如何在代码智能任务中优化应用于 LLMs 的上下文演示？

摘要

源代码预训练模型已广泛应用于众多代码智能任务。近年来，随着模型和语料库规模的不断扩大，大型语言模型（Large Language Models, LLMs）展现出了在上下文学习（in-context learning, ICL）的能力。ICL 通过提供任务说明和若干示例作为演示，随后将这些演示输入 LLMs 进行预测。这种新的学习范式无需额外的训练，并在多种自然语言处理及代码智能任务中表现出令人瞩目的效果。然而，ICL 的表现在很大程度上依赖于演示的质量，尤其是所选示例的质量。因此，系统地研究如何为代码智能任务设计高效的演示显得尤为重要。在本文中，作者探讨了影响代码智能任务中 ICL 性能的三个关键因素：演示示例的选择、顺序和数量。本文作者在多个代码智能任务中开展了广泛实验，包括代码摘要生成、错误修复和程序合成等任务，在本次复现工作中仅涉及代码摘要生成这一任务。实验结果表明，上述三个因素对 ICL 在代码智能任务中的性能具有显著影响。基于这些发现，本文总结了有效构建演示的策略，并提出了具体的建议，以帮助提升演示设计的质量。此外，本文作者还表明，精心设计的演示能够显著优化现有的演示构建方法，并带来实质性改进。例如，在代码总结、错误修复和程序合成等任务中，优质的演示示例使得 BLEU-4 和 EM 值得到了显著提高，具体而言，EM 分别提高了至少 9.90%，175.96% 和 50.81%。

关键词：上下文学习；演示示例；代码智能任务；大型语言模型

1 引言

近些年来，代码智能研究逐渐成为人们关注的热点，旨在减轻软件开发人员的工作负担并提升编程生产力 [32]， [7]。随着大规模开源代码语料库和深度学习技术的进步，各种神经源代码模型已经开发出来，并在各种代码智能任务上取得了最先进的性能，包括代码摘要生成 [1]、错误修复 [25] 和程序合成 [31]。近年来，预训练技术的出现显著推动了这一领域的进步。像是 CodeBERT [9]，PLBART [2] 以及 CodeT5 [26] 等预训练模型，在各种代码智能任务中表现出了卓越的性能。

随着语言模型和训练数据规模的不断扩大，大型语言模型（LLMs）展现出各种涌现能力。其中上下文学习（ICL）便是其中之一，它使得模型能够仅凭几个示例，从特定上下文中学习任务的解决方式。如图 1所示，ICL 利用所提供的任务指令和一些示例，构建演示来描述任务，并将其与查询问题结合，生成语言模型进行预测的输入。ICL 与传统的调优方法之间最显著的区别在于，它不需要任何额外的训练过程，也不需要对模型参数进行更新。正因如此，ICL 使得模型能够直接应用于各种大型语言模型，大大降低了适应新任务所需的训练成本 [5]。最

近的研究表明，ICL 在多个领域取得了显著成果，包括逻辑推理 [28]、对话系统 [12] 和程序修复 [30]， [19] 等，甚至在一些情况下，ICL 的表现超越了在大型任务特定数据上进行监督微调的方法。

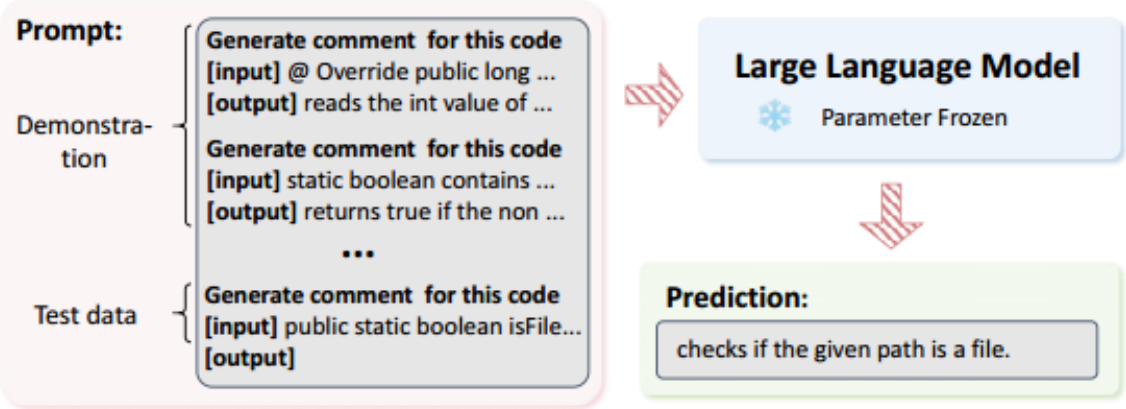

图 1. ICL 应用于代码摘要生成任务的示例图

尽管 ICL 已被证明在代码智能任务中实用性很强，但仍存在问题：ICL 的性能很大程度上依赖于演示的质量 [15]， [16]。现有的研究 [30]， [19] 主要通过随机选择和排列演示示例来构建演示，然而，针对代码智能任务的 ICL 研究仍然相对匮乏。考虑到 ICL 在多个任务中的出色性能表现，探索演示设计的影响因素并研究将 ICL 应用于代码智能任务所面临的挑战，显得尤为重要。于是，本文作者系统地分析了不同的演示构建方法对 ICL 在代码智能任务中性能的影响，如图 2所示，旨在回答以下问题：**如何在代码智能任务中优化应用于 LLMs 的上下文演示？**通过分析上下文演示的原理，作者的研究主要聚焦于三个关键因素：演示示例的选择、顺序和数量。论文对三种热门的代码智能任务——代码摘要生成、错误修复和程序合成，进行了深入的实验研究。具体而言，本文作者主要研究以下四个研究问题（RQs）：

1）在代码智能任务中，哪种演示选择方法能有效提升 ICL 性能?

2）在代码智能任务中，如何安排演示示例的顺序可以优化 ICL 效果?

3）提示中的演示示例的数量如何影响 ICL 在代码智能任务中的表现?

4）这项研究发现是否具有普遍适用性?

考虑到资源的限制，本次课题内容主要围绕前三个 RQs 展开，主要针对代码摘要生成任务进行实验研究。为了回答第一个 RQ，作者比较了多种演示示例选择方法，包括随机选择、基于相似性的选择和基于多样性的选择。此外，作者还对基于相似性选择中的不同检索方法进行了实验，以探讨哪种检索方式能更有效地提升代码智能任务中 ICL 的表现。针对第二个 RQ，本文作者将随机排序与其他两种排序方法（即基于相似性排序和反向相似性排序）进行了对比，旨在探索不同排序方式对 ICL 性能的影响。对于第三个 RQ，作者调整了提示中的演示示例数量，并研究了演示示例数量的增加是否能够带来 ICL 性能的提升。

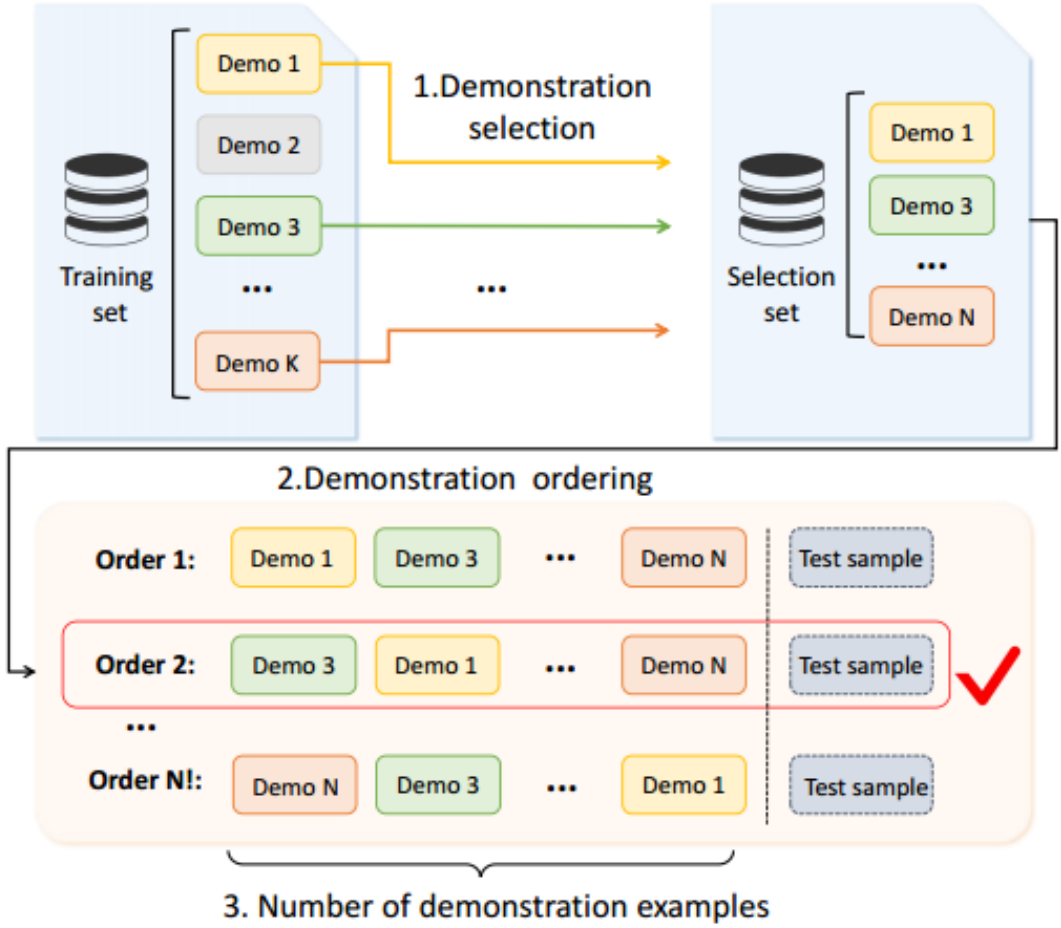

图 2. 上下文演示的设计空间图解

2 相关工作

在详细介绍本次复现的相关工作之前，需要明确一点：ICL 中的演示可以分为两种类型，分别是任务级演示和实例级演示 [21]，[24]。任务级演示对所有测试样本使用相同的演示示例，不考虑每个测试样本的个别差异，而实例级演示则根据不同的测试样本选择不同的演示示例。虽然实例级演示通常在性能上优于任务级演示，但它需要提前标记训练集进行检索。相比之下，任务级演示更加灵活，因为它可以通过选择少量具有代表性的数据进行人工标注，适用于数据稀缺或未标注的场景 [24]。

基于上述分析，复现的过程大致可以分为以下三步：示例检索、示例构造以及对演示示例质量的评估。

2.1 示例检索

示例检索过程主要是针对实例级演示而言的。针对实例级演示，实验需要为每个测试样本选择合适的示例，这一过程的核心在于从数据集中挑选合适的示例便于后续的上下文学习 [15]。作者在检索过程中使用了以下这些方法：

1）**BM-25**：BM-25 是一种经典的稀疏检索方法，广泛应用于信息检索领域，也被广泛用于多个代码智能模型中 [33]， [27]。

2）**SBERT**：SBERT [20] 是一种流行的句子嵌入方法，目前已被广泛应用于文本检索 [20]，[17]。在本文中，作者使用了经过代码数据集微调的版本，以获得更适合代码和文本的表示。

3）**UniXcoder**：UniXcoder [11] 是一个统一的跨模态预训练模型，通过三个序列建模任务和两个基于对比学习的任务进行预训练，它在零样本代码对代码的搜索任务中展现出了出色的性能。

4）**CoCoSoDa**：CoCoSoDa [23] 是一种先进的代码搜索模型，它利用对比学习方法进行代码和文本表示学习，表现出了强大的能力。

对于 BM-25，可以直接调用 gensim 包实现，该方法能够从训练集中检索与测试样本具有最高相似性的样本。对于密集检索方法，实验过程中可以直接采用作者发布的复制包中的预训练模型，无需进一步调整。基于这些预训练模型输出的代码/文本表示，选择与测试样本具有最高余弦相似度的训练样本。此外，作者还借鉴了先前的工作 [21]，提出了一种名为 Oracle 的方法，该方法通过计算测试样本与所有训练集示例之间的输出相似度，来选择演示示例。由于在实际应用中测试样本的输出通常不可得，Oracle 方法通常被视为性能的上限。在 Oracle 中，检索过程由 BM-25 实现，这是因为与其他密集检索方法相比，BM-25 展现出了最佳的性能，如表 1所示。

2.2 示例构造

对于实例级演示，在检索到相关示例后，便可以开始构造示例。而任务级演示则无需进行检索，可以直接构造。在本文中，构造示例的具体过程可以选择不同的选择方法、排列顺序和数量，并生成用于 LLMs 输入的 prompts 文件，以激发 LLMs 的上下文学习能力。

2.3 评估演示示例的质量

根据上一阶段生成的 prompts 文件，调用 LLMs API 接口即可得到一个以 jsonl 格式保存的文本文件。通过对该文件进行简单的划分处理，我们可以进一步计算其评价指标，包括 BLEU-4、ROUGE-L 和 METEOR。

3 本文方法

3.1 本文方法概述

基于**相关工作**部分，本文主要对示例检索和示例构造部分进行了创新性改进。首先，需要将 CSN 数据集按照 8:1:1 的比例划分为训练集、测试集和验证集。

3.2 示例选择

对于 2.1 部分中提到的前四种方法：BM-25、SBERT、UniXcoder 和 CoCoSoDa，本文作者直接使用了现有的预训练模型，因此无需额外的训练。具体的检索过程可以通过以下伪代码进行演示，以下以 Oracle 方法为例进行说明：

该函数实现了基于 BM25 的 “Oracle” 检索过程，具体步骤包括：

1、预处理训练集中的文档注释数据。

2、使用 BM25 模型计算测试集文档注释与训练集文档注释之间的相似度。

Algorithm 1 Oracle

Input: 输入预先处理好的训练集与测试集，同时给出结果保存路径与参数 number——指定从每个测试样本中选择多少个最相关的训练示例

对训练集的每个样本进行处理，将分词连接成字符串
调用已有预训练模型
计算平均逆文档频率（IDF）
for $m = 1 \rightarrow len(test)$ **do** //m 表示迭代次数
    $query \Leftarrow obj['docstring\_tokens']$ //获取测试样本中的文档注释分词
    $score \Leftarrow get\_scores(query, average\_idf)$ //计算 query 与训练集中文档注释之间的相似度分数
    $rtn \Leftarrow sort(score)$//排序并选择前 N 个最相关的样本
    for $i = 1 \rightarrow len(rtn)$ **do** //遍历 rtn
        $candidates\_tokens \Leftarrow 'code\_tokens','docstring\_tokens','score','idx'$ //存储最相关的训练集代码及其相关信息
    end for
    $processed \Leftarrow 'code\_tokens','docstring\_tokens','candidates\_tokens'$//将处理后的数据添加到 processed 列表中
end for
$write\_all(processed)$ //保存结果

3、根据计算的分数选取最相关的训练样本，并将其与测试样本一起保存到输出文件中。

Oracle 方法通过计算测试样本与训练集中所有样本之间的相似性（基于文档注释的相似度），并返回最相关的训练样本，因此通常被视为一种性能的上限。

3.3 示例生成

在示例构造部分，文中共介绍了 9 种方法，其中前两种是基于任务级演示的选择方法：Random 选择方法和 KmeansRND 选择方法。接下来是 4 种实例级演示构造方法（其中 Oracle 方法可以复用 BM25 模型的构造过程），最后是三种示例排序方法。以下将详细介绍 Random 选择方法的构造过程：

Random_selection 函数实现了以下主要功能：

1、随机选择训练集中的样本：从训练集中随机选择 number 个样本，并根据指令构建每个样本的提示（prompt）。

2、构造测试集的提示文本：将训练集生成的提示与测试集中的代码片段结合，形成完整的测试样本提示。

3、保存生成结果：将构造的所有提示保存为 JSON Lines 格式的文件，方便后续输入 LLMs。

Algorithm 2 Random_selection

Input: 输入预先处理好的训练集与测试集，同时给出每个示例的最大长度限制、从训练集中随机选择的示例数目、一些通用的指令或提示文本、针对每个示例输入部分的提示、针对每个示例输出部分的提示以及结果保存路径

1: 打乱训练集顺序 //确保从训练集中随机选择样本
2: $selected \Leftarrow random.sample$ //随机选择训练集中的样本
3: 初始化提示字符串
4: **for** $m = 1 \rightarrow len(selected)$ **do** //m 表示迭代次数
5: $prompt \Leftarrow insturction + input\_insturction + code\_tokens + output\_insturction + docstring\_tokens$ //构造随机选取的训练集示例
6: **end for**
7: **for** $m = 1 \rightarrow len(test)$ **do** //遍历测试集
8: $tmp\_prompt \Leftarrow prompt + insturction + input\_insturction + code\_tokens + output\_insturction$ //创建测试集的提示文本
9: $prompts \Leftarrow 'prompt' : tmp\_prompt, 'label' : docstring\_tokens'$
10: **end for**
11: $write\_all(prompts)$ //保存结果

4 复现细节

4.1 与已有开源代码对比

这篇文章给出了相对完整的检索过程代码、构造过程代码以及调用 LLMs API 的代码，实验过程中无需对代码进行结构性修改。在整个复现过程中，我主要针对一些较为过时的代码进行了更新。原有的开源代码包括了五种预处理方法（在 2.1 节示例检索中有介绍）和四种构造示例的方法（2.2 示例构造中提到的方法），同时也包括了调用多个先进 LLMs API 生成代码摘要的代码。考虑到资源的限制，在复现过程中，我调用了与原文中不同的 LLMs API 接口来完成最后的摘要生成工作，并进行演示示例的质量评估。同时，由于本次复现工作涉及到对代码摘要生成任务的评估，因此需要为该任务收集相应的数据集和预训练模型。

4.2 数据集收集

CSN 是一个从开源 GitHub 存储库中挖掘的大规模源代码数据集，涵盖了六种编程语言的代码摘要数据，包括 Java、Go、JavaScript、PHP、Python 和 Ruby。该数据集按 8:1:1 的比例划分为训练集、验证集和测试集。然而，GitHub 存储库中的 CSN 数据集存在一些未解决的问题，导致它无法直接用于我们的实验。因此，经过一番搜索，我们最终在 Hugging Face 平台上找到了可供实验使用的 CSN 数据集。此外，许多其他预训练模型资源也可以通过 Hugging Face 这一开源平台进行访问。

4.3 实验环境搭建

1、安装 Python 版本 3.9.20

2、安装 Pytorch 版本 2.2.2

https://pytorch.org/get-started/previous-versions/

3、安装其他依赖

pip install -r requirement.txt

这一 txt 文件在源代码文件中已给出。

4、使用服务器硬件加速器进行复现工作

4.4 评价指标

1、BLEU-4

BLEU-4 [18] 是一种用于评估机器翻译结果的指标。BLEU-4 的计算过程分为两个步骤：计算短句惩罚系数和计算 n-gram 的精确匹配率。

计算短句惩罚系数 (BP)。短句惩罚系数主要用于惩罚过长的翻译结果。它通过计算参考答案和机器翻译结果的短句长度比例来确定。如果机器翻译结果的长度超过参考答案的长度，则 BP 小于等于 1，否则 BP 等于 exp(1-参考答案长度/机器翻译结果长度)。短句惩罚系数的作用是防止机器生成过长的翻译结果，从而使得评估结果更加准确。

n-gram 的精确匹配率 (pn)。n-gram 是指由 n 个连续的词组成的序列。计算 n-gram 的精确匹配率需要统计机器翻译结果中 n-gram 的数量，并与参考答案中的 n-gram 进行比较。如果机器翻译结果中的 n-gram 在参考答案中也存在，则认为该 n-gram 是正确匹配的。精确匹配率的计算公式为：pn= 机器翻译结果中正确匹配的 n-gram 数量/机器翻译结果中的 n-gram 数量。

综上所述 BLEU-4 计算公式如下：

$$BLEU = BP \cdot \exp\left(\sum_{N=1}^{4} \omega_n \log(p_n)\right) \tag{1}$$

其中，BP 为短句惩罚系数,pn 为 n-gram 的精确匹配率,exp 为指数函数。

2、ROUGE-L

ROUGE-L [14] 中的”L” 代表最长公共子序列（Longest Common Subsequence, LCS）。该评价指标用于计算输出与参考输出之间的 LCS。为了将 LCS 应用于摘要评估，我们将摘要句子视为单词序列。直观上，两个摘要句子的 LCS 越长，说明它们的相似度越高。

3、METEOR

METEOR [3] 通过显式逐字匹配翻译与参考翻译之间的对应关系来评估翻译质量。如果有多个可用的参考翻译，则针对每个参考独立地对给定的翻译进行评分，并报告最佳分数。该评价指标综合考虑了语料库级别的准确率和召回率、句子流畅性以及同义词对语义的影响，从而解决了 BLEU 标准中存在的一些固有缺陷。

综上所述，这三种评价指标不仅能够对生成的代码摘要进行精准的定量评估，还能从不同角度对生成结果进行全面的质量检测。它们是代码摘要生成任务中不可或缺的工具，能够帮助研究人员和开发者更好地理解模型的表现，提升生成系统的效果。通过结合这些评价指标，能够更准确地衡量模型的翻译或生成能力，从而推动代码摘要生成技术的进一步发展和优化。这些指标目前已在代码摘要生成任务中得到了广泛应用 [22]， [10]。

4.5 创新点

在原文中，作者介绍到，由于时间和资源的限制，所有的实验仅涉及到所选数据集中的两个编程语言部分：Java 和 Python。因此，我尝试在给定数据集的 Go 语言部分验证上述构造演示方法的有效性（进行了两组数据测试），结果较为乐观。同时，在复现过程中，除了调用原文中提到的 GPT-3.5 [5]，我也尝试调用了与原文中不同的 LLMs API 接口来完成最后的摘要生成工作，并对演示示例的质量进行了评估。整个过程主要使用了国内智谱 AI 大模型，经过对 API 文档的学习和理解后，我效仿源代码中的 LLMs API 调用方式，编写出了调用智谱 LLMs API 的接口来生成代码摘要，并根据评估指标对结果进行了分析。

5 实验结果分析

为了避免演示示例顺序不同所可能带来的影响，我们以不同的顺序运行每个实验三次，并报告每个指标的平均结果。此外，我们通过变异系数（CV）进一步评估了每种方法在不同阶数下的敏感性 [4]。CV 由公式 σ/μ 计算，其中 σ 是标准偏差，μ 是平均值。CV 值越低，表明数据波动越小。CV 考虑了数据的规模大小，并在经济学、软件工程等多个领域被广泛用于衡量数据的离散程度 [4]， [29]。

表1展示了在代码摘要生成任务中，演示示例的选择这一因素引起的变化。

从结果可以看出，对于任务级演示，KmeansRND 选择方法明显优于 Random 方法，这表明示例的多样性有助于提升任务级演示的效果。提高任务级演示的多样性不仅能提升 ICL 的平均性能，还能减少不同样本组之间的波动。通过分析 Random 方法的评估指标 CV 值，可以发现，Random 方法在不同上下文中的演示示例表现差异较大，而增加所选示例的多样性则有助于整体减轻这种波动。因此，可以得出结论：提升演示示例的多样性有助于构建更有效的任务级演示，它不仅能提高整体性能，还能带来对不同样本组更为稳定的预测。

在实例级演示中，BM-25 是一种简单有效的实例级演示方法。通过对比不同实例级演示方法的结果，我们发现 BM-25 方法在 ICL 演示选择中的表现，能够与其他密集检索方法相媲美，甚至在某些情况下表现更好。这一结果表明，BM-25 是一个有效的基线方法，值得在未来的代码智能任务中的演示选择研究中予以考虑。因此，对于实例级演示，我们得出结论：演示选择的检索方法会显著影响 ICL 性能，而 BM-25 是一种简单且有效的方法。

在对比任务级演示与实例级演示的性能时，我们发现实例级演示显著优于任务级演示。实例级演示在所有任务中均表现出更好的性能，因此为每个测试样本选择相似的演示示例，可以显著提高 ICL 在代码智能任务中的效果。此外，任务级演示对顺序的敏感度较高。通过比较任务级演示与实例级演示的 CV 值，我们发现实例级演示的性能通常比任务级演示更为稳定，尤其是在面对不同示例顺序时。由此可得出结论：与任务级演示相比，实例级演示不仅能提供更好的性能，而且通常在面对演示顺序变化时具有更强的鲁棒性。

在前面的结论中，我们提到了演示示例的顺序会影响 ICL 在代码摘要生成任务上的性能，特别是在任务级演示中。因此，对于 RQ2，由于资源的限制，本次复现仅针对 Random 方法进行实验。实验结果如表2所示。

首先简要介绍实验中涉及到的三种排序方法。Random 排序方法顾名思义，即对样本进行随机排序；Similarity 排序方法则是根据每个样本与测试样本之间的相似度进行比较排序，相似度较高的样本会被排列在更前的位置。相反，反向相似性排序方法会根据样本与测试样

Approach	**BLEU-4**		**ROUGE-L**		**METEOR**	
	Avg	CV	Avg	CV	Avg	CV
Task-level Demonstration						
Random	18.29	1.59	34.75	1.79	15.6	1.84
KmeansRND	**20.21**	0.90	**37.11**	0.61	**16.09**	0.68
Instance-level Demonstration						
BM-25	**21.91**	0.41	**38.54**	0.49	**16.97**	0.65
SBERT	21.39	0.29	38.01	0.47	16.61	0.29
UniXcoder	21.77	0.49	37.53	0.49	16.23	0.35
CoCoSoDa	21.01	0.56	37.91	0.21	16.29	0.40
Oracle (BM-25)	26.46	0.36	44.11	0.22	18.31	0.21

表 1. 不同的演示示例选择方法对代码摘要生成任务的实验结果

本的相似度进行降序排列。

Approach	Random		
	BLEU-4	**ROUGE-L**	**METEOR**
Random	19.02	35.94	15.71
Similarity	**20.15**	**37.01**	**16.34**
Reverse Similarity	19.87	35.71	15.67

表 2. 不同的演示示例顺序对代码摘要生成任务的实验结果

通过观察表中的结果，可以发现，按照测试样本相似度对演示示例进行升序排序，通常比反向相似性排序方法效果更好。此外，相似性排序方法在大多数情况下表现出三种方法中的最佳性能。然而，值得注意的是，随机排序方法的表现并未明显逊色于其他两种方法，这表明未来的工作可以探索更加复杂、更具综合性的演示排序方法。因此，可以得出结论：演示样例的排序方式会显著影响 ICL 的性能。在大多数情况下，按演示样例与测试样例的相似度升序排列能够获得相对更优的结果。

针对 RQ3，实验过程研究了示例数量的增加是否会提升 ICL 在代码摘要生成任务上的性能。我们将演示示例的数量从 1 个逐步增加至 64 个。基于前两个 RQ 的结果，实验分别采用 BM-25 和相似性排序方法来进行演示示例的检索和排序。如表 3所示，结果表明，ICL 在所有任务中的表现随着演示样例数量的增加而有所提升。

然而，当示例数量超过 16 个时，代码摘要生成任务的 BLEU 指标显著下降，经研究，这一问题主要源于截断问题 [8]， [6]。由于代码通常比自然语言 [13] 更长，截断问题在代码智能任务中更加常见。此外，更多的示例会增加外部 API 的调用成本和推理时间。因此，较少数量的示例可能更适合用于代码智能任务。从实验结果来看，四个演示示例的性能已经足够优秀，因而可以得出结论：考虑到截断问题，增加提示符中的演示示例数量并不总能带来更好的性能。为了节省成本，建议在演示中使用 4 个示例。

Number	**BLEU**
1	20.03
4	21.91
16	**22.63**
64	21.34

表 3. 不同的演示示例数量对代码摘要生成任务的实验结果

6 总结与展望

本文通过实验研究了不同的演示选择方法、演示排序方式和演示示例数量对代码智能任务中上下文学习性能的影响。研究结果表明，与简单随机选择的演示示例相比，经过精心设计的 ICL 演示能够显著提升学习效果。本文作者系统地总结了这些发现，并提出了一些针对代码智能任务演示构建的优化建议，旨在帮助研究人员和开发者改进演示设计，从而提升模型的学习性能。

在本文中，作者全面探索了演示设计对 ICL 性能的影响，突出了几个关键因素，这些因素能够显著提高代码智能任务中的上下文学习表现。实验结果不仅为现有的 ICL 方法提供了新的见解，也为未来的演示设计提供了系统性的指导原则。作者进一步讨论了这些发现对于研究人员和开发者的潜在影响，并探讨了它们在大型语言模型应用中的广泛意义。

在本次复现过程中，我们通过收集数据、配置环境、修改并运行代码等等工作对实验结果进行复现，最后得到了与原文相似的实验表现。针对 RQ1 的结果，可以发现，当前最先进的代码检索模型与 Oracle 之间仍存在较大差距，这表明这些模型未能选择出语义相似度最高的示例。因此，开发更有效的零样本代码到代码搜索的代码表示模型是一个值得深入研究的方向。此外，基于每个任务的先验知识或源代码的属性来设计示例选择策略，也是值得探索的研究方向。针对 RQ2，按演示样例与测试样例的相似度升序排列，相较于随机排序和反向相似性排序，通常能带来更好的性能。然而，这种改善并不始终一致，因此如何为代码智能任务自动设计更优化的排序方法仍然是一个亟待深入探索的问题。对于 RQ3，由于代码片段的长度通常要比自然语言文本冗长得多，因而提示中的示例数量受到了限制，这可能导致陡增的计算成本和时间成本。因此，将程序切片和缩减技术融入 ICL，以降低这些成本，是值得进一步研究的方向。

展望未来，首先，优化检索过程有望显著改善实例级演示的示例选择，且随着检索提示相关话题的逐渐升温，涌现出了不少先进方法。因此，将这些新方法应用于示例检索过程，有望进一步增强 ICL 在代码智能任务中的表现。与此同时，计划深入研究更多与源代码相关的因素，尤其是代码质量和自然性对 ICL 性能的影响。最后，作者将进一步验证本研究成果，并扩展到其他大型语言模型，以确保这些发现具有更广泛的适用性。

参考文献

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. *arXiv preprint arXiv:2005.00653*, 2020.

[2] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333*, 2021.

[3] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[4] Charles E Brown. Coefficient of variation. In *Applied multivariate statistics in geohydrology and related sciences*, pages 155–157. Springer, 1998.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Zihang Dai. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[9] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[10] Shuzheng Gao, Cuiyun Gao, Yulan He, Jichuan Zeng, Lunyiu Nie, Xin Xia, and Michael Lyu. Code structure–guided transformer for source code summarization. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–32, 2023.

[11] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*, 2022.

[12] Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A Smith, and Mari Ostendorf. In-context learning for few-shot dialogue state tracking. *arXiv preprint arXiv:2203.08568*, 2022.

[13] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[14] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[15] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[16] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

[17] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

[18] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[19] Julian Aron Prenner, Hlib Babii, and Romain Robbes. Can openai's codex fix bugs? an evaluation on quixbugs. In *Proceedings of the Third International Workshop on Automated Program Repair*, pages 69–75, 2022.

[20] N Reimers. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[21] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*, 2021.

[22] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. On the evaluation of neural code summarization. In *Proceedings of the 44th international conference on software engineering*, pages 1597–1608, 2022.

[23] Ensheng Shi, Yanlin Wang, Wenchao Gu, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. Cocosoda: Effective contrastive learning for code search. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2198–2210. IEEE, 2023.

[24] Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*, 2022.

[25] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful code changes via neural machine translation. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 25–36. IEEE, 2019.

[26] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[27] Bolin Wei, Yongmin Li, Ge Li, Xin Xia, and Zhi Jin. Retrieve and refine: exemplar-based neural comment generation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 349–360, 2020.

[28] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[29] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. Clear: contrastive learning for api recommendation. In *Proceedings of the 44th International Conference on Software Engineering*, pages 376–387, 2022.

[30] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494. IEEE, 2023.

[31] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

[32] Zhengran Zeng, Hanzhuo Tan, Haotian Zhang, Jing Li, Yuqun Zhang, and Lingming Zhang. An extensive study on pre-trained models for program understanding and generation. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*, pages 39–51, 2022.

[33] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. Retrieval-based neural source code summarization. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1385–1397, 2020.