EvalPlus总结

> EvalPlus通过结合LLM和变异测试技术，创建了更严格、更全面的代码评估框架。研究表明，现有的评估方法高估了LLM生成代码的正确性，而EvalPlus能够暴露先前未被检测的错误。这项工作不仅提供了更可靠的评估工具，还为未来LLM代码生成的研究和应用指明了方向。
>
> 和上一篇一样，也是UIUC的博士写的，甚至都是刘，但不是一个人。发在了NeurIps上，被引量900+。
>
> 体验是这篇文章阅读体验真的真的真的远高于上一篇，问题讲的很清楚，思路明确。上一篇有点在评估最后有点偏离重心，整个的行文逻辑也有点乱。这一篇就很优美简洁。不愧是900+

1. 背景

论文"ChatGPT生成的代码真的正确吗？大型语言模型代码生成的严格评估"针对AI驱动的代码生成领域中一个关键问题：当前评估LLM生成代码的测试框架不足。

现有评估方法的主要问题：

- **测试不足**：当前的编程基准测试如HumanEval每个编码问题平均包含不到10个测试用例，这些测试通常过于简单，无法全面探索边界情况和功能正确性。
- **问题描述不精确**：现有基准测试中的任务描述往往过于模糊，未能明确指定输入域或异常处理要求。
- **不正确的标准解答**：发现HumanEval中约11%的"标准答案"存在缺陷，包括未处理的边界情况和逻辑错误。

2. 研究贡献与创新

主要贡献：

1. **首次深入研究**了当前编程基准测试中测试不足问题，这可能导致功能正确性被大幅高估。

EvalPlus Tests

#	Model	pass@1
1	O1 Preview (Sept 2024)	89
2	O1 Mini (Sept 2024)	89
3	Qwen2.5-Coder-32B-Instruct	87.2
4	GPT 4o (Aug 2024)	87.2
5	DeepSeek-V3 (Nov 2024)	86.6
6	GPT-4-Turbo (April 2024)	86.6
7	DeepSeek-V2.5 (Nov 2024)	83.5
8	GPT 4o Mini (July 2024)	83.5
9	DeepSeek-Coder-V2-Instruct	82.3
10	Claude Sonnet 3.5 (June 2024)	81.7
11	GPT-4-Turbo (Nov 2023)	81.7
12	Grok Beta	80.5
13	Gemini 1.5 Pro 002	79.3
14	GPT-4 (May 2023)	79.3
15	CodeQwen1.5-7B-Chat	78.7

Base Tests

#	Model	pass@1
1	O1 Preview (Sept 2024)	96.3
2	O1 Mini (Sept 2024)	96.3
3	GPT 4o (Aug 2024)	92.7
4	Qwen2.5-Coder-32B-Instruct	92.1
5	DeepSeek-V3 (Nov 2024)	91.5
6	DeepSeek-V2.5 (Nov 2024)	90.2
7	GPT-4-Turbo (April 2024)	90.2
8	Gemini 1.5 Pro 002	89
9	Grok Beta	88.4
10	GPT 4o Mini (July 2024)	88.4
11	GPT-4 (May 2023)	88.4
12	Claude Sonnet 3.5 (June 2024)	87.2
13	DeepSeek-Coder-V2-Instruct	85.4
14	GPT-4-Turbo (Nov 2023)	85.4
15	CodeQwen1.5-7B-Chat	83.5

2. **提出了EvalPlus**：一个评估框架，通过自动化测试生成来严格评估LLM合成代码的功能正确性。
3. **创建了HumanEval+**：将HumanEval基准测试的测试用例扩展了80倍，并修复了其中的错误解答。
4. **开发了测试套件缩减技术**：创建了HumanEval+-Mini，在保持相似测试效果的同时将测试规模缩小了47倍。

3. 研究方法与技术框架

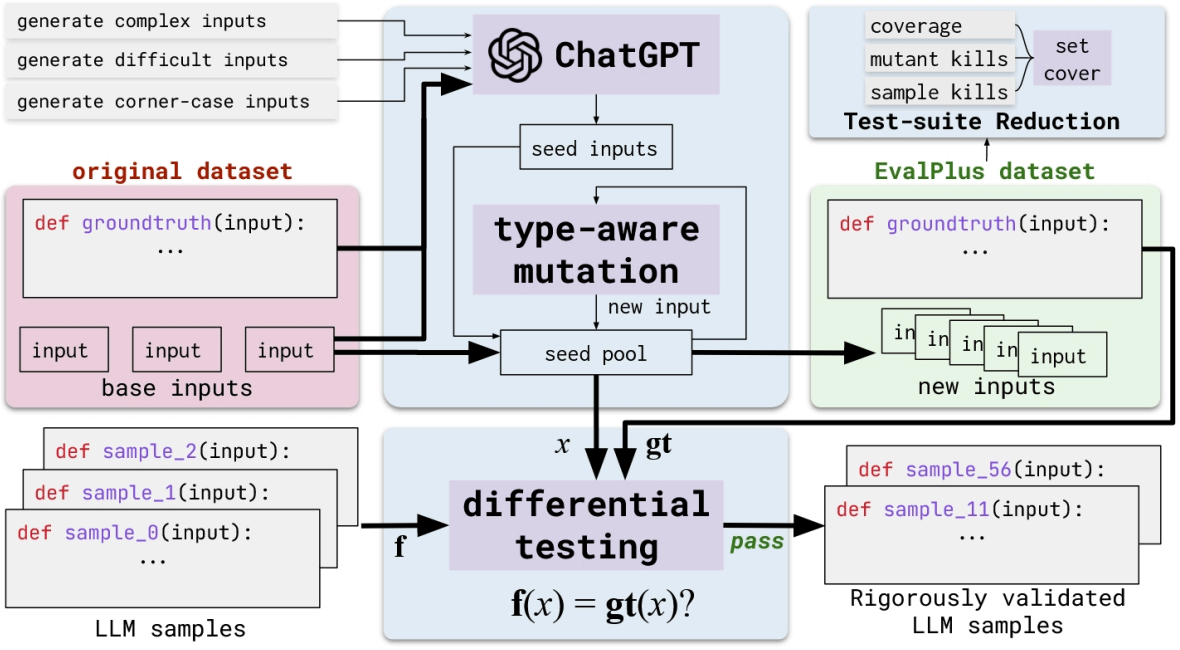

Figure 2: Overview of EvalPlus

EvalPlus工作流：

1. 从原始数据集（original dataset）获取标准答案和基础测试输入
2. 使用ChatGPT生成高质量的种子测试输入
3. 对种子输入进行类型感知变异，扩展生成大量测试用例
4. 使用差分测试评估LLM生成代码的正确性
5. 应用测试套件缩减技术，生成规模更小但效果相当的测试集
6. 最终得到EvalPlus数据集，用于更严格地评估LLM代码生成能力

EvalPlus的核心工作流程包括三个主要组件：

a. 自动化测试输入生成

- **种子初始化**：利用ChatGPT生成高质量的种子输入，为后续变异提供基础

  > 图中左侧显示了三种生成指令："generate complex inputs"、"generate difficult inputs"和"generate corner-case inputs"
  >
  > 这些指令被发送给ChatGPT，引导其生成高质量的种子测试输入
  >
  > ChatGPT可以理解原始问题和现有测试用例的模式，从而生成有意义的新测试用例
  >
  > 这一步骤解决了手动创建复杂测试用例耗时且困难的问题

- **类型感知输入变异**：基于种子输入执行类型感知变异，快速生成大量有效的测试输入

  > 从ChatGPT获得种子输入后，这些输入被存入"seed pool"（种子池）
  >
  > 种子池中的输入随后经过"type-aware mutation"（类型感知变异）模块处理
  >
  > 该模块会根据输入的数据类型执行相应的变异操作（如图中未显示但在论文表1中详述）
  >
  > 例如，对整数进行加减变异，对列表进行元素添加/删除/替换，对字符串进行子串操作等
  >
  > 生成的新输入会被添加回种子池，形成一个迭代过程
  >
  > 这种方法可以快速大量地生成有效测试输入，比纯使用ChatGPT更高效

- **差分测试评估**

  评估阶段采用了差分测试方法，确保了评估的可靠性：

  > 系统将生成的测试输入同时提供给标准解答函数和LLM生成的代码样本
  >
  > 通过比较两者的输出（f(x) = gt(x)?），严格评估LLM代码的功能正确性
  >
  > 这种方法避免了传统评估方法的主观性，提供了客观的正确性度量

b. 测试套件缩减

为解决测试执行成本问题，EvalPlus设计了智能的测试套件缩减机制：

- 系统实现了三种互补的缩减策略：
  - **代码覆盖率**：保留能维持相同代码分支覆盖的最小测试子集
  - **变异体杀死**：确保能检测相同变异错误的最小测试集合
  - **LLM样本杀死**：保证能检测错误LLM样本的最小测试集合

- 通过集合覆盖算法，系统可以将大规模测试集（如HumanEval+平均每任务764.1个测试）缩减至更小规模（如HumanEval+-Mini平均每任务16.1个测试），同时保持相当的测试效果

c. 程序输入契约

为增强评估的精确性，EvalPlus引入了程序契约机制：

- 采用契约式编程理念，通过代码断言形式明确标注函数前置条件
- 这些契约既过滤掉无效输入，又提供了更为明确的函数行为描述
- 确保了评估过程中使用的测试用例都是合理且有意义的

这种多层次、多组件的设计使EvalPlus能够提供比现有评估方法更为严格、全面且高效的代码评估能力，为LLM代码生成领域建立了新的评估标准。

4. 实验评估与关键发现

实验设置

- 对26个流行的LLM模型进行全面测试
- 使用多种温度设置(0.2, 0.4, 0.6, 0.8)和贪婪解码
- 评估pass@k指标(k∈{1, 10, 100})的变化

> pass@k是评估代码生成模型性能的关键指标，它度量的是：**模型从k个生成样本中至少有一个通过所有测试用例的概率**。
>
> pass@1*：使用贪婪解码（温度为0）生成的确定性样本通过测试的概率

关键发现

a. HumanEval+暴露了大量之前未被检测的错误代码

> 表 3：在 HumanEval 和 HumanEval $^+$上评估 LLMs。除 InCoder、CodeGen2、StarCoder 和 SantaCoder（它们执行填充）外，所有模型均使用自回归生成。$k = 1^*$表示使用贪婪解码完成的 pass@1。T_k^*表示最优的 pass@ k温度。

Table 3: Evaluating LLMs on HUMANEVAL and HUMANEVAL+. All models, except for INCODER, CodeGen2, StarCoder and SantaCoder which perform infilling, use auto-regressive generation. $k{=}1^{\star}$ marks pass@1 done with greedy decoding. T_k^* denotes the optimal pass@k temperature.

	Size	pass@k	$k{=}1^{\star}$	$k{=}1$	$k{=}10$	$k{=}100$	T_1^*	T_{10}^*	T_{100}^*
GPT-4 [49]	N/A	base	88.4						
		+extra	76.2						
Phind-CodeLlama [52]	34B	base	71.3	71.6	90.5	96.2	.2	.8	.8
		+extra	67.1	67.0	85.0	92.5	.2	.8	.8
WizardCoder-CodeLlama [38]	34B	base	73.2	61.6	85.2	94.5	.2	.8	.8
		+extra	64.6	54.5	78.6	88.9	.2	.8	.8
ChatGPT [48]	N/A	base	73.2	69.4	88.6	94.0			
		+extra	63.4	62.5	82.1	91.1			
CODELLAMA [54]	34B	base	51.8	52.0	82.4	95.0	.2	.8	.8
		+extra	42.7	43.1	73.7	89.4	.2	.8	.8
	13B	base	42.7	44.6	77.6	92.7	.4	.8	.8
		+extra	36.6	37.4	69.4	88.2	.4	.8	.8
	7B	base	37.8	39.2	69.1	89.7	.2	.8	.8
		+extra	34.1	34.5	61.4	82.9	.2	.8	.8
StarCoder [13]	15B	base	34.1	32.2	56.7	84.2	.2	.8	.8
		+extra	29.3	27.8	50.3	75.4	.2	.8	.8
CodeGen [46]	16B	base	32.9	32.2	56.0	81.5	.2	.6	.8
		+extra	26.8	27.2	48.4	71.4	.2	.6	.8
	6B	base	29.3	27.7	46.9	72.7	.2	.6	.8
		+extra	25.6	23.6	41.0	64.6	.2	.6	.8
	2B	base	24.4	18.4	39.8	66.8	.2	.8	.8
		+extra	20.7	15.1	34.8	55.8	.2	.2	.8
CODET5+ [64]	16B	base	31.7	32.2	58.5	83.5	.2	.6	.8
		+extra	26.2	27.4	51.1	76.4	.2	.6	.8
MISTRAL [26]	7B	base	28.7	28.1	55.2	83.8	.2	.8	.8
		+extra	23.8	23.7	48.5	76.4	.2	.8	.8
CodeGen2 [45]	16B[4]	base	19.5						
		+extra	16.5						
	7B	base	18.3	17.9	30.9	50.9	.2	.6	.8
		+extra	16.5	15.9	27.1	45.4	.2	.6	.8
	3B	base	15.9	15.2	23.9	38.6	.2	.4	.8
		+extra	12.8	12.9	21.2	34.3	.2	.4	.8
	1B	base	11.0	10.2	15.1	24.7	.2	.6	.6
		+extra	9.1	8.7	13.7	21.2	.2	.6	.6
VICUNA [12]	13B	base	16.5	15.3	30.1	54.8	.2	.8	.8
		+extra	15.2	13.9	25.8	46.7	.2	.8	.8
	7B	base	11.6	10.9	23.8	42.3	.2	.6	.6
		+extra	11.0	10.3	20.3	35.0	.2	.6	.6
SantaCoder [2]	1.1B	base	14.6	16.6	29.2	45.4	.4	.6	.8
		+extra	12.8	14.2	26.2	40.6	.4	.6	.8
INCODER [18]	6.7B	base	15.9	15.6	27.7	45.0	.2	.4	.6
		+extra	12.2	12.4	22.2	38.9	.2	.6	.6
	1.3B	base	12.2	10.0	15.9	25.2	.2	.6	.6
		+extra	10.4	7.9	13.5	20.7	.2	.6	.4
GPT-J [63]	6B	base	12.2	11.3	17.7	31.8	.2	.6	.6
		+extra	10.4	9.5	15.2	25.9	.2	.6	.6
GPT-NEO [5]	2.7B	base	7.9	6.5	11.8	20.7	.2	.6	.6
		+extra	6.7	6.0	9.0	16.8	.2	.6	.6
PolyCoder [70]	2.7B	base	6.1	5.9	10.2	17.1	.2	.4	.6
		+extra	5.5	5.3	7.9	13.6	.2	.6	.6
StableLM [60]	7B	base	2.4	2.7	7.5	15.8	.2	.6	.6
		+extra	2.4	2.6	6.2	11.9	.2	.6	.6

- 所有模型在HumanEval+上的pass@k指标显著下降

- 最高下降幅度：pass@1*下降23.1%，pass@1下降19.3%，pass@10下降24.9%，pass@100下降28.9%
- 即使是最先进的模型如GPT-4和ChatGPT也分别下降了13.1%和12.6%

b. 更严格的评估改变了模型排名

- WizardCoder-CodeLlama和Phind-CodeLlama在原始HumanEval上不如ChatGPT
- 在HumanEval+上，这两个开源模型实际上超过了ChatGPT

c. 测试套件缩减仍然保持有效性

> 表 4：HumanEvaL $^{+}$ 的简化测试套件。我们首先展示仅针对每个考虑的指标分别进行集合覆盖时的 pass@ $1^{\star}$ 和平均测试数量（包括基础 HumanEval 测试）（§2.2）。Full 列则显示通过结合所有三个指标得到的最终简化结果。作为参考，原始 HUMANEvAL 和 HUMANEvAL+ 的平均测试数量分别为 9.6 和 774.8（表 2）。

Table 4: Reduced test-suite for HUMANEVAL+. We first show the pass@$1^{\star}$ and average #tests (including base HUMANEVAL tests) by only doing set covering over each considered metric separately (§2.2). The **Full** column then shows the final reduction result by combining all of the three. For reference, the average #tests of original HUMANEVAL and HUMANEVAL+ are 9.6 and 774.8 respectively (Table 2).

	Size	Coverage		Killed mutants		Killed samples		**Full**		Ref. pass@$1^{\star}$	
		pass@$1^{\star}$	#tests	pass@$1^{\star}$	#tests	pass@$1^{\star}$	#tests	pass@$1^{\star}$	#tests	base	+extra
GPT-4	N/A	86.0	11.3	82.9	11.4	78.7	13.8	**78.0**	16.1	88.4	76.2
ChatGPT	N/A	71.3	11.3	69.5	11.4	**65.2**	13.7	**65.2**	16.0	73.2	63.4
StarCoder	15B	32.9	11.3	32.9	11.4	**29.3**	13.6	**29.3**	15.9	34.1	29.3
CodeGen	2B	23.2	11.3	23.8	11.4	**21.3**	13.2	**21.3**	15.4	24.4	20.7
	6B	28.7	11.3	29.3	11.4	**25.6**	13.2	**25.6**	15.4	29.3	25.6
	16B	31.7	11.3	31.1	11.4	**27.4**	13.2	**27.4**	15.4	32.9	26.8
CodeGen2	1B	10.4	11.3	11.0	11.4	**9.1**	13.8	**9.1**	16.0	11.0	9.1
	3B	15.9	11.3	15.9	11.4	**12.8**	13.8	**12.8**	16.0	15.9	12.8
	7B	18.3	11.3	18.3	11.4	**16.5**	13.8	**16.5**	16.0	18.3	16.5
	16B	19.5	11.3	18.9	11.4	**16.5**	13.8	**16.5**	16.0	19.5	16.5
VICUNA	7B	11.6	11.3	11.6	11.4	**11.0**	13.8	**11.0**	16.1	11.6	10.4
	13B	16.5	11.3	16.5	11.4	**15.2**	13.8	**15.2**	16.1	17.1	15.2
SantaCoder	1.1B	14.6	11.3	14.6	11.4	**12.8**	13.8	**12.8**	16.1	14.6	12.8
INCODER	1.3B	12.2	11.3	12.2	11.4	**10.4**	13.6	**10.4**	16.0	12.2	10.4
	6.7B	14.6	11.3	14.6	11.4	**12.2**	13.6	**12.2**	16.0	15.9	12.2
GPT-J	6B	12.2	11.3	12.2	11.4	**10.4**	13.8	**10.4**	16.0	12.2	10.4
GPT-NEO	2.7B	7.3	11.3	7.3	11.4	**6.7**	13.8	**6.7**	16.1	7.9	6.7
PolyCoder	2.7B	6.1	11.3	6.1	11.4	**5.5**	13.8	**5.5**	16.1	6.1	5.5
StableLM	7B	**2.4**	11.3	**2.4**	11.4	**2.4**	13.8	**2.4**	16.1	2.4	2.4

- HumanEval+-Mini使用47倍更少的测试用例，仍能达到与HumanEval+相似的评估效果
- LLM样本杀死策略最为有效，但也消耗更多测试用例

d.发现HumanEval中的标准答案错误

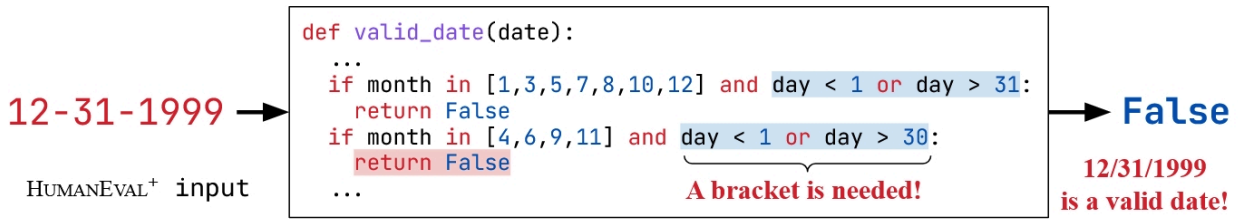

Figure 4: Exemplary incorrect-logic ground-truth solution in HUMANEVAL (#124)

- 检测到18处缺陷(11%的问题)：5个未处理边界情况，10个逻辑错误，3个性能问题
- 图4展示了一个日期验证函数中由于运算符优先级导致的逻辑错误

e.模型在不同难度问题上的表现差异化很大

图 3：通过率分布。X 轴表示所有 164 个问题的条形图，按 HumanEval 通过率排序。Y 轴显示所有 LLM 生成样本的通过率的对数尺度平均值。

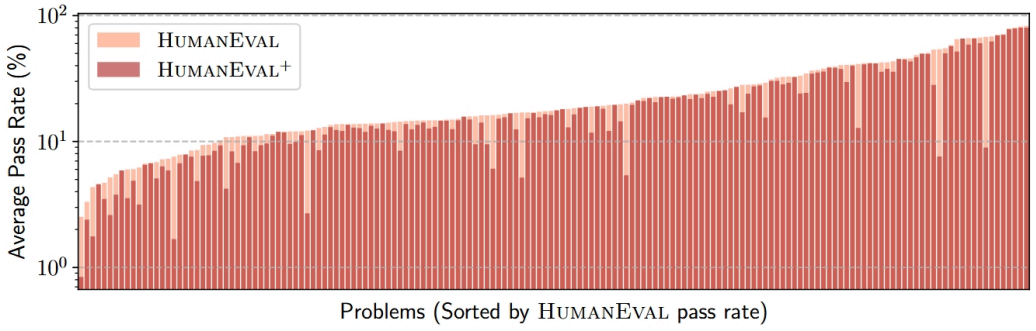

Figure 3: Pass rate distribution. X-axis spans bars for all 164 problems, sorted by the HUMANEVAL pass rate. Y-axis shows the *log*-scale pass rates averaged by all LLM-generated samples.

- 简单问题(如"加两个数字")容易被解决
- 涉及多条件、完整性、推理能力和效率要求的问题对LLM最具挑战性

5. 研究意义与未来方向

研究意义：

- 表明先前流行的代码合成评估结果未能准确反映LLM在代码合成中的真实表现
- 开辟了通过自动化测试改进编程基准测试的新方向
- EvalPlus已被广泛采用，5个月内PyPI包安装超过6000次

未来方向：

- 将EvalPlus应用于更多代码基准测试(如MBPP)
- 集成更正式的验证方法(如Dafny)或验证技术
- 将核心测试生成技术用于AI配对编程(如Copilot)，提醒开发者潜在代码缺陷