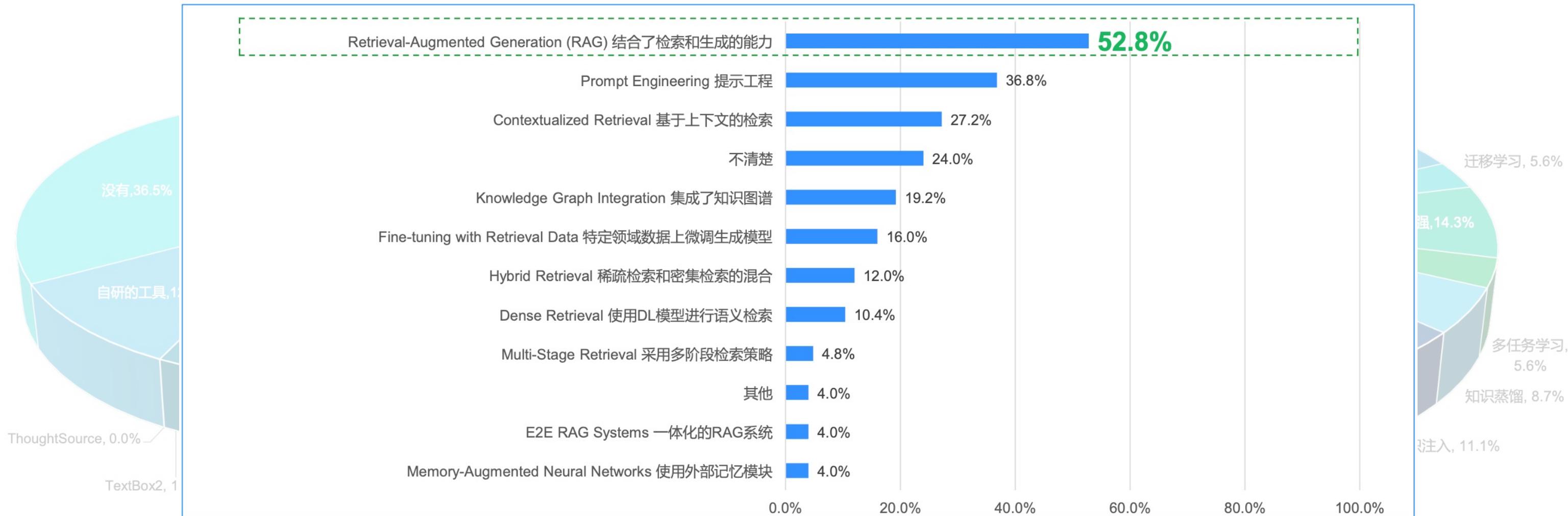


管理或监控LLM工具统计状况



大模型相关技术应用情况

使用RAG技术的统计状况



02

LLM更适合编程工作吗?



“LLM适合编程” 有三大理由

团队中开发人员的成本是最高的



编程语言也是语言



代码数据质量高



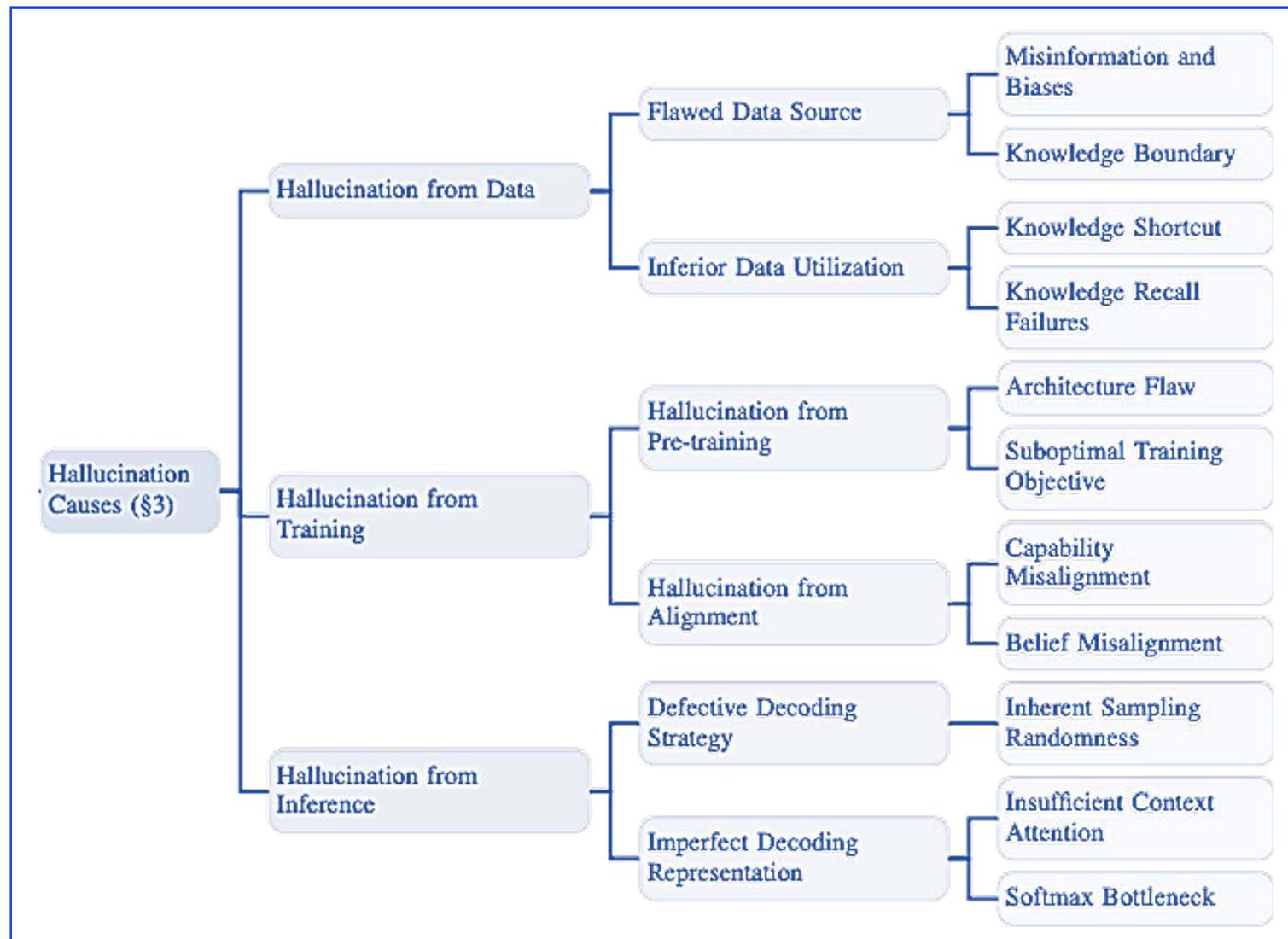
▶ 大模型是概率模型，存在幻觉问题

- 1- **数据产生的幻觉**：错误的數據源、知識邊界、數據利用不當
- 2- **訓練相關的幻觉**：預訓練缺陷、對齊問題、能力不匹配
- 3- **推理相關的幻觉**：解碼策略缺陷、上下文注意力不足

“代码却是需要准确，错一个符号都不行”

那我们需要缓解幻觉问题

- ✓ 数据：增强数据质量、知识边界管理
- ✓ 训练：改进预训练过程、对齐目标优化
- ✓ 推理：增强解码策略、上下文增强



03

代码生成的正确打开方式

悖论：基于代码生成单测

```
public void testReadLines4(String pathname0) throws Exception {
    // setup
    File file = new File(pathname0);

    // run the test
    List result = FileUtil.readLines(file);

    // verify the results
    Assertions.assertEquals(null, result);
}

public String startsWithLiteral(String str) {
    if (str.startsWith("1234567890")) {
        str = str.replace("3", "A");
    } else {
        str = str.trim();
    }

    if (str.charAt(0) == 'x'){
        return str;
    } else {
        return str.toLowerCase();
    }
}
```



▶ 示例：计算年龄

```
1 #计算年龄
2
3 import datetime
4
5 def calculate_age(birthdate_str):
6     """计算年龄，输入日期格式为 YYYY-MM-DD。"""
7     try:
8         birthdate = datetime.datetime.strptime(birthdate_str,
9         "%Y-%m-%d").date()
10    except ValueError:
11        return "Invalid date format"
12
13    today = datetime.date.today()
14    age = today.year - birthdate.year
15
16    if today.month < birthdate.month or (today.month ==
17        birthdate.month and today.day < birthdate.day):
18        age -= 1
19
20    return age
```

大模型生成了Bugly程序

```
>>> python
import unittest
import datetime
from calculate_age import calculate_age

class TestCalculateAge(unittest.TestCase):

    def test_valid_date(self):
        birthdate_str = "1990-05-15"
        age = calculate_age(birthdate_str)
        self.assertEqual(age, datetime.date.today().year - 1990)

    def test_invalid_date(self):
        birthdate_str = "invalid-date"
        age = calculate_age(birthdate_str)
        self.assertEqual(age, "Invalid date format")

    def test_birthday_not_passed(self):
        birthdate_str = f"{datetime.date.today().year}-{datetime.date.today().month}-{da
age = calculate_age(birthdate_str)
self.assertEqual(age, datetime.date.today().year - datetime.date.today().year -

    def test_birthday_passed(self):
        birthdate_str = f"{datetime.date.today().year}-{datetime.date.today().month}-{da
age = calculate_age(birthdate_str)
self.assertEqual(age, datetime.date.today().year - datetime.date.today().year)

if __name__ == '__main__':
    unittest.main()
```

大模型基于程序生成的测试代码

现在让LLM先思考验证点、生成单测

o1-mini

1. 验证点分析

在编写一个“计算年龄”的Python程序时，首先需要明确各种可能的输入情况和应对策略。这有助于确保程序的健壮性和准确性。以下是一些关键的验证点：

1. 输入格式验证

- 有效日期格式：确保输入的日期字符串符合预期的格式（例如，YYYY-MM-DD）。
- 无效日期格式：处理格式错误的输入（例如，DD/MM/YYYY、MM-DD-YYYY 或非日期字符串）。

2. 日期逻辑验证

- 未来日期：出生日期不能是未来的日期。
- 当前日期：如果出生日期是今天，年龄应为0岁。
- 闰年处理：
 - 出生于2月29日：
 - 在非闰年，如何处理（例如，是否视为2月28日或3月1日）。
 - 当前年份为闰年或非闰年时的年龄计算。
- 生日是否已过：在当前年份中，如果生日还未到，年龄应减1岁。

3. 边界条件

- 极端年龄：非常大的年龄（例如，100岁以上）是否能正确处理。
- 最小年龄：出生日期是当天。
- 错误数据类型：非字符串输入（例如，None、数字、列表等）。

4. 性能和异常处理

- 处理异常：程序应优雅地处理各种异常情况，并提供有意义的错误信息。
- 性能：对于大量数据的处理，性能是否满足需求。

2. 测试代码生成

基于上述验证点，我们可以使用Python的 unittest 模块来编写测试代码。以下是一个详细的测试程序示例，涵盖了各种测试用例和测试数据。

```
python
import unittest
from datetime import datetime, date
from age_calculator import calculate_age # 假设你的计算年龄函数在 age_calculator.py 文件中

class TestCalculateAge(unittest.TestCase):
    """测试 calculate_age 函数的单元测试类"""

    def test_valid_birthdate_past(self):
        """测试有效的过去日期"""
        birthdate = "1990-05-15"
        expected_age = self._calculate_expected_age(birthdate)
        self.assertEqual(calculate_age(birthdate), expected_age)

    def test_valid_birthdate_today(self):
        """测试今天的出生日期，年龄应为0"""
        today_str = date.today().strftime("%Y-%m-%d")
        self.assertEqual(calculate_age(today_str), 0)

    def test_future_birthdate(self):
        """测试未来日期，应该处理为错误或特殊值"""
        future_date = (date.today().replace(year=date.today().year + 1)).strftime("%Y-%m-%d")
        with self.assertRaises(ValueError):
            calculate_age(future_date)

    def test_invalid_date_format(self):
        """测试无效的日期格式"""
        invalid_dates = ["15-05-1990", "1990/05/15", "May 15, 1990", "1990.05.15", "invalid-date"]
        for bd in invalid_dates:
            with self.subTest(birthdate=bd):
                with self.assertRaises(ValueError):
                    calculate_age(bd)

    def test_leap_year_birthday_on_leap_year(self):
        """测试出生于2月29日且当前年为闰年"""
        # 设置当前日期为闰年
        original_today = date.today()
        leap_year = 2020 # 选择一个已知的闰年
        birthdate = "2000-02-29"
        expected_age = leap_year - 2000 - (date(leap_year, 2, 29) > date(leap_year, 2, 28))
        # 由于无法更改系统日期，假设 calculate_age 函数使用的年份为 leap_year
        # 这里仅作为示例，实际测试可能需要使用 mocking 框架来模拟日期
        # 因此，这个测试用例可能需要根据具体实现调整
        pass # 需要根据具体实现完成

    finally:
        pass # 恢复原始日期（如果有更改）

    def test_leap_year_birthday_on_non_leap_year(self):
        """测试出生于2月29日且当前年为非闰年"""
        # 同上，此测试需要通过 mocking 当前日期来模拟非闰年情况
        pass # 需要根据具体实现完成

    def test_extremely_old_age(self):
        """测试非常大的年龄，如150岁"""
        birthdate = "1870-01-01"
        expected_age = self._calculate_expected_age(birthdate)
        self.assertEqual(calculate_age(birthdate), expected_age)

    def test_invalid_input_type(self):
        """测试非字符串输入"""
        invalid_inputs = [None, 12345, ["1990", "05", "15"], {"year": 1990, "month": 5, "day": 15}]
        for input_value in invalid_inputs:
            with self.subTest(input=input_value):
                with self.assertRaises(TypeError):
                    calculate_age(input_value)

    def test_negative_age(self):
        """测试出生日期在未来，可能返回负数或抛出异常"""
        future_date = (date.today().replace(year=date.today().year + 1)).strftime("%Y-%m-%d")
        with self.assertRaises(ValueError):
            calculate_age(future_date)

    def test_birthday_not_yet_occurred_this_year(self):
        """测试今年生日尚未到来，年龄应减1"""
        today = date.today()
        birth_year = today.year - 30
        birth_month = today.month + 1 if today.month < 12 else 1
        birth_day = today.day
        birthdate = date(birth_year, birth_month, birth_day).strftime("%Y-%m-%d")
        expected_age = 29 # 30 - 1
        self.assertEqual(calculate_age(birthdate), expected_age)
```

```
try:
    leap_year = 2020 # 选择一个已知的闰年
    birthdate = "2000-02-29"
    expected_age = leap_year - 2000 - (date(leap_year, 2, 29) > date(leap_year, 2, 28))
    # 由于无法更改系统日期，假设 calculate_age 函数使用的年份为 leap_year
    # 这里仅作为示例，实际测试可能需要使用 mocking 框架来模拟日期
    # 因此，这个测试用例可能需要根据具体实现调整
    pass # 需要根据具体实现完成

finally:
    pass # 恢复原始日期（如果有更改）

def test_leap_year_birthday_on_non_leap_year(self):
    """测试出生于2月29日且当前年为非闰年"""
    # 同上，此测试需要通过 mocking 当前日期来模拟非闰年情况
    pass # 需要根据具体实现完成

def test_extremely_old_age(self):
    """测试非常大的年龄，如150岁"""
    birthdate = "1870-01-01"
    expected_age = self._calculate_expected_age(birthdate)
    self.assertEqual(calculate_age(birthdate), expected_age)

def test_invalid_input_type(self):
    """测试非字符串输入"""
    invalid_inputs = [None, 12345, ["1990", "05", "15"], {"year": 1990, "month": 5, "day": 15}]
    for input_value in invalid_inputs:
        with self.subTest(input=input_value):
            with self.assertRaises(TypeError):
                calculate_age(input_value)

def test_negative_age(self):
    """测试出生日期在未来，可能返回负数或抛出异常"""
    future_date = (date.today().replace(year=date.today().year + 1)).strftime("%Y-%m-%d")
    with self.assertRaises(ValueError):
        calculate_age(future_date)

def test_birthday_not_yet_occurred_this_year(self):
    """测试今年生日尚未到来，年龄应减1"""
    today = date.today()
    birth_year = today.year - 30
    birth_month = today.month + 1 if today.month < 12 else 1
    birth_day = today.day
    birthdate = date(birth_year, birth_month, birth_day).strftime("%Y-%m-%d")
    expected_age = 29 # 30 - 1
    self.assertEqual(calculate_age(birthdate), expected_age)
```

```
def _calculate_expected_age(self, birthdate_str):
    """辅助方法，用于计算预期年龄"""
    birthdate = datetime.strptime(birthdate_str, "%Y-%m-%d").date()
    today = date.today()
    age = today.year - birthdate.year
    if (today.month, today.day) < (birthdate.month, birthdate.day):
        age -= 1
    return age

if __name__ == '__main__':
    unittest.main()
```

3. 测试数据和用例说明

1. 测试有效的过去日期

- 输入: "1990-05-15"
- 预期结果: 根据当前日期计算出的年龄。

2. 测试今天的出生日期

- 输入: 今天的日期 (例如, "2024-04-27")
- 预期结果: 0

3. 测试未来日期

- 输入: 明年的同一天 (例如, 如果今天是 "2024-04-27", 则 "2025-04-27")
- 预期结果: 应抛出 ValueError

4. 测试无效的日期格式

- 输入: "15-05-1990", "1990/05/15", "May 15, 1990", "1990.05.15", "invalid-date"
- 预期结果: 应抛出 ValueError 或相应的异常。

5. 测试出生于2月29日且当前年为闰年

- 输入: "2000-02-29", 当前年为闰年 (例如, 2020)
- 预期结果: 准确计算年龄, 考虑闰年的特殊情况。

基于单测再生成程序代码

再基于测试代码class TestCalculateAge(unittest.TestCase), 生成计算年龄的python代码CalculateAge(birthdate_str)

o1-mini

```
python
# age_calculator.py

from datetime import datetime, date

def calculate_age(birthdate_str):
    """计算年龄，输入的日期格式为 YYYY-MM-DD。

    参数:
        birthdate_str (str): 出生日期字符串，格式为 'YYYY-MM-DD'

    返回:
        int: 计算出的年龄

    异常:
        TypeError: 如果输入不是字符串类型
        ValueError: 如果日期格式无效或出生日期在未来
    """
    if not isinstance(birthdate_str, str):
        raise TypeError("birthdate_str 必须是字符串，格式为 'YYYY-MM-DD'。")

    try:
        birthdate = datetime.strptime(birthdate_str, "%Y-%m-%d").date()
    except ValueError:
        raise ValueError("无效的日期格式。预期格式为 'YYYY-MM-DD'。")

    today = date.today()

    if birthdate > today:
        raise ValueError("出生日期不能是未来的日期。")

    age = today.year - birthdate.year

    # 处理出生日期为2月29日的情况
    if birthdate.month == 2 and birthdate.day == 29:
        if is_leap_year(today.year):
            birthday_this_year = date(today.year, 2, 29)
        else:
            # 非闰年，将生日视为2月28日
            birthday_this_year = date(today.year, 2, 28)

        else:
            try:
                birthday_this_year = birthdate.replace(year=today.year)
            except ValueError:
                # 处理2月29日以外的日期替换时可能出现的问题
                raise ValueError("无效的出生日期。")

        if today < birthday_this_year:
            age -= 1

    return age

def is_leap_year(year):
    """判断是否为闰年。

    参数:
        year (int): 年份

    返回:
        bool: 如果是闰年则返回True，否则返回False
    """
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
```

```
# 处理出生日期为2月29日的情况
if birthdate.month == 2 and birthdate.day == 29:
    if is_leap_year(today.year):
        birthday_this_year = date(today.year, 2, 29)
    else:
        # 非闰年，将生日视为2月28日
        birthday_this_year = date(today.year, 2, 28)

else:
    try:
        birthday_this_year = birthdate.replace(year=today.year)
    except ValueError:
        # 处理2月29日以外的日期替换时可能出现的问题
        raise ValueError("无效的出生日期。")

if today < birthday_this_year:
    age -= 1

return age

def is_leap_year(year):
    """判断是否为闰年。

    参数:
        year (int): 年份

    返回:
        bool: 如果是闰年则返回True，否则返回False
    """
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
```

代码说明

1. 函数 calculate_age(birthdate_str) :

- 输入验证:
 - 检查输入是否为字符串类型，若不是则抛出 TypeError。
 - 尝试将输入字符串转换为日期对象，若格式不正确则抛出 ValueError。
 - 检查出生日期是否在当前日期之后，若是则抛出 ValueError。
- 年龄计算:
 - 计算当前年份与出生年份的差。
 - 判断当前日期是否已经过了今年的生日，若未过则年龄减1。
- 闰年处理:
 - 如果出生日期是2月29日，且当前年份是闰年，则将生日视为2月28日进行比较。

“这样生成的代码才是高质量的
UTDD是代码生成的打开方式”

开发时间无需增加，反而会缩短

04

软件研发终极打开方式

LLM适合应用于“需求采集、分析与定义”

帮我全面收集“银行催收系统”的业务需求

GPT-4o mini

收集“银行催收系统”的业务需求可以从以下几个方面进行全面分析：

###1. 系统功能需求- 客户信息管理：

- 客户基本信息录入与维护 - 客户信用评级与风险评估 - 催收任务管理：
- 自动生成催收任务 - 任务分配与跟踪 - 催收进度记录与更新 - 催收方式：
- 电话催收 - 短信催收 - 邮件催收 - 上门催收 - 催收策略：
- 根据客户信用评级制定不同的催收策略 - 催收频率与时间段设置###2. 数据需求- 数据录入：
- 客户欠款信息 - 催收记录与反馈 - 数据分析：
- 催收成功率分析 - 客户还款行为分析 - 催收策略效果评估###3. 用户角色与权限- 角色定义：
- 催收专员 - 主管 - 数据分析师 - 系统管理员 - 权限管理：
- 不同角色的访问权限设置 - 数据修改与删除权限控制###4. 系统集成需求- 与核心银行系统集成：
- 实时获取客户账户信息 - 更新客户催收状态 - 与外部系统集成：
- 短信、邮件服务提供商接口 - 第三方信用评估机构接口###5. 报告与监控需求- 催收报告：
- 定期生成催收工作报告 - 催收效果分析报告 - 实时监控：
- 催收任务进度监控 - 客户反馈与投诉监控###6. 法规与合规需求- 合规性检查：
- 确保催收行为符合相关法律法规 - 客户隐私保护措施###7. 用户体验需求- 界面友好性：
- 简洁易用的用户界面 - 便捷的操作流程 - 培训与支持：
- 提供用户培训与技术支持###8. 性能与安全需求- 系统性能：
- 支持高并发访问 - 快速响应用户请求 - 数据安全：
- 数据加密存储与传输 - 定期备份与恢复机制###结论通过以上的分析，可以全面收集“银行催收系统”的业务需求。这些需求将为系统的设计与开发提供重要依据，确保系统能够有效支持银行的催收工作。

🗨️ 🗨️ 🗨️

Claude 3.5 Sonnet

我将从多个维度全面分析银行催收系统的业务需求：

📄 银行催收系统业务需求分析
点击预览

这份需求分析涵盖了银行催收系统的主要业务场景和功能需求。根据具体项目情况，可以进一步细化和调整各项需求。建议在实际项目中：

1. 与业务部门深入沟通，确认具体需求的优先级
2. 结合银行现有系统情况，确定集成方案
3. 考虑当地监管要求，完善合规要求

GPT-4o mini

聊天工具有了新家！您可以在这里访问所有聊天控件。

2. 业务需求

2.1 高效的债务管理

智能催收系统需具备强大的数据处理能力，能够快速、准确地处理海量债务信息²。通过智能算法分析借款人的行为模式和还款能力，系统能够制定个性化催收策略，提高催收成功率。

2.2 自动化催收流程

系统应实现催收业务的自动化操作，包括账户分配、催收策略、自动将账户分配至相应的催收队列，减少人工干预。

2.3 多渠道催收支持

智能催收系统需支持多种催收渠道，如电话催收、短信催收、邮件催收等，并能根据客户偏好选择最合适的沟通方式¹。多渠道的支持有助于提升催收效率。

2.4 数据统计与分析

系统应具备完善的数据统计与分析功能，对催收过程中的关键指标进行实时监控与分析，银行可以优化催收策略，识别高风险客户，制定有针对性的催收计划。

六、合规与监管要求

1. 监管合规

- 银保监会规定
- 人民银行规定
- 地方金融监管要求

2. 数据合规

- 个人信息保护
- 数据安全要求
- 数据留存要求

3. 催收合规

4. 典型用例

用例1：自动分配催收任务

描述：系统根据预设的策略和账户信息，自动将逾期账户分配至不同的催收队列，并分配给相应的催收人员。

参与者：催收系统、催收经理、催收人员

2.2 自动化催收流程

自动化催收流程旨在通过系统化和智能化的方法，提升催收效率，减少人工干预，确保催收工作的连续性与一致性。以下将详细描述自动化催收流程的各个环节及其具体步骤，以确保系统能够高效、精准地执行催收任务。

2.2.1 流程启动条件

自动化催收流程的启动基于预设的业务规则和客户状态。具体包括：

2.2.1.1 逾期条件触发

- **逾期天数：**当贷款或欠款逾期超过预设天数（如30天）时，触发自动催收流程。
- **欠款金额：**依据欠款金额的大小，确定催收的优先级和策略。
- **客户分类：**根据客户的风险等级、还款能力等因素，决定是否进入自动化催收流程。

2.2.1.2 定期催收周期

- **定期检查：**系统定期（如每日、每周）检查所有客户的欠款状态，识别需要催收的客户。
- **批量处理：**对符合条件的客户进行批量催收任务的生成和分配。

2.2.1.3 特殊事件触发

- **收入变化：**客户收入发生重大变化时，重新评估其还款能力，并决定是否启动催收流程。
- **信用更新：**客户信用评级下降或有负面记录时，触发自动催收措施。

定义用户故事及其验收标准

用例1: 自动分配催收任务相关的用户故事

为确保“自动分配催收任务”功能能够高效支持业务需求，以下是针对业务上常见且高价值用户角色的用户故事。这些用户故事聚焦于关键角色的核心需求，确保系统能够有效提升催收效率和业务绩效。

1. 作为催收主管,

- 我希望系统能够根据客户的逾期金额和风险等级自动分配催收任务,
- 以便更有效地利用资源, 优先处理高风险和大量欠款的客户, 提升整体催收效率和回收率。

2. 作为催收人员,

- 我希望能够接收到系统自动分配的催收任务,
- 以便能够及时开展工作, 提升催收效率和回收率, 确保任务按时完成, 减少逾期风险。

3. 作为系统管理员,

- 我希望能够配置和调整自动分配任务的规则和算法,
- 以便确保分配策略符合最新的业务需求、政策变化和合规要求, 保持系统的灵活性和适应性。

4. 作为客户服务代表,

- 我希望系统在分配催收任务时考虑客户的沟通偏好和历史互动记录,
- 以便制定更具针对性的催收策略, 提高客户响应率和满意度。

5. 作为财务主管,

- 我希望能够通过系统查看自动分配催收任务后的回收情况和财务影响,
- 以便进行财务分析和决策支持, 优化资金流动和风险管理。

用户故事:

作为催收人员,
我希望能够接收到系统自动分配的催收任务,
以便能够及时开展工作, 提升催收效率和回收率,
确保任务按时完成, 减少逾期风险。

为确保该用户故事在各种应用场景下的有效性, 以下是详细的验收标准, 采用行为驱动开发(BDD)的Given-When-Then格式编写。

场景 1: 成功接收自动分配的催收任务

- Given 系统中存在待催收的客户任务,
- And 这些任务符合催收人员的分配规则 (如地域、风险等级等),
- And 催收人员已登录系统,
- When 系统自动分配任务给催收人员,
- Then 催收人员应在其工作界面上看到新分配的催收任务,
- And 系统应发送通知 (如邮件、短信或应用内通知) 提醒催收人员有新任务分配。

场景 2: 催收任务分配依据优先级

- Given 系统中有多个待催收任务, 具有不同的逾期金额和风险等级,
- And 催收人员的工作负载在允许范围内,
- When 系统进行任务自动分配,
- Then 系统应优先分配高逾期金额和高风险等级的任务给催收人员,
- And 催收人员应首先看到这些高优先级的任务。

场景 3: 催收人员接收多任务分配

- Given 催收人员当前未超负荷且有多个待分配任务,
- When 系统自动分配多个催收任务给该催收人员,
- Then 催收人员应在其工作界面上同时看到所有新分配的催收任务,
- And 系统应确保分配任务的总量不会超过催收人员的每日处理能力。

场景 4: 催收人员拒绝或重新分配任务

- Given 催收人员收到系统自动分配的催收任务,
- When 催收人员因某些原因无法处理该任务,
- Then 催收人员应能够在系统中拒绝该任务或请求重新分配,
- And 系统应根据预定义的规则将该任务分配给其他合适的催收人员,
- And 系统应记录任务的重新分配历史。

场景 5: 催收任务到期提醒

- Given 催收任务具有明确的截止日期和提醒时间,
- And 任务尚未完成且接近截止时间,
- When 系统检测到任务截止时间临近 (如提前24小时),
- Then 系统应向催收人员发送提醒通知,
- And 在催收人员的工作界面上高亮显示即将到期的任务。

场景 6: 催收任务完成确认

- Given 催收人员已处理完分配的催收任务,
- When 催收人员标记任务为完成,
- Then 系统应更新任务状态为“已完成”,
- And 系统应记录完成时间和相关备注,
- And 系统应释放资源以便分配新的任务。

场景 7: 系统异常处理

- Given 在任务分配过程中系统发生故障,
- When 系统无法完成任务分配,
- Then 系统应自动记录错误日志,
- And 向系统管理员发送异常通知,
- And 确保未分配的任务在系统恢复后重新尝试分配,
- And 催收人员在异常期间保持已收到任务的准确性。

场景 8: 多设备同步

- Given 催收人员使用多个设备 (如桌面电脑和移动设备) 访问系统,
- When 系统在一个设备上分配新任务,
- Then 其他设备上的催收人员界面应实时同步显示新分配的任务,
- And 确保任务状态在所有设备间的一致性。

场景 9: 任务通知方式选择

- Given 催收人员在个人设置中选择了首选的通知方式 (如邮件、短信或应用内通知),
- When 系统自动分配任务,
- Then 系统应通过催收人员选择的首选方式发送通知,
- And 确保其他通知方式不会重复发送通知。

场景 10: 任务分配优先级调整

- Given 催收主管根据业务需求调整了任务分配的优先级规则,
- When 系统进行新的任务分配,
- Then 系统应基于最新的优先级规则自动分配任务,
- And 催收人员应立即反映新的任务优先级。

基于GWT验收标准生成测试代码

4. 编写步骤定义 (Step Definitions)

在 `src/test/java/stepDefinitions` 目录下创建一个类 `AssignCollectionTasksSteps.java` 并编写步骤定义:

```
java
package stepDefinitions;

import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.*;
import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;

public class AssignCollectionTasksSteps {

    private WebDriver driver;

    @Before
    public void setup() {
        // 使用 WebDriverManager 自动管理浏览器驱动
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
    }

    @After
    public void teardown() {
        if(driver != null) {
            driver.quit();
        }
    }

    @Given("系统中存在待催收的客户任务且符合分配规则")
    public void 系统中存在待催收的客户任务且符合分配规则() {
        // 模拟系统中存在符合分配规则的催收任务
        // 这里可以通过数据库预置数据, 或通过界面操作创建任务
        // 示例: 访问后台管理界面创建任务
        driver.get("http://localhost:8080/admin/createTask");
    }
}
```

5. 创建运行测试的 Runner 类

在 `src/test/java/runners` 目录下创建一个类 `RunCucumberTest.java` :

```
java
package runners;

import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features",
    glue = {"stepDefinitions"},
    plugin = { "pretty", "html:target/cucumber-repo" },
    monochrome = true
)
public class RunCucumberTest {
    // 此类无需编写任何代码, Cucumber 会自动查找特征文件并运行
}
```

9. 完整示例汇总

结合上述步骤, 以下是项目结构的简要展示:

```
css
collection-tasks-automation/
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   └── test
│       ├── java
│       │   ├── runners
│       │   │   └── RunCucumberTest.java
│       │   └── stepDefinitions
│       │       └── AssignCollectionTasksSteps.java
│       └── resources
│           └── features
│               └── assign_collection_tasks.feature
```

10. 扩展实施其他场景

根据前述 10 个验收标准, 可以依次为每个场景编写对应的特性文件和步骤定义。以下是一个额外场景的示例:

```
gherkin
Scenario: 催收任务完成确认
Given 催收人员已接收到催收任务
When 催收人员标记任务为完成
Then 系统应更新任务状态为“已完成”
And 系统应记录完成时间和相关备注
And 系统应释放资源以便分配新的任务
```

对应的步骤定义:

```
java
@Given("催收人员已接收到催收任务")
public void 催收人员已接收到催收任务() {
    // 假设催收人员已登录并接收到任务
    driver.get("http://localhost:8080/collector/tasks");
    WebElement task = driver.findElement(By.xpath("//table[@id='tasksTable']/tbody/tr[1]/td[contains(@class, 'task')]"));
    Assert.assertTrue(task.isDisplayed());
}

@When("催收人员标记任务为完成")
public void 催收人员标记任务为完成() {
    WebElement completeButton = driver.findElement(By.xpath("//table[@id='tasksTable']/tbody/tr[1]/td[contains(@class, 'task')]"));
    completeButton.click();
}

@Then("系统应更新任务状态为“已完成”")
public void 系统应更新任务状态为已完成() {
    WebElement status = driver.findElement(By.xpath("//table[@id='tasksTable']/tbody/tr[1]/td[contains(@class, 'status')]"));
    Assert.assertEquals("已完成", status.getText());
}

@And("系统应记录完成时间和相关备注")
public void 系统应记录完成时间和相关备注() {
}
```

再基于BDD测试代码生成前后端代码

基于上面测试代码，生成支持java sprint cloud的后端程序和前端Vue.js代码

o1-preview

当然！基于您使用 Selenium WebDriver 和 Cucumber 开发的自动化测试代码，我将指导您如何基于 Java Spring Boot（结合 Spring Cloud 功能）创建后端，以及使用 Vue.js 创建前端。这将帮助您建立一个支持您 BDD 场景中描述的功能的全栈应用程序。

使用 Java Spring Boot 和 Spring Cloud 进行后端开发

1. 项目设置

使用 Spring Initializr 创建一个新的 Spring Boot 项目：

- 项目类型：Maven Project
- 语言：Java
- Spring Boot 版本：选择最新的稳定版本
- 项目元数据：
 - Group: com.example
 - Artifact: collection-system
- 依赖项：
 - Spring Web
 - Spring Data JPA
 - H2 Database（用于开发和测试）
 - Spring Security（如果需要身份验证）
 - Spring Cloud Dependencies（如果使用微服务功能）

在您的 pom.xml 文件中添加 Spring Cloud 依赖项：

```
<?xml version="1.0" encoding="UTF-8" ?>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Hoxton.SR12</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

4. 开发服务层

实现一个服务类来处理任务分配和管理的业务逻辑。

```
java
package com.example.collectionsystem.service;

import com.example.collectionsystem.model.CollectionTask;
import com.example.collectionsystem.repository.CollectionTaskRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
public class CollectionTaskService {

    @Autowired
    private CollectionTaskRepository taskRepository;

    public CollectionTask createTask(CollectionTask task) {
        task.setStatus("Pending");
        return taskRepository.save(task);
    }

    public List<CollectionTask> getTasksForCollector(String collectorUsername) {
        return taskRepository.findByCollectorUsernameAndStatus(collectorUsername, "Pending");
    }

    public void assignTasksToCollector(String collectorUsername) {
        // 根据优先级实现任务分配逻辑
        List<CollectionTask> unassignedTasks = taskRepository.findByCollectorUsernameAndStatus(collectorUsername, "Pending");
        for (CollectionTask task : unassignedTasks) {
            task.setCollectorUsername(collectorUsername);
            task.setAssignedAt(LocalDateTime.now());
            taskRepository.save(task);
        }
        // 如有必要，根据工作负载限制分配的任务数量
    }

    public void completeTask(Long taskId) {
        CollectionTask task = taskRepository.findById(taskId).orElseThrow();
        task.setStatus("Completed");
        task.setCompletedAt(LocalDateTime.now());
        taskRepository.save(task);
    }

    // 根据需要添加其他方法
}
```

5. 实现控制器层

创建 REST 控制器以为前端和自动化测试提供端点。

```
java
package com.example.collectionsystem.controller;

import com.example.collectionsystem.model.CollectionTask;
import com.example.collectionsystem.service.CollectionTaskService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/tasks")
public class CollectionTaskController {

    @Autowired
    private CollectionTaskService taskService;

    @PostMapping("/create")
    public CollectionTask createTask(@RequestBody CollectionTask task) {
        return taskService.createTask(task);
    }

    @GetMapping("/collector/{username}")
    public List<CollectionTask> getTasksForCollector(@PathVariable String username) {
        return taskService.getTasksForCollector(username);
    }

    @PostMapping("/assign/{username}")
    public void assignTasksToCollector(@PathVariable String username) {
        taskService.assignTasksToCollector(username);
    }

    @PostMapping("/complete/{taskId}")
    public void completeTask(@PathVariable Long taskId) {
        taskService.completeTask(taskId);
    }

    // 根据需要添加其他端点
}
```

使用 Vue.js 进行前端开发

1. 设置 Vue.js 项目

使用 Vue CLI 创建一个新项目：

```
bash
vue create collection-frontend
```

选择默认预设或根据需要手动选择功能。

2. 安装必要的依赖项

导航到您的项目目录并安装依赖项：

```
bash
cd collection-frontend
npm install axios vue-router
```

3. 配置 Axios 以进行 API 调用

创建一个用于 API 通信的 axios 实例。

```
javascript
// src/plugins/axios.js
import axios from 'axios';

const apiClient = axios.create({
  baseURL: 'http://localhost:8080/api',
  withCredentials: true,
});

export default apiClient;
```

4. 设置 Vue Router

6. 创建仪表板组件

显示分配给催收人员的任务。

```
html
<!-- src/views/Dashboard.vue -->
<template>
  <div class="dashboard">
    <h2>催收任务列表</h2>
    <table id="tasksTable">
      <thead>
        <tr>
          <th>客户名称</th>
          <th>逾期金额</th>
          <th>风险等级</th>
          <th>状态</th>
          <th>操作</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="task in tasks" :key="task.id">
          <td>{{ task.customerName }}</td>
          <td>{{ task.overdueAmount }}</td>
          <td>{{ task.riskLevel }}</td>
          <td class="status">{{ task.status }}</td>
          <td>
            <button class="completeTask" @click="completeTask(task.id)">完成</button>
          </td>
        </tr>
      </tbody>
    </table>
    <div id="newTaskNotification" v-if="showNotification">
      您有一个新的催收任务已分配
    </div>
  </template>

<script>
import apiClient from '@/plugins/axios';

export default {
  data() {
    return {
      tasks: [],
      showNotification: false,
    };
  },
  async created() {
    await this.fetchTasks();
  },
  methods: {
    async fetchTasks() {
      try {
        const response = await apiClient.get('tasks/collector/${this.getUsername()}');
        this.tasks = response.data;
      } catch (error) {
        console.error('Error fetching tasks:', error);
      }
    }
  }
};
```

大模型时代软件研发终极打开方式

```
collection-tasks-automation/  
├── pom.xml  
├── src  
│   ├── main  
│   │   └── java  
│   └── test  
│       ├── java  
│       │   ├── runners  
│       │   │   └── RunCucumberTest.java  
│       │   ├── stepDefinitions  
│       │   │   └── AssignCollectionTasksSteps.java  
│       └── resources  
│           ├── features  
│           │   └── assign_collection_tasks.feature
```

测试脚本

相互验证
/博弈

```
java  
package com.example.collectionsystem.repository;  
  
import com.example.collectionsystem.model.CollectionTask;  
import org.springframework.data.jpa.repository.JpaRepository;  
import java.util.List;  
  
public interface CollectionTaskRepository extends JpaRepository<CollectionTask, Long> {  
    List<CollectionTask> findByCollectorUsernameAndStatus(String collectorUsername, String status);  
}
```

程序/产品代码

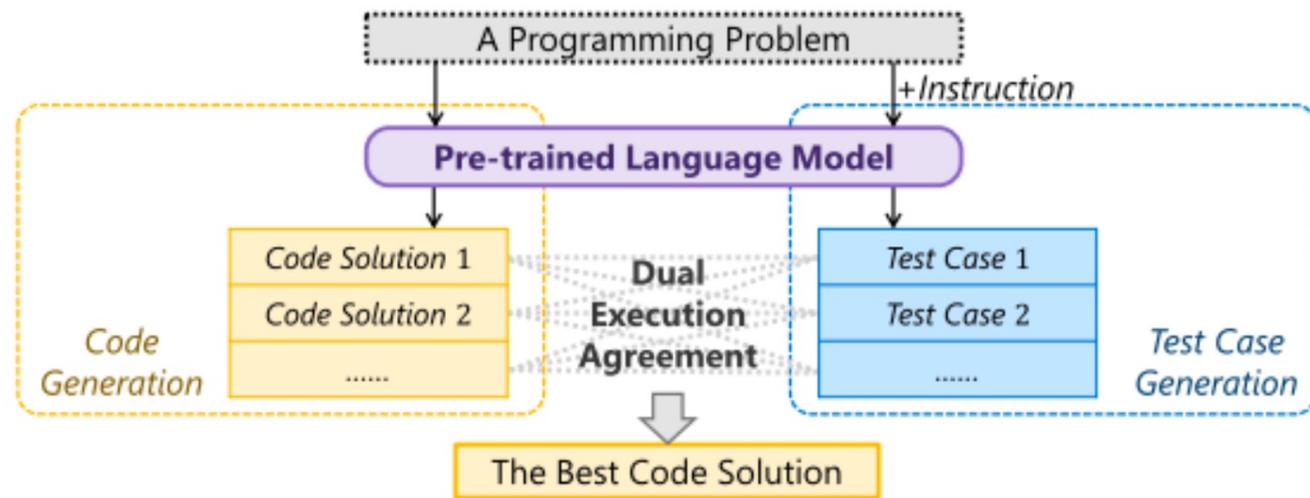
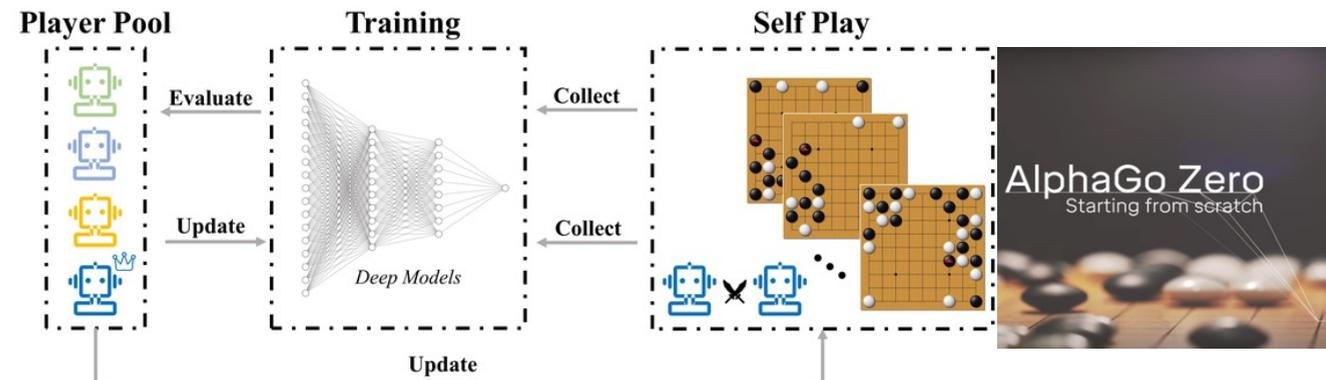
由LLM生成

由LLM生成



由LLM生成需求及其验收标准

背后是研发
工程平台



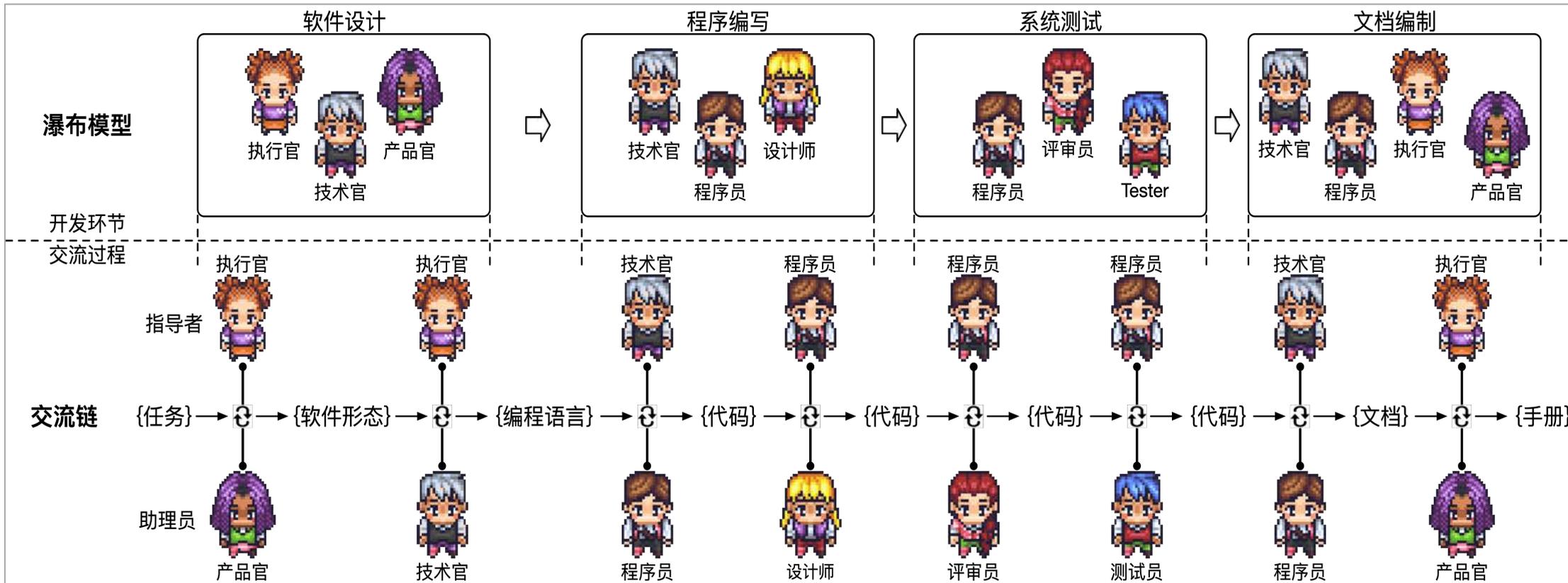
* Refer to our paper: Bei CHEN et. al., CodeT: Code Generation with Generated Tests, <https://arxiv.org/abs/2207.10397>

效果提升明显

05

未来展望

多智能体协同开发软件



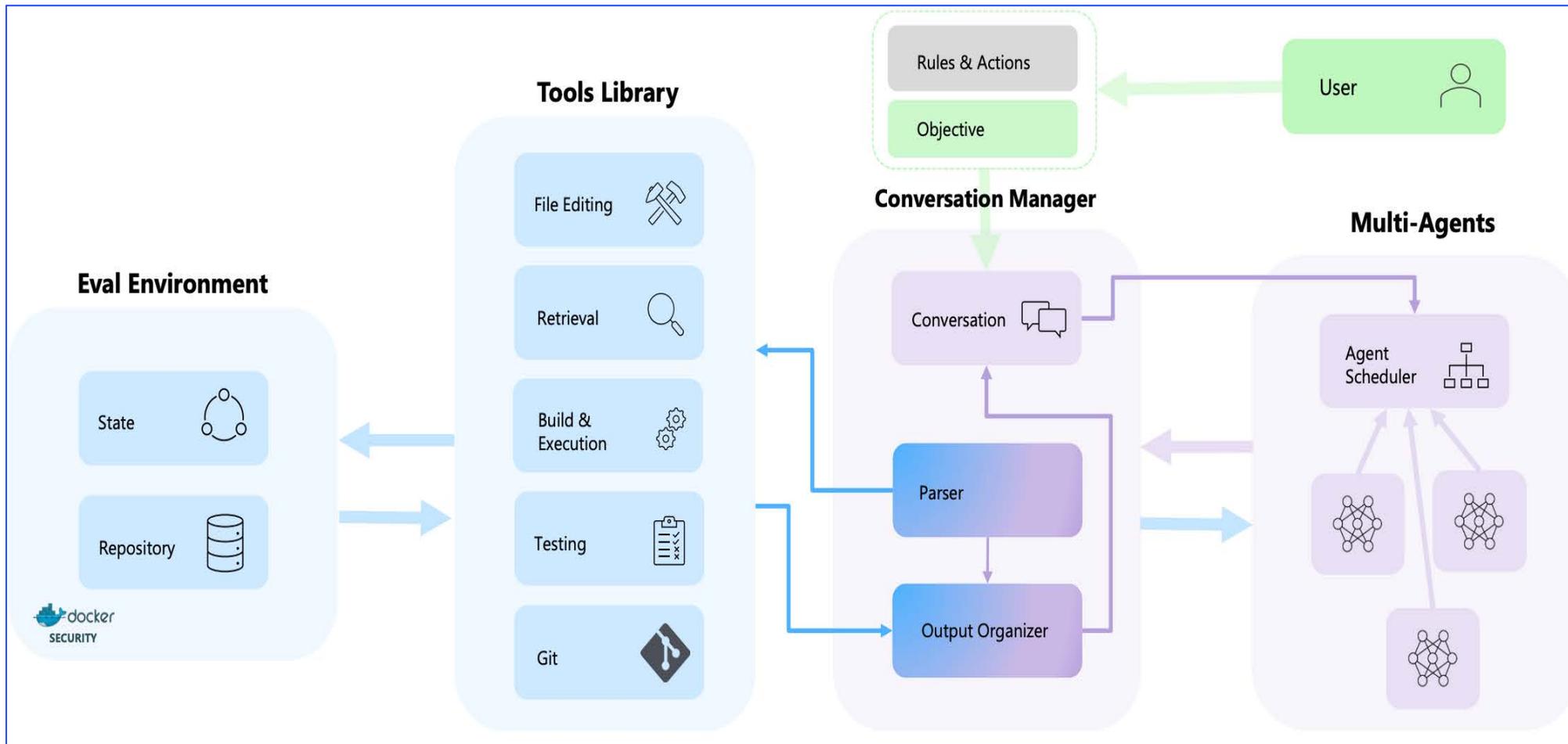
2025 Top 10 Strategic Technology Trends

- AI imperatives and risks**
 - Agentic AI
 - AI Governance Platforms
 - Disinformation Security
- New frontiers of computing**
 - Post-Quantum Cryptography
 - Ambient Invisible Intelligence
 - Energy-Efficient Computing
 - Hybrid Computing
- Human-machine synergy**
 - Spatial Computing
 - Polyfunctional Robots
 - Neurological Enhancement

Gartner

不仅有我们开发出来的智能体、数字人，我们开发人员也是智能体

▶ 具有一定自主能力的智能软件开发平台



- ① Augmented working
- ② Real-time automated decision-making
- ③ Responsible AI
- ④ Generative video
- ⑤ Next-gen voice assistants

- ⑥ AI legislation and regulation
- ⑦ Autonomous AI agents
- ⑧ Navigating a post-truth world
- ⑨ Quantum AI
- ⑩ Sustainable AI

2025

实时决策、自主智能体...

科技生态圈峰会 + 深度研习

—1000+ 技术团队的选择



敦煌站

K+思考周®研习社

时间: 2025.08.29-30



上海站

K+金融专场

时间: 2025.10.17-18



香港站

K+思考周®研习社

时间: 2025.11.25-26



K+峰会详情



上海站

AI+研发数字峰会

时间: 2025.05.17-18



北京站

AI+研发数字峰会

时间: 2025.08.08-09



深圳站

AI+研发数字峰会

时间: 2025.11.28-29



AIDD峰会详情



利用AI技术深化计算机对现实世界的理解
推动研发进入智能化时代

