arXiv:2305.01598v2 [cs.DB] 3 May 2023

从文字到代码：利用数据从自然语言合成程序

Anirudh Khatry
微软
t-anikhatry@microsoft.com

Joyce Cahoon
微软
joyce.cahoon@microsoft.com

Jordan Henkel
微软
jordan.henkel@microsoft.com

Shaleen Deep
微软
shaleen.deep@microsoft.com

Venkatesh Emani
微软
venkatesh.emani@microsoft.com

Avrilia Floratou
微软
avrilia.floratou@microsoft.com

Sumit Gulwani
微软
sumitg@microsoft.com

Vu Le
微软
levu@microsoft.com

Mohammad Raza
微软
moraza@microsoft.com

Sherry Shi
微软
shersh@microsoft.com

Mukul Singh
微软
singhmukul@microsoft.com

Ashish Tiwari
微软
ashish.tiwari@microsoft.com

摘要

创建能够正确处理数据的程序是一项艰巨的任务，因为对于许多不擅长编程的用户而言，底层编程语言和 API 可能难以学习。大型语言模型（大语言模型）展现出了从自然语言生成代码的非凡潜力，但在数据处理领域，除了对预期任务的自然语言（自然语言）描述外，我们还拥有需要执行该任务的数据集，即 数据上下文。现有方法通过简单地将输入数据中的相关信息添加到发送给大语言模型的提示词中，以有限的方式利用了数据上下文。

在本研究中，我们利用可用的输入数据执行大语言模型生成的候选程序，并收集其输出。我们提出了语义重排序，这是一种根据来自程序输出的三种信号对大语言模型生成的程序进行重排序的技术：(a)语义过滤和格式正确性分数调优：程序是否生成格式正确的输出；(b)语义交错：不同候选项的输出如何相互比较；以及 (c) 基于输出的分数调优：这些输出与针对同一任务预测的输出相比如何。我们为语义交错提供了理论依据。我们还引入了温度混合，即结合使用高温度和低温度由大语言模型生成的样本。我们在三个领域广泛评估了该方法，即数据库（SQL）、数据科学（Pandas）和商业智能（Excel 的 Power Query M），并使用了各种新的和现有的基准测试。我们观察到各领域均取得了显著提升，其中 Top-1 准确率最高提升 45%，Top-3 准确率最高提升 34%。

1 引言

Codex [16]和 GPT [14, 43], 等大语言模型（LLM）的出现，从根本上改变了自然语言（NL）程序合成领域，推动自然语言接口（即 [1, 7]）和代码助手 [2] 的快速发展，并使其广泛应用于从业者和开发者。上下文学习 学习 [20] 在这一转变中发挥着关键作用，使大语言模型只需极少输入就能为多种编程语言生成代码。大语言模型的输入通常包括用自然语言描述待执行任务的提示，以及可能提供的少量示例，这些示例用于少样本提示 [12, 14]。借助上下文学习，大语言模型能够获取领域专属知识，从而更好地理解生成代码必须遵循的编程语言语法和结构。基于大语言模型的自然语言转代码方法尤其有价值，因为它无需针对特定编程领域（如 SQL、Python）训练和部署定制模型，从而使设计更加精简。

上下文学习的简洁性和强大能力使其成为合成数据操作程序的绝佳选择，即使用自然语言描述来提取、查询和转换数据的程序。一个使这一问题与众不同的关键方面是需要考虑数据上下文：除了对预期任务的自然语言描述外，我们还掌握了任务所需处理的数据集的上下文信息，例如数据模式和数据值。尽管以往的工作也利用了数据上下文，但主要局限于利用数据上下文来构造更好的提示词 [45, 46, 58]。这些提示词通常包含任务描述以及有关输入数据的信息，例如模式（列名）和样本行。

在本文中，我们进一步展示了如何以更深入的方式利用数据上下文，提升代码生成任务的性能。具体而言，我们发现，给定与某项任务关联的程序以及一份输入数据集样本，我们可以通过在该输入样本上执行程序来生成任务输出。这些任务输出（或者由于生成的代码格式错误等原因而无法构造这些输出的情况）提供了有价值的信息，可在代码生成过程中与任务输入结合使用，从而开辟丰富的新设计可能性。

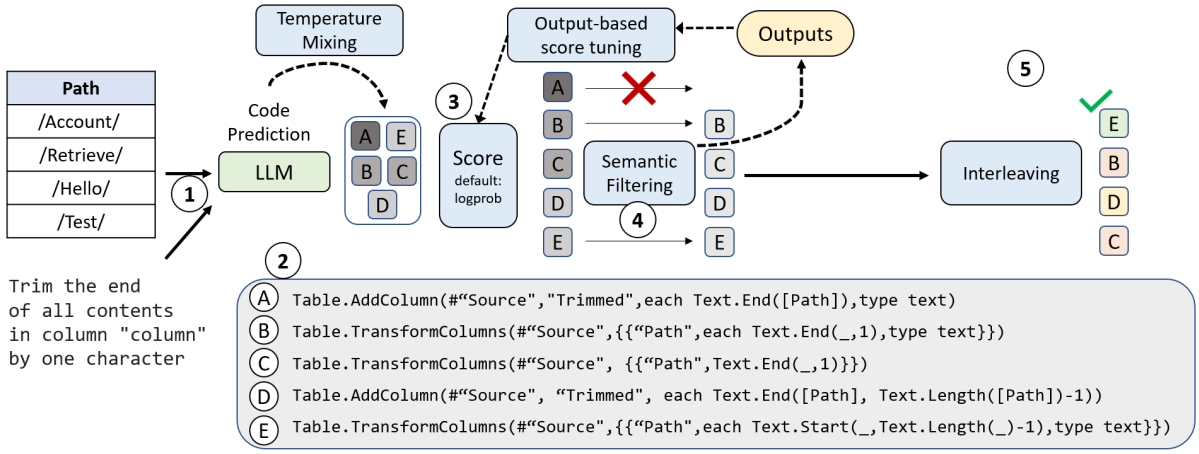


图 1：突出展示系统工作流程的架构图。（1）将数据和自然语言描述一并提供给大语言模型。（2）大语言模型生成 25 个样本，这些样本由使用不同温度生成的样本混合而成，其中五个样本分别以 A、B、C、D 和 E 表示。（3）根据分数对候选项进行排序，默认使用 平均对数概率。（4）执行候选项，并过滤掉无法执行的程序。（5）输出可用于选择性地调优评分函数，根据输出对候选项进行聚类，并将各簇交错排列以生成最终排序列表。虚线箭头表示可选组件。

图 1 展示了我们提出的、以更高级的方式利用数据上下文的架构。我们不再只是向大语言模型提供经过数据增强的提示并接受其首选结果，而是要求大语言模型给出排名前 N 的结果，以形成一组候选程序，并根据其对应语法标记的概率（对数概率 [1]）对这些候选程序进行排序。我们发现，尽管正确的程序可能包含在候选程序集合中，但其排名未必靠前。为了让正确的程序脱颖而出，我们采用一种名为 语义重排序 的新技术对候选程序重新排序，该技术包含两个步骤，即 语义过滤 和 语义交错。语义排序依赖于直接在输入数据样本上执行程序所获得的候选程序输出。如果代码执行失败并抛出异常，则会将该候选项从集合中移除（语义过滤）。完成语义过滤后，每个保留的候选项都有对应的输出。我们利用这些输出将程序分组，并以一种能够提高正确程序排名的可能性的方式重新排序，从而提升排名前 K^2 项的准确率（语义交错）。

语义重排序提升了排名前 K 的代码预测准确率，但我们观察到，对于较少见的目标语言，提升幅度较小。这可能是因为网络上这些语言的训练数据有限。在这种情况下，我们可以利用大语言模型的能力，根据自然语言任务描述和输入数据集，直接为给定任务生成输出数据集（输出预测）。然后，我们将预测输出与执行预测程序生成的输出进行比较，并利用比较结果调整分配给该程序的对数概率分数。这种基于输出的分数调优 能为语义重排序器提供更好的评分函数，从而进一步提升排名前 K 的结果。我们还引入了温度混合，以克服使用大语言模型时温度过低或过高所带来的权衡问题。

我们的贡献。本文提出了一种新颖的方法来合成数据操作程序，以前所未有的方式利用数据上下文，生成更准确的程序。我们的贡献如下。

- 我们为数据处理领域提出了一种新颖的程序合成框架。该框架通过利用任务输入和输出，端到端地利用数据上下文。我们方法的核心是一种新的语义排序技术，它根据程序的输出，将程序的语义多样性纳入考量。

- 我们提出了一种新颖的技术，用于应对大语言模型缺乏特定数据操作语言相关先验知识的情形。该技术利用相关大语言模型任务（即输出预测）中的数据上下文，并将结果作为代码生成任务的输入，从而进一步提高准确率。

- 我们通过多种全新和现有的基准测试，展示了我们的方法在三个领域中的适用性：数据库（SQL）、数据科学（Pandas）和商业智能（Excel 的 Power Query M [4]）。SQL 和 Pandas 是极受欢迎的数据检索与处理语言，多个供应商都在使用；而微软的多款产品则使用 M 来支持商业分析。与 SQL 和 Pandas 相比，大语言模型可能见过的 M 程序示例要少得多。这使我们能够针对模型先验知识不足的场景，对数据处理任务开展分析。广泛的实证评估表明，在这三个领域中，我们

[1]一般而言，大语言模型生成的输出具有较高的对数概率，表明模型认可该输出。
[2]K 是展示给用户的答案数量。

均取得了显著提升，Top-1 准确率最高提高了 45%，Top-3准确率提高了 34%。

2 动机场景

本节将通过一个自然语言状态语句示例以及一个与 Power Query M [6] 表达式[3]相关的真实应用场景，说明我们提出的解决方案。M 是一种数据操作语言，用于微软 Excel 和 Power BI 等商业智能应用，通常用于筛选、转换和合并来自一个或多个数据源的数据。数据库领域的若干先前研究 [9, 47, 68] 以 M 为研究动机，因为它能够动态地查找、可视化、共享和查询来自各种在线和离线数据源的数据。

在图 1 所示的场景中，用户正在 Power Query 中操作一个名为 Source 的表，该表包含一个名为'Path'的列，其中存有一些文本数据。用户发出自然语言查询：*"将"Path"列中所有内容的末尾修剪掉一个字符"*，并希望获得相应的 M 表达式（这是从帮助论坛[4]抓取的一条真实查询）。如果表达式正确，用户便可以继续在自己的数据上执行该表达式并收集输出。

我们首先为大语言模型准备尽可能完善的提示，其中包括这条自然语言查询、Source 表的模式、Source 表中的一些样本行，以及自然语言查询与对应 M 表达式的示例（少样本提示）。我们将提示发送给大语言模型，并要求它针对这条自然语言表述生成多个候选程序（$N = 25$），并附上相应的 M 表达式。图 1 展示了其中一些候选项。如图所示，大语言模型生成的排名第一的程序 A 在语法上是正确的，但在语义上不正确，因为 Text。End 需要两个参数，而不是一个（第二个参数表示要选取的字符数）。

在这个例子中，我们发现正确的程序排在第 #10 位（未展示）。这并不理想，因为用户最多只能合理地期望看到前几个候选项。一个显而易见的解决方案是根据模型提供的平均对数概率对候选程序排序。大语言模型可以为组成生成文本的每个词元返回一个概率。这些以对数形式返回的概率衡量了每个词元在输出文本上下文中出现的可能性。对候选程序排序后，正确的程序如今排在第 #5 位（如图 1 中的 E 所示）。这虽有所改善，但对于大多数应用而言，仍可能不够。

正如我们在图 1 中观察到的，程序 A 表明，大语言模型可能会返回语法或语义上不正确的程序。这些程序在执行时会产生错误。一种合理的做法是，移除所有在程序执行期间产生错误的候选项。执行此语义过滤步骤后（参见第 4.1 节），正确的解答移至第 #4 位。实际上，当候选项不止一个时，这能带来显著改进，但值得探究我们能否进一步提升正确程序的排名。

既然我们已经执行了这些程序并获得了它们的输出，接下来很自然的一步就是考察能否利用这些输出来改进候选程序的排序。仔细观察后，我们发现候选项 B 和 C 返回的列都只包含最后一个字符。因此，这两个程序在语义上可能等价，所以我们可以将候选项 C 下调至候选项 D 和 E 之后，因为它们产生的输出与前面的候选项（B 和 C）产生的输出不同 。经过这一步语义交错 处理后（见第 4.2 节），正确的程序排在第 3 位。

一个自然的问题是，是否还有其他信息可以利用，从而进一步提升正确程序的排名。假设有一个预言机掌握了给定自然语言语句的预期输出数据，我们就能轻松地将生成该输出的程序提升到列表首位。没有预言机时，我们可以利用大语言模型来完成这项任务。具体而言，我们可以让大语言模型直接预测与自然语言语句和输入表相对应的数据输出（而不是 M 代码）。这样，我们就能利用这些额外信息对候选程序重新排序。具体来说，我们注意到，候选项 E 的输出与大语言模型预测的首选输出完全匹配，而候选项 B 和 D则不匹配。因此，我们可以通过优化分配给各候选项的平均对数概率分数，将候选项 E 提升至列表首位。使用此基于输出的分数调优方法后（参见第 5.2 节），正确答案便位于首位。

需要指出的是，为了向读者传达主要见解，我们省略了对图 1 中所示若干步骤的讨论，例如温度混合和分数计算。这些技术将在后续章节中详细讨论。

3 利用数据上下文

我们首先介绍问题设定，随后概述我们整体方法的主要内容。

3.1 问题设定

本研究考虑的问题可表述如下：给定某项任务的自然语言描述 nl 以及执行该任务所依据的数据集 D，我们的目标是生成某种所需目标语言中的表达式或程序 s，以完成该任务。如前所述，所谓数据上下文由数据集 D构成，显然是生成目标程序 s 的重要信息来源。利用数据上下文最常见的方式，是在传递给大语言模型的提示中包含对 D 的摘要。这一步如今已是标准做法 [14, 16, 17, 42]，本文不 讨论提示生成问题。相反，在所有实验中，我们都借鉴现有文献中提出的提示词，为该任务设计了尽可能好的提示。

如第 2 节所示，我们关注的是涉及用户参与的场景。具体而言，用户提交自然语言查询 nl，由此启动生成 K 程序 ($K \geq 1$) 的流程，将其作为该查询的潜在答案。生成程序后，用户会对其进行审查，以判断其中是否有程序符合预期结果。如果有，用户可以在自己的数据集 D 上执行该程序，以获得所需的输出。我们的目标是提高 top-K 准确率，以便

[3]本节讨论的大多数观察和见解都适用于 SQL 和 Python，我们将在第 6 节的实验评估中予以证明。[4] https://stackoverflow.com/questions/72548765/trimming-end-of-column-in-power-query

完成上述任务，也就是确保针对给定的自然语言查询，正确的程序出现在向用户展示的 K 程序中。针对我们的设置，我们对 K 的合理取值作出以下观察：

- **K = 1**。尽管提高前 1 名准确率与先前研究类似 [21,25] 很重要，但用户提出的自然语言问题可能本身就存在歧义，因此只展示一个结果可能无法满足用户的意图。此外，用户可能对自己偏好执行的程序类型有特定偏好。例如，在 SQL 的语境下，即使嵌套 SQL 查询和创建视图的查询都能产生正确输出，用户也可能更倾向于执行前者。因此，考虑 $K > 1$是可取的。
- **K ≫ 1**
  先前研究已针对较大的 K 值（即，在 [16]中，K = 100 ）评估了代码生成任务。然而，在由人工观察生成程序的场景中，返回如此大量的程序并不可行。
- **K ≤ 3**
  正如针对类似场景的相关产品实践所示，[1], 向用户展示 3 个答案确实可行，因为这不仅能提高将正确程序呈现给用户的机会，还有助于应对自然语言固有的歧义。

鉴于上述考量，我们旨在提高较小 K取值下的 top-K 准确率。我们的结果见第 6 节。

大语言模型参数。大语言模型提供了若干可调参数。第一个参数是提示，其中包含有关任务和数据上下文的信息。第二个参数是温度，其取值介于 0 和 1 之间（含两端）。温度值较低会使模型行为更具确定性（即，对于相同输入，模型返回的答案变化较小）；而温度较高则会让模型在回答时更敢于尝试、更具“创造性”。换句话说，温度参数控制模型从输出概率分布中采样时加入的“随机性”程度。这种行为使我们能够从大语言模型获取多样化的回答。模型 API 还提供了参数N，用于控制单次调用模型时针对每个提示生成的回答数量。我们在实验中使用此参数来获取 $N = 25$ 个候选程序。最后，对于生成的每个回答，大语言模型还可以返回与回答中每个词元（w_i）相关联的概率值。这些以对数形式返回的概率（对数概率 [55]）用于衡量在给定提示和先前词元的上下文中，每个词元出现的可能性。对数概率较低表示词语出现的可能性较小，较高则表示可能性较大。

3.2 我们的方法

我们的方法应用于后处理阶段——也就是说，在大语言模型生成 N候选项之后——如图 1 所示。我们的方法总体如下，并在算法1中以伪代码形式概述：

提示生成 描述 nl 以及数据集 D的摘要（通常包含模式和少量 D），会被组合成一个提示，其中还包含一些该任务的少样本示例。

温度混合 我们要求大语言模型使用较高的温度设置来生成 N 候选项，以提高多样性；但我们也可以选择将温度为 0 时生成的排名最高的候选项加入其中。第 5.4 节将进一步讨论温度混合。

分数计算 一些大语言模型会提供候选项中所有词元的对数概率。我们根据每个候选项的词元对数概率的平均值为其评分。稍后我们将介绍如何改进这一分数计算。我们注意到，根据平均对数概率分数对候选项进行排序是标准做法 [55]。候选项列表 L 以及评分函数 score 是应用我们提出的新方法的起点。

语义重排序 我们使用一种称为 语义重排序 的技术，对 L 中的 N 候选项重新排序，具体内容见第 4 节。该技术包括两个步骤：

语义过滤 将 L 中的每个 N 候选项在输入数据集 D 上执行（如果 D 过大，则使用一个小样本）。如果执行失败并引发异常，我们就从排序列表 L中移除该候选项。结果是一个新的排序列表 L'，其中包含 N' 个候选项，且 $N' \leq N$。语义过滤将在第 4.1 节进一步讨论。

语义交错 L' 中的候选项会重新排序，形成一个新的有序列表 L''，其中包含 N' 个元素。重新排序时会参考 N' 个候选项生成的输出。如果某个候选项生成的输出与排名更高的某个候选项相同，就会将它移至排序列表中更靠后的位置。具体而言，我们按输出对 N' 个候选项分组，按分数对每组排序，然后将这些组交错排列，得到最终结果 L''。这样做旨在增加 L''中排名靠前的 K 个候选项所生成输出的多样性。第 4.2 节将讨论语义交错。

输出一致性 第 5.2 节介绍了这一可选步骤：根据 (a) s对输入 D生成的输出 o_{exe}的格式良好性，以及 (b) 大语言模型预测在 D 上执行 nl任务（或在 D的样本 D上执行）时会生成的输出 o_{pred} 与 o_{exe} 之间的相似度，对候选项 s 的分数进行调整（该分数已在上述语义重排序过程中使用）。直观而言，如果输出 o_{exe}与 o_{pred}相同或相近，则候选项的分数会提高；如果输出 o_{exe}格式不良，则候选项的分数会降低。此处“格式良好性”的定义可以自定义。例如，如果 o_{exe}包含一列新的空值，或一列大多数值均缺失的新列，那么 o_{exe}就可能被判定为格式不良。

4 语义重排序

如上一节所述，我们通过提示大语言模型来获取N候选程序。我们经常发现，正确的程序确实包含在返回的候选程序集合中，但排名较低，因此难以呈现给用户。第 2 节讨论的情形展示了一个这样的例子。语义重排序的目标是将最有可能符合预期的程序候选项排到列表靠前的位置，这样，如果我们只向用户呈现少量 K 候选项，预期程序被包含其中的可能性就会很高。

算法1 我们生成排名前 K 的候选项的整体方法
给定自然语言描述 nl 和数据集 D时的程序。

要求： 某项任务的自然语言描述 nl 和一个输入数据集 D
确保： 返回排名前 K 的候选程序，以在 D上执行任务 nl

函数 NL2Code(nl, D)
  提示 ← PreparePrompt(nl, D)
  L,对数概率 ← 大语言模型(提示, tmp $= 0.6, N = 24$) ∪
    大语言模型(提示,tmp $= 0, N = 1$)
  $L', O_{\text{exe}} \leftarrow$ SemFilter(L, D)
  O_{pred}, O对数概率 ← 输出预测(nl, D)
  分数 ← 获取分数(L', 对数概率 O_{pred}, O对数概率 O_{exe})
  $L'' \leftarrow$ SemInterleave(L', O_{exe}, 分数)
  返回 前 K 个候选项，来自 L''

读者可能会想，为什么大语言模型生成的默认顺序在许多情况下表现不佳。这里的主要观察是，与大语言模型返回的 N 程序样本相对应的 N 随机变量是独立同分布（*i.i.d.*）随机变量。每个候选项都从同一个底层概率分布中采样，该分布由模型根据自然语言输入计算得出，因此这些候选项服从相同的分布。（这里，所有可能的程序构成概率空间，而大语言模型被视为根据特定的自然语言表述，在此空间上生成一个概率分布。）此外，采样采用有放回的方式，因为同一个候选项可能会在 N 样本中多次出现。因此，每个程序样本都是相互独立的。由此，大语言模型生成的是独立同分布的样本。

然而，为了向用户呈现最相关的前 K 个候选项，我们希望这些 K 样本彼此相互依赖。事实上，它们应尽可能覆盖由（含糊的）自然语言描述所隐含的所有可能程序构成的空间。具体而言，我们希望第 K 个样本是在已知其他样本并非预期结果的条件下概率最高的样本。因此，例如，选择第二个候选项时，应考虑到第一个候选项并非预期结果这一事实。语义重排序的目标因此是提高在前 K 个候选项中呈现所需程序的概率。语义排序算法利用可用的数据上下文对候选程序重新排序。如算法 2 所示，该算法包含两个步骤：语义过滤和语义交错，下面将进一步详细介绍。

4.1 语义过滤

数据 D 的可用性使我们能够在这些数据上执行大语言模型生成的代码片段。如果执行某个程序时出错，就可以将其移除。这是我们对大语言模型生成的候选项重新排序的第一步。

设 s 为模型生成的一个候选程序样本。我们在输入数据集 $d \subseteq D$ 的一个小样本上执行 s。抽取样本 d 时会考虑潜在的主键/外键关系，以确保执行连接操作时能够得到有意义的结果。如果执行成功（即执行完成且未抛出异常），则称程序 s 为语义有效。在语义过滤步骤中，我们只保留语义有效的候选程序。语义过滤的伪代码以函数 SemFilter 的形式展示在算法 2 中。

算法 2 语义重排序器

要求： 一个 L 候选程序数组、一个输入数据集 D，以及一个函数 score，用于为 L中的每个候选项分配分数
确保： 对 L的一个子集进行重新排序后得到的 rankedL

函数 SemFilter(L, D)
  $d \leftarrow$ 示例 D
  对于 $i \in \{0, \ldots, N-1\}$ **执行**
    try
      $O[i] =$执行 $L[i]$(d)　▷ 在 d 上运行 $L[i]$ 并保存输出
    捕获　▷ 执行时抛出了异常
      从数组 L 中移除 $L[i]$
  返回 L , O
函数 SemInterleave(L, O, scores)
  $L \leftarrow$ 排序(L,分数)
  有序类 ← 新数组[][]
  类代表 ← new Array[]
  遍历 $i \in \{0, \ldots,$ len$(L) - 1\}$ **执行**
    如果 $O[i]$ 是类代表[j] 对某个 j**则**
      将 $L[i]$ 添加到有序类数组 OrderedClasses 的末尾[j]
    else
      $j =$len(类代表)　▷ 创建新类
      类代表[j] ← $O[i]$
      有序类[j] ← [$L[i]$]　▷ 一个新的先进先出队列
  RankedL ← []
  当有序类中仍有程序时**执行**
    对于 $i \in \{0, \ldots,$ len(ClassRepresentative) $- 1\}$ **执行**
      如果 有序类[i] 非空 **则**
        弹出有序类[i][0]，并添加到 RankedL 末尾
  返回 RankedL

请注意，我们在同一个输入数据样本上执行所有候选程序。

语义过滤在两种情况下很有帮助：(1) 识别语法错误的候选程序（即不符合编程语言语法的程序）；(2) 识别语法正确但引用了数据中不存在的表格和列，或使用了不存在的运算符的程序。这些问题可能由模型产生幻觉所致。

图 1 展示了一个示例，其中 M 程序 A 在语义过滤后被剔除，因为执行时会产生错误。正如我们在第 6 节所展示的，通过将这项技术用作基线过滤器，我们可以显著提升大语言模型生成的前 K 个结果的质量。

4.2 语义交错

应用语义过滤后，我们得到一个子集，其中的程序都结构良好且有意义。接下来的主要挑战是从这个子集中向用户展示一组多样化的程序，因为剩余的程序往往只有细微的语法差异，但语义上等价。为此，我们将每个程序划分到等价类中，并从不同的等价类中恰当地选取程序。

首先，我们采用一种常用方法，根据程序样本中每个词元（w_i）对应的对数概率，对候选项进行排序。具体而言，我们利用这些概率为每个候选程序赋予一个“分数”（$\bar{p}$）。

该分数通过计算大语言模型生成的每个程序候选项中各词元的条件对数概率的平均值来得出。换言之：

$$\bar{p}_s = \frac{1}{n_s}\sum_{i=1}^{n_s} \log Pr(w_{s,i}|nl, w_{s,1}, \ldots, w_{s,i-1}),$$

其中 n_s 是生成的程序样本 s 在 N 中包含的词元数量，$w_{s,\ i}$ 表示程序候选项 s中的第 i个词元。我们借鉴了先前提出的做法，即使用平均对数似然作为在样本 [16, 55]中进行选择的标准。这个平均对数概率“分数”用作排序指标，分数越高，程序的排名就越靠前。

使用这种启发式方法对候选项进行排序并选出前 K 个候选项存在两个问题。其一，候选项的长度可能会不公平地影响结果。例如，如果某个候选项较长且包含多个可预测的 词元，这些词元可能会使平均对数概率偏高。例如，考虑某个程序候选项中出现的词元 avg。预测该词元之后会出现的词元是“(”，且其概率很高。这个可预测词元“(”可能会显著提高该候选项的平均对数概率。如果候选项中包含许多此类可预测词元，那么无论该候选项是否真正反映了目标程序，其平均对数概率分数都会更高。即使假设所有候选项包含的可预测词元数量大致相同，且平均对数概率能够很好地替代候选项是目标程序的概率，这种方法仍然存在问题，我们将在第 4.2.1 节中详细讨论。

语义交错的伪代码见算法 2。给定候选项列表 L、语义过滤步骤生成的输出 O 以及评分函数，我们首先使用评分函数对 L 中的候选项进行排序。设 L 为排序列表；也就是说，对于每个 $j > i$，$\text{score}(L[i]) > \text{score}(L[j])$。设 $O[i]$ 表示在数据集样本 d上执行 $L[i]$ 的结果。我们首先根据输出，将 L 中的候选项划分为等价类。设s_j 表示列表 L中的第 j个候选项 $L[j]$。s_i的等价类 $[s_i]$ 定义如下：

$$[s_i] \quad = \quad \{s_j \in L \mid O[j] = O[i]\}$$

等价类在算法 2 的第 13 至 19 行构建，并继承 L中的排序；也就是说，等价类的分数等于其排名最高的成员的分数。

$$[s_j] > [s_i] \quad \text{if} \quad \begin{array}{l}\text{there exists } s \in [s_j] \text{ s.t. } \text{score}(s) > \text{score}(s') \\ \text{for all } s' \in [s_j]\end{array}$$

设 $[s_1] > [s_2] \cdots > [s_m]$为根据上述排序从候选项列表 L中得到的 m个等价类。设类别$[s_i]$包含按分数排序的候选项$s_{i,\ 1}$，$s_{i,\ 2}$，$s_{i,\ 3}$，.....。那么，语义交错的结果就是以下按序排列的候选项集合：

$$s_{1,1}, s_{2,1}, \ldots, s_{m,1}, s_{1,2}, s_{2,2}, \ldots, s_{m,2}, s_{1,3}, s_{2,3}, \ldots, s_{m,3}$$

换句话说，我们按顺序考察各个等价类，先从每个等价类中选出排名最高的成员，再选出每个等价类中的第二优成员，依此类推。算法 2 第 21 至 24 行执行了这种等价类交错。

4.2.1 语义重排序的依据。 设 PS 为给定用户自然语言 nl描述和数据上下文 D时所有程序构成的空间。对于任意程序 $s \in PS$，令 $Pr(s \mid nl, D)$表示用户在给定自然语言描述 nl和输入数据集 D时接受 s的概率。当我们使用大语言模型生成代码时，模型会（在内部）在程序空间 PS上生成一个概率分布 Pr，该分布以提示为条件；并假设大语言模型生成 $Pr(_ \mid nl, D)$。

为简单起见，假设我们只想向用户展示 $K = 2$ 条建议。这个论证可以推广到任意 K。为了最大限度地提高用户能在向其展示的 $K = 2$ 个程序中找到目标程序的概率，我们应选择两条建议 s_1 和 s_2，使其中任意一条成为目标程序的概率最大：

$$\underset{s_1,s_2}{\text{argmax}}\ Pr(s_1 \cup s_2 \mid nl, D) \tag{1}$$

当大语言模型生成 N 条建议 L，而我们从候选列表 L 中选出平均对数概率 $\bar{p}$最高的两个程序样本时，这相当于（近似地）尝试最大化：

$$\underset{s_1,s_2}{\text{argmax}}\ Pr(s_1 \mid nl, D) + Pr(s_2 \mid nl, D) \tag{2}$$

这通常无法为用户带来良好的体验，因为该目标与目标 1 并不相同。事实上，我们知道

$$\begin{aligned} & Pr(s_1 \cup s_2 \mid nl, D) \\ &= \ Pr(s_1 \mid nl, D) + Pr(\neg s_1 \wedge s_2 \mid nl, D) \\ &= \ Pr(s_1 \mid nl, D) + Pr(s_2 \mid nl, D) \cdot Pr(\neg s_1 \mid s_2, nl, D) \end{aligned}$$

表达式 2 与上面的表达式之间的差异在于因子 $Pr(\neg s_1 \mid s_2, nl, D)$。在我们期望的目标中，概率 $Pr(s_2 \mid nl, D)$会被这个额外因子折减。概率 $Pr(\neg s_1 \mid s_2, nl, D)$表示：给定用户说出了 nl、给定 D，且给定用户会接受 s_2时，用户不接受 s_1 的概率。现在，假设对于任意两个程序s_1, s_2，$Pr(\neg s_1 \mid s_2, nl, D)$的值都接近 1，那么按对数概率选出排名靠前的$K$个候选项（公式 2）就足够好了，因为这几乎等同于我们期望的目标（公式 1）。然而，这个假设并不合理。具体而言，如果 s_1 和 s_2在用户的输入表 D上产生相同的输出，那么如果用户接受 s_2，他们很可能也会接受 s_1，因此$Pr(\neg s_1 \mid s_2, nl)$会非常小。

令 $[s]$ 表示程序样本 s 的等价类，该等价类由所有在用户的输入表 D上产生相同输出的程序组成：

$$[s] = \{r \in PS \mid \text{exec } s(D) = \text{exec } r(D)\} \tag{3}$$

我们的直觉是，如果 $[s_1] = [s_2]$，那么 $Pr(\neg s_1 \mid s_2,\ nl,\ D)$ 将会非常小：

$$Pr(\neg s_1 \mid s_2, nl, D) = \left\{ \begin{array}{ll} 1 - \epsilon & \text{if } [s_1] \neq [s_2] \\ \epsilon & \text{if } [s_1] = [s_2] \end{array} \right\} \tag{4}$$

其中，ϵ 是某个足够小的常数。

如果假设 4 成立，那么只需识别出满足 s_2 不属于 $[s_1]$ 且 $Pr(s_2 \mid nl)$ 最大的程序 s_2，即可实现目标 1。语义交错正是这样做的：我们将概率最大的程序选作 s_1，然后将概率最大的、不属于同一等价类的程序选作 s_2

作为 s_1。我们继续这一过程，将第三个和第四个样本选为概率最大的、与其前序程序不属于同一等价类的程序。我们一直这样做，直到所有等价类都用尽，然后返回每个类别，选取其中概率第二大的程序。

定理 4.1。令 $Pr(s \mid nl, D)$ 表示用户在给定描述 nl 和输入数据集 D 的情况下接受程序 s 的概率。假设 分数 是一个函数，它为每个候选项 s 赋予一个与 $Pr(s \mid nl, D)$ 成比例的数值。令 $L \subset PS$ 为可能候选程序的有限样本，O 为在输入数据集 D 上运行 L 中的候选程序所获得的输出。若假设 *4* 成立，则由 SemInterleave返回的 K 个程序构成的列表 $L^* :=$ SemInterleave(L, O, 分数) 满足该要求

$$L^* = \underset{L' \subset L, |L'|=K}{\operatorname{argmax}} \; Pr(L \mid nl, D), \quad \forall 1 \le K \le |Distinct(O)|.$$

当 K 大于等价类的数量时，选择任意 K 个候选项都必然会从某些等价类中选出多个候选项。根据我们的假设，这些候选项（从与现有候选项相同的等价类中选出）对目标 1 的贡献微不足道，无论选取哪些候选项都是如此。至此，我们完成了对语义交错的形式化论证。

定理 4.1 表明，初始候选项集合 L 至关重要——首先要有一个多样化的集合 L，这样我们就能从大量等价类中选出排名靠前的K 候选项。定理 4.1 还从形式上说明了，为什么大语言模型生成的排名前K（按排序）的候选项并非最适合展示给用户的候选项，以及重排序为何十分重要。

5 高级技术

语义重排序（包括语义过滤和交错）是一种强大的技术，可利用数据上下文提高前K 项的准确率。本节介绍另外两种技术，进一步提升语义重排序带来的效果。这些高级技术对于非主流且在大语言模型训练数据中代表性不足的目标语言尤其有效。

5.1 替代任务技术

如果大语言模型难以完成某项任务（例如，根据自然语言描述和输入数据集生成 M 代码），我们可以借助一个略有不同的任务来帮助完成原始任务。替代任务技术是指设计一个与原始任务相似但并不完全相同的任务，然后利用该替代任务中的信息来完成原始任务。

假设我们有一个任务：给定随机变量 X，预测随机变量 Y；也就是说，给定 $X = x$，我们希望模型生成概率分布 $Pr(Y \mid X = x)$。假设我们可以找到一个替代随机变量 Z（与 Y相关），并能非常准确地估计 $Pr(Y = y, Z = z \mid X = x)$。例如，$Y$和 Z可能与某个确定性函数（可能可逆）相关。在这种情况下，我们可以要求大语言模型也预测 $Pr\ Z \mid X = (\ )$。之后，我们可以分别获取大语言模型生成的$(Z \mid X = x)$的样本和 y，以及 $Pr(Y \mid X = x)$，然后利用我们对 $Pr(Y = y, Z = z \mid X = x)$的了解，改进对 Y的预测。

算法 3 基于输出预测的分数调优

要求： 自然语言描述 nl、输入样本数据集 d、候选项 L，及其对数概率、执行结果 O_{exe}（针对 L），以及一个函数 sim，该函数用于衡量两个输出之间的相似度；当两个输出相等时返回 1，否则返回 0 到 1 之间的值
确保： 为 L中的每个候选项计算分数

```
1: 函数 输出预测(nl, d)
2: 提示 ← PreparePromptForOutputPrediction(nl, D)
3:     O_pred,O对数概率 ← 大语言模型(提示,tmp = 0.6, N = 25)
4:     return O_pred,O对数概率
5: 函数 EstPr(s, o, O_exe)
6:     o_exe ← O_exe[s]                          ▷ 获取由 s生成的输出
7:     m ← sim(o, o_exe)
8:     返回 m                          ▷ 概率 Pr 的估计值 s | o, nl, d
9: 函数 获取分数(L, 对数概率, O_pred, O对数概率, O_exe)
10:    对于 s ∈ L 执行                                  ▷ 对于每个候选项 s
11:        分数[s] ← 对数概率[s]          ▷ 将分数初始化为平均对数概率
12:        对于 o ∈ O_pred 执行 ▷ 对于每个预测输出
13:            分数[s] ← 分数[s] + O对数概率[o] ∗ EstPr(s, o, O_exe)
14:    返回 O_pred,O对数概率
```

这样做更好的原因在于，当大语言模型被要求预测Z时，可能会采用“不同的神经元放电序列”，因此更有可能为我们带来新知识。

5.2 基于输出预测的分数调优

现在，让我们为自然语言到代码生成任务实例化替代任务技术。这里，X由自然语言描述 nl和数据集 D定义。随机变量 Y取值于所需目标语言中的程序空间。第二个随机变量 Z则取值于程序的输出空间；也就是说，我们希望预测程序在D上运行时会生成什么输出。我们观察到，当数据上下文可用时，大语言模型 Codex 能够根据自然语言表述生成高度准确的输出。在某些情况下，当目标代码是用 Power Query 的 M 语言编写时，Codex 甚至能在给出以下提示后直接预测输出：

> 当收到关于转换任务的描述时，助手会通过展示数据
> 在 *Power Query* 中的转换方式来回答有关表格的问题。

回想一下，语义交错使用分数对等价类排序，并对等价类中的候选项排序；这最终会影响重排序以及对排名前 K 个候选项的选取。默认分数只是大语言模型提供的平均对数概率值。我们利用预测输出进一步优化该分数。基于输出预测的分数调优见算法 3，具体如下：

(1) 在第 3 行，我们使用大语言模型，给定 nl 和 D（温度为 0.6），预测 25 个可能的输出 O预测及其对数概率 Ologprobs。

（2）在第 11—13 行中，我们按如下方式为列表 L 中的每个候选项 s 分配新的分数：

$$\texttt{score}(p) = \texttt{logprobs}(s) + \sum_{o \in O_{\text{pred}}} \texttt{Ologprobs}(o) * Pr(s \mid o, nl, D),$$

其中，$Pr(s \mid o, nl, D)$是这样一个概率的估计值：给定o在D上的输出是所需输出，s是所需程序的概率。

（3）我们通过在输出空间上使用任意相似度指标（第 7 行）来比较o与执行输出$o_{执行}$，从而估计该概率。最简单的指标是在$o == o_{执行}$时返回 1，否则返回 0；不过，也可以使用其他指标。

回到替代任务技术，需要注意的是，$\text{Ologprobs}(o) \cdot Pr(s \mid o, nl, D)$是$Pr(s, o \mid nl, D)$的估计值，而后者正是替代任务技术描述中使用的通用$Pr(Z = z, Y = y \mid X = x)$，在我们的案例中具体化为：$x$是（$nl$，$D$），$y$是$s$，而$z$是$o$。

使用大语言模型进行输出预测也可以通过其他方式实现。我们使用预测输出对候选项重新排序的整体方法，并不取决于如何 预测输出。我们可以使用输入nl和D生成面向不同目标（而非用户所需目标）的代码，然后执行这些代码来预测输出。这是替代任务技术的另一种实例，其进一步研究留待未来工作。

5.3 基于格式良好性的分数调优

我们在分配给程序s的分数中加入一个额外信号，即由s生成的输出o_{exe}的格式良好性。请回忆一下，语义过滤会移除无法生成输出的s。然而，即使程序s成功生成了输出，某些输出也不太可能是期望的输出。例如，如果输出表中包含一个全为空值的新列，那么我们可以将该输出标记为不太可能是期望输出。具体来说，我们将模型为候选项s分配的对数概率值$\text{logprobs}(s)$乘以一个系数$\text{DataQualityMetric}(o_{\text{exe}})$。同样，我们可以在此使用任何数据质量指标，但在实验中，我们采用了一种简单的指标：如果o_{exe}存在空值列，或者o_{exe}是空表，就对s进行惩罚。为了报告实验结果，我们也将基于格式良好性的分数调优纳入“基于输出预测的分数调优”中。

5.4 温度混合

我们之前介绍的技术都假设正确的程序位于大语言模型生成的候选程序列表中。然而，如果大语言模型对数据操作语言的先验知识不足，正确的程序可能不在大语言模型生成的候选集内。在这种情况下，使用额外的程序扩充候选集是合理的。我们可以利用大语言模型本身以及它会进行温度采样这一事实来完成扩充 [14]。

大语言模型通过对随机变量进行采样来预测下一个词元。采样温度是一个控制采样过程随机性的超参数。采样温度越高，采样过程的随机性就越强。具体而言，如果V是词汇表的大小，那么第i个词元以概率$\frac{e^{x_i/T}}{\sum_{j=1}^{V} e^{x_j/T}}$被采样，其中$x_j$是模型为词元$x_j$学习到的权重，$T$是温度。温度会增加熵。如果温度较高，模型输出最高$x_j$值之外的词元的概率也会相当高，从而使生成的输出更加多样。

在代码生成中，我们注意到，大语言模型合成的程序质量会随着温度的变化而显著波动。事实上，这其中存在一种权衡。温度较高时，我们能得到多样化的N样本，但前 1 名准确率会下降，因为N样本可能不包含平均对数概率最高的那个候选项（例如，温度设为 0 时会输出的程序）。另一方面，温度较低时，我们能得到平均对数概率最高的候选项，但多样性会降低，而且N样本往往会多次包含同一个候选项，导致$K > 1$的前K名准确率与前 1 名准确率非常接近，使重排序失去意义。

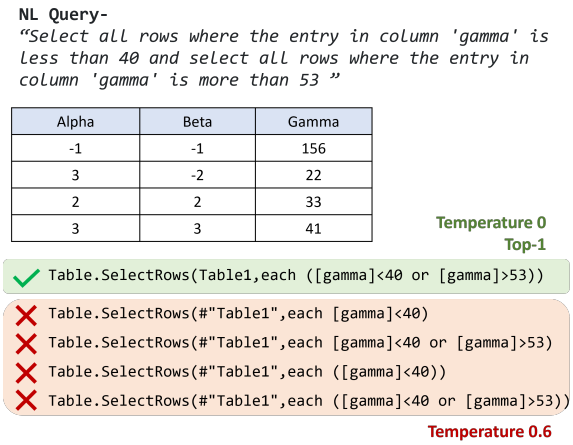


图2：温度混合示例：在温度 *0.6*下，根据对数概率排序，正确的程序排在第二位。添加温度 *0* 候选项后，正确的程序升至第一位。

为缓解这些问题并避免漏掉正确的程序，我们将温度混合引入方法中。具体来说，我们分别以低温度和高温度（即 0 和 0.6）生成程序，将结果拼接起来（如算法1第 3 行所示），然后应用基于语义过滤、交错和输出预测的重排序方法。当模型对输出越不确定时，温度混合就越有效。这种情况可能是因为 Query 含糊不清或非常复杂，也可能是因为模型不熟悉目标语言。在这些情况下，低温度采样十分重要，因为模型计算出的概率分布本身熵就很高（不确定性更大），降低温度有助于减少不确定性。低温度下采样得到的候选项排名更高，因此，即使温度混合没有增加新的候选项，也可能有所帮助；参见图 2。第 6 节将讨论通过这一候选项扩增过程取得的性能提升。

6 实验评估

在本节中，我们在三个数据处理领域评估我们的方法，并分析实验结果。

6.1 实验设置

6.1.1 目标。我们在三种不同的目标语言上进行评估：SQL、Power Query 公式语言 M，以及

Python Pandas。我们的评估任务是根据自然语言描述生成代码（使用这三种目标语言之一）。此外，我们假设给定一个输入数据集，目标是为自然语言查询生成一个目标语言公式或表达式，使其能够在给定的数据集上运行。

SQL 和 Pandas 都是流行的语言，预计预训练大语言模型在训练数据中见过这两种语言的大量表达式；而 M 是一种不那么流行的语言，大语言模型在训练集中接触到的 M 表达式可能没有那么多。

6.1.2 大语言模型。我们使用 code-davinci-002（俗称 Codex 模型）进行评估。该模型后来已被更新的模型取代，例如 text-davinci-003 和一些新的聊天模型；不过，由于时间、成本和可用性限制，我们在 code-davinci-002 上进行了全面评估。我们还在较新的模型上进行了一些初步实验，并确认本文报告的趋势在较新的模型上同样成立。

如第 3.1 节所述，大语言模型提供了会影响其行为的超参数。我们设计实验时遵循的原则是，为超参数选择尽可能好的设置，以定义基线方法（针对每个领域分别设置），并检验我们提出的方法能否取得更好的效果。对于温度，我们采用了设置 $T = 0.6$ 或 $T = 0.8$，因为这些设置在基线评估中表现最好。

我们通常要求模型生成 $N = 25$ 个候选项；除非另有说明，对于所考虑的任意 K 值，基线方法都会根据 25 个解的平均对数概率选取排名前 K 的候选项。大语言模型的停止序列设为提示中指定的标记，用于表示 SQL、M 表达式或 Pandas 表达式的结束。模型的其他超参数均采用默认值，详见 [3]。

6.1.3 基准测试。为了评估我们针对 Pandas 的方法，我们使用了 “Jigsaw” 数据集 [27]。由于 M 表达式语言的功能有限，且没有可用的公开基准测试，我们利用 Jigsaw 数据集创建了一个 M 基准测试。具体而言，我们筛选了 Jigsaw 数据集，只提取 M 支持的转换，构建了 M 的 “JigsawM” 基准测试集。此外，我们还使用了另外两个数据集：第一个由我们通过抓取 PowerQuery 帮助论坛创建，收集了其中 M 表达式的自然语言描述和 M 表达式，在表格中称为 “Forum”。第二个是从 PowerQuery 团队获得的一组基准测试，在表格中称为 “Product”。对于 SQL，我们使用了 “Spider” 开发集 [64] 和 “KaggleDBQA” [29] 数据集。

表 1 展示了有关基准测试的一些统计数据。每个基准测试都由一组自然语言语句及其对应的代码对组成。如表中所示，我们总共在 2526 个（自然语言，代码）对上评估了我们的技术。具体而言，我们分别使用了 SQL 的 1306 个查询、M 的 501 个查询和 Pandas 的 793 个查询。表 1 的最后两列给出了自然语言描述和代码的平均字符数。

6.1.4 指标。 我们使用执行匹配准确率 作为评估指标。如果候选项程序在输入数据集上运行时返回的输出与真实结果程序完全相同，则认为候选项执行匹配成功。

表 1：数据集统计

目标	基准测试	# NL 问题	平均自然语言问题长度	平均代码长度
SQL	Spider	1034	68.04	108.32
SQL	KaggleDBQA	272	55.79	96.00
M	论坛	25	88.28	96.96
M	产品	34	38.08	68.94
M	JigsawM	442	65.66	75.83
Pandas	Jigsaw	793	70.49	56.47

表 2：使用我们的方法获得的前 1 名、前 3 名和前 5 名执行匹配准确率，以及相较于相应基线方法的提升幅度（括号内）。

Task		执行匹配准确率 @K		
目标	基准测试	K=1	K=3	K=5
SQL	Spider	76.0 (+02.8)	90.5 (+12.1)	92.8 (+11.9)
SQL	KaggleDBQA	63.2 (+00.5)	78.4 (+12.5)	81.1 (+12.5)
M	论坛	61.6 (+17.6)	74.4 (+19.2)	74.4 (+10.4)
M	产品	72.3 (+10.6)	75.3 (+02.4)	76.5（+00.6）
M	JigsawM	64.7（+45.0）	72.52（+33.8）	73.7（+25.7）
Pandas	Jigsaw	74.1（+02.9）	86.8（+13.4）	89.0（+12.9）

我们在下方表格中报告执行匹配成功的基准测试所占的百分比。我们还收集了完全匹配准确率的数据，用于检验候选项在语法上是否与真实结果匹配。这些数值较低，但两种指标呈现的改进趋势相似，因此下文的展示将重点采用其中一种指标。

6.2 实验结果

我们首先介绍相对于基线所获得的净改进，这些改进来自结合使用我们的所有技术（语义重排序和基于输出的方法），并根据每个目标和基准测试类别的适用情况加以应用。表 2 展示了这些结果。对于不同的基准测试集，我们报告在 $K \in \{1、3、5\}$ 时获得的执行匹配准确率（百分比）。基线通过关闭我们的方法所添加的附加步骤获得。表 2 中括号内报告了相对于基线数值的绝对 改进。这里的绝对 是指，基线准确率等于括号外的数字减去括号内的数字。

请注意，基线方法使用的提示与我们的方法相同。该提示包括五个少样本示例、输入表名和模式，以及至少三行样本行（三行）。换句话说，基线方法使用的是我们能够为该任务设计的最佳提示。我们没有将任何利用定制机器学习模型或需要微调大型语言模型的方法（见第8节）视为基线，因为我们不希望对训练数据的可用性做出任何假设。还值得注意的是，我们的基线已经通过提示工程超越了最先进水平，如 Spider 排行榜 [8]所示。

表 3：添加语义过滤带来的提升。

Task		执行匹配准确率 @K		
目标	基准测试	K=1	K=3	K=5
SQL	Spider	76.0（+2.8）	80.5（+2.1）	82.8（+1.9）
SQL	KaggleDBQA	63.2（+0.5）	66.5（+0.6）	68.6 (+0.0)
M	论坛	51.2 (+7.2)	64.8 (+9.6)	69.6 (+5.6)
M	产品	71.2 (+9.5)	75.3 (+2.4)	76.5 (+0.6)
M	Jigsaw	39.5 (+19.8)	55.1 (+16.4)	61.9 (+14.0)
Pandas	Jigsaw	74.1 (+2.9)	75.7 (+2.3)	78.1 (+2.0)

表 4：添加语义交错带来的提升。

Task		执行匹配准确率 @K	
目标	基准测试	K=3	K=5
SQL	Spider	90.5 (+10.1)	92.84 (+10.1)
SQL	KaggleDBQA	78.4 (+11.9)	81.1 (+12.5)
M	论坛	67.2 (+2.4)	70.4 (+0.8)
M	产品	75.3 (+0.0)	76.5 (+0.0)
M	Jigsaw	60.5 (+5.4)	67.9 (+6.0)
Pandas	Jigsaw	86.8 (+11.1)	89.0 (+10.9)

对于 $K = 1$，我们看到提升幅度从 0.5%（Kaggle 数据集上的 SQL）到 45%（JigsawM 数据集上的 M）不等。对于 M 而言，前 1 名的提升显著高于 SQL 和 Pandas，因为如果没有我们的增强，大语言模型在面对 M 这种不熟悉的语言时会举步维艰。当 $K = 3$ 和 $K = 5$时，我们看到了我们的方法在排序方面的优势：我们的技术使期望的解决方案更接近顶部。对于 $K = 3$，我们持续看到超过 12% 的提升（Product 上的 M 除外）。对于 $K = 5$，我们持续看到超过 10% 的提升（同样不包括 Product 上的 M）。Product 基准测试上的增益较小，因为这些基准测试规模更小、也更简单：表 1 显示，Product 的自然语言问题和真实代码的平均长度在所有 M 任务中都是最短的；而且在 $K = 1$ 处，Product 的基线准确率在所有 M 任务中最高（为 61.7）。

现在，我们将介绍整体方法中各个组件带来的提升。我们从基线开始，然后逐一启用各个组件。我们报告每一步达到的准确率（因此也报告相较于前一步的提升）。

语义过滤带来的提升

为了衡量语义过滤带来的提升，我们首先让模型针对当前的自然语言问题生成 25 个回答。然后，我们按顺序取出模型生成的候选项列表，并在一部分输入表上执行这些候选项。如果执行成功，我们就称该候选项在语义上有效。我们只保留在语义上有效的候选项（同时保持原有顺序），并报告表 3 中前 K 个候选项的准确率。

表 3 展示了对大语言模型的输出进行语义过滤所带来的提升。我们比较在添加语义过滤与不添加语义过滤的基线条件下，$K \in$个观测值K处的执行匹配准确率，其中包括{1、3、5}个观测值。基线不使用语义过滤，但其他设置均保持不变（即提示词、超参数等）。

执行匹配准确率从未下降，因为语义过滤只会移除那些必定无法通过执行匹配检查的候选项。虽然我们观察到各方面都有不错的提升，但 M 的提升更为显著。这是因为模型对 M 语言不太熟悉，因此更有可能生成无法成功执行的 M 表达式。

语义过滤步骤后，Spider 数据集的候选程序平均剩余 8.4 个，KaggleDBQA 数据集为 6.3 个，三个 M 数据集为 8.3 个，Jigsaw Pandas 数据集为 5 个。相比之下，每种情况下原本都有 25 个，数量显著减少。被过滤掉的程序最多为 24/25 个，最少为 0 个。程序执行过程中遇到的错误包括列名和表名错误（例如，列操作名 operationFrom 被写成 operation_from），以及查询中多出或缺少字符。请注意，语义过滤会移除语法错误的程序，也会移除能够解析但会引发运行时错误的程序。

语义交错带来的改进

为衡量交错带来的改进，我们将上述生成的语义有效候选项作为输入，交由基于交错的重新排序函数处理，从而得到候选项的不同排序。表 4 报告了前 K 个候选项的准确率。请注意，表 4 中报告的改进幅度（方括号内）是相较于表 3 所报告数值的绝对增幅。我们注意到，语义交错不会改变排名最高的候选项，因此不会影响 Top-1 准确率，所以我们仅展示 $K = 3$ 和 $K = 5$处的执行匹配准确率。

我们观察到，对于 Pandas 和 SQL，语义交错使语义匹配准确率绝对提升约 10%，而对于 M，提升幅度通常较小。这几乎与我们观察到的语义过滤效果相反：语义过滤对M的提升高于对 Pandas 和 SQL 的提升。可以这样理解：对于 Pandas 和 SQL 这类在网络上有较充分表示的目标，大语言模型生成语法正确且可执行的表达式并不困难，但要正确地对这些表达式排序则有一定困难；而对于 M 这类在网络数据中代表性不足的目标，情况则恰好相反。

表 3 和表 4 中报告的提升可以叠加，因此，对于大多数基准测试类别，采用语义过滤和交错带来的总提升幅度超过 10%；对于 $K = 3$ 中 JigsawM 上的 M，提升幅度最高可达 21.8%。

基于输出的分数调优带来的提升

接下来，我们评估采用基于输出的分数调优所带来的提升，其中包括基于输出预测的调优和基于格式正确性的调优。正如前文所述，这项技术旨在用于大语言模型训练数据中代表性不足的目标语言。因此，我们这里只报告 M 的数据，并指出，我们没有观察到采用这项技术在 Pandas 和 SQL 上带来任何显著提升。

表 5 报告了在额外加入基于输出的排名后，$K \in${1、3、 5}在M基准测试 上观测到的执行准确率。括号中的数字表示相较于表 4 所报告的准确率的提升；在表 4 中，我们未使用基于输出的

表 5：基于输出的分数调优带来的提升（仅 M）。

Task		执行匹配准确率 @K		
目标	基准测试	K=1	K=3	K=5
M	论坛	60.0 (+8.8)	73.6 (+6.4)	74.4 (+4.0)
M	产品	71.2 (+0.0)	75.3 (+0.0)	76.5 (+0.0)
M	JigsawM	61.3 (+21.8)	72.5 (+12.0)	73.7 (+5.8)

表 6：温度混合带来的提升。

Task		执行匹配准确率 @K
目标	基准测试	K=1
M	论坛	61.6（+1.6）
M	产品	72.3（+1.1）
M	Jigsaw	64.7 (+3.4)

排名。基于输出的排名能够显著提升 M 的 3 个基准测试类别中的 2 个类别的前 1 名准确率；事实上，提升幅度最高可达 21。8%。这符合我们的直觉：基于输出的排名能够潜在地利用大语言模型生成候选项的新替代路径，这对于直接使用大语言模型会产生较差结果的基准测试尤其有帮助。对于所有类别，它都能将前 1 名准确率提升至 60%–70%。在产品类别中，我们没有看到改进；其中的前 1 名准确率很可能受前文提到的原因影响。我们看到，基于输出的分数调优使第 2 个和第 95 个基准测试受益，K=1 分别来自论坛和 JigsawM。

我们观察到 $K \in \{3,5\}$ 的准确率有所提升，尽管与 $K = 1$ 时相比，提升幅度较小。尽管如此，即使对于 $K \in \{3,5\}$，基于输出的分数调优通常也会采取一切必要措施，使准确率达到约 75% 的水平。

温度混合带来的提升

接下来，我们评估温度混合带来的提升。这也是一种有助于改善大语言模型训练数据中代表性不足的语言（例如 M）效果的技术。温度混合只会添加一个来自温度为 0 的运行结果的候选项，因此通常只会影响 Top-1 准确率。因此，表 6 中我们仅报告 M 的 $K = 1$ 数值。$K = 3,5$ 的准确率与表 5 中的准确率相同，因此这些情况下的提升为 0%。然而，对于 $K = 1$，执行匹配准确率提升了 0% 至 3.8%。这些提升看似微不足道，但十分有用，因为它们提高了 Top-1 准确率。

7 讨论与未来工作

我们几种重排序技术的一个关键假设是，大语言模型生成的候选项可以在 try-catch 块中执行。对于副作用很少或没有副作用的语言，这一假设很容易满足。PowerQuery M 目标语言就是这种情况。对于这类语言，我们可以在生产环境中使用基于执行的交错。然而，对于 Python 这类功能更丰富、用途更广泛的语言，Codex 等模型可能会推荐带有负面副作用的程序（例如删除操作系统等）。在这种情况下，我们有两种选择。

表 7：在 Jigsaw 基准测试中，语义交错在有执行和无执行情况下带来的提升。

Task		执行匹配准确率 @K	
目标	基于执行	K=3	K=5
Pandas	Yes	86.8 (+11.1)	89.0 (+10.9)
Pandas	No	79.1 (+3.4)	81.4 (+3.3)

第一种选择是在沙盒中执行来自大语言模型的不可信代码。另一种成本更低的方案是设计非基于执行的技术，尝试近似基于执行的技术。

我们在自然语言到 Pandas 的场景下，针对这一方向开展了一些初步研究。其思路是通过对候选项进行聚类来计算等价类（而不是通过执行候选项来计算）。因此，我们基于所使用的程序语法生成特征，并利用这些特征进行程序聚类。我们开发了一个定制的 ANTLR4[44] 解析器，为任何使用 Pandas 库的 Python 查询提供逻辑表示。我们使用定制解析器提取的特征（例如 Pandas 运算符），将代码片段归入各自的类别。表 7 所示结果表明，在通过执行获得的完整 11% 提升中，仅通过聚类我们就能取得约 3.3% 的提升。显然，这里仍有很大的改进空间。具体而言，在代码生成和交错领域，一个引人关注的挑战在于能否快速确定两个查询的语义等价性，因为自然语言到代码的大多数应用都要求低延迟。

8 相关工作

8.1 少样本提示

我们的贡献与少样本提示无关，但我们利用它来构建基线方法。少样本提示是指在提示中加入任务的一些具体示例。研究表明，这有助于大语言模型生成优质的程序推荐 [14, 16, 17, 42],，包括对较不流行语言的推荐 [26]。关于少样本提示的研究十分丰富，涵盖了设计提示模板 [23, 55, 56, 67],、考察示例的排列组合 [38, 66],，以及增加少样本示例的数量 [62]。我们参考这些工作来构建基线方法，并采用广为人知的 KATE（Knn 增强的上下文示例选择）方法 [36]，从而为少样本提示选择要纳入的示例：使用嵌入模型将自然语言查询转换为向量表示，并 k 从知识库中检索示例，这些示例在嵌入空间中与用户的自然语言查询互为最近邻。鉴于大语言模型对提示十分敏感，已有许多关于提示聚合[10], 或训练能够自行执行聚合的模型 [28, 49], 的研究，也有关于思维链提示 [35], 以及近期修复方法 [18, 57],的研究；不过，我们将这些留作未来工作的潜在方向。

8.2 数据上下文

由于我们所处的领域中数据可用，我们测试了在提示中总结相关输入数据的各种方式，因为众所周知，提示中的微小变化可能会对生成的程序产生显著影响 [39]。示例

包括使用编码将输入数据嵌入 CREATE SQL 语句中，引入诸如 <T> 之类的新词元来标记表名，以及使用简单的字典列出每个表及其关联的列属性和类型 [50, 53]。与现有工作类似 [24], , 我们在提示中为每个表包含了 3-8 行示例数据。我们的主要贡献在于利用数据上下文进行后处理，但我们确实会在提示中总结数据，以定义基线。

8.3 自然语言到代码

在数据库语境中，早期构建自然语言接口的工作涉及利用本体、中间语言以及各种用于连接路径选择的启发式方法 [30, 48, 52]。随着 Transformer 的出现 [59], , 在关系数据库之上提供此类功能变得简单得多。Spider 排行榜 [8]包含了利用机器学习进行文本到SQL生成，并在 Spider 数据集上进行评估的工作列表。这些方法分为三类：定制机器学习模型（例如 [15, 22, 31, 63]）、使用 Codex 和 GPT-4 [45, 46], 等预训练语言模型进行提示工程，以及微调的大语言模型 [51, 54]。我们的工作属于第二类，因为我们假设没有足够的数据来训练定制模型或微调大语言模型。该类别中表现最佳的结果来自 [46] 的工作。这项工作使用 GPT-4 和 Codex 模型，在 Spider 开发测试集（我们也用于评估的数据集）上分别取得了 74.2% 和 69.9% 的前 1 名执行准确率。我们的方法使用 Codex 模型取得了 76% 的前 1 名执行准确率，表明通过提示工程和大语言模型，我们能够超越最先进方法。此外，我们的新技术还显著提升了前 K。

在 Pandas 的背景下，与我们最相关的工作发表于 [27]。主要区别在于，他们的方法需要用户提供输入/输出测试用例。这些测试用于验证和改进大语言模型生成的程序，或修改大语言模型生成的代码，使其能够通过测试用例。相比之下，我们的方法仅依赖自然语言表述，不需要任何额外输入。

8.4 重排序

从自然语言生成代码颇具挑战性 [13, 16, 34, 64] 。

由于采样多个程序时更有可能生成所需的代码，因此已有大量工作致力于设计重排序技术，包括基于执行的重排序技术，以从多个样本中选出最佳候选项 [34, 41, 55, 65]。然而，大多数工作都侧重于提高 Top-1 准确率 [41, 55, 65], , 而我们关注的是提升 top-K 的技术。与我们的工作不同，有些工作会考虑另一种重排序信号：将代码反向翻译为自然语言并检查一致性，这与最大化互信息目标以选出排名第一的候选项有关 [32, 37, 65], ；我们可以将其整合到基于评分的重排序框架中。近期有一项工作 [34] 考虑了与我们类似的技术来提高 Top-K 准确率；然而，该工作主要针对竞赛级编程，且采用的规模与实际应用不符。例如，它会生成数十万个样本，将其筛选至数千个，再进行聚类并选取 10 个，通过交错完成。与包括 [34], 在内的所有先前工作相比，一个关键区别在于，我们的贡献不只是经验性的，还为交错提供了理论依据。此外，我们还提出了新的替代任务技术，以及将其应用于基于输出预测的分数调优。

8.5 大语言模型的数据管理

上下文学习是一个相对较新的概念，关于其在数据处理方面潜在益处的研究有限。Chen 等人 [19] 提出了一种利用上下文学习查询异构数据湖的系统。 [11]提出了另一种方法，该方法需要更少的空间，但需要更多预处理。上下文学习也已应用于数据整理 [40] 和处理 SQL 查询 [58], , 但这些方法需要手动进行提示设计。由于文档中存在多种格式、属性类型和主题，这对数据管理系统而言可能是一项挑战。大多数使用大语言模型的方法都局限于单个表格。 [60] 指出，从关系数据库的完整结构中学习具有巨大潜力，包括可为上下文化表示提供重要信息的相邻表格。最后， [61] 开展了一项初步研究，探讨如何利用大语言模型扩展数据整理，以完成数据集成和清理任务。他们发现，提示方式对这类应用颇具吸引力，因为它可以重复使用同一个预训练模型来处理多种任务和表格。然而，要设计高质量、针对特定任务和数据的提示词，需要丰富的专业知识和大量人工投入，这对于拥有数千个不同表格的企业数据库而言并不可行。另一方面，微调的人工成本较低，但存储成本较高。为了兼得两者的优势，他们将前缀调优 [33] 用作一种参数高效的替代方案，应用于数据整理任务。类似的方法也可能有利于其他自然语言到 X 任务。

9 结论

本文介绍了一种基于上下文学习、用于数据操作程序的新型程序合成框架。我们的方法结合任务输入和输出，端到端地利用数据上下文。我们在三个不同领域（数据库、数据科学和商业智能）中，使用多种新基准测试和现有基准测试对该框架进行了评估。结果显示，在这三个领域中，top-K 准确率均有显著提升。

参考文献

[1] [无日期]. PowerAutomate 中的 AI。 https://powerautomate.microsoft.com/en-us/。[2] [无日期]. GitHub Copilot。 https://github.com/features/copilot。[3] [无日期]. OpenAI API。 https://platform.openai.com/docs/api-reference/completions/ create。[4] [无日期]. Power Query M。https://learn.microsoft.com/en-us/powerquery-m/。[5] [无日期]. Power BI。https://powerbi.microsoft.com。[6] [无日期]. PowerQuery M。 https://learn.microsoft.com/en-us/powerquery-m/。[7] [无日期]. Power BI 中的问答。 https://learn.microsoft.com/en-us/power-bi/natural- language/q-and-a-intro。[8] [无日期]. Spider 排行榜。https://yale-lily.github.io/spider。[9] Hotham Altwaijry、Sharad Mehrotra 和 Dmitri V Kalashnikov。2015年。Query：一种将实体解析与查询处理相结合的框架。*VLDB*基金会论文集9，3（2015年），120–131。[10] Simran Arora、Avanika Narayan、Mayee F Chen、Laurel J Orr、Neel Guha、Kush Bhatia、Ines Chami、Frederic Sala 和 Christopher Ré。2022年。Ask Me Anything：一种提示语言模型的简单策略。*arXiv* 预印本 *arXiv:2210.02441*（2022年）。

[11] Simran Arora、Brandon Yang、Sabri Eyuboglu、Avanika Narayan、Andrew Hojel、Immanuel Trummer 和 Christopher Ré。2023年。语言模型让生成异构数据湖结构化视图的系统变得简单。*arXiv* 预印本 *arXiv:2304.09433*（2023年）。[12] Robert L. Logan IV au2、Ivana Balažević、Eric Wallace、Fabio Petroni、Sameer Singh 和 Sebastian Riedel。2021年。减少提示词和参数：利用语言模型进行简单的少样本学习。arXiv:2106.13353 [计算语言学][13]Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le 等。2021年。利用大型语言模型进行程序合成。*arXiv* 预印本 *arXiv:2108.07732*（2021年）。[14] Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。2020年。语言模型是少样本学习器。神经信息处理系统进展33（2020年），1877–1901。[15] Ruisheng Cao、Lu Chen、Zhi Chen、Yanbin Zhao、Su Zhu 和 Kai Yu。2021年。LGESQL：结合混合局部与非局部关系的线图增强型文本到SQL模型。载于第*59*届计算语言学协会年会暨第*11*届自然语言处理国际联合会议论文集（第*1*卷：长篇论文）。计算语言学协会，在线，2541–2555。https://doi.org/10.18653/v1/2021.acl-long.198[16] Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Ponde de Oliveira Pinto、Jared Kaplan、Harri Edwards、Yuri Burda、Nicholas Joseph、Greg Brockman、Alex Ray、Raul Puri、Gretchen Krueger、Michael Petrov、Heidy Khlaaf、Girish Sastry、Pamela Mishkin、Brooke Chan、Scott Gray、Nick Ryder、Mikhail Pavlov、Alethea Power、Lukasz Kaiser、Mohammad Bavarian、Clemens Winter、Philippe Tillet、Felipe Petroski Such、Dave Cummings、Matthias Plappert、Fo- tios Chantzis、Elizabeth Barnes、Ariel Herbert-Voss、William Hebgen Guss、Alex Nichol、Alex Paino、Nikolas Tezak、Jie Tang、Igor Babuschkin、Suchir Balaji、Shan- tanu Jain、William Saunders、Christopher Hesse、Andrew N. Carr、Jan Leike、Josh Achiam、Vedant Misra、Evan Morikawa、Alec Radford、Matthew Knight、Miles Brundage、Mira Murati、Katie Mayer、Peter Welinder、Bob McGrew、Dario Amodei、Sam McCandlish、Ilya Sutskever 和 Wojciech Zaremba。2021年。评估基于代码训练的大型语言模型。arXiv: 2107 03374 [ ]。机器学习[17]Wenhu Chen、Xueguang Ma、Xinyi Wang 和 William W Cohen。2022年。思维程序提示：将计算与数值推理任务中的推理过程分离。*arXiv* 预印本 *arXiv:2211.12588*（2022年）。[18] Xinyun Chen、Maxwell Lin、Nathanael Schärli 和 Denny Zhou。2023年。教大型语言模型进行自我调试。*arXiv* 预印本 *arXiv:2304.05128*（2023年）。[19] Zui Chen、Zihui Gu、Lei Cao、Ju Fan、Sam Madden 和 Nan Tang。2023年。Symphony：面向多模态数据湖的自然语言查询问答。CIDR。[20] Qingxiu Dong、Lei Li、Damai Dai、Ce Zheng、Zhiyong Wu、Baobao Chang、Xu Sun、Jingjing Xu、Lei Li 和 Zhifang Sui。2023年。上下文学习综述。arXiv: 2301 00234 [ ]。计算语言学 Han Fu、Chang Liu、[21]Bin Wu、Feifei Li、Jian Tan 和 Jianling Sun。2023年。CatSQL：迈向真实世界的自然语言转SQL应用。*VLDB*基金会论文集16，6（2023年），1534–1547。[22] Han Fu、Chang Liu、Bin Wu、Feifei Li、Jian Tan 和 Jianling Sun。2023年。CatSQL：迈向真实世界的自然语言转SQL应用。*VLDB*基金会论文集 16, 6 2023 1534 1547 10 14778 3583140 3583165（apr），–。[23] 2020https://doi.org/。/。Tianyu Gao、Adam Fisch 和 Danqi Chen。。让预训练语言模型成为更好的少样本学习器。*arXiv* 预印本 *arXiv:2012.15723*（2020年）。[24] Carlos Gemmell 和 Jeffrey Dalton。2023年。生成、转换、回答：表格数据的特定问题的工具合成。*arXiv* 预印本 *arXiv:2303.10138*（2023年）。[25] Zihui Gu、Ju Fan、Nan Tang、Lei Ju Cao、Bowen Jia、Sam Madden 和 Xiaoyong Du。2023年。使用结构与内容提示学习进行少样本文本到SQL转换。SIGMOD。[26] Amr Hendy、Mohamed Abdelrehim、Amr Sharaf、Vikas Raunak、Mohamed Gabr、Hitokazu Matsushita、Young Jin Kim、Mohamed Afify 和 Hany Hassan Awadalla。2023年。GPT模型的机器翻译能力如何？一项全面评估。*arXiv* 预印本 *arXiv:2302.09210*（2023年）。[27] Naman Jain、Skanda Vaidyanath、Arun Iyer、Nagarajan Natarajan、Suresh Parthasarathy、Sriram Rajamani 和 Rahul Sharma。2022年。Jigsaw：大型语言模型与程序合成相遇。载于第*44*届国际软件工程会议论文集。1219–1231。[28] Zhengbao Jiang、Frank F Xu、Jun Araki 和 Graham Neubig。2020年。我们如何知道语言模型知道什么？计算语言学协会汇刊8（2020年），423–438。[29] Chia-Hsuan Lee、Oleksandr Polozov 和 Matthew Richardson。2021年。KaggleD-BQA：对文本到SQL解析器的真实评估。*arXiv* 预印本 *arXiv:2106.11455*（2021年）。[30] Fei Li 和 Hosagrahar V Jagadish。2014年。NaLIR：用于查询关系数据库的交互式自然语言界面。载于*2014*年*ACM SIGMOD*数据管理国际会议论文集（美国犹他州斯诺伯德）（*SIGMOD '14*）。计算机协会，纽约，美国，709–712。https://doi.org/10.1145/2588555.2594519

[31] Haoyang Li、Jing Zhang、Cuiping Li 和 Hong Chen。2023年。RESDSQL：为文本到SQL解耦模式链接与骨架解析。arXiv: 2302 05965 [ ]。计算语言学[32]2016Jiwei Li、Michel Galley、Chris Brockett、Jianfeng Gao 和 Bill Dolan。《促进多样性的神经对话模型目标函数》。载于《2016年计算语言学协会北美分会人类语言技术会议论文集》。计算语言学协会，110 119 10 18653 1 16 1014–。[33]2021https://doi.org/。/v /N - Xiang Lisa Li 和 Percy Liang。《前缀微调：优化用于生成的连续提示》。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（第一卷：长篇论文）》。4582–4597。[34] Yujia Li、David Choi、Junyoung Chung、Nate Kushman、Julian Schrittwieser、Rémi Leblond、Tom Eccles、James Keeling、Felix Gimeno、Agustin Dal Lago 等。2022年。AlphaCode：达到竞赛水平的代码生成。《科学》378，6624（2022年），1092–1097。[35] Aiwei Liu、Xuming Hu、Lijie Wen 和 Philip S Yu。2023 年。对 ChatGPT 零样本文本到 SQL 能力的全面评估。arXiv 预印本 arXiv:2303.13547（2023年）。[36] Jiachang Liu、Dinghan Shen、Yizhe Zhang、Bill Dolan、Lawrence Carin 和 Weizhu Chen。2021年。什么样的上下文示例适合 GPT-3? arXiv 预印本 arXiv:2101.06804（2021年）。[37] Michael Xieyang Liu、Advait Sarkar、Carina Negreanu、Benjamin G. Zorn、Jack Williams、Neil Toronto 和 Andrew D. Gordon。2023年。“它想让我说什么”：弥合终端用户程序员与代码生成大型语言模型之间的抽象鸿沟。载于《2023年计算机系统人因国际会议论文集》。ACM，598:1–598:31。https://doi.org/10. 1145/3544548.3580817[38] Yao Lu、Max Bartolo、Alastair Moore、Sebastian Riedel 和 Pontus Stenetorp。2021年。奇妙排序的提示及其发现之处：克服少样本提示顺序敏感性。arXiv 预印本 arXiv:2104.08786（2021年）。[39] Sewon Min、Xinxi Lyu、Ari Holtzman、Mikel Artetxe、Mike Lewis、Hannaneh Hajishirzi 和 Luke Zettlemoyer。2022年。重新思考示范样例的作用：上下文学习如何发挥作用? arXiv 预印本 arXiv:2202.12837 （ 2022 年）。[40] Avanika Narayan 、 Ines Chami 、 Laurel Orr 和 Christopher Ré。2022年。基础模型能帮你整理数据吗? arXiv 预印本 arXiv:2205.09911（2022年）。[41] Ansong Ni、Srini Iyer、Dragomir Radev、Ves Stoyanov、Wen-tau Yih、Sida I Wang 和 Xi Victoria Lin。2023年。LEVER：通过执行学习验证语言到代码生成。arXiv 预印本 arXiv:2302.08468（2023年）。[42] Erik Nijkamp、Bo Pang、Hiroaki Hayashi、Lifu Tu、Huan Wang、Yingbo Zhou、Silvio Savarese 和 Caiming Xiong。2022年。CodeGen：支持多轮程序合成的开放式大型语言模型。arXiv 预印本 arXiv:2203.13474（2022年）。

[43] 2023 4 2303 08774 [ ] OpenAI。《GPT- 技术报[44]2013 4告》。arXiv: 。计算语言学 Terence Parr。《ANTLR权威参考》。The Definitive ANTLR 4 Reference（2013年），1–326。[45] Gabriel Poesia、Alex Polozov、Vu Le、Ashish Tiwari、Gustavo Soares、Christopher Meek 和 Sumit Gulwani。2022年。Synchromesh：基于预训练语言模型的可靠代码生成。载于《第十届国际学习表征会议（ICLR 2022）》，线上会议，2022年4月25日至29日。OpenReview.net。https: // openreview.net/forum?id=KmtVD97J43e [46] Mohammadreza Pourreza 和 Davood Rafiei。2023年。DIN-SQL：通过自我纠错进行分解式文本到SQL上下文学习。arXiv: 2304 11015 [ ]。计算语言学[47]2020Mohammad Raza 和 Sumit Gulwani。《使用混合式程序合成进行网页数据抽取：结合自顶向下与自底向上推断》。载于《2020年ACM SIGMOD数据管理国际会议论文集》。1967–1978。[48] Diptikalyan Saha、Avrilia Floratou、Karthik Sankaranarayanan、Umar Farooq Minhas、Ashish R. Mittal 和 Fatma Özcan。2016年。ATHENA：面向关系型数据存储自然语言查询的本体驱动系统。9，12（2016 10 14778 2994509 2994536）。https://doi.org/。/。Timo Schick 和[49]2020Hinrich Schütze。《重要的不只是规模：小型语言模型同样是少样本学习器》。arXiv 预印本 arXiv:2009.07118（2020年）。[50] Torsten Scholak、Nathan Schucher 和 Dzmitry Bahdanau。2021年。PICARD：语言模型约束自回归解码的增量解析。arXiv 预印本 arXiv:2109.05093（2021年）。[51] Torsten Scholak、Nathan Schucher 和 Dzmitry Bahdanau。2021年。PICARD：对语言模型约束自回归解码进行增量解析。CoRR abs/2109.05093（2021年）。arXiv:2109.05093 https://arxiv.org/abs/ 2109.05093[52] Jaydeep Sen、Chuan Lei、Abdul Quamar、Fatma Özcan、Vasilis Efthymiou、Ayushi Dalmia、Greg Stager、Ashish Mittal、Diptikalyan Saha 和 Karthik Sankara-narayanan。2020年。ATHENA++：面向复杂嵌套SQL查询的自然语言查询。13, 12 2020 10 14778 3407790 3407858 ( )。https://doi.org/。/。[53]Peter Shaw、Ming-Wei Chang、Panupong Pasupat 和 Kristina Toutanova。2020年。组合泛化与自然语言变体：语义解析方法能否兼顾两者？arXiv 预印本 arXiv:2010.12725（2020年）。

[54] Peter Shaw、Ming-Wei Chang、Panupong Pasupat 和 Kristina Toutanova。2021年。组合泛化与自然语言变体：语义解析方法能否兼顾两者？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（第一卷：长篇论文）*。计算语言学协会，在线，922–938。https://doi.org/10.18653/v1/2021.acl-long.75[55] Freda Shi、Daniel Fried、Marjan Ghazvininejad、Luke Zettlemoyer 和 Sida I Wang。2022年。借助执行实现自然语言到代码的转换。*arXiv* 预印本 *arXiv:2204.11454*（2022年）。[56] Taylor Shin、Yasaman Razeghi、Robert L Logan IV、Eric Wallace 和 Sameer Singh。2020年。Autoprompt：利用自动生成的提示从语言模型中引出知识。*arXiv* 预印本 *arXiv:2010.15980*（2020年）。[57] Noah Shinn、Beck Labash 和 Ashwin Gopinath。2023年。Reflexion：具备动态记忆和自我反思能力的自主智能体。*arXiv* 预印本 *arXiv:2303.11366*（2023年）。[58] Immanuel Trummer。2022年。CodexDB：使用 GPT-3 Codex 根据自然语言指令合成查询处理代码。*VLDB基金会论文集*15，11（2022年），2921–2928。[59] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Lukasz Kaiser 和 Illia Polosukhin。2017年。注意力机制就是你所需要的一切。arXiv: 1706 03762 [ ] 。计算语言学[60] Liane Vogel、Benjamin Hilprecht 和 Carsten Binnig。[无日期。]面向关系数据库的基础模型 [愿景论文]。载于*NeurIPS 2022年首届表格表示研讨会*。[61] David Vos、Till Döhmen 和 Sebastian Schelter。2022年。利用前缀微调实现数据整理任务的高效自动化。载于*NeurIPS 2022年首届表格表示研讨会*。[62] Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Ed Chi、Quoc Le 和 Denny Zhou。2022年。思维链提示引发大型语言模型中的推理。*arXiv* 预印本 *arXiv:2201.11903*（2022年）。[63] Peng Xu、Dhruv Kumar、Wei Yang、Wenjie Zi、Keyi Tang、Chenyang Huang、Jackie Chi Kit Cheung、Simon J.D. Prince 和 Yanshuai Cao。2021年。在小型数据集上优化更深层的Transformer。发表于计算语言学协会第*59*届年会暨第*11*届自然语言处理国际联合会议论文集（第*1*卷：长篇论文）。计算语言学协会，在线，第2089–2102页。https://doi.org/10.18653/v1/2021.acl-long.163[64] Tao Yu、Rui Zhang、Kai Yang、Michihiro Yasunaga、Dongxu Wang、Zifan Li、James Ma、Irene Li、Qingning Yao、Shanelle Roman 等。2018年。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。*arXiv* 预印本 *arXiv:1809.08887*（2018年）。[65] Tianyi Zhang、Tao Yu、Tatsunori B Hashimoto、Mike Lewis、Wen-tau Yih、Daniel Fried 和 Sida I Wang。2022年。用于代码生成的编码器-审阅器重排序。*arXiv* 预印本 *arXiv:2211.16490*（2022年）。[66] Zihao Zhao、Eric Wallace、Shi Feng、Dan Klein 和 Sameer Singh。2021年。使用前校准：提升语言模型的少样本性能。发表于国际机器学习会议。PMLR，第12697–12706页。[67] Zexuan Zhong、Dan Friedman 和 Danqi Chen。2021年。事实探测是 [掩码]：学习与学习回忆。*arXiv* 预印本 *arXiv:2104.05240*（2021年）。[68] Erkang Zhu、Yeye He 和 Surajit Chaudhuri。2017年。自动连接：利用转换连接表格。*VLDB基金会论文集*10，10（2017年），1034–1045。