arXiv:2302.08468v3 [cs.LG] 1 Sep 2023

LEVER：学 习如何通过执行验证语言到代码生成

Ansong Ni [1 †] **Srini Iyer** [2] **Dragomir Radev** [1] **Ves Stoyanov** [2] **Wen-tau Yih** [2] **Sida I. Wang** [2 *] **Xi Victoria Lin** [2 *]

摘要

在代码上训练的大语言模型（代码大语言模型）问世后，语言到代码生成取得了显著进展。该领域最先进的方法将大语言模型解码与样本筛选和重排序相结合，利用测试用例或基于执行结果的启发式方法。然而，许多现实世界的语言到代码应用难以获得测试用例，而且启发式方法无法充分捕捉执行结果的语义特征，例如数据类型和取值范围，而这些特征往往能够指示程序的正确性。在本研究中，我们提出了 LEVER，这是一种简单的方法，通过学习依据生成程序的执行结果来验证程序，从而改进语言到代码生成。具体而言，我们训练验证器，根据自然语言输入、程序本身及其执行结果，判断从大语言模型中采样得到的程序是否正确。通过将验证分数与大语言模型的生成概率相结合，并对执行结果相同的程序进行边缘化处理，对采样程序进行重排序。在涵盖表格问答、数学问答和基础 Python 编程领域的四个数据集上，LEVER 始终优于基础代码大语言模型（使用 code-davinci-002 时提升 4.6% 至 10.9%），并在所有数据集上取得了新的最先进结果。

1. 引言

将自然语言映射为可执行代码的能力，是数据库接口（Pasupat 和 Liang, 2015年；Yu 等人, 2018年；Shi 等人, 2020年）、机器人控制（Zhou 等, 2021年；Shridhar 等, 2020年）和虚拟助手（Agashe 等, 2019；Lai 等, 2022年）等众多人工智能应用的基石。大语言模型（LLM）近期取得了进展（Brown 等, 2020年；Wei 等, 2021年；Chowdhery 等, 2022 年），尤其是那些在代码上预训练的模型（代码大语言模型）（Chen 等人, 2021a；Fried 等，2022；Nijkamp 等，2022；Li 等，2022a）已经表明，其在此类任务中通过上下文少样本学习（Shi 等人，2022；Chen 等人，2022a；Zhang 等，2022）展现出巨大潜力。然而，它们的性能仍远非完美（Chen 等人，2021a）。考虑到微调此类模型的计算成本，探索在不改变其参数的情况下改进它们的方法颇具吸引力。

一个关键观察是，尽管大语言模型在少样本设置下难以做到精确，但只要抽取足够多的样本，往往就能生成正确的输出。先前工作表明，大规模抽取样本时，多数投票和依据测试用例进行筛选能够显著提升模型的表现（Chen 等人，2021a；Austin 等人，2021；Li 等，2022a）。Shen 等人（2021）和 Cobbe 等人（2021）进一步证明，训练验证器并利用验证分数对数学文字题的候选解答重新排名是有效的。与仅依赖执行一致性和错误剪枝的方法相比，训练好的验证器能够利用模型解答中丰富的语义特征，例如数据类型、取值范围和变量属性；这些特征都可以有力地表明程序是否正确。Cobbe 等人（2021）及后续工作（Li 等，2022b；Kadavath 等人，2022）主要关注利用语言模型验证自然语言解答，一个自然而然的问题是，同样的方法能否应用于程序解答。

在本研究中，我们提出借助执行来学习验证（ LEVER）代码大语言模型的语言到代码生成。更具体地说，我们训练了一个验证器，使其基于自然语言描述、程序表面形式及其执行结果的联合表示，学习区分并拒绝不正确的程序。我们进一步将验证概率与大语言模型的生成概率相结合，并对具有相同执行结果的程序进行边缘化。我们将这一聚合概率用作重排序分数，并输出执行结果概率最高的程序。

我们在涵盖文本到

[†]大部分工作在 Meta AI 实习期间完成。[*]同等贡献 [1]耶鲁大学 [2]Meta AI。通讯作者：Ansong Ni <ansong.ni@yale.edu>，Xi Victoria Lin <victori- alin@meta.com>，Sida I. Wang <sida@meta.com>。

载于 40th 国际机器学习会议论文集，美国夏威夷檀香山。PMLR 202，2023年。版权归作者所有，2023年。

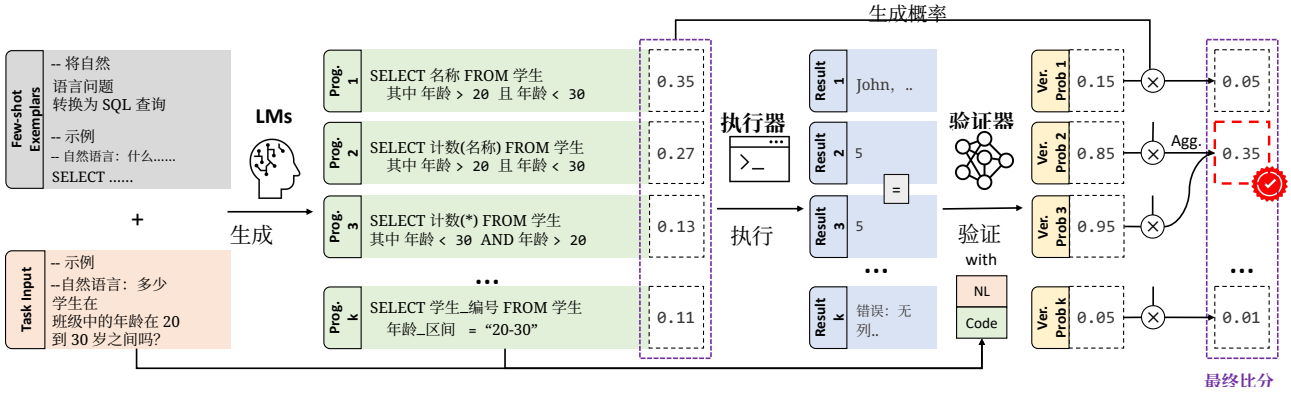


图1：以文本到 SQL 为例，展示 LEVER 的工作流程。它包含三个步骤：1) 生成：根据任务输入和少样本示例，从代码大语言模型中采样程序；2) 执行：使用程序执行器获取执行结果；3) 验证：使用学习型验证器，根据自然语言、程序和执行结果，输出程序正确的概率。

SQL语义解析、表格问答、数学推理和基础 Python 编程等领域的四个不同语言到代码基准测试上开展了广泛实验。使用三种不同代码大语言模型的实验结果表明，LEVER 持续提升了生成程序的执行准确率。值得注意的是，与采用执行错误剪枝的强基线相比，LEVER 与 code-davinci-002 结合后将执行准确率提升了 4.6% 至 10.9%，并在全部四个基准测试上取得了新的最先进结果，同时无需使用任务专用模型架构或提示方法。消融研究表明，执行结果对于验证至关重要；此外，LEVER 在低资源和弱监督设置下也带来了显著改进。[1][2]

2. 方法

下面我们将介绍 LEVER 的详细形式化定义和训练流程。其关键组成部分如图1所示。

2.1. 使用代码大语言模型进行语言到代码生成

语言到代码任务的输入通常包括自然语言（NL）描述，以及可选的编程上下文（例如，数据存储、断言等）。我们将此类输入记作 x。给定 x，生成模型 $P(y|x)$ 会生成一个程序 y，随后通过执行器 $\mathcal{E}(\cdot)$执行该程序，以获得结果[3] $\mathcal{E}(y)$。对于少样本学习，使用大型语言模型时，生成过程也常以一组固定的 m 示例作为条件，$\{(x_i, y_i)\}_{i<m}$。因此，使用代码大语言模型进行少样本语言到代码生成可表述为：

$$P_{\mathbf{LM}}(y|x) = P(y|\,\text{prompt}(x, \{(x_i, y_i)\}_{i<m})), \quad (1)$$

其中，$\text{prompt}(x,\ \{(x_i,\ y_i)\}_{i<m})$ 是对整体输入的字符串表示。通常使用贪心搜索来找到生成概率（近似）最高的程序，即 $\hat{}y_{\text{greedy}} \approx \text{argmax}_y\, P_{\mathbf{LM}}(y|x)$。

2.2. 程序候选项重排序

我们方法背后的关键观察是，从 $P_{\mathbf{LM}}(y|x)$中获取一个足够大的样本集，通常会包含正确的程序。这表明，对程序候选项进行重排序可能会显著改善结果。判别式重排序（Shen 等人，2004年；Collins 和 Koo，2005年）的思路是学习一个评分函数$R(x,\ \hat{}y)$，用于衡量$\hat{}y$ 作为输入 x的最佳输出的可能性。给定 $R(\cdot)$，重排序器会从候选项集合 S中输出重排序分数最高的程序：

$$\hat{y}_{\text{rerank}} = \arg\max_{\hat{y}\in S} R(x, \hat{y}) \quad (2)$$

接下来，我们将介绍如何采用训练好的验证器来验证并重排序从代码大语言模型中采样得到的程序候选项，使得$\hat{}y_{\text{rerank}}$ 优于$\hat{}y_{\text{greedy}}$。

从代码大语言模型中进行程序采样。给定输入 x，我们不进行贪心搜索，而是通过温度采样，从 $P_{\mathbf{LM}}(y|x)$ 中获取 k 个程序，即 也就是 $\{\hat{}y_i\}_{i=1}^{k} \sim P_{\mathbf{LM}}(y|x)$。由于相同的程序可能会被采样多

[1]为确保实验可复现，我们已开源实验代码：https://github.com/niansong1996/lever。[2]根据 Cobbe 等人（2021）的定义，"验证"是指评估程序的功能是否符合程序员意图。这与编程语言中的形式化验证概念不同（Seligman 等人，2015）。[3]一些数据集（如 Spider（Yu 等人，2018））要求输入值与程序一同生成，因此 y^* 可直接执行。其他数据集则要求程序可执行

在单独提供的测试用例上，例如，MBPP（Austin 等人，2021年）。为简洁起见，我们采用这一记法。

次，我们会进行去重，形成由 n唯一的程序候选项 $S = \{\hat{}y_i\}_{i=1}^{n}$ 构成的集合，其中 $n \leq k$。我们选择采样而非束搜索，主要有两个原因：1）近期研究表明，由于会生成退化程序，代码生成中的束搜索通常会导致性能更差（Austin 等人，2021年；Zhang 等，2022）；2）对于我们测试的所有大语言模型，束搜索并非都可用或都能得到高效实现（例如，Codex）。

基于执行的验证。 我们将问题描述 x、候选程序 $\hat{}y$及其执行结果 $\mathcal{E}(\hat{}y)$ 的表示简单拼接，作为重排序器的输入。受近期研究（Cobbe 等人，2021年；Li 等，2022b）的启发，我们将判别式重排序器参数化为验证（即 二分类）模型 $P_\theta(v|x,\ \hat{}y,\ \mathcal{E}(\hat{}y))$，其中 $v \in \{0,1\}$。在实践中，重排序器可使用任何二分类架构实现。我们在 §B.2 中报告了使用 T5（Raffel 等人，2020年）和 RoBERTa（ Liu 等人，2019年）进行的实验。

给定输入 x 和候选程序 $\hat{}y \in S$，我们将重排序概率视为生成且通过验证的联合概率：

$$P_R(\hat{y}, v_{=1}|x) = P_{\mathbf{LM}}(\hat{y}|x) \cdot P_\theta(v_{=1}|x, \hat{y}, \mathcal{E}(\hat{y})) \quad (3)$$

执行结果聚合。 由于语义相同的程序可能具有不同的表面形式，我们进一步聚合在 S中执行后得到相同结果的程序的重排序概率。这样，我们便降低了对表面形式的依赖，转而关注执行结果。因此，用于重排序的最终评分函数为：

$$R(x, \hat{y}) = P_R(\mathcal{E}(\hat{y}), v_{=1}|x) = \sum_{y \in S, \mathcal{E}(y)=\mathcal{E}(\hat{y})} P_R(y, v_{=1}|x)$$

由于可能有多个程序共享概率最高的执行结果，因此在输出程序时，我们会随机打破平局。

2.3. 学习验证器

前文介绍了如何在推理时使用验证器。接下来，我们将介绍其训练过程。

训练数据创建。 对于语言到代码数据集，每个样例通常是一个三元组（x，y^*，z^*），其中$z^* = \mathcal{E}(y^*)$是标准执行结果，y^* 是标准程序。由于标注程序需要领域专业知识，对于某些可以直接获得最终结果的数据集，学习时只提供 z^*，而不提供 y^*（Artzi & Zettlemoyer，2013年；Cheng & Lapata，2018年；Gold- man et al.，2018年）。这称为弱监督设置。为收集训练数据，我们为训练集中的每个输入 n 获取一组唯一程序候选项 $\hat{}S = \{\hat{}y_i\}_{i=1}^{n}$，首先从 $P_{\mathbf{LM}}$（$\hat{}y|x$）中采样 k 个程序，然后移除所有重复程序，与推理时的做法类似。接着，对于每个程序候选项$\hat{}y \in S$，我们通过比较其执行结果$\hat{}z = \mathcal{E}$（$\hat{}y$）与标准[4] 执行结果 z^*，来获取其二元验证标签，即 $v = \mathbb{1}$（$\hat{}z = z^*$）。对于包含标准程序 y^*的数据集，我们会追加（x，y^*，z^*，$v_{=1}$）作为额外的验证训练示例；对于弱监督数据集，则跳过此步骤。这样，我们就为每个输入 x创建了一组验证训练示例 $\{$（x，$\hat{}y_i$，$\hat{}z_i$，v_i）$\mid \hat{}y_i \in S\}$。

学习目标。 给定这组验证训练示例，我们使用负对数似然函数来构造输入 x 的损失，并按程序候选项的数量进行规范化

$$\mathcal{L}_\theta(x, S) = -\frac{1}{|S|} \cdot \sum_{\hat{y}_i \in S} \log P_\theta(v_i|x, \hat{y}_i, \hat{z}_i) \quad (4)$$

规范化步骤十分重要，可防止包含大量独特程序候选项的示例主导训练。

3. 实验设置

3.1. 数据集

我们在涵盖语义解析、表格问答、数学推理和基础 Python 编程领域的四个语言到代码数据集上进行实验。这四个数据集的主要设置见表 1。验证所需的更详细设置见附录中的表 7。

▷ **Spider**（Yu 等人，2018年）是一个语义解析数据集，旨在根据自然语言问题生成 SQL 查询。它包含 7k 条平行训练数据，也非常适合用于微调生成器；

▷ **WikiTableQuestions（WikiTQ）**（Pasupat 和 Liang，2015 年）是一个表格问答数据集，我们尝试通过在源表上生成并执行 SQL 查询来解决其中的问题。我们使用 Shi 等（2020年）预处理后的表格，并采用他们标注的 SQL 查询，为原本的弱监督数据集添加标准程序；

▷ **GSM8k**（Cobbe 等人，2021年）是一个用于解答小学数学文字题的基准测试。遵循先前工作（Chowdhery 等，2022年；Chen等，2022b；Gao等，2022），我们通过根据自然语言问题生成 Python 程序来处理这一基准测试，这些程序在执行后应能得出正确答案。原始数据集只有自然语言内容，没有程序解答，因此它是

[4]对于提供多个测试用例的数据集，只有当程序在所有测试用例上的执行结果都与真实程序一致时，我们才将其标记为正确。

	Spider	WikiTQ	GSM8k	MBPP
领域	表 QA	表 QA	Math QA	基础编程
包含程序	✓	✓*	✗	✓
目标	SQL	SQL	Python	Python
数据统计				
# 训练	7,000	11,321	5,968	378
# 开发集	1,032	2,831	1,448	90
# 测试集	-	4,336	1,312	500
少样本生成设置				
输入 Format	模式 + NL	模式 + NL	NL	断言 + NL
# 提示示例数	8‡	8	8	3
# 样本（训练 / 测试集）	20/50†	50/50	50/100	100/100
生成长度	128	128	256	256

表1：本研究所用数据集概述。 *：WikiTableQuestions 中约 80% 的样例由 Shi 等（2020）标注了 SQL。 †：为提高上界，InCoder 和 CodeGen 使用 50/100。 ‡：由于上下文长度和硬件限制，InCoder 和 CodeGen 仅使用 8 个示例中的前 2 个。

用于语言到代码转换的弱监督数据集；
▷ **MBPP** （Austin 等人，2021）包含用自然语言描述的基础 Python 编程程序。每个样例都配有 3 个测试用例，用于检查程序的正确性。遵循先前工作（Shi 等人，2022；Zhang 等，2022），我们将第一个测试用例作为提示的一部分，以便模型生成正确的函数签名，并使用全部三个测试用例评估正确性。

3.2. 代码大语言模型

我们使用三种不同的代码大语言模型评估 LEVER：
▷ **Codex** （Chen 等人，2021a）是 OpenAI 开发的一系列不同规模的代码大语言模型。具体而言，我们使用 code-davinci-002 API[5]，通过其官方 Python 绑定调用。

▷ **InCoder** （Fried 等，2022）是一系列参数规模最高达 60 亿、使用宽松许可的代码语料库进行训练的代码大语言模型。我们对 InCoder-6B 进行实验，并将其用于从左到右生成。

▷ **CodeGen** （Nijkamp 等，2022年）是一个代码大语言模型系列，我们评估了 CodeGen-16B-multi 版本。尽管 CodeGen 的训练语料库中不包含 SQL 文件，但我们发现它在 SQL 生成任务上的表现仍然相当不错，这可能是因为 SQL 查询与其他编程语言的源文件混合在一起。

[5]https://openai.com/api/

3.3. 基线和评估指标

基线。 我们将 LEVER 与以下使用代码大语言模型生成程序的基线方法进行比较。
▷ **贪心**：在每个解码步骤中选择最可能的词元。▷ **最大似然（ML）**：从 k 个采样得到的程序候选项中，选择生成对数概率最高的程序，即 $\log P_{\mathbf{LM}}(\hat{y}|x)$（或归一化生成对数概率 $\log P_{\mathbf{LM}}(\hat{y}|x)/\lceil\hat{y}|\rceil$）。我们通过开发集上的实证结果，确定每个数据集是否使用归一化概率。更多详情见附录 A。

▷ **错误剪枝 +ML（EP +ML）**：剔除存在执行错误的程序候选项，然后选择最大似然的程序；

▷ **错误剪枝 + 投票（EP + 投票）**：对无错误程序的执行结果进行多数投票，选出得票最多的执行结果及其对应的程序。

我们重点与 EP+ML 基线进行比较，因为它是一种利用执行信息的简单重排序方法，在不同数据集和代码大语言模型上均能持续取得有竞争力的结果。

评估指标。 遵循先前工作（Xie 等，2022；Liu 等，2021；Ni 等，2022；Zhang 等，2022），我们将执行准确率 作为所有数据集的主要评估指标，该指标衡量能够得出标准执行结果或通过所有测试用例的样本所占的比例。

3.4. 实现细节

验证器训练。 我们按照表1所述的采样预算，从训练集上的大语言模型中采样，构建验证器训练数据。更多生成的训练数据统计信息见附录中的表7。学习验证器时，如公式4所示，训练损失通过对每个样本的所有程序样本取平均值计算。由于我们会将同一样本对应的程序样本放在一起组成批次，有效批次大小也会随样本数量增加。当样本数量较大时（我们的实验中最高为100），这可能会造成问题，因为它们可能无法一次全部装入 GPU 内存。因此，我们会在每次迭代中对每个样本用于学习的程序进行下采样。随机下采样在每个训练轮次开始时进行，使验证器在每个训练轮次都能看到不同的程序。具体批次大小和下采样系数见附录中的表7。

执行结果表示。 验证器的输入由任务输入、候选程序及其执行结果拼接而成。对于 Spider 和 WikiTQ，我们将 SQL 执行生成的线性化结果表用作

方法	Dev
无微调的先前工作	
Rajkumar 等（2022年）	67.0
MBR-Exec（Shi 等人，2022年）	75.2
Coder-Reviewer（Zhang 等人，2022年）	74.5
以微调为基础的先前工作	
T5-3B（Xie 等，2022年）	71.8
PICARD（Scholak 等，2021年）	75.5
RASAT（Qi 等，2022年）	80.5
使用 *code-davinci-002* 的本工作	
贪心	75.3
EP + ML	77.3
LEVER	**81.9**$_{\pm 0.1}$

表 2：Spider 数据集上的执行准确率。标准差是基于三次使用不同随机种子的运行结果计算得出的（以下表格中呈现标准差时亦同）。

方法	Dev	Test
无微调的先前工作		
Codex QA* (Cheng 等，2022)	50.5	48.7
Codex SQL（Cheng 等，2022年）	60.2	61.1
Codex Binder（Cheng 等，2022年）	**65.0**	64.6
以微调为基础的先前工作		
TaPEX* （Liu 等，2021年）	60.4	59.1
TaCube（Zhou 等，2022年）	61.1	61.3
OmniTab* （Jiang 等，2022年）	-	63.3
使用 *code-davinci-002* 的本工作		
贪心	49.6	53.0
执行程序 + 机器学习	52.7	54.9
LEVER	64.6$_{\pm 0.2}$	**65.8**$_{\pm 0.2}$

表 3：WikiTQ数据集上的执行准确率。 *：建模为端到端问答，不以生成程序作为中间过程。

执行结果。对于 GSM8k，我们将执行程序后名为“answer”的变量的值用作执行结果。对于 MBPP，我们使用函数返回的类型和值（转换为字符串）。所有执行错误均表示为“ERROR: ⟦reason⟧”，例如“ERROR: Time out”。不同数据集的这些验证器输入示例见表 11。

验证器模型选择。 我们使用开发集来选择最佳验证器模型。在主要实验中[6]，我们选择 T5-base 作为 Spider 的基础语言模型，选择 T5-large 作为 WikiTQ 和 MBPP 的基础语言模型，并选择 RoBERTa-large 作为 GSM8k 验证器所使用的基础语言模型。选择过程详见 § B.2。对于 T5模型（Raffel 等人，2020年），给定验证器输入后，我们训练模型针对每个正例/负例输出“yes/no”标记，并在推理过程中将生成“yes”的概率作为验证概率。对于 RoBERTa（Liu 等人，2019年），我们在其 ⟦CLS⟧ 头之上添加一个线性层，遵循使用仅编码器模型进行序列分类的标准做法（Devlin 等人，2019年）。

大语言模型采样、少样本提示构建和特定数据集设置的详情见附录 A。

4. 主要结果

我们展示了 LEVER 与 Codex-Davinci 结合后的性能，并将其与先前工作中针对 Spider（表 2）、WikiTQ（表 3）、GSM8k（表 4）和 MBPP（表 5）所报告的最先进的微调性能和少样本性能进行比较。此外，我们还在 Spider 和 GSM8k 上使用 InCoder 和 CodeGen 模型评估了 LEVER（表 6）。

[6]我们尝试以少样本方式将代码大语言模型本身用作验证器，但其表现不如执行程序+机器学习。

方法	Dev	Test
无微调的先前工作		
PAL（Gao 等人，2022年）-		72.0
Codex +SC† （Wang 等人，2022年）	-	78.0
PoT-SC（Chen 等人，2022b）	-	80.0
以微调为基础的先前工作		
Neo-2.7B +SS（Ni 等，2022）	20.7	19.5
Neo-1.3B +SC（Welleck 等，2022）	-	24.2
DiVeRSe*† （Li 等，2022b）	-	83.2
本研究与 *codex-davinci-002*		
贪心	68.1	67.2
执行程序 + 机器学习	72.1	72.6
LEVER	**84.1**$_{\pm 0.2}$	**84.5**$_{\pm 0.3}$

表 4：GSM8k 数据集上的执行准确率。 *：微调模型与 Codex 结合（类似于 LEVER）； †：生成自然语言解决方案，而非程序。

4.1. LEVER 的有效性。

LEVER 在所有任务上均持续提升了所有代码大语言模型的性能：相较于 Codex-Davinci 的贪婪解码基线，Spider 上提升了 6.6%，WikiTQ 上提升了 17.3%。对于 InCoder 和 CodeGen 等较弱模型，我们观察到在 Spi- der 上最高提升 30.0%，在 GSM8k 上最高提升 15.0%。此外，LEVER 与 Codex-Davinci 结合后，还在全部四个数据集上取得了新的最先进结- 果，提升幅度从 WikiTQ 上的 1.2% 到 MBPP 上的 2.0% 不等。在具有挑战性的文本- 到-SQL 数据集 Spider 上，此前的最先进结果是通过微调一个经关系感知自注意力增强的 T5-3B 模型获得的；我们使用 Codex-Davinci + LEVER 取得了更好的结- 果，其中验证器使用 T5-base 模型进行微调。LEVER 还使用 InCoder 和 CodeGen 分别将 Spider 上此前的最佳结果提升了 13.2% 和 20.6%。

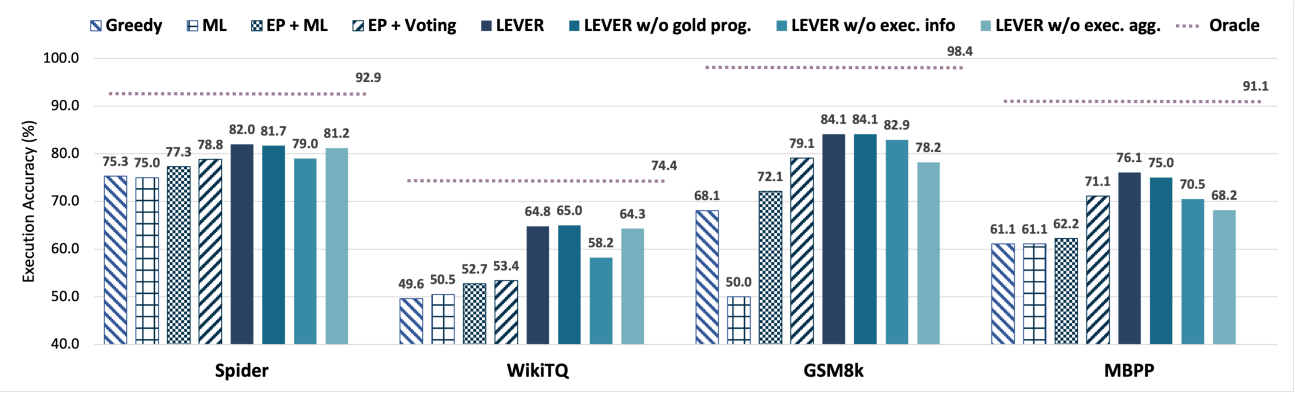


图2：比较 LEVER与使用 Codex-Davinci 的基线。LEVER 及其消融实验结果以实心条形表示 .

方法	Dev	Test
无微调的先前工作		
MBR-Exec（Shi 等人，2022年）	-	63.0
Reviewer（Zhang 等，2022）	-	66.9
本研究使用 *codex-davinci-002*		
贪心	61.1	62.0
执行程序 + 机器学习	62.2	60.2
LEVER	**75.4**$_{\pm 0.7}$	**68.9**$_{\pm 0.4}$

表 5：MBPP 数据集上的执行准确率。

由于 LEVER 是一种将少样本语言模型生成与学习型验证器相结合的简单方法，它有望进一步受益于更先进的提示方法（Li 等，2022b；Cheng 等，2022）或模型架构（Qi 等，2022；Wang 等，2020），这将留待未来工作探索。

4.2. 使用 LEVER 的消融实验

我们使用 Codex-Davinci 对 LEVER 进行消融研究，并与 § 3.3 中提到的基线进行比较，结果如图2所示。我们还对 InCoder 和 CodeGen 进行了相同的消融实验，结果见表 6。在这些结果中，我们纳入了**“预言机”**性能，即只要正确程序出现在样本集中，便始终选择该正确程序所获得的性能。

纳入执行结果的影响。如图 2 所示，从验证器输入中移除执行结果后，四个基准上的性能均大幅下降，这表明执行结果对验证器训练十分重要。这种影响在不同数据集上有所差异。WikiTQ 和 MBPP 的绝对性能分别下降 6.6% 和 5.6%，而 Spider 和 GSM8k 的降幅较小，分别为 3.0% 和 1.2%。我们发现，WikiTQ 和 MBPP 的代码样本包含更多执行错误，这解释了为什么我们的方法在这两个数据集上更为有效。表 6 显示了类似的趋势

方法	InCoder-6B		CodeGen-16B	
	Spider	GSM8k	Spider	GSM8k
先前工作：				
MBR-EXEC	38.2	-	30.6	-
Reviewer	41.5	-	31.7	-
基线：				
贪心	24.1	3.1	24.6	7.1
ML	33.7	3.8	31.2	9.6
执行程序 + 机器学习	41.2	4.4	37.7	11.4
执行程序 + 投票	37.4	5.9	37.1	14.2
LEVER	54.1	**11.9**	51.0	**22.1**
–金标准程序	53.4	-	**52.3**	-
–执行信息	48.5	5.6	43.0	13.4
–执行聚合	**54.7**	10.6	51.6	18.3
预言机	71.6	48.0	68.6	61.4

表 6：使用 InCoder 和 CodeGen 作为代码大语言模型，并以 T5-base 作为验证器，在开发集上进行评估的结果。先前工作的结果摘自 Zhang 等（2022）。

对于 Spider 和 GSM8k 上的 InCoder-6B 和 CodeGen-16B，规模较小的语言模型少样本性能更差；从验证器中移除执行信息，往往会导致更大的性能下降。此外，我们发现 LEVER 总体上优于 EP+ML 基线，这表明验证器能够利用简单执行错误之外的线索。关于执行信息何时有帮助的更详细定量分析见图 6。

执行结果聚合的效果。 将执行结果相同的程序聚合起来是一种简单且广泛使用的技术（Chen 等，2022b；Cheng 等，2022）。我们发现，执行聚合对 Python 输出的数据集上的 LEVER 效果良好，但对 SQL 数据集的帮助有限。一个可能的原因是，与 SQL 等领域特定语言相比，Python 代码的结构更加灵活。在数据库查询领域，错误程序更有可能

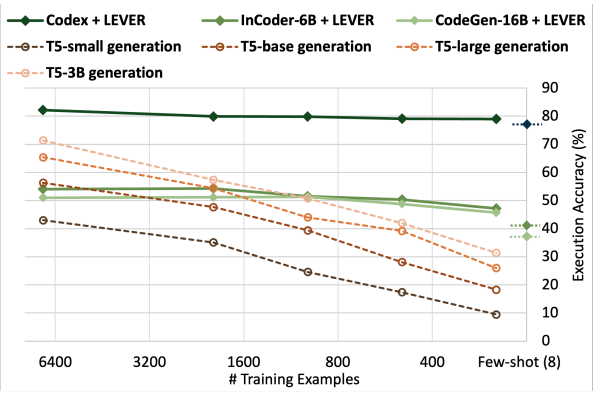


图3：Spider 训练样例数量减少时的验证与生成性能。y轴上的数据标记表示 EP+机器学习基线，而x轴采用对数刻度。LEVER 使用 T5-base 作为基础模型。WikiTQ 和 GSM8k 的结果见附录中的图7。

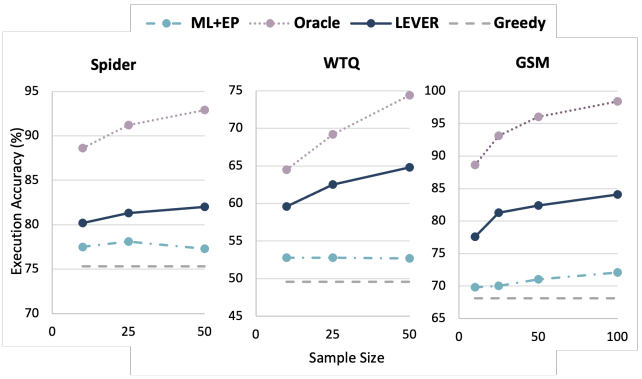
执行出一些无关紧要但错误的结果（例如，“0”或空表）。聚合后，这类错误结果可能累积足够的概率质量，超过正确结果，从而对性能产生负面影响。

弱监督设置。我们还比较了 LEVER 在完全监督和弱监督设置下的性能。图2和表 6显示，当不提供标准程序而采用弱监督设置（§2.3）时，LEVER 的性能基本得以保持，绝对性能仅下降至多 1.1%。这表明 LEVER 在弱监督设置下表现良好；与执行结果相比，程序本身对于验证所提供的信息较少。

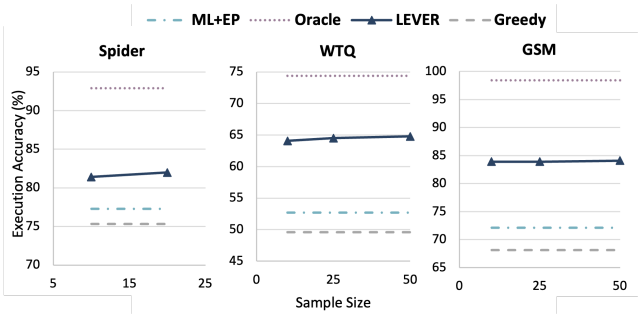
5. 分析

5.1. 训练样例规模扩展

图 3 展示了以 Spider 为例，LEVER 的性能如何随着训练样例减少而变化。关于 WikiTQ 和 GSM8k 的更多结果见 § B.3。即使只提供 250 个样例，LEVER 相较于基础大语言模型的提升仍然稳定，在不同数据集和大语言模型上的提升幅度为 1.7% 至 10.0%。这表明 LEVER 能够在少资源设置下发挥作用。此外，不同数据集和代码大语言模型的表现趋势也有所不同。例如，使用 Codex 作为大语言模型时，LEVER 在 WikiTQ 上的性能下降了 6.4%，而在 Spider 上仅下降了 3.2%。然而，即使是在 Spider 上，使用 InCoder 和 CodeGen 时，性能也分别下降了 6.9% 和 5.3%。这表明，对于难度更高的数据集和能力较弱的语言模型，增加 LEVER 的训练样例会带来更大的影响。



(a) LEVER在推理时的样本规模消融实验，训练时的样本规模固定为表1中的设置。



(b) 训练验证器时，每个样例采样不同数量程序的性能。推理时的样本规模固定为表1中的设置。

图 4：训练期间的样本量和推理时的样本量如何影响 LEVER +Codex-Davinci 的性能。

结合图3，我们还比较了 LEVER 与使用相同数量训练样例直接微调以生成代码的 T5模型的性能。只需数百个样例即可学会验证，但可用训练样例较少时，微调后的 T5模型性能会大幅下降。例如，对于500个样例，基于 InCoder/CodeGen 的 T5-base验证器比微调后的 T5-3B生成器高出 ∼ 7%。

5.2. 样本规模扩展

由于从大语言模型中抽取样本在计算上可能代价高昂，我们在此研究训练和推理时的样本数量如何影响性能。如图 4a 所示，在推理时，将每个示例的程序样本数从 50 降至 10 时，LEVER 的性能下降了 1.8%（Spider）至 5.2%（WikiTQ）。这表明 LEVER 对推理时的样本数量较为敏感，这也在预期之中，因为样本数量也会显著影响预言机结果（即，重排序的上界）。相比之下，图 4b 显示，提供训练

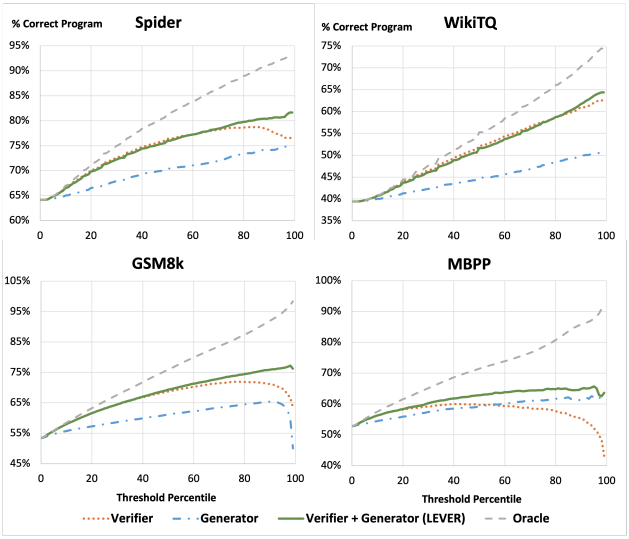


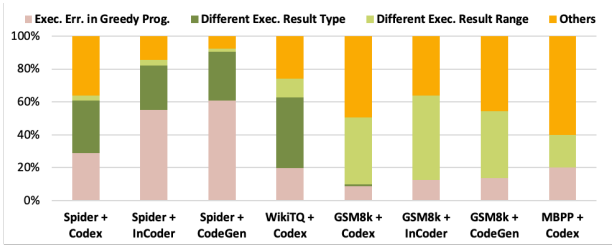
图5：验证器、生成器（Codex- Davinci）及二者组合概率（LEVER 使用）的校准。首先根据模型概率对采样程序进行排名。 x轴表示阈值筛选后排除的样本百分比， y轴表示剩余样本中正确程序的百分比。为确保不同程序的评分相互独立，这组图中未应用执行聚合。

数据时，LEVER 对样本数量极不敏感，在三个数据集上的性能差距均低于 1%。总体而言，结果表明，较高的采样预算在测试时更有帮助。

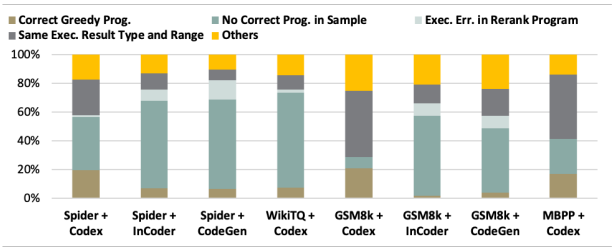
5.3. 验证器和生成器校准

我们研究了验证器和生成器在识别正确程序方面的校准程度。理想情况下，正确的程序样本应获得更高的概率，因此，当程序越接近顶部时，我们应观察到更高比例的程序是正确的。为此，我们对验证器、生成器和 LEVER 的预测分数（如公式 3 所示）进行排序，并移动百分位阈值，测量排名靠前的程序中正确程序所占的比例。根据图5，验证器通常比生成器具有更好的校准效果，尤其是在阈值处于较低百分位时。这表明，由于执行结果属于其输入的一部分，验证器更容易识别程序中的明显错误。有趣的是，在区分排名靠前的程序时，四个受测试数据集中的三个数据集里的验证器校准效果较差[7]。然而，生成器在这一范围内通常具有更好的校准效果，将验证器和生成器的概率结合起来，则在全部四个基准测试上都取得了最佳结果。更具体地说，

[7]我们的假设是，排名靠前的程序在形式和执行结果上非常相似（例如，具有相同的类型和范围），这使得规模较小但经过微调的模型难以区分它们。



(a) LEVER 将正确程序重新排至首位，但贪婪解码失败时。



(b) LEVER 未能将正确程序排至首位时。

图 6：定量分析 LEVER 在哪些情况下能够或无法提升代码大语言模型相较于贪婪解码的表现。

在 GSM8k 数据集上，尽管两个模型对排名靠前程序的校准效果都相当差，但它们的联合概率却出人意料地得到了良好校准，表明这两个模型在该数据集上能够相互补充。

5.4. 定量分析

我们对 LEVER 成功或未能提升大语言模型性能的原因进行了定量分析。如图 6 所示，当 LEVER 对程序进行重排序，以替换生成概率更高的另一个程序时，往往是因为执行结果提供了关键信息，例如执行错误、变量类型和范围。这与我们在 § 4.2 中关于执行结果对 LEVER 重要性的发现一致。还值得注意的是，在某些情况下，即使无错误的执行结果与贪心程序的执行结果具有相同的类型和范围，LEVER 仍能对正确程序进行重排序；这类情况属于“其他”类别。我们的假设是，此时程序本身成为验证器可利用的主要特征。此外，当 LEVER 未能将正确程序排在首位时，最常见的原因是样本中找不到正确程序（即达到上界），这种情况在较弱的语言模型中尤为常见。LEVER 失败的第二个最常见原因是，重排序时错误程序的执行结果与样本中正确程序的执行结果具有相同的类型和范围。在这种情况下，执行结果无法为验证器提供丰富的信息，因此 LEVER 未能提升代码大语言模型的性能。

6. 相关工作

语言到代码生成。将自然语言转换为代码是人工智能发展各个时代长期面临的挑战，其中包括基于规则的系统（Woods，1973；Templeton 和 Burger，1983）、结构化预测（Zelle 和 Mooney，1996；Zettlemoyer 和 Collins，2005年；Gulwani 和 Marron，2014）以及深度学习（Xiao 等人，2016；Dong 和 Lapata，2016；Rabinovich 等人，2017；Zhong 等人，2017；Lin 等人，2017）。近年来，预训练代码语言模型（Chen 等，2021年a；Wang 等，2021年；Fried 等，2022年；Nijkamp 等，2022年；OpenAI，2022年）在这一问题上展现出令人惊讶的强大能力，涵盖多种编程语言（Lin 等人，2018年；Yu 等人，2018年；Austin 等人，2021年；Cobbe 等人，2021年；Li 等，2022年a）。研究者提出了多种方法来改进大语言模型的样本选择，包括基于测试用例执行（Li 等，2022年a）、跨样本相似度（Chen 等，2021年a；Li 等，2022年a；Shi 等人，2022年）和最大互信息（Zhang 等，2022年）进行筛选。我们的工作提出了一种可学习验证模块，用于判断大语言模型的样本输出，从而进一步提升其性能。

带执行的代码生成。以往的代码生成工作以不同方式利用执行结果。弱监督学习方法（Berant 等人，2013年；Pasupat 和 Liang，2015年；Guu 等人，2017年）将程序建模为潜变量，并利用执行结果得出监督信号。中间执行结果被用于指导训练阶段（Chen 等人，2019年；2021b）和推理时（Wang 等人，2018年）的程序搜索。在大规模采样时，基于执行结果的多数投票已被证明有助于候选项选择（Li 等，2022a；Cobbe 等人，2021年）。Shi 等（2022）将这一原则推广为：选择在执行结果上与其他样本共识度最高的样本。我们提出训练一个验证模型，在考虑执行结果的基础上判断代码生成的正确性。

学习验证。先前工作已证明，在数学问答（Shen 等人，2021；Cobbe 等人，2021）和常识问答（Li 等人，2022b）等领域中，学习型验证器对于样本筛选是有效的；在这些领域，解决方案主要以自然语言描述。虽然更常见的做法是独立于生成器训练验证器（Cobbe 等人，2021；Li 等人，2022b），但 Shen 等人（2021）在同一时间对二者进行了联合微调。先前工作还为验证器使用了不同的基础语言模型。Cobbe 等人（2021）使用 GPT-3（Brown 等，2020），而 Li 等人（2022b）使用 DeBERTa（He 等人，2020）。除了特定任务的验证器之外，Kadavath 等人（2022）表明，大型语言模型能够在少样本设置下对其输出进行自我验证，且适用于广泛的任务。Chen 等人（2022a）以及其他工作（Tufano 等人，2020；Li 等人，2022a）使用语言模型生成测试用例，而不是直接判断输出程序的正确性。相比之下，LEVER 的设置更接近 Li 等人（2022b）：我们独立训练验证器，并为其使用规模小得多的语言模型（约为生成器参数规模的 0.5%）。我们首次对自然语言到代码任务进行了全面评估，并利用了程序执行结果[8]。

判别式重排序。判别式重排序方法长期以来被用于进一步提升序列生成任务的性能，包括摘要（Wan 等人，2015年）、机器翻译（Shen 等人，2004年；Lee 等人，2021年）、对话回复生成（Olabiyi 等人，2018年），以及近年来的代码生成（Yin 和 Neubig，2019年）。LEVER 可视为一种判别式重排序框架。

7. 局限性

在本研究中，我们使用执行信息来验证 LEVER 中的程序。然而，程序的执行至少依赖一组输入（例如，函数参数）和适当的执行环境（例如，数据库），而某些应用可能无法提供这些条件。此外，我们不能总是假定模型生成的程序可以安全执行。另外，我们在实验中采用 PASS@1 作为主要评估指标。对于文本到 SQL 和数学推理等应用而言，这一指标很理想，因为用户只是在寻找问题的答案；但对于 MBPP 等通用编程任务，PASS@k 或 N@k 等指标或许能提供不同的视角。

8. 结论

我们提出了 LEVER，这是一种简单的方法，通过学习独立的验证模型来判断生成程序的正确性，并将其执行结果纳入考量，从而提升代码大语言模型在自然语言到代码任务上的表现。我们表明，利用有监督基准数据集训练的验证器，其规模可以约为生成器的 0.5%。我们还表明，相比直接根据验证器的输出进行拒绝采样，将生成概率与验证概率结合起来进行采样排名效果更好。LEVER 持续提升代码大语言模型在四项自然语言到代码任务上的表现，并在所有任务上取得了新的最先进结果。进一步分析表明，程序执行结果对验证至关重要，而且所提出的方法可以推广到不同的大语言模型。

[8]尽管 Kadavath 等人（2022）也报告了在 HumanEval 上进行自我验证的结果，但他们的方法并未利用执行。

致谢

作者感谢 Xi Ye、Tianyi Zhang、Mengzhou Xia、LukeZettlemoyer，以及匿名审稿人提出的有益讨论和意见。

参考文献

Agashe, R.、Iyer, S. 和 Zettlemoyer, L. JuICe：一个用于开放领域上下文代码生成的大规模远程监督数据集。载于*2019*年自然语言处理实证方法会议和第九届国际自然语言处理联合会议论文集（*EMNLP-IJCNLP*），第 5436–5446 页，中国香港，2019年11月。计算语言学协会。doi：10.18653/v1/D19-1546。URL：https: // aclanthology.org/D19-1546。

Artzi, Y. 和 Zettlemoyer, L. 用于将指令映射到动作的语义解析器弱监督学习。计算语言学协会汇刊，1:49–62，2013年。

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry,M., Le,Q.,et al. 使用大语言模型进行程序合成。*arXiv* 预印本 *arXiv:2108.07732*，2021年。

Berant, J., Chou, A., Frostig, R., and Liang, P. 基于问答对的 Freebase 语义解析。载于自然语言处理实证方法会议（*EMNLP*），2013年。

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. 语言模型是少样本学习器。《神经信息处理系统进展》，33：1877–1901，2020年。

Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., 和 Chen, W. Codet：使用生成的测试集进行代码生成，2022a。网址：https://arxiv.org/abs/2207.10397。

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., 等。评估在代码上训练的大语言模型。*arXiv* 预印本 *arXiv:2107.03374*，2021a。

Chen, W., Ma, X., Wang, X., 和 Cohen, W. W. 程序思维提示：将计算与数值推理任务中的推理分离。*arXiv* 预印本 *arXiv:2211.12588*，2022b。

Chen, X.、Liu, C. 和 Song, D. 执行引导的神经程序合成。发表于国际会议学习表征，2019年。网址：https:// openreview.net/forum?id=H1gfOiAqYm。

Chen, X.、Song, D. 和 Tian, Y. 超越领域特定语言的神经程序合成潜在执行。载于 Ranzato, M.、Beygelzimer, A.、Dauphin, Y. N.、Liang, P. 和 Vaughan, J. W.（编），神经信息处理系统进展第*34*卷：*2021*年神经信息处理系统年会，*NeurIPS 2021*，*2021*年*12*月*6*日至*14*日，线上，第 22196–22208 页，2021b。

Cheng, J. 和 Lapata, M. 使用生成式排序器进行弱监督神经语义解析。载于第*22*届计算自然语言学习会议论文集，第 356–367 页，2018年。

Cheng, Z.、Xie, T.、Shi, P.、Li, C.、Nadkarni, R.、Hu, Y.、Xiong, C.、Radev, D.、Ostendorf, M.、Zettlemoyer, L. 等。将语言模型绑定到符号语言中。*arXiv* 预印本 *arXiv:2210.02875*，2022年。Chowdhery, A.、Narang, S.、Devlin, J.、Bosma, M.、Mishra, G.、Roberts, A.、Barham, P.、Chung, H. W.、Sutton, C.、Gehrmann, S. 等。Palm：借助 Pathways 扩展语言建模。*arXiv* 预印本 *arXiv:2204.02311*，2022年。Cobbe, K.、Kosaraju, V.、Bavarian, M.、Hilton, J.、Nakano, R.、Hesse, C. 和 Schulman, J. 训练验证器以解答数学应用题。*arXiv* 预印本 *arXiv:2110.14168*，2021年。Collins, M. 和 Koo, T. 自然语言解析的判别式重排序。《计算语言学》，31(1):25–70，2005年。Devlin, J.、Chang, M.、Lee, K. 和 Toutanova, K. BERT：用于语言理解的深度双向变换器预训练。载于 Burstein, J.、Doran, C. 和 Solorio, T.（编），*2019*年北美计算语言学协会分会年会：人类语言技术论文集，*NAACL-HLT 2019*，美国明尼苏达州明尼阿波利斯，*2019*年*6*月*2*日至*7*日，第*1*卷（长篇与短篇论文），第 4171–4186 页。计算语言学协会，2019 年。doi: 10.18653/v1/n19-1423。URL https://doi.org/ 10.18653/v1/n19-1423。

Dong, L. 和 Lapata, M. 利用神经注意力将语言转换为逻辑形式。载于计算语言学协会第*54*届年会论文集，*ACL 2016*，*2016*年*8*月*7*日至*12*日，德国柏林，第*1*卷：长篇论文。计算语言学协会，2022年。doi: 10.18653/v1/p16-1004。网址：https://doi.org/10.18653/v1/p16-1004。

Fried, D.、Aghajanyan, A.、Lin, J.、Wang, S.、Wallace, E.、Shi, F.、Zhong, R.、Yih, W.-t.、Zettlemoyer, L. 和 Lewis, M. InCoder：用于代码填充和合成的生成模型。*arXiv* 预印本 *arXiv:2204.05999*，2022年。

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL：程序辅助语言模型。*arXiv* 预印本 *arXiv:2211.10435*，2022年。

Goldman, O., Latcinnik, V., Nave, E., Globerson, A., and Berant, J. 使用抽象示例进行弱监督语义解析。载于计算语言学协会第*56*届年会论文集（第*1*卷：长篇论文），第1809–1819页，2018年。

Gulwani, S. and Marron, M. Nlyze：通过自然语言交互式编程，以分析和处理电子表格数据。载于 Dyreson, C. E., Li, F., and ¨Ozsu, M. T.（编），*2014*年国际数据管理会议（*SIGMOD*），美国犹他州斯诺伯德，*2014*年*6*月*22*日至*27*日，第803–814页。ACM，2014年。doi：10. 1145/2588555.2612177。网址：https://doi.org/ 10.1145/2588555.2612177。Guu, K., Pasupat, P., Liu, E. Z., and Liang, P. 从语言到程序：连接强化学习与最大边际似然。载于计算语言学协会（*ACL*），2017年。He, P., Liu, X., Gao, J., and Chen, W. DeBERTa：采用解耦注意力的解码增强型 BERT，2020年。网址：https://arxiv.org/abs/2006.03654。Jiang, Z., Mao, Y., He, P., Neubig, G., and Chen, W. Omnitab：通过自然数据和合成数据预训练，实现少样本表格问答。载于*2022*年计算语言学协会北美分会人类语言技术会议论文集，第932–942页，2022年。Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S., El-Showk, S., Jones, A., Elhage, N., Hume, T., Chen, A., Bai, Y., Bowman, S., Fort, S., Ganguli, D., Hernandez, D., Jacobson, J., Kernion, J., Kravec, S., Lovitt, L., Ndousse, K., Olsson, C., Ringer, S., Amodei, D., Brown, T., Clark, J., Joseph, N., Mann, B., McCandlish, S., Olah, C., and Kaplan, J. 语言模型（大多）知道自己知道什么，2022年。网址：https://arxiv.org/abs/2207.05221。

Lai, Y.、Li, C.、Wang, Y.、Zhang, T.、Zhong, R.、Zettlemoyer, L.、Yih, S.W.-t.、Fried, D.、Wang, S. 和 Yu, T. Ds-1000：一个用于数据科学代码生成的自然且可靠的基准测试。*arXiv* 预印本 *arXiv:2211.11501*，2022年。

Lee, A.、Auli, M. 和 Ranzato, M. 神经机器翻译的判别式重排序。载于 计算语言学协会第*59*届年会暨自然语言处理国际联合会议第*11*届会议论文集（第*1*卷：长篇论文），第7250–7264页，2021年。

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D. 等。使用 AlphaCode 的竞赛级代码生成。*arXiv* 预印本 *arXiv:2203.07814*，2022年a。

Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G. 和 Chen, W. 关于提升语言模型作为推理器能力的进展。*arXiv* 预印本 *arXiv:2206.02336*，2022年b。

Lin, X. V.、Wang, C.、Pang, D.、Vu, K.、Zettlemoyer, L. 和 Ernst, M. D. 使用循环神经网络从自然语言进行程序合成。技术报告 UW-CSE-17-03-01，华盛顿大学计算机科学与工程系，美国西雅图，2017年3月。Lin, X. V.、Wang, C.、Zettlemoyer, L. 和 Ernst, M. D. Nl2bash：用于Linux操作系统自然语言接口的语料库和语义解析器。载于 第十一届语言资源与评估国际会议论文集 *LREC 2018*，日本宫崎，*2018*年*5*月*7*日至*12*日。，2018年。Liu, Q.、Chen, B.、Guo, J.、Ziyadi, M.、Lin, Z.、Chen, W. 和 Lou, J.-G. TaPEX：通过学习神经SQL执行器进行表格预训练。载于 国际学习表征会议，2021年。Liu, Y.、Ott, M.、Goyal, N.、Du, J.、Joshi, M.、Chen, D.、Levy, O.、Lewis, M.、Zettlemoyer, L. 和 Stoyanov, V. RoBERTa：一种鲁棒优化的 BERT 预训练方法。*CoRR*，abs/1907.11692，2019年。网址：http://arxiv.org/abs/1907.11692。Ni, A.、Inala, J. P.、Wang, C.、Polozov, O.、Meek, C.、Radev, D. 和 Gao, J. 从自采样的正确程序和部分正确程序中学习。*arXiv* 预印本 *arXiv:2205.14318*，2022年。Nijkamp, E.、Pang, B.、Hayashi, H.、Tu, L.、Wang, H.、Zhou, Y.、Savarese, S. 和 Xiong, C. 一种用于程序合成的对话范式。*arXiv* 预印本 *arXiv:2203.13474*，2022年。Olabiyi, O.、Salimov, A.、Khazane, A. 和 Mueller, E. T. 对抗学习框架中的多轮对话响应生成。*CoRR*，abs/1805.11752，2018年。网址：http://arxiv.org/abs/1805.11752。

OpenAI。ChatGPT：优化用于对话的语言模型，2022年11月。网址：https://openai. com/blog/chatgpt/。

Pasupat, P. 和 Liang, P. 半结构化表格上的组合语义解析。载于计算语言学协会第*53*届年会暨第*7*届国际自然语言处理联合会议论文集（第*1*卷：长篇论文），第1470–1480页，中国北京，2015年7月。计算语言学协会。doi: 10.3115/v1/P15-1142。URL https://aclanthology.org/P15-1142。

Qi, J., Tang, J., He, Z., Wan, X., Zhou, C., Wang, X., Zhang, Q. 和 Lin, Z. RASAT：将关系结构集成到用于文本到 SQL 的预训练序列到序列模型中。*arXiv* 预印本 *arXiv:2205.06983*，2022年。

Rabinovich, M., Stern, M. 和 Klein, D. 用于代码生成和语义解析的抽象语法网络。载于计算语言学协会第*55*届年会论文集（第*1*卷：长篇论文），第1139–1149页，加拿大温哥华，2017年7月。计算语言学协会。doi: 10.18653/v1/P17-1105。URL https: // aclanthology.org/P17-1105。

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. 和 Liu, P.J. 探索统一文本到文本迁移模型的迁移学习极限。机器学习研究杂志，21:140:1–140:67，2020年。网址：http://jmlr.org/papers/v21/20-074.html。

Rajkumar, N., Li, R. 和 Bahdanau, D. 评估大语言模型的文本到 SQL 能力。*arXiv* 预印本 *arXiv:2204.00498*，2022年。

Scholak, T.、Schucher, N. 和 Bahdanau, D. PICARD：对语言模型的受约束自回归解码进行增量解析。载于*2021*年自然语言处理实证方法会议论文集，第9895–9901页，2021年。

Seligman, E.、Schubert, T. 和 Kumar, M. V. A. K. 形式化验证：现代 *VLSI* 设计的必备工具包。摩根·考夫曼出版社，旧金山，加利福尼亚州，美国，2015年。ISBN 9780128008157。

Shen, J., Yin, Y., Li, L., Shang, L., Jiang, X., Zhang, M., 和 Liu, Q. 生成与排序：用于数学应用题的多任务框架。载于计算语言学协会发现：*EMNLP 2021*，第2269–2279 页，蓬塔卡纳，多米尼加共和国，2021年11月。计算语言学协会。doi: 10.18653/v1/2021.findings-emnlp. 195。网址：https://aclanthology.org/2021. findings-emnlp.195。

Shen, L., Sarkar, A., 和 Och, F. J. 面向机器翻译的判别式重排序。载于北美分会人类语言技术会议论文集：计算语言学协会 *HLT-NAACL 2004*，第 177–184 页，2004 年。

Shi, F.、Fried, D.、Ghazvininejad, M.、Zettlemoyer, L. 和 Wang, S. I. 利用执行进行自然语言到代码翻译。*arXiv* 预印本 *arXiv:2204.11454*，2022年。

Shi, T.、Zhao, C.、Boyd-Graber, J.、Daum´e III, H. 和 Lee, L. 论词汇逻辑对齐在面向 SQL 查询的语义解析中的潜力。载于计算语言学协会发现：*EMNLP 2020*，第1849–1864 页，线上，2020年11月。计算语言学协会。doi：10.18653/v1/2020.findings-emnlp. 167。网址：https://aclanthology.org/2020. findings-emnlp.167。

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L. 和 Fox, D. Alfred：用于解读日常任务具身指令的基准测试。载于*2020*年 *IEEE/CVF* 计算机视觉与模式识别会议（*CVPR*），第 10737–10746 页。IEEE，2020年。

Templeton, M. 和 Burger, J. D. DSMS 自然语言接口中的问题：以 EU-FID 为例。载于第 *1* 届应用自然语言处理会议（*ANLP 1983*），米拉玛喜来登酒店，美国加利福尼亚州圣莫尼卡，*1983*年*2*月*1*日至*3*日，第 3–16 页。ACL，1983年。doi: 10.3115/974194.974197。URL https://aclanthology.org/A83-1002/。

Tufano, M., Drain, D., Svyatkovskiy, A., Deng, S. K. 和 Sundaresan, N. 使用 Transformer 生成单元测试用例。*CoRR*，abs/2009.05617，2020年。网址：https: // arxiv.org/abs/2009.05617。

Wan, X., Cao, Z., Wei, F., Li, S. 和 Zhou, M. 通过判别式摘要重排序进行多文档摘要生成。*CoRR*，abs/1507.02062，2015年。网址：http: //arxiv.org/abs/1507.02062。

Wang, B., Shin, R., Liu, X., Polozov, O. 和 Richardson, M. RAT-SQL：用于文本到 SQL 解析器的关系感知模式编码与链接。载于第*58*届计算语言学协会年会论文集，第7567–7578页，2020年。

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang, P.-S., Mao, Y., Polozov, O., 和 Singh, R. 通过执行引导解码实现鲁棒的文本到 SQL 生成，2018年。网址：https://arxiv.org/abs/1807.03100。

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., 和 Zhou, D. 自洽性提升了语言模型中的思维链推理。*arXiv*预印本 *arXiv:2203.11171*，2022年。

Wang, Y.、Wang, W.、Joty, S. 和 Hoi, S. C. CodeT5：面向标识符的代码理解与生成统一预训练编码器-解码器模型。载于*2021*年自然语言处理实证方法会议论文集，第 8696–8708 页，线上会议，蓬塔卡纳，多米尼加共和国，2021年11月。计算语言学协会。doi：10.18653/v1/2021.emnlp-main.685。网址：https:// aclanthology.org/2021.emnlp-main.685。

Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., 和 Le, Q. V. 微调后的语言模型是零样本学习器。载于国际学习表征会议，2021年。Welleck, S., Lu, X., West, P., Brahman, F., Shen, T., Khashabi, D., 和 Choi, Y. 通过学习自我纠正来生成序列。*arXiv*预印本 *arXiv:2211.00053*，2022年。Woods, W. A. 自然语言理解的进展：在月球地质学中的应用。载于美国信息处理协会联合会：*1973*年全国计算机会议，*1973*年*6*月*4*日至*8*日，美国纽约州纽约市，收录于*AFIPS*会议论文集第42卷，第441–450页。AFIPS出版社/ACM，1973年。doi：10. 1145/1499586.1499695。URL https://doi.org/10.1145/1499586.1499695。Xiao, C., Dymetman, M., 和 Gardent, C. 用于语义解析的基于序列的结构化预测。载于第*54*届计算语言学协会年会论文集（第*1*卷：长篇论文），第1341–1350页，德国柏林，2016年8月。计算语言学协会。doi：10.18653/v1/P16-1127。URL https://aclanthology.org/P16-1127。Xie, T., Wu, C. H., Shi, P., Zhong, R., Scholak, T., Yasunaga, M., Wu, C.-S., Zhong, M., Yin, P., Wang, S. I., 等。UnifiedSKG：利用文本到文本语言模型统一结构化知识落地并实现多任务。*arXiv*预印本 *arXiv:2201.05966*，2022年。Yin, P. 和 Neubig, G. 用于神经语义解析的重排序。载于第*57*届计算语言学协会年会论文集，第4553–4559页，意大利佛罗伦萨，2019年7月。计算语言学协会。doi：10.18653/v1/P19-1447。URL https://aclanthology.org/P19-1447。

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., 和 Radev, D. Spider：用于复杂、跨领域语义解析和文本到 SQL 任务的大规模人工标注数据集。载于*2018* 年自然语言处理经验方法会议论文集，第3911–3921页，比利时布鲁塞尔，2018年10月至11月。计算语言学协会。doi：10.18653/v1/D18-1425。URL https: //aclanthology.org/D18-1425。

Zelle, J. M. 和 Mooney, R. J. 使用归纳逻辑编程学习解析数据库查询。载于 Clancey, W. J. 和 Weld, D. S.（编），第十三届全国人工智能会议暨第八届人工智能创新应用会议论文集，*AAAI 96*，*IAAI 96*，美国俄勒冈州波特兰，*1996*年*8*月*4*日至*8*日，第 *2* 卷，第 1050–1055 页。AAAI 出版社 / 麻省理工学院出版社，1996年。网址 http://www.aaai.org/Library/AAAI/1996/aaai96-156.php。

Zettlemoyer, L. S. 和 Collins, M. 将句子映射为逻辑形式的学习：使用概率范畴语法进行结构化分类。载于*UAI '05*，《人工智能中的不确定性》第 *21* 届会议论文集，爱丁堡，苏格兰，*2005*年*7*月*26–29*日，第 658–666 页。AUAI 出版社，2005年。网址：https://dslpitt.org/uai/displayArticleDetails. jsp?mmnu=1&smnu=2&article_id=1209& proceeding_id=21。

Zhang, T., Yu, T., Hashimoto, T. B., Lewis, M., Yih, W.-t., Fried, D. 和 Wang, S. I. 面向代码生成的 Coder 审阅者重排序。*arXiv* 预印本 *arXiv:2211.16490*，2022年。

Zhong, V., Xiong, C. 和 Socher, R. Seq2SQL：使用强化学习从自然语言生成结构化查询。*CoRR*，abs/1709.00103，2017年。网址：http://arxiv.org/abs/1709.00103。

Zhou, F., Hu, M., Dong, H., Cheng, Z., Han, S., 和 Zhang, D. TaCube：预计算数据立方体，以回答有关表格数据的数值推理问题。*arXiv* 预印本 *arXiv:2205.12682*，2022年。

Zhou, S.、Yin, P. 和 Neubig, G. 通过自然语言对情境化智能体进行分层控制。*arXiv* 预印本 *arXiv:2109.08214*，2021年。

A. 其他实现细节

	Spider	WTQ	GSM8k	MBPP
验证设置				
输入格式	NL+ SQL+ 执行	NL+ SQL+ 执行	NL+ 程序+ 执行	NL+ 程序+ 执行
规范化	No	No	Yes	Yes
生成概率				
批次大小*	8	8	8	8
下采样†	20	5	5	4
正标签比例				
Codex	68.0%	47.6%	61.1%	56.6%
InCoder	9.2%	-	2.2%	-
CodeGen	8.6%	-	4.9%	-
唯一程序占比				
Codex	30.6%	30.1%	90.3%	93.8%
InCoder	94.2%	-	99.2%	-
CodeGen	94.5%	-	99.3%	-

表 7：用于训练验证器的详细数据集特定设置和统计信息。 *：训练期间每个批次的样本数。 †：批次中每个样本对应的程序样本数。所有百分比均以每个样本所抽取的程序为总体，并且均在训练集上测得。

代码大语言模型采样。 我们按照表 7 中所述的输入格式和采样超参数，使用温度采样来生成程序候选项。参考原论文（Fried 等，2022；Nijkamp 等，2022）中使 PASS@k 达到最佳效果的最优温度，我们将 Codex 的温度设为 $T = 0.6$，将 In-Coder 和 CodeGen 的温度设为 $T = 0.8$。关于采样预算的消融研究见 §5.1。

少样本提示构建。 表1展示了不同数据集的提示中包含的少样本示例数量。除 MBPP 外，所有示例均从训练集中随机抽取并排序；对于 MBPP，我们使用原始数据集提供的 3 个示例。各数据集使用的完整提示见附录 C。

特定数据集设置。 特定数据集的详细实验设置见表 7。具体而言，我们对 GSM8k 和 MBPP 数据集使用归一化概率，因为我们发现这两个数据集能够从这种规范化中受益。我们认为，这是因为 Python 程序由于语法灵活且表达能力更强，长度变化更大。此外，正标签所占的比例也代表了“随机”基线，即从

目标大语言模型 & 机器学习+执行程序 基线		源大语言模型（% 正标签） Codex (64.0%)	InCoder (9.2%)	CodeGen (8.6%)
Codex	77.3	82.0 (+4.7)	81.7 (+4.4)	80.8 (+3.5)
InCoder	41.2	46.4 (+5.2)	54.1 (+12.9)	47.6 (+6.4)
CodeGen	37.7	44.7 (+7.0)	48.9 (+11.2)	51.0 (+13.3)

（a）Spider 上代码大语言模型之间的迁移结果。

目标大语言模型 & 机器学习+执行程序 基线		源大语言模型（% 正标签） Codex (53.4%)	InCoder (2.3%)	CodeGen (5.0%)
Codex	72.1	83.7 (+11.6)	70.0 (-2.1)	71.9 (-0.2)
InCoder	4.3	8.3（+4.0）	11.9（+7.6）	12.3（+8.0）
CodeGen	9.6	18.4（+8.8）	20.7 (+11.1)	22.1 (+12.5)

(b) 代码大语言模型在 GSM8k 上的迁移结果。

表 8：在源代码大语言模型采样的程序上训练验证器，并将其应用于目标代码大语言模型时的执行准确率。每行中最佳和 次佳的表现分别以相应方式突出显示。

采样得到的程序。这进一步展现了代码大语言模型的能力，以及为它们学习验证器的难度。

验证器输入示例 这里我们展示了表 11 中不同数据集对应的验证器输入示例。

B. 其他结果

B.1. 代码大语言模型之间的迁移学习

避免从代码大语言模型中采样所需成本的另一种方法，是使用来自一个大语言模型的样本训练验证器，然后将其直接应用于从另一个大语言模型采样得到的程序，即，在大语言模型间迁移。我们在表 8 中展示了这种方法在 Spider 和 GSM8k 上的结果。从结果中，我们首先可以观察到，LEVER 大多数情况下仍能显著提升基线性能，但在 GSM8k 数据集上，从 InCoder 和 CodeGen 迁移到 Codex 时例外。这表明验证器学到的知识可以泛化到不同大语言模型的输出。此外，我们可以看到，正标签的比例越接近，迁移通常效果越好；与 Codex 相比，InCoder 和 CodeGen 模型之间的迁移更为成功。这些结果表明，大语言模型间迁移是一种颇有意义的方法，可以减少 LEVER 所需的训练数据。

B.2. 基础语言模型验证消融实验

在本研究中，我们将验证器所用基础模型的选择视为一个超参数，并选用表现最佳的模型开展后续实验。这里展示了我们在四个数据集上尝试过的所有基础模型的性能，结果见

基础语言模型	Spider	WTQ	GSM8K	MBPP
T5-base	**82.0**	64.8	82.4	76.8
T5-large	81.9	**65.0**	82.5	**77.3**
T5-3B	83.1	64.7	**84.4**	-
RoBERTa-large	-	64.3	**84.4**	-

表9：验证器使用不同基础模型的消融实验。-：该数据集未测试基础语言模型。

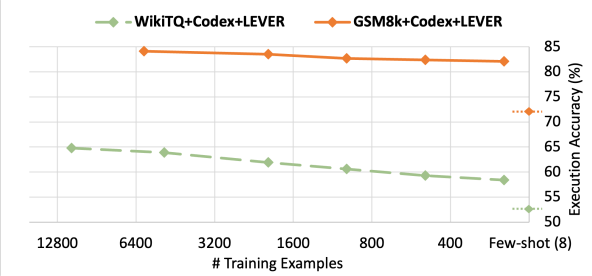


图7：Codex+LEVER 在 WTQ 和 GSM8k 数据集上针对训练样例数量的消融实验。 y轴上的数据标记表示 ML+EP 的性能，作为基线。LEVER 使用 T5-base。

表9中的结果。

B.3. WTQ 和 GSM8k 的训练样例规模扩展

受篇幅限制，我们只能展示 Spider 上训练样例数量的消融实验。在图7中，我们还展示了 WikiTQ 和 GSM8k 上的结果。从结果可以看出，LEVER 在这两个基准测试上的学习同样具有很高的数据效率：即使仅提供 250 个训练样例，也能观察到明显的改进。

B.4. 使用官方评估器的 WikiTQ 结果

按照 Cheng等人（2022年）的方法，我们通过规范化单位、布尔值等来修正 WikiTQ 的官方评估器。这里，我们还在表10中报告了 LEVER 与基于官方 WikiTQ 评估器的先前工作相比的表现。从结果可以看出，在这种设置下，LEVER 仍取得了最先进结果。

B.5. 案例研究

这里我们通过表12中的一些具体示例，说明 LEVER 的工作方式及其失效的情形。在来自 Spider 数据集的第一个示例中，我们可以看到，程序候选项 $\hat{}y_2$ 从错误的表中进行查询，从而导致执行错误。验证器很容易检测到这一点，因此会为此类程序赋予较低的验证概率。同时，程序 $\hat{}y_1$ 的执行结果 $\hat{}z_1$ 似乎更有可能是问题的答案，因而

方法	Dev	Test
无微调的先前工作		
Codex 问答‡ (Cheng 等，2022)	49.3	47.6
Codex SQL‡ (Cheng 等，2022)	57.6	55.1
Codex Binder‡ (Cheng 等，2022)	**62.6**	61.9
以微调为基础的先前工作		
TaPEX* (Liu 等，2021年)	57.0	57.5
TaCube (Zhou 等，2022年)	59.7	59.6
OmniTab (Jiang 等，2022年)	-	**62.8**
本研究使用 *code-davinci-002*		
贪心	47.2	50.9
ML	48.3	50.9
执行程序 + 机器学习	50.1	52.5
执行程序 + 投票	50.6	53.6
LEVER	$61.1_{\pm 0.2}$	$\mathbf{62.9}_{\pm 0.2}$
预言机	70.9	74.6

表10：使用官方 WTQ 执行器在 WTQ 数据集上的执行准确率。 ‡：官方执行器中添加了一个用于识别日期的规范化器。

更容易被验证器接受。然而，在来自 WikiTQ 的第二个示例中，执行结果 $\hat{}z_1$ 和 $\hat{}z_2$ 并未提供明确的信息，因为它们都是县名。在这种情况下，验证器所掌握的有用信息并不比生成器多，因此无法识别出错误的程序。

C. 少样本生成的提示词

最后，我们附上用于对代码大语言模型进行少样本提示的完整提示词：Spider（表13、表14、表15）、WikiTQ（表16、表17）、GSM8k（表18、表19）和 MBPP（表20）。

SPIDER/WIKITQ：问题 + SQL + 线

输入：-- 问题：列出按年龄排序的系主任的姓名、出生州和年龄。 |- SQL： |select name, bornstate, age from head join management on head.head id =management.headid order by age |- 执行结果： |/*| 姓名 出生州 年龄 | Dudley Hart 加利福尼亚州 52。0 | Jeff Maggert 特拉华州 53。0 |Franklin Langham 康涅狄格州 67。0 |Billy Mayfair 加利福尼亚州 69。0 |K. J. Choi 阿拉巴马州 69。0 |*/Q

utput: no

GSM8K：问题 + 惯用程序 + 答案变量

输入:
卡莉最近毕业了，正在寻找与她所学专业相关的工作。她向本州的公司投递了200份求职申请，向外州公司的投递数量是其两倍。计算一下，到目前为止她一共投递了多少份求职申请。|

```
n_j求职申请 _i州内状态 = 200
州外求职申请 = 州内求职申请 * _2
答案 = 州内求职申请 + 州外求职申请 -|
‘答案’: 600
```

输出：是

MBPP：任务描述 + 函数 + 返回类

输入:
描述
编写一个函数，使用 lambda 函数求列表中各个元素的 n 次幂。

```
# 程序 def nth nums(nums,n): 结果列表 = list(map(lambda x:
x ** n, nums)) 返回 (resultlist)

# 执行
# 返回: (列表) =[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# 返回: (列表) =[1000, 8000, 27000]
# 返回: (列表) =[248832, 759375]
```

输出：是

表 11：示例 数据集上的验证器输入。为便于展示，手动插入了一些换行符 .

SPIDER 示例

问题 x：找出每位球员的排名积分总数及其名字。
程序 $\hat{}y_1$(正确)：
选择名字、排名积分总数，来自 球员连接排名，连接条件为 球员标识符 = 排名.球员标识符按名字分组
程序 $\hat{}y_2$(错误)：
选择名字，并对 排名与球员表进行连接，条件是 rankings = players.player id 按 player id 分组
执行信息 $\hat{}z_1$：
Aastha | 68; Abbi | 304; Abbie | ...
执行信息 $\hat{}z_2$：
错误：未找到名为 ... 的列

WIKITQ 示例

问题 x：按家庭收入中位数从高到低对各县排名时，排名第一的会是哪个县?
程序 $\hat{}y_1$(不正确)：
按家庭收入中位数从高到低对主表中的县进行排序 限制 1 条
程序 $\hat{}y_2$(正确)：
按家庭收入中位数对主表中的县进行排序 降序排列，限制 1 条
执行信息 $\hat{}z_1$：县 | jefferson
执行信息 $\hat{}z_2$：县 | sanders

表12：WikiTQ 和 Spider 数据集的案例研究。在两个示例中，程序 $\hat{}y_1$ **的排名高于** 程序 $\hat{}y_2$。导致错误的 SQL 程序中的主要差异已突出显示。

```
-- 将自然语言问题转换为 SQL 查询。

-- 示例ample:

-- 数据库比赛_伤病:
-- 表 体育场：标识符、名称、主场_比赛、平均_观众人数、总_观众人数、容量_百分比
-- 表 比赛：体育场_标识符、标识符、赛季、日期、主场_球队、客场_球队、比分、赛事
-- 表 伤病_事故：比赛_标识符、标识符、球员、伤病、比赛_场数、来源
-- 问题：2010 赛季之后发生了多少种不同种类的伤病?
-- SQL:
SELECT count(DISTINCT T1.伤病) FROM 伤病_事故 AS T1 JOIN 比赛 AS T2 ON T1.比赛_标识符 = T2.标识符 WHERE T2.
    赛季 > 2010

-- Ex ample:

-- 农场数据库:
-- 城市表：城市_标识符、官方_名称、状态、面积_平方公里_2、人口、人口普查_排名
-- 农场表：农场_标识符, 年份, 马匹_总数, 役马_, 牛类_总数、阉牛、公牛、母牛、猪,
    绵羊_和_山羊
-- 表 农场表_赛事：赛事_标识符、年份、主题、主办方_城市_标识符、主办方
-- 表 赛事_记录：赛事_标识符、农场_标识符、排名
-- 问题：返回主题不是外星人的赛事主办方?
-- SQL:
SELECT 主办方 FROM 农场_赛事 WHERE 主题 != '外星人'
```

表13：我们用于 Spider 数据集的提示，旨在通过代码大语言模型进行少样本生成。此处仅展示前2个示例，这也是 InCoder/CodeGendue 受模型长度和计算资源限制而使用的全部示例。Codex 共使用8个示例，其余示例见表14和表15。

```
-- 示例ample:

-- 数据库 学校                 _金融:
-- 表 学校: 学校                 _id, 学校    _name, 地点      , 吉祥物 , 在校人数     , IHSAA _班级  , IHSAA _美式足球 _班级  ,
        县
-- 表 预算金额: 学校_标识符、年份、预算金额、总数_预算_百分比_预算金额、投资金额,
        总数_预算_百分比_投资金额、预算_投资金额_百分比
-- 表 捐赠基金: 捐赠基金                 _id, 学校    _id, 捐赠者   _name, 金额
-- 问题: 显示所有学校的平均在校人数、最大在校人数和最小在校人数。
-- SQL:
SELECT avg(在校人数) , max(在校人数) , min(在校人数) FROM 学校

-- 示例:

-- 数据库创建_文档_和_执行程序:
-- 表引用_文档_类型: 文档_类型_代码、文档_类型_名称、文档_类型_描述
-- 表引用_预算_代码: 预算_类型_代码、预算_类型_描述
-- 表项目: 项目_标识符、项目_详情
-- 表文档: 文档_标识符、文档_类型_代码、项目_标识符、文档_日期、文档_名称、
        文档_描述, 其他_详情
-- 表 报表: 报表                         _ID, 报表        _详情
-- 表 费用文档: 文档_标识符, 预算_类型_代码, 文档_详情
-- 表 账户: 账户                     _ID, 报表        _ID, 账户      _详情
-- 问题: 返回文档数量超过两个的项目所对应的标识符和详情。
-- SQL:
SELECT T1.Project_标识符 , T1.Project_详情 FROM 项目 AS T1 JOIN 文档 AS T2 ON T1.Project_标识符  =  T2.
        项目_标识符 按 T1.项目_标识符 分组, 满足计数(*)  > 2

-- 示例ample:

-- 数据库 local_govt_in_alabama:
-- 服务表: 服务_标识符、服务_类型_代码
-- 参与者表: 参与者_标识符、参与者_类型_代码、参与者_详情
-- 事件表: 事件                 _ID, 服务      _ID, 事件   _详情
-- 参与者表_在_事件中: 事件_标识符, 参与者_标识符
-- 问题: 按字母顺序列出服务的类型。
-- SQL:
SELECT Service_Type_代码 FROM Services ORDER BY Service_Type_代码

-- Ex  ample:

-- 数据库 cre_主题_公园:
-- 表 Ref_Hotel_Star_Ratings: star_rating_code, star_rating_description
-- 地点表: 地点                     _ID, 地点       _Name, 地址      , 其他   _详情
-- 表 Ref_景点_类型: 景点_类型_代码, 景点_类型_描述
-- 表 游客: 游客                     _ID, 游客      _详情
-- 特色设施表: 特色                 _ID, 特色      _详情
-- 酒店表: 酒店_标识符、星级_评分_代码、宠物_是否允许_、价格_范围、其他_酒店_详情
-- 游客_景点: 游客_景点_标识符、景点_类型_代码、地点_标识符、如何_前往___、名称
        , 描述, 营业_小时, 其他_详情
-- 街头市场表         _: 市场                 _ID, 市场     _详情
-- 商店表: 商店_标识符, 商店_详情
-- 表 博物馆: 博物馆                 _ID, 博物馆  _详情
-- 表 Royal_Family: Royal_Family_标识符, Royal_Family_详情
-- 表 主题            _公园: 主题      _Park_ID, 主题   _Park_详情
-- 访问表: 访问                 _ID, 游客      _景点          _ID, 游客      _ID, 访问   _Date, 访问   _详情
-- 照片表: 照片_标识符, 游客_景点_标识符, 名称, 描述, 文件名, 其他_详情
-- 员工表: 员工                 _ID, 游客      _景点          _ID, Name, 其他   _详情
-- 表 游客                 _景点          _特征: 游客                 _景点          _ID, 特色      _ID
-- 问题: 展示获得五星级评分且允许携带宠物的酒店的平均价格范围。
-- SQL:
SELECT avg(price_range) FROM 酒店 WHERE star_rating_code  =  "5" AND pets_allowed_yn  = 1
```

表14：我们用于 Spider 数据集少样本生成的代码大语言模型提示（第2部分），续表13。

```
-- Ex  ample:

-- 保险数据库                  _首次损失通知:
-- 客户表：客户                        _ID, 客户      _name
-- 服务表：服务                    _ID, 服务     _name
-- 可用_保单表：保单_标识符、保单_类型_代码、客户_电话
-- 客户_保单表：客户_标识符、保单_标识符、_开立日期、_结束日期
-- 首次_损失_通知_表：首次损失通知_标识符、客户_标识符、保单_标识符、服务_标识符
-- 索赔表：索赔              _ID, FNOL_ID, 生效       _Date
-- 结算表：结算                        _ID, 索赔  _ID, 生效       _Date, 结算        –金额
-- 问题：查找所有电话号码。
-- SQL:
SELECT 客户_电话 FROM 可用_保单

-- 示例ample:

-- 数据库 cre_主题_公园:
-- 表 Ref_Hotel_Star_Ratings：star_rating_code、star_rating_description
-- 地点表：地点                        _ID, 地点      _Name, 地址     ’ 其他  –详情
-- 表 Ref_景点_类型：景点_类型_代码，景点_类型_描述
-- 游客表：游客                    _ID, 游客     –详情
-- 特色设施表：特色                _ID, 特色     –详情
-- 酒店表：酒店_标识符，星级_评分_代码，是否_允许_携带宠物，价格_范围，其他_酒店_详情
-- 表 游客_景点：游客_景点_标识符，景点_类型_代码，地点_标识符，如何_前往_那里_，名称
      ，描述，开放_小时，其他_详情
-- 表 街头            _市场：市场          _ID, 市场    –详情
-- 表 商店：商店_标识符、商店_详情
-- 表 博物馆：博物馆            _ID, 博物馆 –详情
-- 表 皇家_家族：皇家_家族_标识符、皇家_家族_详情
-- 表 主题          _公园：主题     _Park_ID, 主题  _Park_详情
-- 表 访问记录：访问          _ID, 游客     –景点          _ID, 游客     _ID, 访问  _Date, 访问  –详情
-- 照片表：照片_标识符，游客_景点_标识符，名称，描述，文件名，其他_详情
-- 员工表：员工              _ID, 游客     _景点          _ID, Name, 其他  –详情
-- 表 游客              _景点          _特征：游客              –景点         _ID, 特色     _ID
-- 问题：游客前往旅游景点最常使用哪种交通方式?
-- SQL:
SELECT 如何_前往_那里_ FROM 游客_景点 GROUP BY 如何_前往_那里_ ORDER BY 计数(*) DESC LIMIT 1
```

表15：我们用于 Spider 数据集的代码大语言模型少样本生成提示（第3部分），接续表13和表14。

-- 将自然语言问题转换为 SQL 查询。 -- 示例： -- 数据库 204 126:_ -- 表 main_table：标识符 (1), 聚合 (0), 名次 (t1), 名次_number (1.0), 球员 (拉里·纳尔逊), 国家 (美国), 比分 (70-72-73-72=287), 比分_result (287), 比分_number (70), 比分_number1 (70), 比分_number2 (72), 比分_number3 (73), 比分_number4 (72), to_par (-1), to_par_number (-1.0), 奖金_lrb_rrb (延长赛), 奖金_lrb_rrb_number (58750.0) -- 问题：第一名与标准杆相差多少？ -- SQL: select to_par from main_table where place_number = 1-- 示例： -- 数据库 204 522:_ -- 表 main_table：标识符 (1), 聚合 (0), boat_count (4911), boat_count_number (4911), boat_count_minimum (4951), boat_count_maximum (4955), 名称 (ha-201), 建造者 (佐世保海军工厂), laid_down (01-03-1945), laid_down_number (1), laid_down_parsed (1945-01-03), laid_down_year (1945), laid_down_month (1), laid_down_day (3), 下水 (23-04-1945), launched_number (23), launched_parsed (1945-04-23), launched_year (1945), launched_month (4), launched_day (23), 完工 (31-05-1945), completed_number (31), completed_parsed (1945-05-31), completed_year (1945), completed_month (5), completed_day (31), 结局 (退役于 30-11-1945。于 01-04-1946 在五岛列岛附近凿沉) -- 问题：哪艘船是在 ha-206 之前紧接着下水的？ -- SQL: select name from main_table where launched_parsed < (select launched_parsed from main_table where name = 'ha -206') order by launched_parsed desc limit 1 -- 示例： -- 数据库 204 877:_ -- 表 main_table：标识符 (1), 聚合 (0), 名次 (1), 名次_number (1.0), 位置 (mf), number (4), number_number (4.0), 名称 (ryan hall), 联赛_two (10), 联赛_two_number (10.0), fa_cup (1), fa_cup_number (1.0), league_cup (0), league_cup_number (0.0), fl_trophy (3), fl_trophy_number (3.0), 总数 (14), total_number (14.0) -- 问题：格兰特和贝尼恩谁得分更多？ -- SQL: select name from main_table where name in (' 安东尼 · 格兰特 ' , ' 埃利奥特 · 贝尼恩 ') order by total_number desc limit 1 -- 示例： -- 数据库 204_400: -- 表 main_table：标识符 (1), 聚合 (0), 选区 (1), 选区_number (1.0), 参议员 (肯尼斯·P·拉瓦勒), 政党 (共和党), 党团 (共和党), first_elected (1976), first_elected_number (1976), counties_represented (suffolk), counties_represented_length (1) -- 表 t_counties_represented_list：m_id (1), counties_represented_list (suffolk) -- 问题：2000 年后有多少共和党人当选？ -- SQL: select count (*) from main_table where party = 'republican' and first_elected_number > 2000

表16：我们用于 WTQ 数据集的代码大语言模型少样本生成提示（第1部分）。

```
-- 示例ample:

-- 数据库203 _208:
-- 表 main_table: 标识符 (1), 聚合 (0), 球队（明斯克迪纳摩）, 地点（明斯克）, 场馆（迪纳摩, 明斯克), 容量
(41040), 容量_数字 (41040.0), 位置_在_1993_94 (1), 位置_在_1993_94_数字 (1.0)
-- 表 t_场馆_地址: m_标识符 (1) , 场馆_地址 (迪纳摩)
-- 问题: 位于博布鲁伊斯克的球队数量是多少?
-- SQL:
select count ( 球队 ) from main_table where 地点 = '博布鲁伊斯克'

-- 示例:

-- 数据库203 _60:
-- 表 main_table: 标识符 (1), 聚合 (0), 结果（冠军）, 否（1）, 否_数字（1.0）, 日期（1981年7月20日）,
日期_数字（20）, 日期_已解析（1981-07-20）, 日期_年份（1981）, 日期_月份（7）, 日期_天（20）, 锦标赛
（巴斯塔德, 瑞典）, 场地类型（红土）, 决赛对手_于_该_决赛（Anders Jarryd）, 决赛比分_于_该_决赛（6-2, 6-3）,
决赛比分_于_该_决赛_时长（2）
-- 表 t_锦标赛_地址: m_标识符 (1), 锦标赛_地址 (巴斯塔德)
-- 表 t_决赛_比分_列表: m_标识符 (1) , 决赛_比分_列表 (6-2)
-- 表 t_决赛_比分_列表_第一项: m_标识符 (1) , 决赛_比分_列表_第一项 (6-2)
-- 表 t_决赛_比分_列表_第二项: m_标识符 (7) , 最终列表第二项的比分 (1-7)
-- 表 t_最终列表第一项的比分_数字: m_标识符 (1) , 比分_在_最终_列表_中_第_一个数字 (6)
-- 表 t_最终列表第一项的比分_数字1: m_标识符 (1) , 在最终列表中的比分_____第一个_数字1 (6)
-- 表 t_最终列表第一项的比分_数字2: m_标识符 (1) , 最终列表中第一个数字的比分2 (2)
-- 表 t 中最终列表第二个数字的比分: m 标识符 (7) , 最终列表中的第二个数字的比分 (1)
-- 表 t 中最终列表第二个数字1的比分: m 标识符 (7) , 最终列表中的第二个数字1的比分 (1)
-- 表 t 中最终列表第二个数字2的比分: m 标识符 (7) , 计分_在_该_最终_列表_第二_数字2 (7)
-- 问题: 锦标赛最多在哪个月份举行?
-- SQL:
select 日期_月份 from main_表 group by 日期_月份 order by count ( * ) desc limit 1

-- 示例ample:

-- 数据库203 _462:
-- 表 main_表: 标识符 (1), 聚合 (0), 年份 (2006), 年份_数字 (2006), 组别 (4), 组别_数字 (4.0),
联赛 (USL PDL), 常规_赛季 (第4, 心脏地带), 常规_赛季_长度 (2), 季后赛 (未获得参赛资格),
公开杯（未能晋级）
-- 表 t_常规_赛季_名单: m_标识符 (1), 常规_赛季_名单（第4名）
-- 问题: 哪一年表现更好, 2012年还是2007年?
-- SQL:
select year_number from main_table where year_number in ( 2012年 , 2007年 ) order by 常规_赛季 limit 1

-- 示例ample:

-- 数据库 204 _139:
-- 问题: 除了伊格尔农场机场, 还有其他类型为"军用/民用"的机场吗?
-- 表 main_表: 标识符 (1), 聚合 (0), 社区 (Antil Plains), 机场_名称 (Antil Plains Aerodrome), 类型 (
军用), 坐标（19 26'36"s 146 49'29"e/19.44333 s 146.82472 e）
-- SQL:
选择（选择计数（机场_名称）从主_表中, 其中类型 = '军用/民用' 且机场_名称 != '
伊格尔农场机场' ) > 0
```

表 17：我们用于 WTQ 数据集上代码大语言模型少样本生成的提示（第 2 部分）。

```
## Cristina、John、Clarissa 和 Sarah 想在母亲生日时送给她一本相册。Cristina
      带来 7 张照片，John 带来 10 张照片，Sarah 带来 9 张照片。如果相册有 40 个空位
      ，Clarissa 需要带来多少张照片才能填满相册?
n_照片_Cristina = 7
n_照片_John = 10
n_照片_Sarah = 9
n_照片_总数 = n_照片_Cristina + n_照片_John + n_照片_Sarah
n_槽位 = 40
n_槽位_剩余 = n_槽位 - n_照片_总数
答案 = n _槽位 _left

## Katy、Wendi 和 Carrie 去参加了一个烘焙派对。Katy 带了三袋 5 磅的面粉。Wendi
      带的面粉是 Katy 的两倍，但 Carrie 比 Wendi 带的面粉金额少 5 磅
      。Carrie 比 Katy 多带了多少盎司面粉?
磅_面粉_katy= 3 * 5
磅_面粉_wendi =磅_面粉_katy * 2
磅_面粉_carrie = 磅_面粉_wendi - 5
磅_差值_Carrie_Katy =磅_面粉_Carrie - 磅_面粉_Katy
盎司_差值_Carrie_Katy = 磅_差值_Carrie_Katy * 16
答案 = 盎司_差值_Carrie_Katy

## James 每 6 小时服用 2 片每片含 375 毫克的泰诺。他一天服用了多少毫克?
毫克_泰诺_每_片剂 = 375
毫克_泰诺_每次_服用_时间 = 2 * 毫克_泰诺_每_片剂
每天_小时_ = 24
每天_次_ =每天_小时_ / 6
每天_毫克_ = 每次服用_毫克泰诺_, 每天_次_ * _每天_次
答案 = 毫克_每天_

## Kyle烤了60块曲奇饼和32块布朗尼。Kyle吃了2块曲奇饼和2块布朗尼。Kyle的妈妈吃了1块曲奇饼和2
      块布朗尼。如果Kyle以1美元的价格卖一块曲奇饼，以1.50美元的价格卖一块布朗尼，那么如果他.....，Kyle能赚多少钱
      卖掉了他所有烘焙食品?
n_曲奇饼 = 60
n_布朗尼 = 32
n_曲奇饼_剩下_之后_Kyle = n_曲奇饼 - 2
n_布朗尼_剩下_之后_Kyle = n_布朗尼 - 2
n_曲奇饼_剩下_之后_Kyle_妈妈 =n_曲奇饼_剩下_之后_Kyle - 1
n_布朗尼_剩下_在_kyle_妈妈 = n_布朗尼_剩下_在_kyle - 2
奖金_赚得_kyle = n_曲奇饼_剩下_在_kyle_妈妈 * 1 + n_布朗尼_剩下_在_kyle_妈妈 * 1.5
答案 = 奖金_赚得_kyle

## 院子里有63个复活节彩蛋。Hannah找到的数量是Helen的两倍。Hannah找到了多少个复活节彩蛋
      ?
n_复活节_彩蛋 = 63
unit_ = 2次
总数_单位 = 单位 _ + 1次
_个复活节彩蛋_每_单位 = _个复活节彩蛋 / 总数_单位
_个复活节彩蛋_Helen = _个复活节彩蛋_每_单位 * 1
n_复活节_彩蛋_Hannah =n_复活节_彩蛋_每_单位 * 2
答案 = n_复活节_彩蛋_Hannah

## Ethan 正在读一本有 360 页的科幻小说。他在星期六早上读了 40 页，晚上又读了 10
      页。第二天，他读的页数是星期六总页数的两倍。他还剩多少页
      要读?
n_页 = 360
总共_页_星期六 = 40 + 10
总共_页_接下来_天 = 总共_页_星期六 * 2
总共_页_已读 = 总共_页_星期六 + 总共_页_接下来_天
还剩 n_页_ = n_页 - 总页数_已读页数_
答案 = 还剩 n_页_
```

表18：我们用于代码大语言模型少样本生成的 GSM8k 数据集提示（第1部分）。

```
## 一个图书馆有若干本书。其中 35% 是供儿童阅读的，104 本供成人阅读。图书馆里共有多少本书？ percent_本书_供_儿童阅读 = 0。35
percent_本书_供_成人阅读 = 1。0- percent_本书_供_儿童阅读 n 本书供成人阅读 = 104_ _ _n_本书_的_总数 = n_本书_供_成人
阅读 / percent_本书_供_成人阅读 答案 = n 本书总数_ _ _## Tyler 有 21 张 CD。他把三分之一的 CD 送给了朋友。然后他去音像店买了
8 张全新的 CD。Tyler 现在有多少张 CD？ n_张_Tyler 的 CD = 21 percent_张_送给_朋友的 CD = 1。0 /3.0 n_张_送出_后_剩下的_CD =n_
张_Tyler 的 CD - n_张_Tyler 的 CD * percent_张_送给_朋友的 CD n_张_新_购买的 CD = 8n_张_现在的 CD = n_张_送出_后_剩下的_CD +
n_张_新_购买的 CD 答案 = n 张 CD 现在_ _
```

表1 9：我们用于 GSM8k 数据集的代码大语言模型少样本生成提示（第 2 部分） .

```
# 编写 Python 函数以完成任务并通过断言测试集。

### 任务开始 ###
# 以下是针对你的函数的断言：
assert similar_elements((3 , 4, 5, 6),(5, 7, 4, 10)) == (4 , 5)

""" 编写一个函数，从给定的两个元组列表中找出相似元素。 """
def similar_elements(测试集_tup1, 测试集_tup2):
    res = tuple(set(test_tup1) & set(test_tup2))
    return (res)
### 任务结束 ###

### 任务开始 ###
# 以下是针对该函数的断言：
assert is_not_prime(2) == False

""" 编写一个 Python 函数，用于识别合数。 """
import math
def is_not_prime(n):
    结果 = False
    对于 i 在 range(2,int(数学.sqrt(n)) + 1):
        如果 n % i == 0:
            结果 = True
    返回结果
### 任务结束 ###

### 任务开始 ###
# 以下是针对你的函数的断言：
断言 heap_queue_largest( [25, 35, 22, 85, 14, 65, 75, 22, 58],3)==[85, 75, 65]

""" 使用堆队列算法从给定的数字列表中找出最大的整数。 """
导入 heapq 作为 hq
def heap_queue_largest(nums,n):
    最大_数字 = hq.nlargest(n, nums)
    return largest_nums
### 任务结束 ###
```

表 20：我们用于代码大语言模型在 MBPP 数据集上进行少样本生成的提示。