CODEGEN：A Conversational Paradigm for Program Synthesis

程序综合的会话范式

Paper:

- A Conversational Paradigm for Program Synthesis

摘要

程序合成力求生成一个计算机程序作为给定问题规范的解决方案。我们提出了一种通过大型语言模型的会话程序合成方法，该方法解决了在先前方法中面临的巨大程序空间和用户意图规范中搜索的挑战。我们的新方法将编写规范和程序的过程转换为用户和系统之间的多轮对话。它将程序合成视为序列预测问题，其中规范用自然语言表达，期望的程序被有条件地采样。我们在自然语言和编程语言数据上训练了一系列大型语言模型，称为 CODEGEN。随着数据监督薄弱以及数据大小和模型大小的扩大，会话能力从简单的自回归语言建模中出现。为了研究会话程序合成的模型行为，我们开发了一个多轮编程基准（MTPB），其中解决每个问题都需要通过用户和模型之间的多轮对话进行多步合成。我们的研究结果显示了会话能力的出现以及所提出的会话程序合成范式的有效性。此外，我们的模型 CODEGEN（在 TPU-v4 上训练了多达 16B 个参数）在 HumanEval 基准测试中优于 OpenAI 的 Codex。我们将包括检查点在内的训练库JAXFORMER 作为开放源码贡献给大家：https://github.com/salesforce/CodeGen.

1、介绍

自计算机时代开始以来，创建程序通常需要人工手动输入代码。程序合成的目标是使编码过程自动化，并生成满足用户指定意图的计算机程序。有人将其称为计算机科学的圣杯（Manna 和 Waldinger，1971；Gulwani 等人，2017）。成功的程序合成不仅可以提高有经验的程序员的工作效率，还可以让更广泛的受众能够接触到编程。

在努力实现程序合成时会出现两个关键挑战：（1）搜索空间的巨大，以及（2）正确指定用户意图的难度。为了解决这些挑战，我们提出并训练了 CODEGEN，一种用于程序合成的交互式代码生成模型。此外，我们开发了一个新的多轮编程基准来研究 CODEGEN 的编程综合能力。

1.1、用大型语言模型在程序搜索空间中导航

为了保持搜索空间的表现力，人们需要一个大的搜索空间，这给高效搜索带来了挑战。以前的工作（Joshi等人，2002；Panchekha等人，2015；Cheung等人，2013）利用特定领域的语言来限制搜索空间；然而，这限制了合成程序的适用性。

我们选择使用通用编程语言（如Python），这样我们的方法所合成的程序就能广泛适用。为了浏览通用编程语言引起的巨大程序空间，我们将程序合成视为一个序列预测问题，并利用具有大规模自我监督预训练的转化器（Vaswani等人，2017）。变换器通过注意力机制（Bahdanau等人，2014）捕捉序列元素之间的依赖性，并且具有高度的可扩展性，允许它从大型数据集中学习。这种方法已经成功应用于自然语言处理（Devlin等人，2018；Liu等人，2019；Lewis等人，2019；Raffel等人，2019；Brown等人，2020）、计算机视觉（Dosovitskiy等人，2020）、音频和语音处理（Oord等人，2018；Baevski等人，2020）以及生物信息学（Rives等人，2021；Jumper等人，2021）。之前的一些工作，如CuBERT（Kanade等人，2020）、CodeBERT（Feng等人，2020）、PyMT5（Clement等人，2020）和CodeT5（Wang等人，2021），已经将转化器应用于代码理解，但这些大多侧重于代码检索、分类和程序修复。我们假设，从大量未标记的程序数据中学习到的转化器将很好地捕捉到程序元素之间的统计依赖性，并允许在给定的用户意图规格下进行有效的搜索。

在我们的实验中，我们发现，通过在大规模数据集上学习的大型基于 Transformer 的语言模型和优化的训练方案，NLP 中使用的常用采样方法，如贪婪解码和 top-p 采样（Holtzman et al., 2019) 是有效且高效的搜索方法。这些模型生成通过专家编写的测试用例的有效程序。

1.2、通过多轮对话的用户意图规范

用于程序合成的用户意图可以是一个逻辑规范（它规定了程序的输入和输出之间的逻辑关系），输入输出的例子，或用自然语言指定的请求。一方面，一个完整的正式或非正式的规范需要用户付出太多的努力，这可能与编写程序本身的任务相当，从而偏离了程序合成的目的。另一方面，仅仅基于输入-输出例子的规范可能会对问题的规范性不足，导致不理想的解决方案。

我们选择使用自然语言，程序员在其中进行大部分的思考，利用大型语言模型的语言理解能力（Brown等人，2020）。我们提出了一种交互式的程序合成方法，用户通过逐步提供自然语言的规范与合成系统进行交互，同时从系统中获得合成子程序的回应，这样，用户与系统一起在多轮对话中交互地完成程序。

两种考虑促使我们引入这种方法。首先，将一个潜在的长而复杂的规范分解成多个步骤，将使模型更容易理解，从而导致一个更成功的程序合成系统。在多回合的方法中，模型可以专注于与一个子程序相关的规范，避免艰辛地跟踪子程序之间复杂的依赖关系。在我们的实验中，我们确实观察到：(1)以语言模型的迷惑性为代表（详见第3.3节），当规范分多步而不是一步指定时，模型对规范的理解会得到改善；(2)多轮方法导致更成功的合成程序。

提出方法的第二个动机是，自然语言注释和用编程语言编写的子程序的交错模式出现在常规代码数据中。程序员写的评论往往是为了解释评论后面的代码的功能而存在的。鉴于语言建模是训练目标（即预测下一个标记的历史条件），该数据模式为模型提供了监督信号，以发展对话能力，特别是预测给定的自然语言描述的代码。然而，这种监督是有噪音的或弱的。这种交错的描述-代码块只出现在数据的一部分，一些评论没有提供完整或正确的代码描述，一些评论可能出现在相应的代码之后而不是之前。然而，数据大小和模型大小的缩放可能会克服监督的噪音和不完全性，因此，在大量数据和大模型的情况下，对话能力可能会在弱监督下出现。通过我们目前工作中开发的多轮编程基准，我们证实了这个假设，表明一个简单的自回归语言模型可以通过与用户的多轮对话来合成程序。类似的方法在面向任务的对话领域也有研究（Budzianowski和Vuli´c，2019；Hosseini-Asl等人，2020)

如前所述，为了衡量模型在对话式程序合成方面的能力，我们开发了一个多轮编程基准。为了解决该基准中的问题，一个模型需要通过与用户的多轮对话，在多个步骤中合成一个程序，用户在每轮对话中用自然语言指定意图。该基准的性能是通过专家编写的测试案例的通过率来衡量的。据我们所知，这是第一个多轮程序合成基准，它允许对对话式程序合成进行定量分析。随着有利于问题解决的大型语言模型中对话能力的出现（Austin等人，2021），我们相信这个基准将促进对话式程序合成的未来研究。

1.3、本文的贡献

最近和同时进行的几项工作也在探索采用大型语言模型进行程序合成。Chen等人（2021）继续在代码数据上训练GPT-3（Brown等人，2020），产生了Codex，并分析了其Python代码编写能力。Austin等人（2021）在文本和代码数据上训练大型因果语言模型，用于代码生成。Li等人（2022）招募了一个编码器-解码器模型，在GitHub的通用代码数据上训练，并在竞争性编程数据上进行微调，以解决竞争性编程问题。

我们的工作与上述作品一样，采用语言模型进行程序合成的基本思路，并以单次用户意图规范研究程序合成能力。此外，我们在以下三个方面做出了贡献：

1. 我们研究了在缩放定律下从简单的自回归语言建模中产生的会话能力，并利用它来引入会话程序合成方法；
2. 我们使用新开发的多轮编程基准来研究它的属性；
3. 我们开源了一个自定义库 JAXFORMER，其中包括我们的训练代码以及模型检查点。

到目前为止，似乎还没有与 Codex 竞争的基于大型语言模型的程序合成模型开源，这阻碍了该领域的进展，因为训练这些大型模型所需的昂贵计算资源只有数量有限的工业研究机构。我们相信开源代码和模型将使广大研究人员能够研究这些模型的特性并将其应用于解决实际问题，这将极大地促进程序合成的研究进展。

2、Model Training

为了评估缩放定律下会话编程能力的出现，我们采用标准的基于Transformer的自回归语言模型，改变（1）模型参数的数量（350M、2.7B、6.1B、16.1B）和（2）训练语料库中的编程语言标记。为了扩展模型，开发了一个自定义库 JAX_{FORMER}，用于在 TPU-v4 硬件上训练大型语言模型，并将作为开源发布，包括训练后的模型权重。

Dataset	Language	Raw Size	Final Size	Final Tokens
THEPILE	Natural Language	825.18 GiB	1159.04 GiB	354.7B
	Code	95.16 GiB	95.16 GiB	31.6B
BIGQUERY	C	1772.1 GiB	48.9 GiB	19.7B
	C++	205.5 GiB	69.9 GiB	25.5B
	Go	256.4 GiB	21.4 GiB	9.6B
	Java	335.1 GiB	120.3 GiB	35.4B
	JavaScript	1282.3 GiB	24.7 GiB	9.7B
	Python	196.8 GiB	55.9 GiB	19.3B
BIGPYTHON	Python	5558.1 GiB	217.3 GiB	71.7B

Table 1. 沿预处理步骤训练语料库的近似统计数据。

2.1、Datasets

CODEGEN 模型系列在三个数据集上按顺序训练：THEPILE、BIGQUERY 和 BIGPYTHON。

自然语言数据集THEPILE是Gao等人（2020）为语言建模而收集的825.18GiB英语文本语料。该数据集由22个不同的高质量子集构成，其中一个是从GitHub存储库中收集的编程语言数据，其中有超过100颗星，占数据集的7.6%。由于THEPILE的大部分是英文文本，我们把在该数据集上训练的模型称为自然语言CODEGEN模型（CODEGEN-NL）。

多语言数据集BIGQUERY是谷歌公开的BigQuery数据集的一个子集，它包括多种编程语言的代码，如C、Python、Ruby。对于多语言训练，我们选择了以下6种编程语言。C, C++, Go, Java, JavaScript, 和Python。因此，我们把在BIGQUERY上训练的模型称为多语言CODEGEN模型（CODEGEN-MULTI）。

单一语言的数据集BIGPYTHON包含了大量的编程语言Python的数据。我们在2021年10月汇编了来自GitHub的公开的、非个人的信息，包括许可的Python代码。因此，我们将在BIGPYTHON上训练的模型称为单语言CODEGEN模型（CODEGEN-MONO）。

对于每个数据集，预处理都有以下步骤：(1)过滤，(2)重复数据删除，(3)标记化，(4)洗牌，和(5)连接。关于THEPILE的细节，我们参考了Gao等人(2020)。对于BIGQUERY和BIGPYTHON，在(1)中，文件按文件扩展名过滤，删除平均行长<100个字符，最大行长1000个，以及>90%的字符是十进制或十六进制数字的文件。对于(2)，基于SHA-256哈希值的完全重复的文件被删除，这相当于原始数据中由于分叉和储存库的副本而产生的相当大的一部分。对于(3)，GPT-2的BPE词汇被特殊的标记所扩展，这些标记代表了标签和白空间的重复标记。在BIGQUERY的多语言设置中，前缀是用来表示编程语言的名称。对于(4)，每一年的数据都是随机洗牌的。对于(5)，序列被连接起来，以填补上下文长度的2,048个标记，用一个特殊的标记作为分隔符。表1总结了训练语料库的统计数据。

CODEGEN-NL 模型在 THEPILE 上随机初始化和训练。 CODEGEN-MULTI 模型从 CODEGEN-NL 初始化，然后在 BIGQUERY 上进行训练。 CODEGEN-MONO 模型从 CODEGEN-MULTI 初始化，然后在 BIGPYTHON 上进行训练。

2.2、Model

我们的模型是自回归变换器，以常规的下一个标记预测语言模型为学习目标。CODEGEN模型家族被训练成各种规模，有350M、27B、61B和161B的参数。前三种配置可以与在文本语料库上训练的开源大型语言模型GPT-NEO(350M，2.7B)(Black等人，2021)和GPT-J(6B)(Wang和Komatsuzaki，2021)直接比较。模型规格见表2。

Model	Dataset	Hyper-parameter	350M	2.7B	6.1B	16.1B
CODEGEN		Number of layers	20	32	33	34
		Number of heads	16	32	16	24
		Dimensions per head	64	80	256	256
		Context length	2,048	2,048	2,048	2,048
		Batch size	500k	1M	2M	2M
		Weight decay	0.1	0.1	0.1	0.1
CODEGEN-NL	THEPILE	Learning rate	3.0e−4	1.6e−4	1.2e−4	0.9e−4
		Warm-up steps	3k	3k	3k	3k
		Warm-up / Total steps	350k	350k	350k	350k
CODEGEN-MULTI	BIGQUERY	Learning rate	1.8e−4	0.8e−4	0.4e−4	0.5e−4
		Warm-up steps	3k	3k	3k	3k
		Total steps	150k	150k	150k	150k
CODEGEN-MONO	BIGPYTHON	Learning rate	1.8e−4	0.8e−4	0.4e−4	0.5e−4
		Warm-up steps	3k	3k	3k	3k
		Total steps	150k	150k	150k	150k

Table 2. 用于 CODEGEN 模型系列的模型规范和优化的超参数。

该架构遵循一个标准的变压器解码器，具有从左到右的因果掩码。对于位置编码，我们采用旋转位置嵌入(Su等人，2021)。对于前向传递，我们按照Wang和Komatsuzaki(2021)的做法，并行执行自我注意和前馈电路，以提高通信开销，即 $x_{t+1} = x_t + \text{mlp}(\ln(x_t + \text{attn}(\ln(x_t))))$ 被改变为 $x_{t+1} = x_t + \text{attn}(\ln(x_t)) + \text{mlp}(\ln(x_t))$，其中自我注意、attn()和前馈(mlp())的计算与层规范(ln())是同步的。架构和超参数的选择是为TPU-v4的硬件布局专门优化的。

2.3、Training

大型语言模型的扩展需要数据和模型的并行性。谷歌的TPU-v4硬件具有高速环形网状互连，自然可以实现高效的并行性。为了有效利用硬件，模型的训练是在JAX中实现的(Bradbury等人，2018)。对于JAX中的并行评估，采用了 $pjit()$ 操作符。该操作符实现了一种名为单程序多数据(SPMD)代码的范式，它指的是一种并行技术，即在不同的设备上对不同的输入数据并行运行相同的计算。具体来说，$pjit()$ 是JAX中XLA SPMD分区器暴露的API，它允许在计算的逻辑网中以等效语义并行评估给定的函数。

我们的库JAXFORMER招募了一个指定的协调器节点，通过一个自定义的TCP/IP协议来协调TPU-VM的集群。对于数据并行，协调者对一个批次进行分区，并将分区分配给各个TPU-VMs。对于模型的并行性，支持两种模型参数的分片方案。(1)TPU-VM内，参数在物理TPU-v4板内的MXU核9之间分片，并按照Shoeybi等人(2019)；Wang和Komatsuzaki(2021)进行跨板复制；(2)TPU-VM间，参数在TPU-v4板上分片，并按照Rajbhandari等人(2020)进行激活复制。

两种分片方案都是基于我们特定的 $pjit()$ 实现的，逻辑网格规范 (r, p, c) 具有参数的 r 个副本，参数的 p 个分区，以及在 n_b 个 TPU 板上的每个板的 c 个逻辑核心，每个 n_c 逻辑核心使得 $d \times p = n_b$ 和 $r \times p \times c = n_b \times n_c$。

对于参数大小小于或等于6B的模型，采用intra-TPU-VM方案，模型和优化器参数的总量适合单个TPU-v4板的组合HBM内存。例如，$n_b = 64$ 和 $n_c = 4$ 的 TPU-v4-512 切片将配置为 $(r, p, c) = (64, 1, 4)$。也就是说，参数在 $r = 64$个板上被复制，其中 $p = 1$ 个板间分区和 $c = 4$个逻辑芯片上的板内并行性。在此配置中，平均梯度通过 with_sharding_constraint() 跨板累积，有效模拟 $xmap()$运算符的行为。

对于超过6B参数大小的模型，采用TPU-VM之间的方案，对于这些模型和优化器的参数必须在TPU-v4板上进行分片。例如，一个 $n_b = 64$ 和 $n_c = 4$ 的TPU-v4-512片将被配置为（r, p, c）=（$1, 64, 4$）。对于较大的切片，如 $n_b = 128$ 的TPU-v4-1024，可以在参数分片中引入冗余，例如，（r, p, c）=（$2, 64, 4$）。在这种配置下，通过with_sharding_constraint()，激活被复制到各个板块。此外，(r, p, c) 允许在逻辑硬件布局从 $c = 8$ 的TPU-v3过渡到 $c = 4$ 的TPU-v4时，通过调整 p 来实现向后兼容，不需要重新分片。

对于优化，表 2 总结了超参数。我们采用 Adam (Kingma and Ba, 2014) 优化器，其 $(\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 1e-08)$ 和全局梯度范数裁剪 (Pascanu et al., 2013) 为 1.0。随着时间的推移，学习率函数遵循 GPT-3 (Brown et al., 2020) 的预热步骤和余弦退火。

3、Single-Turn Evaluation

虽然早期的工作采用基于 n-gram 匹配的指标(Yin 和 Neubig，2017；Miceli Barone 和 Sennrich，2017；Iyer 等人，2018)，但最近的技术使用功能正确性来评估程序合成模型，因为基于 n-gram 匹配指标不能解释功能等效的程序(Hendrycks 等人，2021 年；奥斯汀等人，2021 年)。我们在单轮评估基准和多轮评估基准上根据其功能正确性评估合成程序。这两个基准都是手写的，以避免从训练数据中泄漏数据。我们的模型在本节的单轮评估基准和下一节的多轮基准上进行评估。

由Chen等人(2021)介绍的HumanEval是一个衡量Python模型程序合成能力的基准。它包含164个手写的编程问题。每个问题都提供了一个提示，包括要生成的函数的描述、参数名称、函数名称和断言形式的测试案例示例。模型需要根据提示完成一个函数，使其能够通过所有提供的测试案例。由于用户意图是在一个提示中指定的，并提供给模型一次，我们将HumanEval上的评价视为单次评价，以区别于用户意图被分解为多个步骤的多次评价。继Chen等人(2021)之后，我们使用核抽样(Holtzman等人，2019)与top-p，其中p=0.95来产生程序样本。我们测试了一系列的采样温度，即t 在$\frac{\exp(x/t)}{\sum_k \exp(x_k/t)}$ ，特别是$t \in 0.2, 0.6, 0.8$。当遇到以下标记之一：'\nclass', '\ndef', '\n#', '\nif', 或 '\nprint'时，采样就会停止。我们根据Chen等人(2021)提出的无偏估计器(详见附录A)，对每个问题用200个样本计算pass@k。

3.1、HumanEval Performance Scales as a Function of Model Size and Data Size

我们将我们的模型与 Codex 模型(Chen 等人，2021 年)进行了比较，后者展示了 HumanEval 的最新性能。此外，我们的模型与开源大型语言模型 GPT-NEO(Black 等人，2021 年)和 GPT-J(Wang 和 Komatsuzaki，2021 年)进行了比较。这些是在 THEPILE(Gao 等人，2020)上训练的，因此在训练数据和模型大小方面类似于我们的 CODEGEN-NL 模型。所有模型都使用温度 $t \in 0.2, 0.6, 0.8$ 进行评估，我们计算 pass@k，其中每个模型的 $k \in \{1, 10, 100\}$。为了直接比较陈等人的结果(2021 年)，我们为每个 k 选择产生最佳 pass@k 的温度。表 3 总结了我们的模型和基线的结果。我们的 CODEGEN-NL 模型(350M、2.7B、6.1B)的表现优于或与各自的 GPT-NEO 和 GPT-J 模型相当。

Model	pass@k [%]		
	$k=1$	$k=10$	$k=100$
GPT-NEO 350M	0.85	2.55	5.95
GPT-NEO 2.7B	6.41	11.27	21.37
GPT-J 6B	11.62	15.74	27.74
CODEX 300M	13.17	20.37	36.27
CODEX 2.5B	21.36	35.42	59.50
CODEX 12B	28.81	46.81	72.31
CODEGEN-NL 350M	2.12	4.10	7.38
CODEGEN-NL 2.7B	6.70	14.15	22.84
CODEGEN-NL 6.1B	10.43	18.36	29.85
CODEGEN-MULTI 350M	6.67	10.61	16.84
CODEGEN-MULTI 2.7B	14.51	24.67	38.56
CODEGEN-MULTI 6.1B	18.16	28.71	44.85
CODEGEN-MONO 350M	12.76	23.11	35.19
CODEGEN-MONO 2.7B	23.70	36.64	57.01
CODEGEN-MONO 6.1B	26.13	42.29	65.82
CODEGEN-MONO 16.1B	**29.28**	**49.86**	**75.00**

Table 3: 对HumanEval基准的评估结果。每个模型的每个pass@k（其中k∈{1,10,100}）用三个采样温度（t∈{0.2,0.6,0.8}）计算，并显示三个采样温度中最高的一个，这与（Chen等人，2021）的评估程序一致。对于每个问题，在给定的温度下生成n=200个样本。详细说明见第3.1节。CODEGEN-MONO 16.1B的训练仍在进行，性能不断提高。

在多语言编程语言数据（BIGQUERY）上进一步训练CODEGEN-NL，可以得到CODEGEN-MULTI。多语言CODEGEN模型在很大程度上超过了在THEPILE（GPT-NEO、GPT-J、CODEGEN-NL）上训练的模型。然后，我们在Python数据集（BIGPYTHON）上对CODEGEN-MULTI进行了微调，形成了CODEGEN-MONO。程序合成能力得到了进一步的大幅提高。因此，Python程序的合成能力随着Python训练数据量的增加而增强。对于几乎所有的模型，正如预期的那样，增加模型的规模可以提高整体性能。

与目前最先进的模型Codex相比，我们的Python-monolingual CODEGEN模型具有竞争性或改进的性能。当k=100时，CODEGEN-MONO 2.7B的性能低于CODEX 2.5B，但当k∈{1,10}时则优于它。虽然它只有CODEX 12B的一半，但我们的CODEGEN-MONO 6.1B展示了接近表现最好的CODEX 12B的pass@k分数。我们的CODEGEN-MONO 16.1B（其训练正在进行中）开始超过它的表现。

3.2、Temperature Selection Based on the Number of Chances Permitted

为了证明采样温度对程序合成的影响，我们在图1中显示了所有三种温度下的通过率。在图中的每个面板上，对于每个采样温度 $t \in \{0.2, 0.6, 0.8\}$ ，我们将pass@k绘制成 $k \in \{1, 10, 100\}$ 的函数。在所有9个CODEGEN模型中，我们观察到一个普遍的模式，即最佳性能温度随着k的增加而增加。考虑到允许的采样数量（由计算和时间限制和/或应用场景），我们可以相应地设置采样温度以优化合成系统的性能。当一个模型只允许一个或几个样本时（例如，用一个样本通过所有的测试案例），更有可能通过的候选者可以通过低温度，利用学到的模型分布，产生模型的高置信度样本。当一个模型被允许有更多的机会（例如，通过@10，通过@100），用更高的采样温度倾斜所学到的模型分布，允许它探索不同的样本，从而更可能合成一个正确的程序。

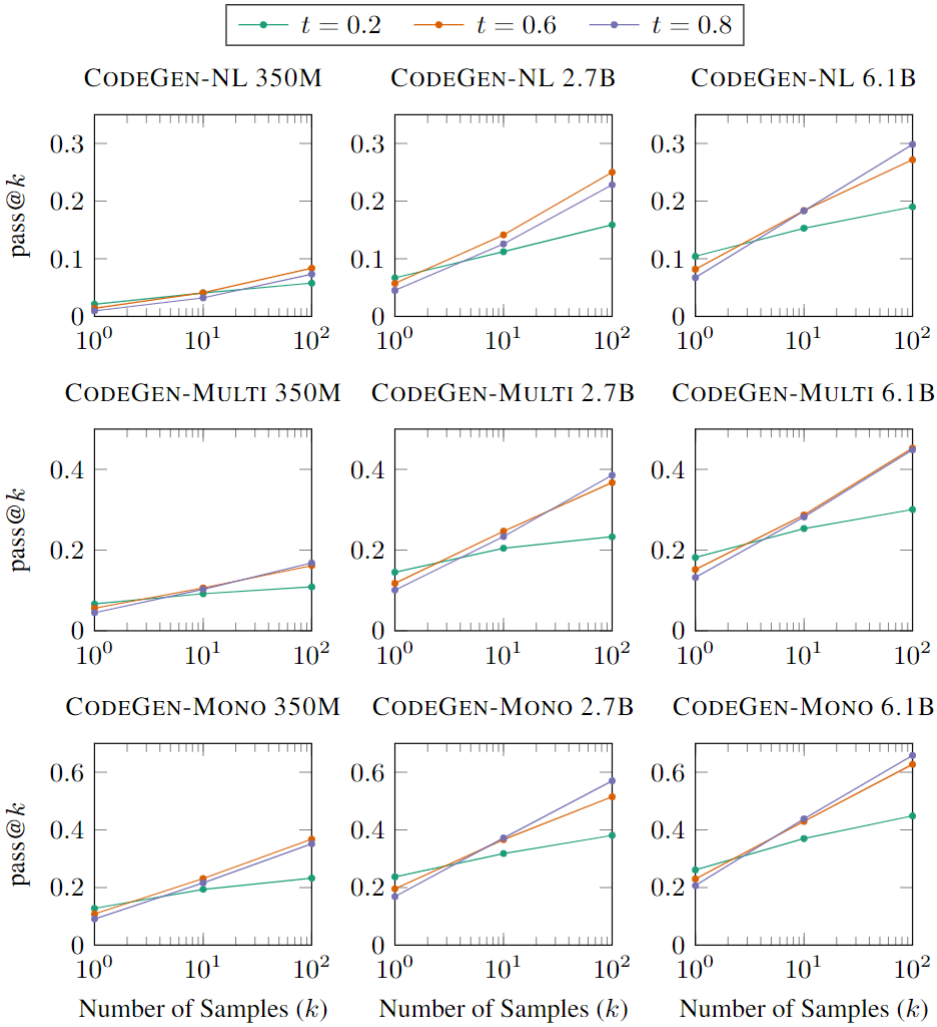

Figure 1: 在 HumanEval 基准测试中，k ∈{1,10,100} 的 pass@k 的温度效应。有关详细信息，请参阅第 3.2 节。

3.3、Better User Intent Understanding Yields Better Synthesized Programs

一个程序合成系统的成功在很大程度上取决于它如何理解用户的意图。当系统以语言模型为基础时，问题提示的困惑度可以代表系统对用户意图规格的理解。一个模型下的意图规范的低困惑度表明，这个意图规范与模型从训练数据中学习到的知识是兼容的。我们研究了以较低的提示迷惑度为代表的更好的提示理解是否会导致更好的综合程序。

Model	Pass	Non-Pass
CODEGEN-MONO 2.7B	3.66	4.37
CODEGEN-MONO 6.1B	3.535	3.88

Table 4: 提示通过问题和未通过问题的困惑。

我们将所有问题划分为通过与不通过问题。一个通过的问题是200个样本中至少有一个样本12通过了所有的测试案例，而一个不通过的问题是200个样本中没有一个通过所有的测试案例。我们根据CODEGEN-MONO 2.7B和CODEGEN-MONO 6.1B的样本，计算出通过问题和未通过问题的平均困惑度，其温度为0.2。结果显示在表4中。通过问题的提示比非通过问题的提示具有更低的困惑性。这一发现意味着，当用户的意图说明被模型更好地理解时，程序合成更容易成功。正

如第1.2节所讨论的，一些训练数据包含自然语言和编程语言的交错序列，其中自然语言序列是由程序员写的评论，用来描述以下代码的功能。因此我们推测，类似于这种模式的用户意图规范会被模型更好地理解，从而导致更好的程序合成。

4、Multi-Turn Evaluation

在本节中，我们提出并研究了一个对话式的程序合成范式，在这个范式中，合成程序的过程被视为用户和系统之间的多轮对话。为了研究这样一个范式，我们首先开发了一个多轮编程基准（MTPB）。MTPB包括50个由专家编写的问题。解决每个问题的描述被分解成多个步骤，每个步骤都包括一个自然语言的描述（提示）。为了解决一个问题，模型需要(1)按照当前步骤的描述，(2)考虑先前步骤的描述和合成的子程序（例如，先前步骤中定义的函数和/或变量的正确反向引用），合成语法和功能上正确的子程序。图2中显示了一个说明性的例子。

我们猜想，将用户的意图分解成多个步骤，会使模型更容易理解规范，从而合成出完成用户目标的程序。这一假设在MTPB中得到了证实，具有多圈因子化规范的模型比单圈规范的模型表现出更高的通过率。此外，我们通过MTPB系统地研究了基于语言模型的程序合成系统在多轮对话程序合成中的行为。

4.1、Benchmark Construction

我们通过策划一组50个需要不同编程知识的问题来构建MTPB，包括数学问题、数组操作、字符串操作、算法、数据科学以及需要其他知识的问题。我们在附录B中列出了这些问题的描述。对于每个问题，我们都包括多轮提示，作为模型的逐步指导，确保这些(1)至少形成三个轮子，(2)不在一个轮子中归结问题的解决。13 为了评估合成的程序的功能正确性，我们为每个问题包括5个测试案例。在创建测试用例时，我们包括那些只有在正确的解决方案下才会通过的非琐碎的测试用例。

与HumanEval不同的是，模型要完成一个部分定义的功能，MTPB问题只提供提示，因此模型必须从头生成解决方案。虽然自由形式的生成可能允许更多潜在的解决方案，但由于缺乏提供测试用例输入的入口，在不同的测试用例上测试生成的代码是一个挑战。为了克服这一挑战，我们将测试用例输入嵌入到提示中。具体来说，提示语是用Python的格式化字符串编写的，当某个测试用例被应用到问题上时，输入值会被替换成变量名。例如，一个提示，"定义一个名为's'的字符串，其值为{var}。"，加上测试用例的输入var = 'Hello'，将被格式化为 "定义一个名为's'的字符串，其值为'Hello'。

4.2、Generation with an Expanding History

在每个回合，我们的模型考虑到了作为Python注释的提示（即" "，它可以通过合成的子程序获得。为了整合这两者，我们在每个转弯处用提示和生成的样本的交错串联来喂养模型。串联的历史长度在转弯时线性增加，如果超过了模型的最大输入长度，则从头开始截断。我们在图2中说明了这个过程。

由于将生成的输出反馈到下一回合的输入，早期回合中的错误可能会对后面回合的生成产生负面影响。然而，我们在实践中并没有系统地观察到这种现象。我们为未来的工作留下更有效的历史条件:

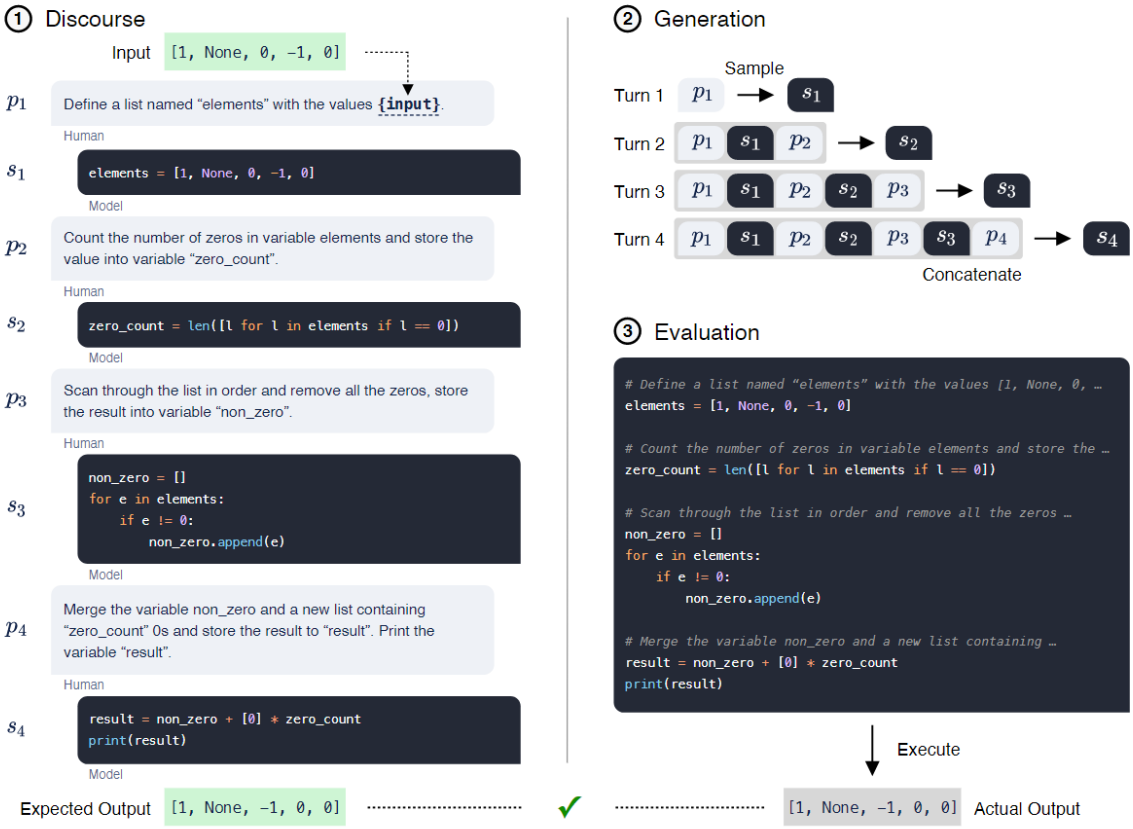

Figure 2: 多轮编程基准的一个说明性例子。 (1) 每个回合由一个提示和一个生成的响应组成。前者可以是一个模板，接受各种输入测试案例（即{input}）。对于每个问题，我们提供5个测试用例，每个测试用例包括(a)一个输入，在提示中取代{input}，然后再送入模型；(b)一个答案，生成程序的输出与之比较，以评估其功能正确性（见图中（3）的执行部分）。在显示的例子中，提供的输入是[1, None, 0, -1, 0]，它取代了p1中的{input}，而黄金答案是[1, None, -1, 0, 0]。 (2) 我们的模型的条件是将交错的过去的提示和生成的回答按时间顺序串联起来。(3) 每一回合的生成的回答都被串联起来并执行，其中输出与答案进行比较。

4.3、Execution Environment

MTPB 的目标是评估和量化模型在多轮对话中解决问题的能力程度。因此，基准测试需要执行和验证生成的解决方案。

Data	Model	Pass Rate↑ [%]	
		2.7B	6.1B
THEPILE	GPT-NEO & GPT-J	7.06	17.84
THEPILE	CODEGEN-NL	11.18	16.34
BIGQUERY	CODEGEN-MULTI	23.55	27.18
BIGPYTHON	CODEGEN-MONO	47.14	51.91

Table 5: 多轮编程基准的评估结果。对话式程序合成性能随着模型大小（列）和代码数据大小（行）的变化而变化。

Prompt	PPL↓		Pass Rate↑ [%]	
	2.7B	6.1B	2.7B	6.1B
Single-Turn	10.11	9.44	26.15	26.91
Multi-Turn	8.19	7.67	47.14	51.19

Table 6: 多轮用户规范和串联的单轮规范在困惑度 (PPL) 和程序综合性能（通过率衡量）方面的比较。使用 CODEGEN-MONO 2.7B 和 CODEGEN-MONO 6.1B 计算困惑度和通过率。

为了执行，成对的提示和生成的完成度的历史被串联成一个独立的程序（例子见图2中的3）。然后按照单圈HumanEval基准（Chen等人，2021）在一个孤立的Python环境中执行该程序。然而，HumanEval中的问题是以这样一种方式构建的，即完成了一个已知的函数签名，因此在一组功能单元测试下调用生成的代码是很简单的。在我们的多轮案例中，没有这样的入口点（或返回值）被保证生成。为了规避返回签名（或值）缺失的问题，MTPB中的多轮问题的最后一个提示总是被指定为向终端打印出结果状态。然后，基准执行环境重载了Python print(args)函数，并将args存储在栈上。如果一个问题的最后一个提示的采样代码不包括print()语句，这是Python或具体的Jupyter笔记本中在终端上打印的有效约定，那么生成的代码的AST将被突变以注入print()的调用。最后，针对问题的预定义黄金输出，对args进行类型放松的等价检查（例如，列表和图元之间的隐式转换），以确定测试的失败或成功。

4.4、Conversational Programming Capacity Scales as a Function of Model Size and Data Size

在这个分析中，我们研究了两个因素，模型大小(就模型参数的数量而言)和数据大小(就评估领域的数据大小和当前案例中Python预训练数据的大小而言)，如何影响多轮对话范式中的程序合成能力。我们在以下数据集上以2.7B和61B两种规模训练模型。THEPILE、BIGQUERY、BIGPYTHON，它们拥有越来越多的Python数据(详见2.1节)。GPT-NEO、GPT-J、CODEGEN-NL 2.7B和CODEGEN-NL 6.1B是在THEPILE上训练的。CODEGEN-MULTI 2.7B和CODEGEN-MULTI 6.1B分别用CODEGEN-NL 2.7B和CODEGEN-NL 6.1B初始化，然后对BIGQUERY进行训练。CODEGEN-MONO 2.7B和CODEGEN-MONO 6.1B分别用CODEGEN-MULTI 2.7B和CODEGEN-MULTI 6.1B初始化，然后在BIGPYTHON上训练。在MTPB中，每个问题有5个测试案例，我们用每个模型对每个测试案例抽样40个样本，在此基础上计算出每个问题的通过率。我们的CODEGEN模型和基线的MTPB评估结果(平均通过率)见表5。很明显，MTPB的性能随着模型大小和数据大小的增加而增强。这表明对话式程序合成的能力是随着模型大小和数据大小的变化而变化的。这些模型只是用自回归语言建模的目标进行训练。当模型和数据的规模扩大时，对话能力就会出现。

```
# Import libraries.

import numpy as np

# Initialize the variable named lst1 with a
    list ['abc', 'ab10c', 'a10bc', 'bcd'].

lst1 = ['abc', 'ab10c', 'a10bc', 'bcd']

# Create a function called num_in_str() to
    check whether a string contains a number
    .

def num_in_str(s):
    for i in s:
        if i.isdigit():
            return True
        else:
            return False

# Call the function num_in_str() to find
    strings in lst1 that have numbers and
    assign them to a list named lst2

lst2 = []
for i in lst1:
    if num_in_str(i):
        lst2.append(i)

# Print out lst2

print(lst2)
```

```
# Import libraries.

import numpy as np

# Initialize the variable named lst1 with a
    list ['abc', 'ab10c', 'a10bc', 'bcd'].

lst1 = ['abc', 'ab10c', 'a10bc', 'bcd']

# Create a function called num_in_str() to
    check whether a string contains a number
    .

def num_in_str(string):
    """
    This function checks whether a string
    contains a number.
    """
    if any(char.isdigit() for char in string
    ):
        return True
    else:
        return False

# Call the function num_in_str() to find
    strings in lst1 that have numbers and
    assign them to a list named lst2

lst2 = [string for string in lst1 if
    num_in_str(string)]

# Print out lst2

print(lst2)
```

CODEGEN-MONO 2.7B　　CODEGEN-MONO 6.1B

Figure 3: 两个CODEGEN模型在 "检测数字 "问题上的比较。CODEGEN- MONO 2.7B未能写出检测字符串任意位置的数字的程序，而右边的CODEGEN- MONO 6.1B正确地使用any()来实现前面提示中的规范。

4.5、Better User Specification Understanding with Multi-turn Factorization

我们假设，多轮分解使模型更好地理解用户的意图规范，这反过来可能导致更高的程序合成能力。在下文中，我们将测试这一假设。我们比较了模型对多轮规格的理解和对单轮输入到模型中的串联规格的理解。正如第3.3节所讨论的，我们采用提示性的困惑作为用户意图理解的代理。因此，我们比较了CODEGEN-MONO 2.7B和CODEGEN-MONO 6.1B下的多圈提示的困惑性和串联的单圈提示的困惑性。

两种模型在MTPB中所有问题的平均困惑度（计算细节见附录C）显示在表6的左侧面板。对于这两个模型，单圈规格的平均困惑度比多圈规格的高。这意味着多圈用户规格可以被模型更好地理解。我们还注意到，大模型下的多转和单转意图规范的平均困惑度（6.1B）略低于小模型下的平均困惑度（2.7B），说明大模型比小模型更能理解用户意图。

此外，我们还将多轮提示与串联单轮提示的程序合成性能进行了比较。 CODEGEN-MONO 2.7B 和 CODEGEN-MONO 6.1B 的结果显示在表 6 的右侧面板中。多圈规格的性能几乎是单圈规格的两倍。连同上面的困惑分析，似乎将用户规范分解为多个步骤并利用大型语言模型出现的会话能力使他们能够更容易地消化规范并更成功地综合程序。

4.6、Comparison by Category

在下面的分析中，我们比较了CODEGEN-MONO 2.7B和CODEGEN-MONO 6.1B在图4中提出的每一类问题的性能，以衡量在某些方面的问题解决能力。每个问题都被专门分配到以下类别之一：数学、数据科学、阵列、字符串、算法和其他。将一个问题分配到一个类别是基于完成该任务所需的知识。我们参考附录B，以获得一个完整的问题列表和所分配的类别。

较大模型的改进在数组类别上最为显着，同时在除字符串类别之外的其他类别上也表现出增强。我们还观察到两种模型在算法问题上的表现最差。与其他类型的问题相比，这些问题（例如，二和问题、嘶嘶声问题）通常很难并且需要更深入的推理。

4.7、Qualitation Examples

为了进一步了解模型行为在模型大小上的差异，我们检查了CODEGEN-MONO 2.7B与CODEGEN-MONO 6.1B相比的失败案例。虽然两个模型都能在每次给定的提示下生成合理的代码，但CODEGEN-MONO 2.7B表现出更高的误解提示的倾向，从而导致错误的实现。我们在图3中展示了一个代表性的例子。在这里，CODEGEN-MONO 2.7B生成的num_in_str函数，如果第一个字符不是数字，就不会检查给定字符串s的其余部分，尽管提示说 "一个字符串包含一个数字"。这个错误在CODEGEN-MONO 6.1B的样本中没有出现，该函数正确地使用any()函数来捕获任意位置的数字。我们在附录D中包括了其他样本的并列比较。

5、Conclusion

我们研究用在代码数据上训练的大型因果语言模型进行程序合成。我们假设，随着模型规模和数据规模的扩大，对话能力会从简单的语言建模中出现。利用对话能力并观察到更好的用户意图理解会导致更好的程序合成，我们提出了一种对话式程序合成方法，其中程序合成是通过用户和系统之间的多轮对话实现的。在我们的研究中，我们还开发了多轮编程基准（MTPB）来研究我们的模型在这种对话范式下的程序合成能力。我们的实验证实，对话能力确实是从语言建模中产生的，而且对话编程能力是作为模型大小和数据大小的函数来扩展的。在多个步骤中指定的意图规范更容易被模型消化，并导致更准确的程序合成。我们开放了训练代码和模型检查点，以促进这一领域的未来研究和实际应用。

A Pass@k Estimator

我们使用Chen等人（2021）提出的无偏估计器来计算 pass@k。对于每个任务，$n \geq k$ 的样本被取样。特别是，我们使用 $n = 200$，$k \leq 100$。假设 c 是 n 个样本中，通过所有单元测试的正确样本的数量。那么无偏估计器定义如下：

$$\text{pass } @k = \mathbb{E}_{\text{Problems}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

直接计算这个估计器在数值上是不稳定的。图5显示了由(Chen et al, 2021)介绍的一个数值稳定的numpy实现方式。

```
def pass_at_k(n, c, k):
    """
    :param n: total number of samples
    :param c: number of correct samples
    :param k: k in pass@$k$
    """
    if n - c < k: return 1.0
    return 1.0 - np.prod(1.0 - k / np.arange(n - c + 1, n + 1))
```

Figure 5: 计算pass@k的无偏估计的数值稳定实现。