数据集。然而,当部署它来比较存储库中的整体功能语义时,
它不如接下来介绍的模型有效。

UniXCoder - 代码搜索任务微调

前面提到的模型都接受了克隆检测任务的训练,这
使用交叉编码器范例微调模型。调查模型是否
在双编码器范例中进行微调可以更好地生成单独的代码嵌入,
然后我们在代码搜索任务上对 UniXCoder 进行了微调,该任务旨在语义上查找
给定自然语言查询的相似代码。

我们在 AdvTest9 上使用提供的微调脚本 8 微调 UniXCoder
数据集,来源于 CodeSearchNet10。这个过程花了大约 3 个小时
A40 服务器一个时代。

然后我们将在克隆检测任务上微调的 UniXCoder 与这个进行了比较
模型。在使用 bi 比较代码对时,两种模型都表现出良好的性能
编码器方法,但后一种模型在以下情况下产生了明显更好的嵌入
比较存储库相似性。评估部分提供了更多详细信息。

5.2 比较存储库之间的相似性

该项目的另一个重要任务是比较和识别语义相似
存储库。通过结合源代码语义比较和元数据的结果
比较,我们可以将存储库的语义信息表示为嵌入和
比较它们的相似性,如 Repo2Vec11 中所示。在这里,我们使用了文档
inspect4py 提取的字符串作为每个存储库的元数据组件。我们

8 郭大亚等。 UniXcoder:代码表示的统一跨模态预训练。 2022.doi:10.48550/ARXIV.2203.03850。网址:https://arxiv.org/abs/2203.03850。

9帅路等。 “CodeXGLUE:用于代码理解和识别的机器学习基准数据集一代”。在:CoRR abs/2102.04664 (2021)。

10Hamel Husain 等人。 “Codesearchnet 挑战:评估语义代码搜索的状态”。在:arXiv 预印本 arXiv:1909.09436 (2019)。

11Md Omar Faruk Rokon 等人。 Repo2Vec:一种用于确定存储库相似性的综合嵌入方法。 2021.doi:10.48550/ARXIV.2107.05112。网址:https://arxiv.org/abs/2107。 05112。

通过将所有函数和文档字符串提取到一个中,采用了一种直接的方法
维度集,并比较函数集和文档字符串集
以某种方式存储库。

5.2.1 交叉编码器和匈牙利算法

匈牙利算法最初是作为一种有效解决问题的方法而提出的
分配问题 12.在我们的例子中,它是寻找单边完美匹配的有用工具
两组之间。这意味着较小集合中的每个元素都被分配给一个
较大集合中的唯一元素,使得分配的总相似性得分为
最大化,基于两个集合之间每个匹配的给定相似性分数。

我们使用我们的 GraphCodeBERT 模型来计算每个独特的相似度分数
对两个函数集的笛卡尔积。然后,我们应用了匈牙利语
算法找到具有最大总相似性得分的分配。

我们在两个 GitHub 存储库上测试了这个方法,这两个存储库都是各种不同的集合
Python中的算法实现。结果表明,虽然该方法
成功识别出许多实现相同算法的代码对,有
一些不可忽视的不可避免的问题:

1. 效率低 这种方法的最大问题是效率低下。对于两个回购
   分别具有 m 和 n 个独特功能的故事,这将需要计算
   m·n 函数对的相似度得分。通常,GraphCodeBERT 模型采用
   在 NVIDIA T4 Tensor Core GPU 上预测一个代码对的时间超过 0.7 秒。
   在这种情况下,如果我们有两个存储库,每个存储库都有 100 个独特的函数,对所有
   对将需要近2个小时。这意味着该方法使用起来不切实际,因为
   具有 100 多个函数的存储库在实践中很常见。

2. 概率估计不佳 由于 GraphCodeBERT 模型使用线性
   分类器对输出嵌入进行二分类,相似度

12哈罗德·W·库恩。 “分配问题的匈牙利方法”。在:海军研究后勤季刊 2.1-2（1955 年 3 月）,第 83-97 页。内政部:10.1002/nav.3800020109。

通过在其输出 logits 上应用 softmax 函数产生的分数往往具有
阳性病例的得分非常高（例如 0.9997），或者得分非常低（例如 0.01）
对于负面案例。这不仅导致不可靠的不确定性估计，而且
即使预测不正确，也显示出高置信度。在那个时间
在撰写这份报告时，我们考虑过使用概率校准 13来查看它是否
改进了模型的概率估计，但由于时间限制，这将
留待以后研究。

3.牺牲了一些最佳匹配为了找到与
最大相似度得分，匈牙利算法可能会牺牲一些局部操作
timal 匹配有利于更高的全局匹配分数。因此，一些算法
可能与完全不同的算法相匹配。

5.2.2 双编码器方法：RepoSim

由于交叉编码器方法不可扩展，我们提出 RepoSim：一种使用
双编码器范式，包括以下步骤：

1.将存储库的所有函数和文档字符串提取到函数集和文档字符串中
分别设置，使用inspect4py。

2.使用UniXCoder获取每个函数的code embeddings并求平均
计算平均代码嵌入。

3.使用docstring embedding模型获取每个docstring的sentence embeddings
并计算它们的平均嵌入。

4.确定两个repos之间的code similarity score和docstring similarity score
通过计算它们的平均代码嵌入的余弦相似度及其
平均文档字符串嵌入。

13F。佩德雷戈萨等人。 “Scikit-learn：Python 中的机器学习”。载于：机器学习研究杂志 12 (2011)，第 2825–2830 页。网址：https://scikit-learn.org/stable/modules/calibration。
html#校准。

将存储库转换为其平均代码嵌入和平均文档的可视化

字符串嵌入如图 5.2 所示。

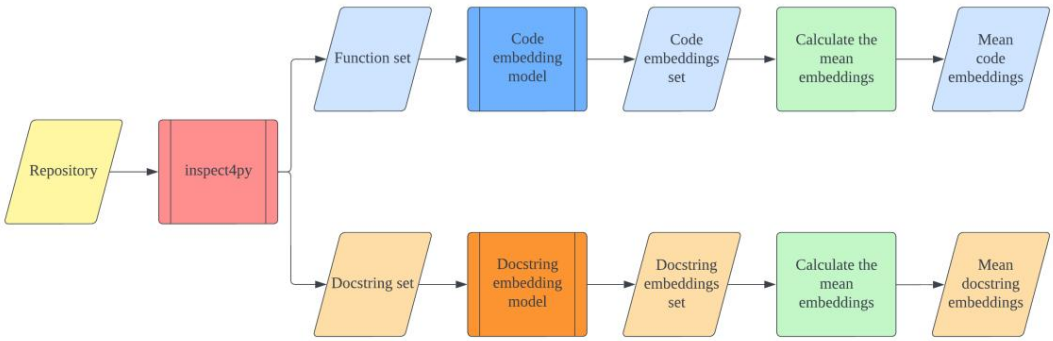

图 5.2:嵌入的存储库

这种方法比以前的方法更有效,因为每个函数和

每个文档字符串只需要编码一次,比较之间的余弦相似度

嵌入比运行语言模型生成它们花费的时间要少得多。这

允许我们在合理的时间范围内比较更多的存储库。在后面的评估中

评估阶段,该方法在对存储库进行分组方面也展示了可喜的结果

具有各种主题并提供合理的相似性分数。

文档字符串嵌入模型的选择

我们选择了一系列知名的高性能预训练句子转换器

楷模。通过让他们为每个存储库中的文档字符串生成嵌入,并且

通过接收器操作特性测试它们的性能,我们确定了最佳

执行文档字符串嵌入模型以在代码搜索任务上对 UniXCoder 进行微调。

可以在评估部分找到更多详细信息。

第6章

评估

在这个评估部分,我们调查了各种模型在 determin 中的性能
使用 Type IV Python 克隆和我们提出的方法的有效性,包括
确定存储库相似性的匈牙利算法和 RepoSim。我们评估
克隆检测任务上的自定义 GraphCodeBERT 和 UniXCoder 等模型,以及
评估类似存储库的不同编程语言和自然语言模型
使用余弦相似度和 AUC 度量。结果提供了对模型的见解
性能和我们发现的影响,并解决一些局限性和
提出潜在的未来研究方向。

本节重点关注两个主要方面:

1. 评估模型在确定 IV 型 Python 克隆方面的性能。

2. 评估我们的方法在确定存储库相似性方面的有效性。

为了量化这两项任务的表现,我们需要为每项任务定义指标。

6.1 评估中使用的指标

6.1.1 精确率、召回率和 F1 分数

精确率、召回率和 F1 分数是用于评估性能的常用指标
分类或排名模型。这些指标是根据

真阳性、假阳性、真阴性和假阴性。这是一个定义
这些条款：

- 真阳性 (TP)：被正确分类为阳性的阳性实例
  模型。
- 假阳性 (FP)：被错误分类为阳性的阴性实例
  该模型。
- True Negative (TN)：被正确分类为负例的负例
  该模型。
- 假阴性 (FN)：被错误分类为阴性的阳性实例
  该模型。

精确率、召回率和 F1 可以使用以下公式计算：

$$\text{精度} = \frac{TP}{TP + FP}$$

$$\text{回忆} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \quad \text{精度} \quad \text{召回}}{\text{准确率} + \text{召回率}}$$

总之，精度是衡量模型预测正例的准确性的指标，
召回率（也称为敏感性）评估模型识别积极实例的程度，
F1 分数是精确率和召回率的调和平均值，这在两者都适用时很有用
准确率和召回率很重要。

准确率和召回率的重要性在很大程度上取决于具体任务。为了
例如，在代码抄袭检测中，我们希望尽可能少的误报，使得
精度更重要。在这个项目中，我们主要关注 F1 分数
评估我们模型的性能。

6.1.2 ROC 和 AUC

接受者操作特征 (ROC) 曲线是
分类模型在各种分类阈值下的性能 1。它绘制了
以下两个参数：

- 真阳性率 (TPR)：正确识别的阳性实例的比例
  分类为阳性。它与上面定义的召回率（或灵敏度)相同：

$$TPR = \frac{TP}{TP + FN}$$

- 假阳性率 (FPR)：不正确的阴性实例的比例
  直接归类为阳性。它是使用以下公式计算的：

$$FPR = \frac{\text{计划生育}}{FP+TN}$$

可以通过改变分类来改变正预测和负预测的数量
阳离子阈值。阈值越低，越多的实例被归类为正例，
从而增加真阳性和假阳性病例。 ROC 曲线示例
如下所示：

1分类：Roc曲线和AUC 机器学习 google developers。 2022 年 7 月。网址：https：//developers.google.com/machine-learning/crash-course/classification/roc-and-auc。

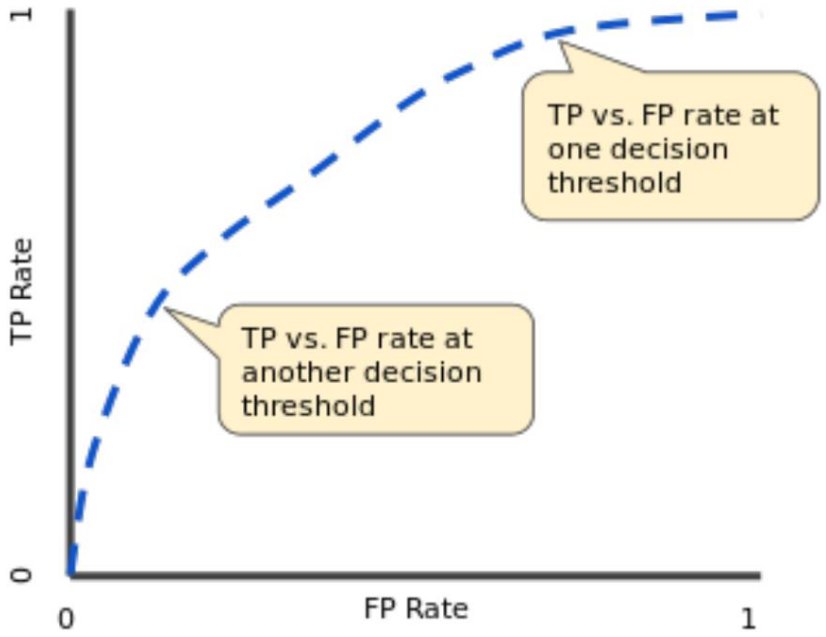

图 6.1:ROC 曲线 2

顾名思义,ROC 曲线下的面积 (AUC) 是曲线下的面积
ROC 曲线范围从 0 到 1,其中 1 代表一个完美的分类器,0.5 代表一个
随机分类器。它可以用来总结二元分类的性能
跨越所有可能的分类阈值的模型。在这个项目中,我们将使用 AUC,因为
metric 允许我们比较不同的分类器并找到一个最佳阈值
关于存储库是否相似的二元决定。

6.2 IV型蟒蛇克隆检测评价

模型	微调任务	克隆检测			
		Recall Precision F1 分数 AUC			
GraphCodeBERT 克隆检测 96.42 UniXcoder 克隆检测 93.64			97.56	96.99	
			96.34	94.97	98.5
UniXcoder	代码搜索	80.98	85.96	83.4	90.4

表 6.1:使用 C4 数据集对克隆检测任务的评估

2[32]

衡量模型在克隆检测任务中性能的最常用指标
是精确率、召回率和 F1 分数。在这个项目中,我们评估了所有模型,包括
为克隆检测任务微调的自定义 GraphCodeBERT,UniXCoder 很好
针对克隆检测任务进行了调整,而 UniXCoder 针对代码搜索任务进行了微调,在
C4 数据集3来测试它们在 IV 型克隆检测任务上的性能。

我们的 GraphCodeBERT 可以用作二元分类器并生成 True/False pre
C4 数据集中每个代码对的字典。对于两个 UniXCoder 模型,我们生成
通过计算每个代码对的余弦相似度分数来进行预测,然后找到一个最优的
具有最高 F1 分数的阈值,并使用它根据相似性对每个代码对进行分类
分数。评价结果见表6.1。

由于我们使用双编码器范式来执行克隆检测任务,因此 F1 分数
两个 UniXCoder 模型中的一个不如 GraphCodeBERT,这是一个交叉
编码器。但是,我们仍然可以看到它们具有相对较好的性能,尤其是对于
UniXCoder 在克隆检测任务上进行了微调,其 F1 分数非常接近
GraphCodeBERT 模型。此外,我们还绘制了 ROC 曲线并计算了
两个 UniXCoder 模型的 AUC,如表 6.1 和图 6.2 所示。我们可以考虑
两个分类器都很好,因为它们都达到了高于 0.9 的 AUC。

3陈宁陶等。 “C4:对比跨语言代码克隆检测”。在:2022 年 IEEE/ACM 第 30 届程序理解国际会议 (ICPC)。 2022 年,第 413–424 页。 doi: 10.1145/
3524610.3527911。

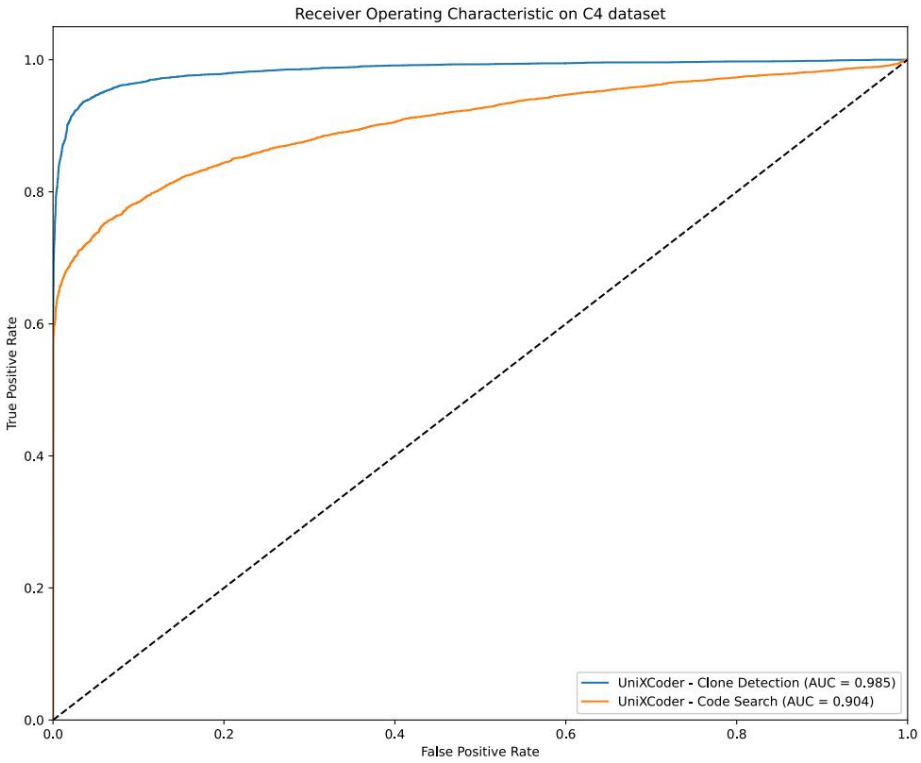

图 6.2:在 C4 数据集上的克隆检测任务上评估的每个 UniXCoder 模型的 ROC 曲线

它们的余弦相似度得分分布之间的差异在 Ap 中给出

附录 B。

6.3 库相似性比较评价

6.3.1 GraphCodeBERT与匈牙利算法

我们使用我们的 GraphCodeBERT 模型并应用匈牙利算法进行比较

两个 GitHub 存储库:keon/algorithms4和 TheAlgorithms/Python5。这些

4基翁。算法。 https://github.com/keon/algorithms。 2022.
5算法。 Python。 https://github.com/TheAlgorithms/Python。 2022.

两个存储库特别适合语义比较,作为不同的实现
相同算法的不同版本共享相同的意图或语义,但它们的代码可能看起来
非常不同。

结果分配,包括每个匹配项的相似性分数,存储在
CSV 文件并在提交中上传。该 CSV 文件的一部分如图所示
6.3,我们可以看到相同算法的许多实现被匹配在一起
这个任务。

	name1	name2	docs1
2	gcd	euclidean_gcd	gcd Euclid's Algorithm. gcd{a,b}=gcd{−a,b}=gcd{a,−b}=gcd{−a,−b} See proof: https://proofwiki.org/wi
3	num_perfect_squares	perfect_square	num_perfect_squares Returns the smallest number of perfect squares that sum to the specified num
4	_l2_distance	euclidean	_l2_distance Calculate l2 distance from two given vectors.
5	cosine_similarity	cosine_similarity	cosine_similarity Calculate cosine similarity between given two vectors
6	magic_number	is_magic_gon	magic_number Checks if n is a magic number
7	solve_chinese_remainder	chinese_remainder_theorem2	solve_chinese_remainder for a system of equations. The system of equations has the form: x % nums
8	extended_gcd	extended_gcd	extended_gcd Return s, t, g such that num1 * s + num2 * t = GCD(num1, num2) and s and t are co-pri
9	modular_inverse	modular_exponential	modular_inverse a and m must be coprime
10	decimal_to_binary_util	decimal_to_binary	decimal_to_binary_util Convert 8-bit decimal number to binary representation
11	decimal_to_binary_ip	local_binary_value	decimal_to_binary_ip Convert dotted-decimal ip address to binary representation with help of decima
12	euler_totient	euler_modified	euler_totient Time Complexity: O(sqrt(n)).
13	base_to_int	base85_encode	base_to_int Note : You can use int() built-in function instead of this.
14	modular_exponential	naive_cut_rod_recursive	modular_exponential Time complexity - O(log n) Use similar to Python in-built function pow.
15	get_primes	sieve_er	get_primes Using sieve of Eratosthenes.
16	power	combination_sum_iv	power Calculate a ^ n if mod is specified, return the result modulo mod Time Complexity : O(log(n)) S
17	power_recur	_modexpt	power_recur Calculate a ^ n if mod is specified, return the result modulo mod Time Complexity : O(lo
18	lcm	lcm	lcm Computes the lowest common multiple of integers a and b.
19	gcd_bit	get_bitcode	gcd_bit Similar to gcd but uses bitwise operators and less error handling.
20	find_next_square2	check1	find_next_square2 Alternative method, works by evaluating anything non-zero as True (0.000001 -->
21	gen_strobogrammatic	line_length	gen_strobogrammatic Given n, generate all strobogrammatic numbers of length n.
22	recursive_binomial_coefficient	binomial_coefficient	recursive_binomial_coefficient Time complexity is O(k), so can calculate fairly quickly for large values
23	generate_key	get_bit	generate_key k is the number of bits in n
24	is_strobogrammatic2	str_eval	is_strobogrammatic2 Another implementation.
25	prime_check	validate	prime_check Else return False.
26	find_order	gcd	find_order Find order for positive integer n and given integer a that satisfies gcd(a, n) = 1.
27	find_primitive_root	quadratic_roots	find_primitive_root Returns all primitive roots of n.
28	factorial	pollard_rho	factorial If mod is not None, then return (n! % mod) Time Complexity - O(n)
29	factorial_recur	factorial_recursive	factorial_recur If mod is not None, then return (n! % mod) Time Complexity - O(n)
30	find_nth_digit	get_digits	find_nth_digit 1. find the length of the number where the nth digit is from. 2. find the actual number w
31	cycle_product	vol_spheres_union	cycle_product cycle index of a symmetry group), compute the resultant monomial in the cartesian pr

图 6.3:匈牙利算法赋值

由于这种方法效率低下,我们没有足够的预测来提供
对其性能的准确评价。因此,我们将跳过这一部分。

6.3.2 使用不同模型对 RepoSim 进行评估

我们的存储库示例是从 awesome-python6列表中获得的。此列表类别
按相关主题整理数百个 Python 存储库,例如算法、Au
dio、身份验证、作业调度程序、自然语言处理、机器
学习等等。它们可以用作基本事实,表示语义相似
存储库。

我们为每个存储库的函数和文档字符串生成了平均嵌入,
按照图 5.2 中描述的过程,使用不同的编程语言和
自然语言模型。对于每对可能的存储库,我们计算了它们的余弦
相似。最后,为了评估每个模型的性能,我们使用了 AUC 指标。

与其他自然语言模型不同,UniXCoder 是一个跨模态的预训练模型
能够生成文档字符串和函数嵌入。图 6.4 显示
所有模型的 ROC 曲线,而表 6.2 总结了它们的 AUC 分数。

6维塔。很棒的蟒蛇。 https://github.com/vinta/awesome-python。 2023.

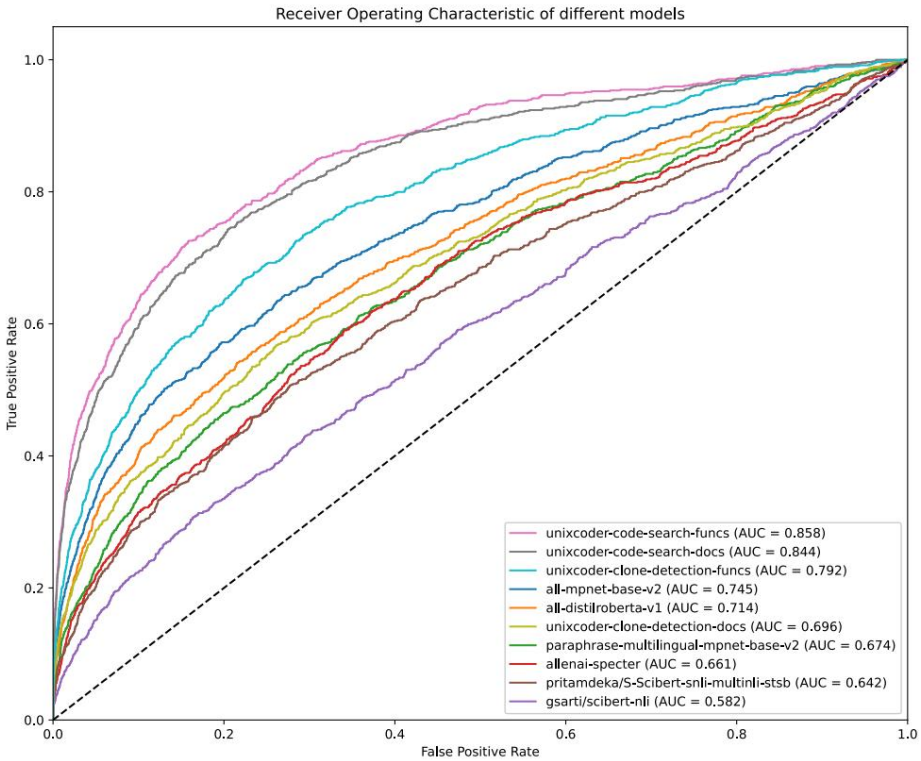

图 6.4:存储库相似性比较中不同模型的 ROC 曲线

模型	AUC	
	文档字符串嵌入代码嵌入 58.2	
gsarti/scibert-nli		
pritamdeka/S-Scibert-snli-multinli-stsb	64.2	
allenai-specter	66.1	
释义-多语言-mpnet-base-v2 all-distilroberta-	67.4	
v1	71.4	
all-mpnet-base-v2	74.5	
unixcoder-clone-检测	79.2	69.6
unixcoder-代码搜索	84.4	85.8

表 6.2:存储库嵌入比较中不同模型的 AUC

评估结果表明,我们的 UniXCoder 模型在代码搜索上进行了微调
任务,在比较存储库相似性方面具有最佳性能。

6.4 最佳表现嵌入的可视化

图 6.5 展示了 UniXCoder 生成的代码嵌入的可视化,很好
在使用维度将它们减少到二维之后,调整代码搜索任务
缩减算法称为 t-SNE7。在这个图中,每个散点代表一个存储库和
标有其名称,而散点的颜色代表一个主题。

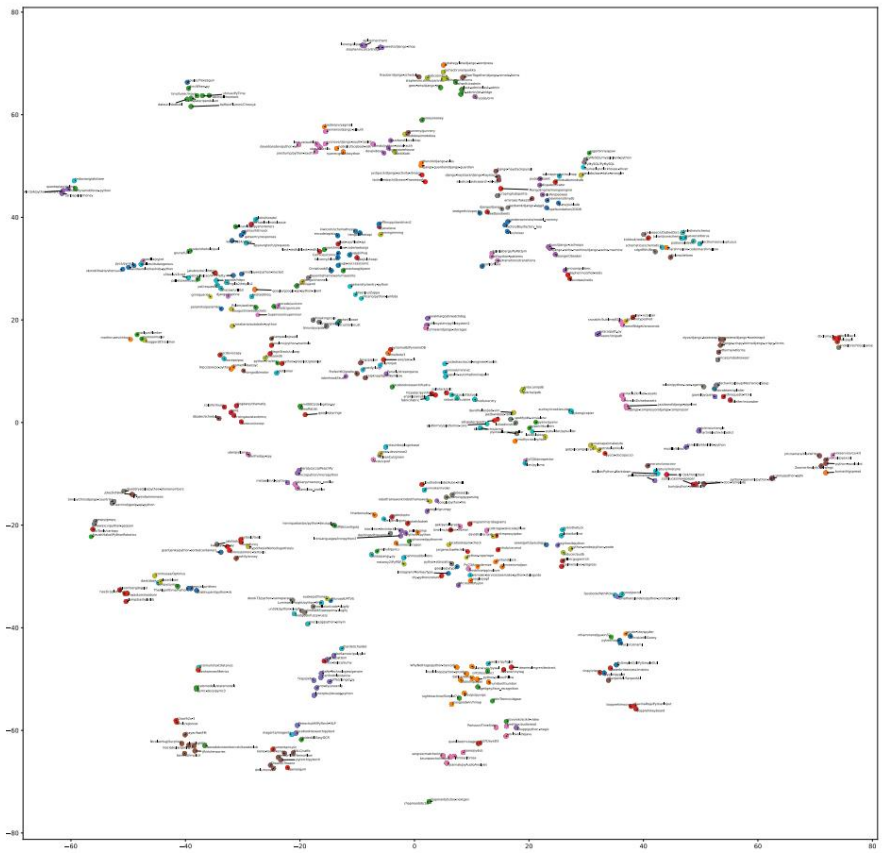

图 6.5:存储库均值代码嵌入的 T-SNE 二维可视化

7Laurens van der Maaten 和 Geo rey Hinton。 “使用 t-SNE 可视化数据”。见:机器学习研究杂志 9.86 (2008),第 2579–2605 页。网址:
http://jmlr.org/papers/v9/vandermaaten08a。
HTML。

此可视化中具有相似主题的一些存储库集群包含在
附录 A。

第7章

再现性

我们用于微调模型的硬件：

组件规格	
显卡	A40 (48GB) 1
中央处理器	15 个 vCPU AMD EPYC 7543 32 核处理器

7.1 在克隆检测上微调 GraphCodeBERT 模型

GraphCodeBERT 的微调脚本可在 orig 的 GitHub 存储库中获得
最终作者：https://github.com/snoop2head/CloneDetection。运行训练
脚本，首先安装必要的要求，并创建一个wandb(https://wandb.ai)
在罚款期间跟踪超参数、模型预测和其他相关数据的帐户
调整。接下来，更新 train.py 脚本如下：

1. 去掉第44行的参数use auth token=True。 _

2.直接在源代码中分配从wandb获得的WANDB AUTH KEY
或者通过第 131 行的 shell 环境变量。

3.将第160行的entity参数的值替换成你创建的实体名称
在魔杖上。

训练脚本会自动下载 5fold 数据集并存储在缓存中
目录。要指定缓存目录，请在训练的第 2 行添加以下行
脚本：

```
导入操作系统
操作系统en vi ron [ 变形金刚缓存 - ] = <你的缓存目录 > - -
操作系统en vi ron [- HF HOME ] = <你的缓存目录 >
导入魔杖
```

图 7.1:微调 GraphCodeBERT

如图 7.1 所示,该模型是在 5 倍数据集的第一倍上训练的

在 A40 服务器上大约 11.5 小时。为了以后对 C4 数据集的评估,我们

选择 checkpoint-22000,因为它在评估期间达到了 0.984 的最佳评估精度

微调。

7.2 在克隆检测上微调 UniXCoder 模型

UniXCoder作者提供了克隆检测任务的微调脚本:https://
github.com/microsoft/CodeBERT/tree/master/UniXcoder/downstream-tasks/clone-detection/
BCB。但是,我们需要准备自己的 Python 数据集,因为原始脚本使用
BigCloneBench,一个 Java 克隆数据集。微调 UniXCoder 克隆检测
5 倍数据集,我们上传了一个 Python 脚本,将其转换为数据集 di
微调脚本的目录。

由于 5 倍数据集的大小,微调脚本无法处理所有训练
数据不超过我们的 GPU RAM,要求我们将训练大小减少到 $\frac{1}{10}$。
在我们的 A40 GPU 服务器上进行了 10 多个小时的训练。

7.3 代码搜索微调 UniXCoder 模型

在代码搜索上微调 UniXCoder 更直接,因为微调
提供了脚本和 Python 数据集。要重现,请遵循的 AdvTest 部分
原始来源中的数据下载和微调设置部分:https://github.com/
微软/CodeBERT/tree/master/UniXcoder/downstream-tasks/code-search。如图
ure 7.2展示了在我们的A40服务器上训练UniXCoder的过程,耗时不到
3小时完成。

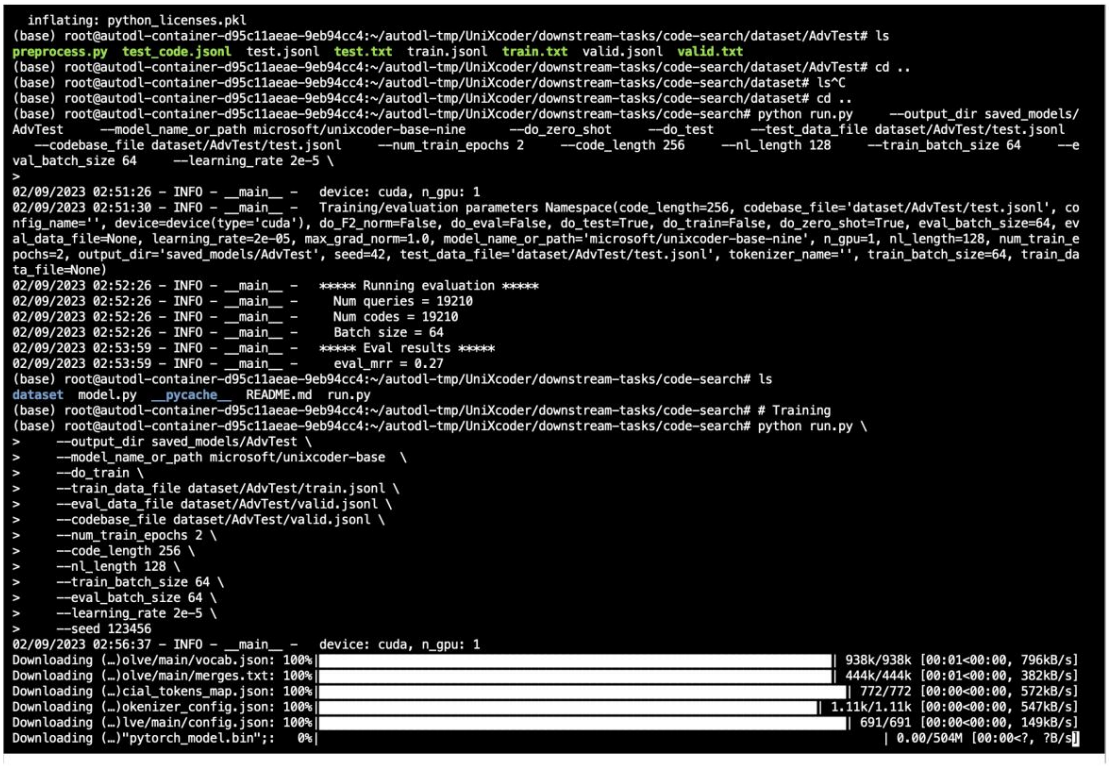

图 7.2:在代码搜索任务上微调 UniXCoder

7.4 复制存储库相似性比较

我们已将我们的工作作为 Jupyter 笔记本上传到 GitHub。笔记本目录

包含两个子目录:Cross-Encoder 和 Bi-Encoder,每个子目录都有笔记本

用于使用中描述的相应方法生成和评估嵌入

实施和评估部分。

我们的评估结果可以使用直接在 Google 上运行的笔记本进行复制

Colab（https://colab.research.google.com）。有关内容的更多信息

关于此存储库,请参考提交根目录中的 README.md 文件。

第8章

结论和未来的工作

本论文研究了 NLP 模型的演变，从递归神经网络
变形金刚及其扩展的最新进展。我们微调了两个预
经过训练的编程语言模型、GraphCodeBERT 和 UniXCoder，并进行了评估
他们在成对克隆检测任务中的表现。此外，我们设计了一种方法
有效地比较具有不同主题的存储库的语义相似性和
评估了不同语言模型嵌入在这种情况下的性能。我们的
评估显示 GraphCodeBERT 模型在
Python 克隆检测任务，而 UniXCoder 模型在代码搜索任务上进行了微调
擅长根据代码嵌入识别语义相似的存储库。

对于未来的工作，我们计划进一步完善我们的 GraphCodeBERT 模型的相似性
通过应用概率校准技术对它们进行归一化来获得分数。我们也打算
探索代码搜索任务是否更适合提取语义信息
从源代码与克隆检测任务相比。

此外，最近出现的大型语言模型如 ChatGPT、LLaMa、
巴德彻底改变了自然语言处理，使更强大的下
代码相似性比较、代码 com 等任务的站立和生成能力
完成、代码摘要等。我们渴望通过
将我们的方法应用于这些模型生成的代码嵌入并进行评估

它们在存储库语义相似性比较中的表现。这项研究可以促进
提供有关高级语言模型对这项任务的有效性的宝贵见解，以及
为领域的进步做出贡献。

附录 A

嵌入簇

本附录部分展示了所有 repos 的二维可视化中一些集群的图形
itories 的平均代码嵌入。同一集群中的许多存储库共享相似的
与他人相比的意图：

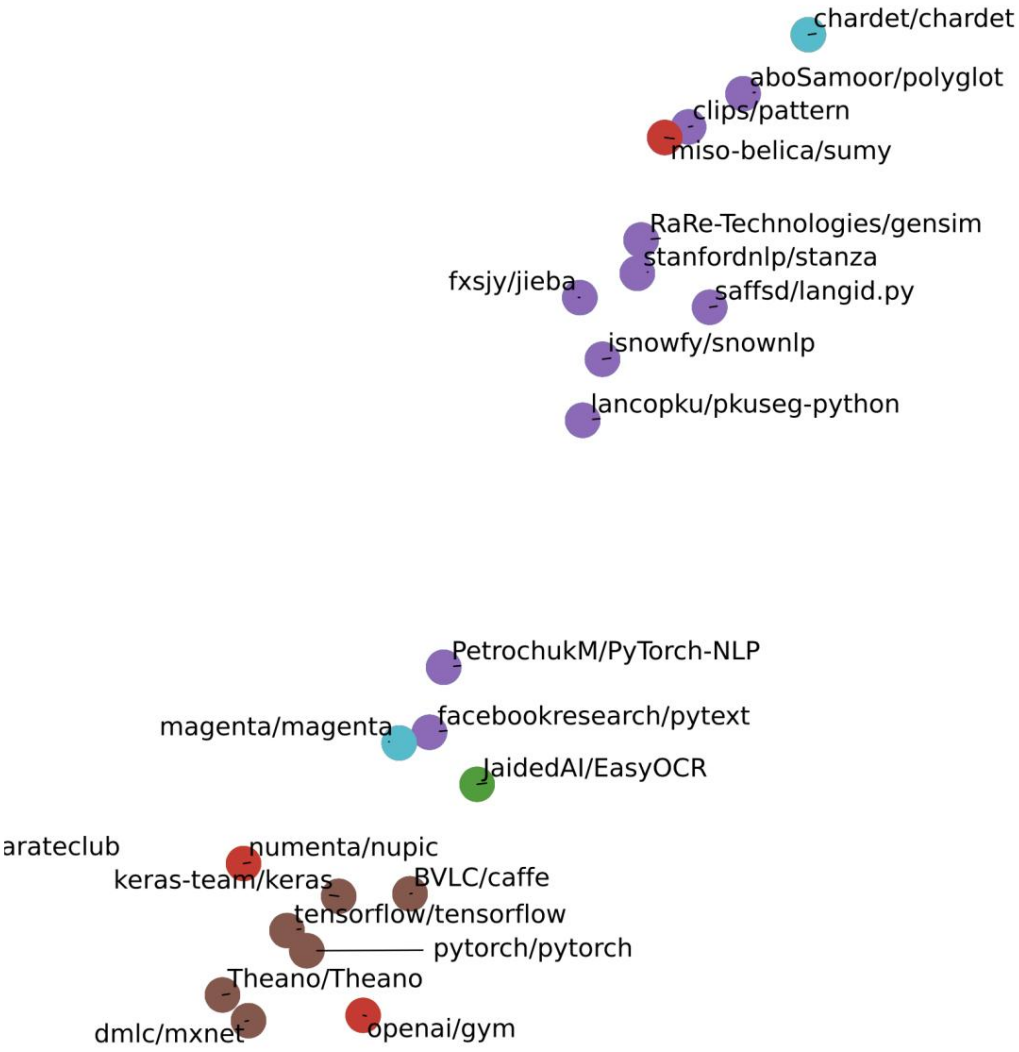

图 A.1:机器学习和 NLP 存储库集群

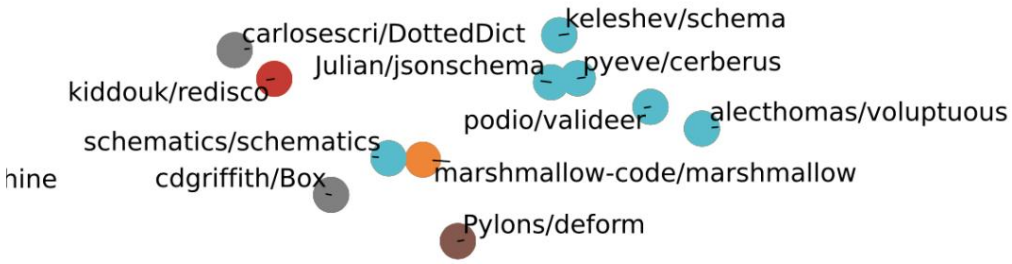

图 A.2：验证/模式存储库集群

图 A.3：OAuth 存储库集群

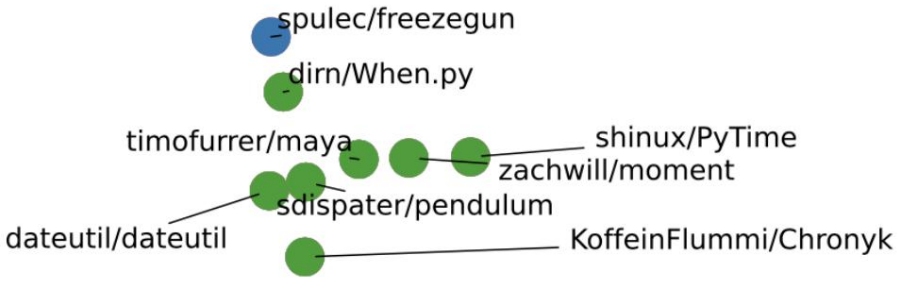

图 A.4:日期/时间存储库集群

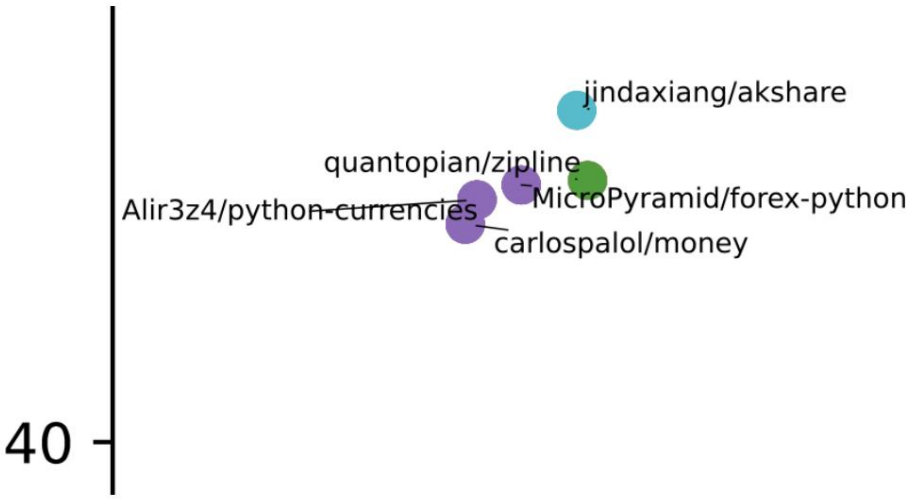

图 A.5:货币存储库集群

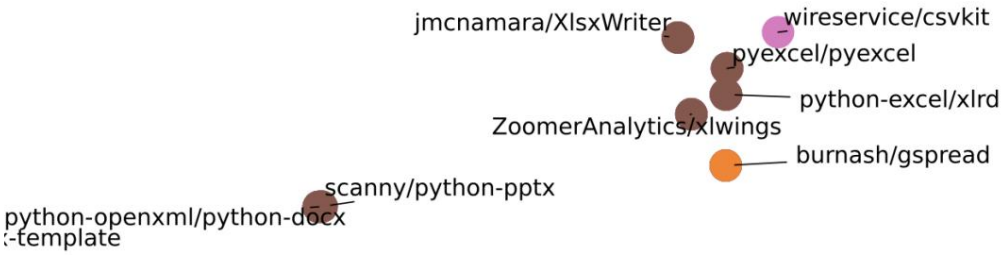

图 A.6:电子表格存储库集群

尽管只为每个存储库分配一个主题存在局限性,但嵌入可视化揭示了属于不同主题的存储库之间的隐藏关系通过他们的集群。例如,存储库 wireservice/csvkit 被标记为属于主题“CSV”,而 burnash/gspread 被标记为属于主题“第三方 API”。尽管如此,它们都与有效的存储库聚集在一起使用 Excel 文档,这是合乎逻辑的,因为它们都可用于处理电子表格。

附录 B

余弦相似度分布

图 B.1 说明了两个 UniXCoder mod 的余弦相似度分数分布
埃尔斯。与具有极化相似性得分的克隆检测模型相比
分布，代码搜索模型表现出更均匀的相似性分布，并且是
不太可能给出非常低的相似性分数。

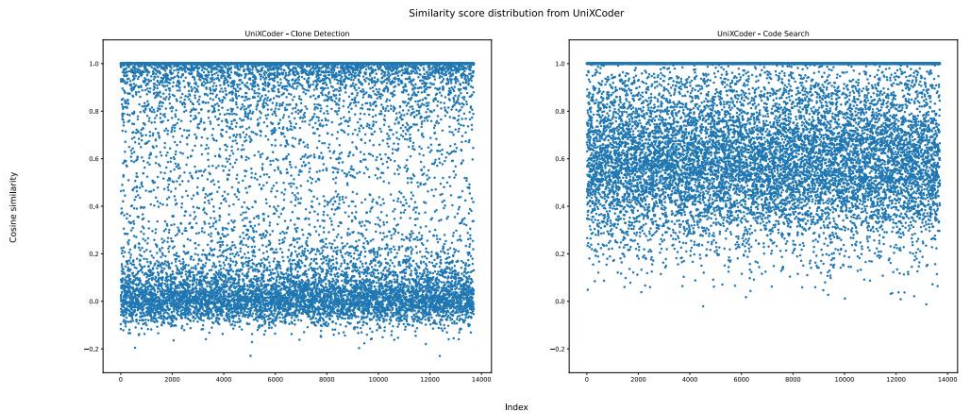

图 B.1:两个 UniXCoder 模型的余弦相似度分布