专题:利用注意力从源代码中学习存储库嵌入技术

Agathe Lherondelle* , Yash Satsangi* , Fran Silavong* , Shaltiel Eloul* , Sean Moran †
摩根大通，伦敦，英国
电子邮件。*firstname.lastname@jpmchase.com, †sean.j.moran@jpmchase.com

摘要--
源代码上的*机器*学习（MLOnCode）有希望改变软件的交付方式。通过挖掘软件工件之间的背景和关系，MLOnCode通过代码自动生成、代码推荐、代码自动标记和其他数据驱动的增强功能，增强了软件开发人员的能力。对于许多这样的任务，代码的脚本级表示就足够了，然而，在许多情况下，考虑到各种依赖关系和资源库结构的资源库级表示是必须的，例如，用主题自动标记资源库或资源库代码的自动文档等。现有的计算资源库级别表示的方法存在以下问题：（a）依赖代码的自然语言文档（例如README文件）；（b）对方法/脚本级别的表示进行天真的聚合，例如，通过连接或平均。本文介绍了*Topical*，这是一个深度神经网络，可以直接从源代码中生成公开的GitHub代码库的库级嵌入。Topical包含了一个注意力机制，将源代码、完整的依赖关系图和脚本级文本信息投射到密集的资源级表示中。为了计算资源库级别的表示，Topical被训练来预测与资源库相关的主题，在公开的GitHub资源库的数据集上，这些资源库被抓取，同时还有它们的地面真实主题标签。我们的实验表明，Topical计算的嵌入能够超越多个基线，包括通过平均或串联在仓库自动标记任务中天真地结合方法层面的表示的基线。此外，我们表明，当从现有方法产生的脚本级表示中计算资源库级表示时，Topical的注意力机制优于天真的聚合方法。Topical是一个轻量级的框架，用于计算代码库的库级表示，它可以随着主题的数量和数据集的大小而有效扩展。

I. 简介

像GitHub这样的代码托管网站已经彻底改变了代码开发，为开发者提供了一个协作环境，在这个环境中，他们可以发现相关的代码，跟踪技术进步，学习和孵化新应用的想法。开放源代码库的数量是巨大的，例如，GitHub托管了超过2亿个资源库[1].自动工具，能够更快、更有效地基于搜索访问相关的资源库，例如用语义关键词对资源库进行自动标记[1]，对其主题进行建模[2]，对于管理这一信息洪流至关重要。为有用的应用程序处理源代码的挑战由MLOnCode[3]领域解决，导致了各种诸如重复检测[4]、软件开发的设计模式[5]、代码质量和自动生成[6]，以及直接从代码中提取软件开发人员的技能组合[3]等有用的应用。开发这些工具和应用的主要挑战之一是建立一个合适的方式来表示代码，如代码嵌入或存储库嵌入[7]，[8]。

近年来，MLOnCode研究的主要重点是*方法级*或*片段级*的预测任务[9]---。
[11]的颗粒度。最近，深度神经网络[12]在理解和操作方法级代码片段方面表现出令人印象深刻的能力[3]，[9]，[10]。代码嵌入的进展可以与自然语言模型中带有*注意机制*的变换器的引入联系起来，如BERT[13]和相关模型[14]-[16]。类似的神经模型也适用于源代码嵌入[9]，[10]。[17]-
[21].这些模型在脚本或方法层面取得了令人印象深刻的结果，然而，这些工作中只有少数[1]，[7]解决了将许多脚本的信息汇总到资源库层面的表示的任务。这些方法[1]、[7]、[22]解决了汇编版本库级别表示的挑战，主要依靠代码的自然语言文档，例如，READMEs和其他伴随典型代码库的文档。

纯粹的文本信息，如在README文件或日志中可能发现的，确实拥有关于代码的重要背景，然而文本在这些模型中被用作代码本身的"代理"信息。依靠READMEs将正确和有效的文档责任放在开发者身上，可以说是增加了他们的工作。代码文档是很繁琐的，许多GitHub仓库要么完全没有README，要么有不准确或不一致的README，这并不奇怪。事实上，存储库完全没有文档，或者甚至包含"不准确"的文档（例如，使用不相关的标签来增加可见度）的情况并不少见，这很容易误导那些严重依赖这些信息的方法。

我们如何才能有效地从源代码中直接生成资源库级别的代表？本文试图回答这个研究问题。我们介绍了一个新的工具，我们称之为*Topical*，它可以学习一个灵活的资源库表示，可以应用于多个下游任务，如资源库标签和总结源代码。Topical通过利用三个领域来学习一个资源库级别的嵌入

[1] https://en.wikipedia.org/wiki/GitHub, www.GitHub.com

在一个代码库中。1）将*依赖关系*编码为脚本或库内和跨脚本或库的函数调用图，2）*逻辑代码内容和结构*，以及3）代码中的相关*文档字符串和方法或文件名*。Topical为每个脚本文件生成这三个领域的嵌入，并通过使用注意机制产生一个资源库级别的表示。为了训练Topical，我们确定了自动标记公开可用的Github资源库的任务，因为这些资源库及其策划的标签在Github上是公开可用的。我们抓取了这些Github资源库的数据集，以及它们的地面真实标签。我们的实验表明，Topical及其关注机制在自动标记代码库方面明显优于平均或串联方法级表示的基线。Topical并不依赖开发者生成READMEs与他们代码的自然语言文档，因此减轻了软件开发者的代码文档负担。

综上所述。

- **我们提出了Topical：一个基于注意力的深度神**经网络工作架构，它从代码库的三个领域提取并结合信息。1）依赖性。(2)代码内容；(3)生成资源库级别的表述的文档串。Topical与GitHub爬虫一起，提取资源库和策划的"特色主题"，用于训练Topical产生资源库级别的嵌入。
- **我们表明，Topical可以作为一个通用的轻量级框架，**通过结合任何其他现有的方法级表示法，例如import 2vec[8]，来计算资源库级别的表示法。我们的实验表明，采用方法级表示法的平均/总和或串联来计算资源库的

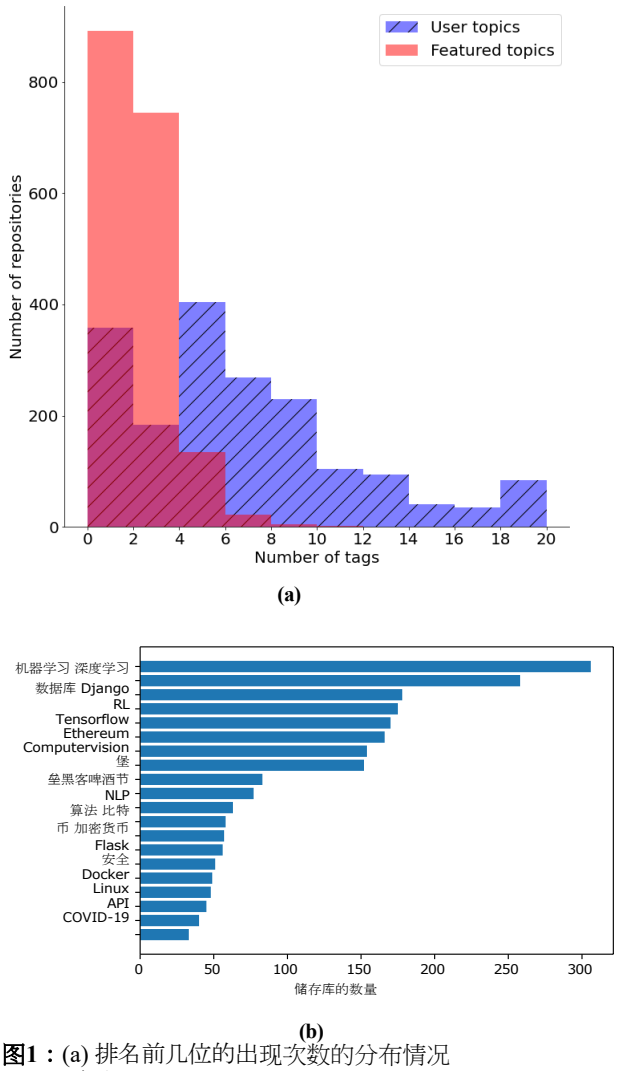


图1：(a) 排名前几位的出现次数的分布情况20个特色主题（红色），用户主题（蓝色），模糊匹配（混合）之前和之后。(b) 数据集中各主题的资源库分布。

与Topical所采用的注意力机制相比，水平嵌入会受到很大影响。

- **我们表明，在GitHub资源库自动标记任务中，Topical**的表现优于多个基线。不仅仅是Topical优于现有嵌入的库级表示，Topical也优于通过使用注意力机制和现有方法级表示生成的库级表示。

本文的其余部分组织如下：在第二节中，我们介绍了爬虫和我们的数据集，并讨论了Topical模型的架构。随后，在我们的实验结果中（第三节），我们在数据库上对Topical进行了基准测试，并与TF3D进行了比较，TF3D是一个新颖的基线模型，我们认为它是与Topical进行比较的有力依据。在第四节中，我们提供了结论和在该领域进一步研究的指针。

II. 方法论

A. 数据集和GitHub爬虫

我们首先生成一个资源库级别的数据库，以及资源库的注释主题。现有的开放源代码数据集[11],[21]被指定为文件或方法级别的任务，不包括研究文件集合的足够信息，如导入、元数据和提交/git历史。在这个阶段，我们只考虑Python资源库，但是我们的方法可以很容易地扩展到其他语言。据我们所知，没有任何基准数据库包括可以直接应用于本研究的资源库和它们的注释。为了促进我们的研究和未来对源代码主题建模的研究，我们建立了一个GitHub爬虫工具，用于从开放源码库中生成数据库，以及为本研究组成的数据库。

GitHub仓库通常由其所有者使用用户定义的主题进行分类，这些主题可能包含缩写、错别字和重复的内容。由于主题名称差异较大，GitHub还定义了480个特色主题，这些有限的预定义主题将由仓库的

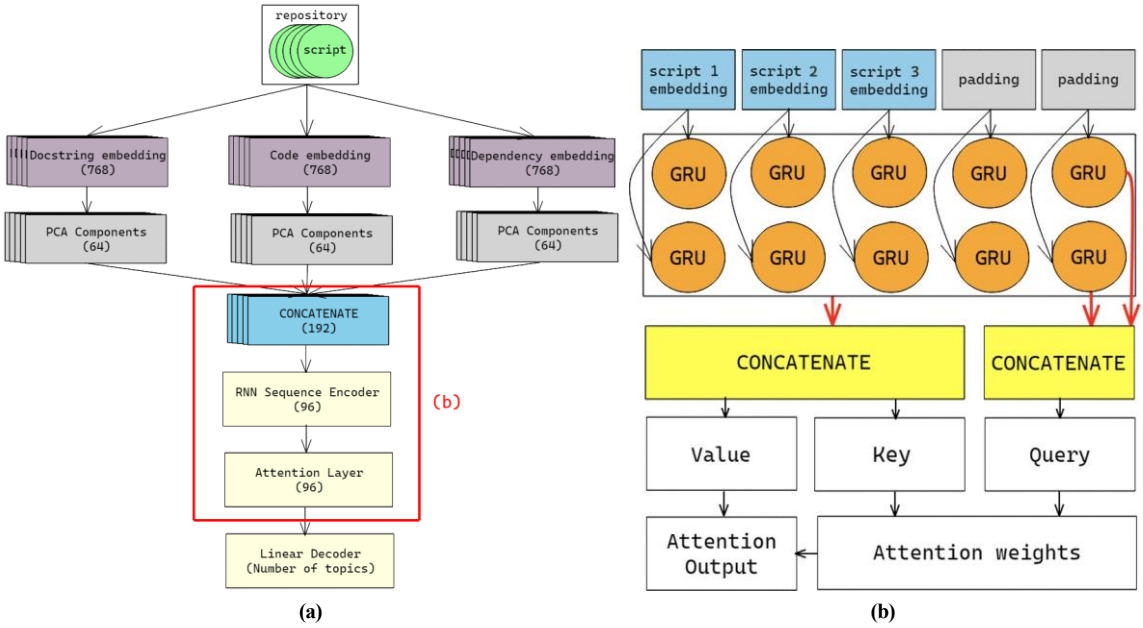


图2：（a）包括下游分类层的一般模型图。(b) 脚本嵌入上的RNN序列编码器与Attention层，以计算存储库的单一嵌入表示。

所有者。为了有一个标准的标签集，爬虫使用字符串匹配法[23],
[24]将用户定义的主题映射到GitHub的特色主题，依靠"模糊匹配"（Levenshtein比率[25]）的单词距离的阈值（90%）来识别相似的主题。图1a比较了映射到特色主题前后与资源库相关的主题数量的分布。图1a中具有一定数量标签的仓库数量的减少表明，GitHub用户倾向于将许多主题与他们的仓库相关联，这些主题实际上可能非常相似，以提高他们在GitHub搜索引擎中的可见度。请注意，大多数用户定义的主题都与一个特色主题有效地相关。此外，大量（约32%）的仓库被发现与任何主题都不相关，当限制在特色主题时，这个数字还会增加。因此，Topical可以帮助用户在有限的集合中自动生成主题标签的过程。为了收集数据集，我们从20个主题的初始集开始（例如，ML、NLP、数据库等），对于每个主题，爬虫收集与之相关的固定数量的资源库。由此产生的数据集由大约3000个资源库组成，所有资源库中共有大约92383个python脚本。然而，这些资源库中约有32%没有一个特色主题。由于一个被抓取的资源库可以有多个特色主题，因此特色主题的分布会向某些主题倾斜。在最后的分类任务中，选择了前20个最频繁的特色主题，与这些主题对应的数据集包括1600个资源库。最后，主题分布如图1（b）所示。该分布显示，与前几个主题相比，后几个主题相当罕见，但它们仍然是
构成足够的资源库，因此，分类器要想获得总体上的好成绩，就必须对所有的主题进行准确分类。

B. 模型

我们的主要目标是能够在资源库层面上表示源代码。因此，我们的模型，Topical，由编码器和基于深度神经网络的注意力机制组成。图2展示了该模型的各个组成部分的架构图。该模型可以分为三个阶段。在第一阶段，我们将脚本嵌入到一个资源库中。为此，我们利用预先训练好的BERT基础转化器模型，为资源库中的每个脚本生成嵌入向量，特别是为每个脚本生成三个嵌入向量，如后面所述（第二部分-
C）。在第二阶段，我们引入了一个关注机制，从对其每个脚本计算的嵌入中获得一个集体嵌入。在最后一个阶段，我们增加了一个分类器单元，用于对资源库主题进行多标签/单标签分类的任务。我们在下面详细说明这些主要阶段的每一个。

C. 脚本级嵌入

Topical利用资源库中的三个领域作为编码器的输入：源代码的内容和结构，源代码中的文本信息，以及通过对脚本中的方法、类或外部库的方法调用而形成的脚本之间的依赖图。代码内容和文本信息（*如*文档字符串、文件名）通常被用于代码分类[10]。依赖图域捕获了许多库的结构和模式，不仅对主题标签有用，而且对软件架构中的模式识别也有用[8]。因此，我们认为，使用所有这三个领域将提供一个多用途的和

全面嵌入一个资源库。图3说明了脚本中包含的不同领域的代码（图3a）被分离和处理的方式（图3b），然后使用适当的表示 信息和标记器进行标记（图3c）。

1) 代码内容和结构。代码内容是使用GraphCodeBERT RoBERTa基础[10]嵌入-演绎的。Graph-CodeBERT基础是一个在多种代码语言上预先训练的BERT模型，它的标记器结合了方法的原始代码内容和"数据流 "信息。数据流"提供了一个代码方法中变量的浅层关系图。在这种情况下，我们对每个脚本使用512个输入标记大小，因为这是使用预训练的 GraphCodeBERT的默认值。

2) 文本信息--Docstrings 嵌入。虽然GraphCodeBERT也处理代码中的注释，但在这里我们指定了一个单独的嵌入，用于从注释和预处理的方法名称中检索文本信息。这使我们能够对源代码中的文本信息进行预处理，以实现资源库级别的任务。我们提取文档字符串、函数文本名称和文件名称。为了得到一个固定大小的最终嵌入，我们将文件名和方法名整合到一个句子中，并使用特殊的标记将它们与脚本文档串分开，并将标记化的输入编码到DistilBERT[26]（一个自然语言em-bedding，它是一个预先训练的英语语言模型）。与代码嵌入类似，我们将脚本中的注释和函数名称连接起来，并使用最大512个标记，作为DistilBERT的向量输入。

3) 依赖图的嵌入。以前的工作已经探索了图书馆导入语句，通过在版本库层面将加载的包列为抽象树，来获得导入的图书馆作为依赖关系的嵌入[8]。然而，它只依赖于资源库中的软件包装载声明。这可能在很大程度上误导了代码中的实际代码通信和库的使用。在这里，我们引入了软件仓库中方法和脚本之间的完整通信图的嵌入。对于每个脚本，我们检索其相应的边，这些边将脚本中的所有方法与其他类和其他方法联系起来，这些方法在同一个软件库中实现，但也在外部库调用中实现，如图3a-b所示。为了获得这样的图，我们利用PyCG，这是一个开源的库，可以从静态的python代码中提取依赖图[27]。由于方法和包名的描述性，我们使用预先在英语上训练的DistilBERT模型来嵌入图。

为了将依赖图标记到DistilBERT模型中，我们专门用一个特殊的标记来表示图中两个节点之间的联系，一个方法名称和它的类导入或其他方法的使用。然后，图中的所有第一等级的边都会与分离标记依次串联，然后再传递给DistilBERT模型。图3b展示了我们如何从PyCG输出中检索节点，图3c展示了我们在引入*[C]*作为DistilBERT特殊标记后如何对其进行标记。在一个训练有素的自然语言模型中，路径作为一个句子的用法被认为是可取的

```python
from collections import Counter
import time
import logging


def all_unique(nums):
    """returns True if all elements of a list
    are unique otherwise, return False"""
    begin = time.time()
    count = dict(Counter(nums))
    for i in count.values():
        if i > 1:
            logging.info(time.time() - begin)
            return False
    logging.info(time.time() - begin)
    return True
```

(a)

```
{
    "test_pycg": [],
    "test_pycg.all_unique": [
        "collections.Counter",
        "<builtin>.dict",
        "time.time",
        "logging.info"
        ],
    "time.time": [],
    "collections.Counter": [],
    "<builtin>.dict": [],
    "logging.info": []
}
```

(b)

```
['test','_','p','##y','##c','##g','all','_','unique','[C]
','collections','counter','time','time','logging','info']
['returns','true','if','all','elements','of','a',
'list','are','unique','otherwise',',','return','false']
['from','Ġcollections','Ġimport','ĠCounter','Ċ','import',
'Ġtime','Ċ','import','Ġlogging','ĊĊ','Ċ','def','Ġall',
'_','unique','(','n','ums', ...]
```

(c)

图3：（a）代码片段。(b) PyCG输出的代码片段的依赖关系图。(c) PyCG依赖关系、文档串和源代码的标记化序列。

因为我们期望得到与函数调用（也是词）之间的距离有"某种 "关系的嵌入。

D. 存储库级嵌入

Topical应用注意机制，从不同的预训练（GraphCodeBERT，DistilBERT）BERT模型嵌入中产生一个透明的嵌入，如上文所讨论的，针对资源库中的每个脚本。该模型的详细方案显示在图2a-b。我们减少了每个嵌入的组件的数量，以便为我们的最终脚本表示作出贡献。这主要是为了减少计算量，使我们的模型能够有效地扩展到大数据集。减少嵌入中的成分数量可以减少参数数量，并可以根据具体的下行任务进行优化。我们使用PCA（主成分分析）将768个嵌入向量的维度降低到192维。降维后，我们使用一个注意力单元将来自不同脚本的信息结合到一个单一的密集的资源库表示。该注意单元包括一个RNNs序列编码器，然后是一个自我注意机制[28]。我们使用双向门控递归单元（GRU）[29]作为递归单元，因为之前的研究[30]倾向于使用GRU单元来实现更好的性能。

在大序列的小数据集上。GRU单元允许将许多脚本嵌入为一个代表资源库的序列。自我关注层允许Topical优化脚本的权重分布。这对分类主题特别有用，因为"主题"可以表现在资源库的一小部分，或者通过独特的关系---

脚本集合中的主题之间的运输。我们的注意力范例详见图2b。对于一个给定的资源库R，我们得到$R^d = \{x_0, x_1, \ldots, x_n\}$，作为脚本嵌入对于每个域$d$，其中$d$属于任一代码结构。docstrings，或依赖性。x_t，因此是一个单一的获得的脚本嵌入，其中t是来自资料库的脚本序列中的脚本位置，它被用作GRU隐藏层的输入。由于我们在这里不考虑脚本序列的顺序，我们利用了对双向GRU的关注。前向GRU的隐藏层被表示为$\overrightarrow{h_t}$，而后向GRU的隐藏层则表示为$\overleftarrow{h_t}$，在这里我们计算一下。

$$\overrightarrow{h_t} = GRU(x_t, \overrightarrow{h_{t-1}}) \quad (1)$$

$$\overleftarrow{h_t} = GRU(x_t, \overleftarrow{h_{t+1}}) \quad (2)$$

对于一个由n个脚本组成的资源库，我们检索两个隐藏层的最后一个隐藏状态，并将其连接起来，如下所示。

$$h_n = [\overrightarrow{h_n}, \overleftarrow{h_n}] \quad (3)$$

我们还检索了GRU的输出，它是其所有隐藏状态的张量。

$$y = [h_i]_{0 \le i \le n}, h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}] \quad (4)$$

事实上，h_n包含所有其他隐藏状态的信息，因此允许代表整个脚本。然后，y被用作注意力层的键和值，而最后的隐藏状态h_n被用作查询。

*1) 从资源库中抽出脚本。*由于y应该有一个固定的形状，在脚本总数小于n的情况下，我们在序列中加入填充嵌入（*例如*充满零的脚本嵌入）。在这项工作中，我们根据经验选择n=15作为嵌入单个资源库的最大脚本文件数。仓库可能包含更多的脚本，但使用低数量可以包括正在工作中的原始仓库。如果我们选择从大型资源库中随机抽取脚本，那么来自复制的第三方库的文件有可能会在抽样中占主导地位。为了尽量减少这种影响，我们再次利用PyCG对嵌入过程中的脚本进行采样。这是通过使用资源库中最高目录的路径来实现的，并选择涉及到函数调用路径的脚本。脚本是按照它们在路径中出现的顺序来检索的。在探索完路径后，我们随机抽取一个新的路径，直到我们填充完所有的15个脚本。

在计算注意力输出时，我们在这些填充嵌入上应用一个注意力屏蔽[28]。屏蔽矩阵计算将注意力权重设置为0，在填充嵌入上使用注意力作为。

$$F = softmax(\frac{Q \times k^T + M}{\sqrt{d_k}}) \times V \quad (5)$$

其中M的掩码矩阵为。

$$M_{t,i} = \begin{cases} 0 & \text{如果} x_t \text{是一个脚本嵌入} \\ -\infty & \text{如果} x_t \text{是一个填充嵌入物} \end{cases} \quad (6)$$

和Q是查询矩阵，k是密钥向量，V是值矩阵，如图2(b)所示，d_k是密钥向量的尺寸。

*2) 多标签分类。*GitHub上的大多数资源库都属于一个以上的主题。此外，一些特色主题是其他主题的子主题（图1b）。例如，专注于NLP的资源库可能会被归入更广泛的机器学习主题。我们根据它们的频率挑选了5-20个有代表性的主题，这些主题也可以与其他主题分开找到。为各种测试选择的主题在后面的表一中进行了总结。为了实现多标签分类任务，我们在注意力机制的基础上增加了一个线性层，与一个sigmoid激活函数相配。整个架构被训练成预测和地面真实标签之间的交叉熵损失最小。使用验证集，我们在解码器的sigmoid输出上固定一个阈值。这个阈值是通过在验证集上最大化地面真相二元标签向量l和分数输出向量s之间的F1分数来优化的，转化为二元标签向量$\hat{l}$。对于每个主题i和输出分数s_i。预测的主题标签l_i，计算方法如下。

$$\hat{l}_i = \begin{cases} 1 & \text{如果} s_i \ge \textit{阈值} \\ 0 & \text{否则} \end{cases} \quad (7)$$

E. 基线

为了给我们基于注意力的模型提供结论性的结果，我们开发了多个有竞争力的基线模型，TF3D、GraphCodeBert和Import2vec。TF3D是一个基于术语频率的模型。这个模型允许我们将Topical与一个非深度学习模型进行比较，但具有类似的嵌入信息（源代码、文档串和依赖关系）。第二个基线，GraphCodeBERT只使用基于代码内容的资源库的嵌入。最后，我们在四个新的变量中使用资源库嵌入模型Import2vec，这些变量是建立在预先训练好的Import2vec模型之上的。这些基线的表现突出了Topical在与现有的脚本/文件级嵌入相结合时的灵活性和有效性。

*1) TF3D - 一个统计学的NLP基线。*为了将具有注意力机制的深度BERT嵌入与传统的统计模型进行比较，我们开发了一个有竞争力的统计基线，称为TF3D。TF3D是基于代表术语频率，如TF-IDF，但在这种情况下适应于脚本/方法的集合。典型的基于统计术语的源代码模型最近被证明对源代码分析和相似性检测是有效的[31]-[33]。

与注意力模型类似，我们结合了三个源代码特征域：（a）代码结构，通过使用代码中每个方法的AST（抽象语法树）特征；（b）*文档字符串*，这是方法层面的任何注释，以及函数名称，最后，（c）脚本文件的依赖/库。我们将一个资源库表示为n个方法的集合
与它们在资源库源代码中的相应特征向量（m），i，R^d = m_1 ,i { m_2 , } ...., m_n
其中d属于代码结构、文档串或依赖关系。对于每个d中的每个资源库，R^d i
，我们使用聚合来表示一个资源库中特征的概率向量。

$$S_t^d = \frac{Pj\ mj}{¥m¥} \tag{8}$$

通过对N 个 资源库的训练集R_0 ， R_1 ， . . ., R_N { 我们通过使用S^d 的对数的算术平均值来计算n_T主题中每个主题（T）的术语矩阵$C(3\ n_T)$。

$$C(d, T) = \frac{(\ln S_{i \in T})}{(\ln S_{i \in /T})} \tag{9}$$

请注意，这里我们修改了标准的TF-IDF[34]和
引入对数来惩罚R_i
中的脚本或方法中过度重复的术语，这些术语会支配频率向量（而不是像通常使用的那样惩罚频率的倒数）。方程9让人想起信息检索领域的清晰度评分[35]，该评分衡量标记分布与背景的不同程度。然后，我们计算余弦相似度，以获得代表训练和测试中每个资源库i的嵌入矩阵。

集。$E_i\ (3 \times n_T) =$ $\frac{CT}{\text{登宁省}}$.最后，我们用嵌入的方式矩阵进行分类，使用标准的随机森林
regressor分类器对库房进行多标签标记。

2) GraphCodeBERT： GraphCodeBERT本身结合了代码内容，其*数据流*，以及在方法中发现的注释。GraphCodeBERT嵌入了各种代码相关任务的方法级源代码。因此，我们用它作为一个基线来展示我们的注意力模型的效率。

值得注意的是，我们已经试验了其他的方法级源代码嵌入模型，如使用注意力掩码的CodeBERT注入代码AST。在该模型中，CodeBERT只依赖于源代码内容，而使用AST图连接作为注意力掩码能够捕获代码数据流。然而，GraphCodeBERT提供了明显更好的结果，因此我们只报告了GraphCodeBERT作为竞争基准，它使用预先训练的方法级嵌入来表示一个资源库。与结合了文档串、代码和依赖性嵌入的Topical相比，GraphCodeBERT基线只使用 代码嵌入。

*3) Import2Vec。*我们比对的最后一条基线是一个基于Import2vec嵌入的模型[8]。这组基线也显示了Topical架构如何被使用
与现有的基于方法的嵌入，产生库级嵌入。Import2vec为软件库提供矢量嵌入，由脚本导入，基于
这些库之间的语义相似性。学习这种嵌入的想法是为了捕捉类似的软件库，基于它们被同时导入的频率。这些表征是通过训练一个深度神经网络来预测一对软件库在代码库的大型数据集中至少一个源文件中被一起导入或不被导入的概率而获得的。Theeten等人[8]提供了预训练的Import2vec嵌入，为大多数现有的python库/包返回一个vector表示，例如numpy、tensorflow等。

为了将Import2vec嵌入用于下游任务，如我们案例中的分类，我们首先提取了

存储库中导入的所有软件库的列表。
对于每一个被版本库导入的库，我们获得它的Import2vec向量。这些向量的维数可以在60到200之间变化。[8]报告说，嵌入尺寸为100就足够了，所以我们在这项工作中把向量的维数设为100。将这些向量
，用于分类的直接方法是取其平均值/均值或通过
将它们按固定顺序连接起来，然后使用分类器。我们使用注意力机制将这些向量组合为
在Topical架构中提出。这可以通过代表每个资源库$R = x_0$, x_1 , . . ., x_n
作为资源库导入的Import2vec向量的列表。{ 这个资源库R } 现在可以用2.4节中描述的同样方法来处理。我们得到4个变化的
作为基线的Import2vec嵌入。

- **I2V-conc-linear:**将一个资源库导入的所有库的Import2vec embeddings按照预定的顺序串联起来，然后训练一个线性神经网络。
  用于自动标记存储库的层。
- **I2V-conc-attn：**以与i2V-conc-linear相同的方式对Import2vec嵌入进行串联，然而，这些嵌入向量然后使用Topical
  attention机制架构进行组合，如图2（b）所示，并在第2.4节中描述。
- **I2V-mean-linear:**由一个存储库导入的所有存储库的Import2vec em- bedding向量的平均值，之后是线性层
- **I2V-mean-attn：**由资源库导入的所有库的Import2vec嵌入向量的平均值，然后由Topical attention机制处理。

*4）评价指标。*在评估中，我们比较了基于阈值的分类整体性能的F1分数，也评估了LRAP（标签排名平均精度）分数，该分数计算了基于排名的平均精度（不需要使用阈值）。这是文献中用于评估多标签分类的一个流行指标[1]。
[36] LRAP的计算方法如下。

$$LRAP(y, \hat{f}) = \frac{1}{\text{实验样本}} \sum_{i=0}^{nsamples-1} \frac{1}{||yi||_0} \sum_{j:yij=1} \frac{|L|ij}{\text{级别}ij} \tag{10}$$

$$y \in \{0, 1\}^{nsamples \times nlabels}$$

$$\hat{f} \in^{nsamples \times nlabels}$$

$$L_{ij} = \{k : y_{ik} = 1, \hat{f}_{ik} \geq \hat{f}_{ij}\}$$

$$\text{等级}_{ij} = |\{k : \hat{f}_{ik} \geq \hat{f}_{ij}\}|$$

其中y是地面真实标签的二元指标矩阵，$\hat{f}$是每个标签的预测分数向量。等级$_{ij}$ ，提供了有序预测向量的索引，$_{ij}$ ，是所有等级以上的索引的真实预测数$_{ij}$ 。因此，$|L|$ 所有样本的比率的平均值给出了一个可靠的多标签分数的度量。LRAP通过首先对模型预测的标签进行排名，然后推理出正确标签出现的顺序来衡量多标签分类的质量。如果正确的标签出现在最高等级，那么就会得到1分（或接近1分），但是如果正确的标签出现在较低的等级，那么根据它们的等级，就只能得到1分的一小部分分数。

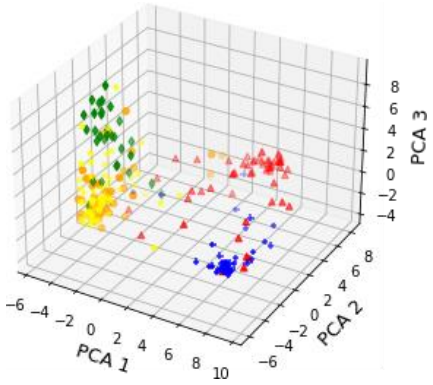
F1分数。我们报告F1分数（除了LRAP），因为它是分类任务中最常见的报告和最容易理解的指标之一。我们报告了通过计算整个数据集的全局真阳性、假阳性和假阴性而计算的微观平均F1分数[37]。一般来说，LRAP和F1的区别在于它们惩罚分类器错误的方式。LRAP通过考虑标签的等级来惩罚这些额外的标签分配，而F1则通过计算全局精度来惩罚这些分配，以0-1的方式。

III. 结果

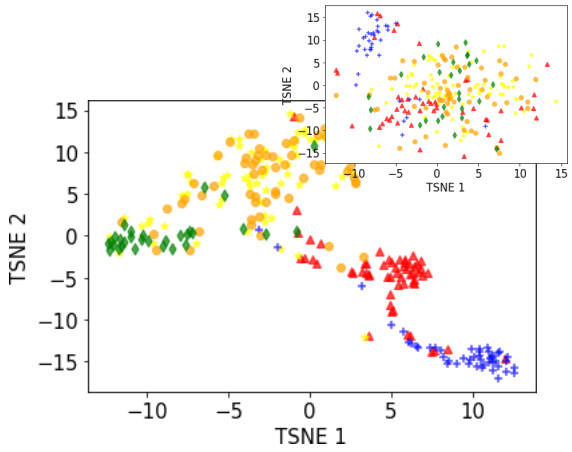
我们首先分析了我们模型的嵌入输出。在第二部分，我们比较了各种基线在不同数量的主题上的自动标签的性能。我们还研究了使用注意力机制的算法与标准聚合的性能对比。这些实验背后的主要想法是：(a)考察注意力机制的性能和有效性，以及
(b)
使用依赖性和文档串嵌入与代码嵌入相结合的效果。最后，我们比较了各种算法在数据集大小上的表现，以比较它们在数据上的扩展程度。所有的算法都是通过最小化交叉熵损失来训练的，固定的学习率为0.002，使用ADAM优化器和权重衰减[38], [39]。

图4介绍了使用注意力模型对Topical的资源库嵌入的潜在可视化。我们嵌入了从GitHub上直接抓取的5个流行主题（见表一）的资源库。我们在二维潜伏空间上使用TSNE投影，在三维投影中使用PCA的前三个成分。图4显示了在对GitHub主题进行分类的任务中，来自Topical的嵌入pro-
jections之间明显的分离。值得注意的是，两个潜伏成分并不能完全捕捉到存储库之间的抽象差异，如存储库的背景或主题。然而，与没有注意机制的嵌入相比，Topical嵌入的分离仍然明显更明显（图4b）（图4b的插图）。

PCA投影（图4a）至少在视觉上也认可了我们在分类器头之前降低嵌入维度的决定。图4b的插图也显示了



(a)



(b)

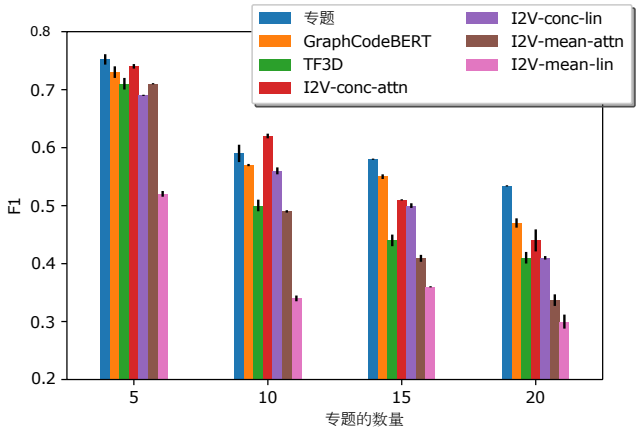
图4：(a)按频率排列的前5个主题的存储库嵌入的三维PCA投影。(b)前5个主题的存储库嵌入的二维TSNE投影。内页显示了没有关注机制的前5个主题的二维TSNE投影嵌入。前5个主题。ML、DL、数据库、Django和RL分别由橙色、黄色、红色、蓝色和绿色的点表示。

没有注意机制的嵌入的投影。它表明，使用平均嵌入（*即*没有注意机制）可以是有用的，但在聚类库的类似嵌入方面的效率远远低于亲身提出的注意模型。

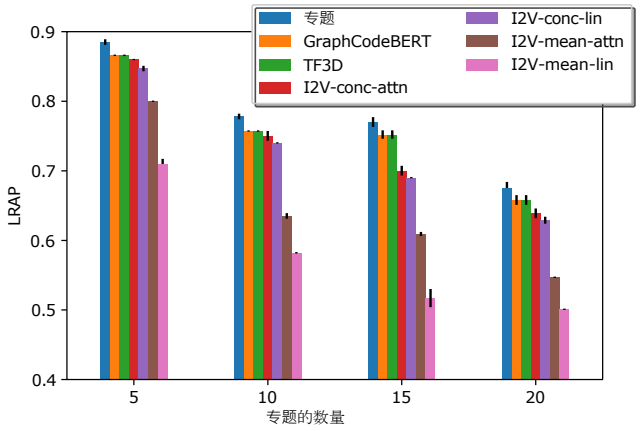
在第一个实验中，我们比较了各种基线对5、10、15、20个主题进行多标签分类的性能。表一和图一分别列出了数据集中的主题和它们的频率。与基线相比，Topical的分类结果显示在图5中。Topical显示出总体上比TF3D和GraphCodeBERT基线更好的F1分数，而LRAP分数则高出6%-10%。在F1分数方面也是如此，Topical比其他基线做得更好。

总的来说，基于Import2vec的基线表现比GraphCodeBERT和TF3D略差，这表明处理完整的源代码比只看存储库导入的软件库有明显的优势。在这个实验中显示的第二个重要的见解是，当把预训练的嵌入与注意力机制结合起来时，可以导致显著的性能提升。对于

例如，与串联和平均等天真的聚合机制相比，使用Topical attention时，Import2vec嵌入的性能明显增长。各种主题数量的结果所揭示的另一个趋势是，随着主题数量的增加，性能开始下降。然而，一个可能的原因是类似主题之间的重叠。



(a)



(b)

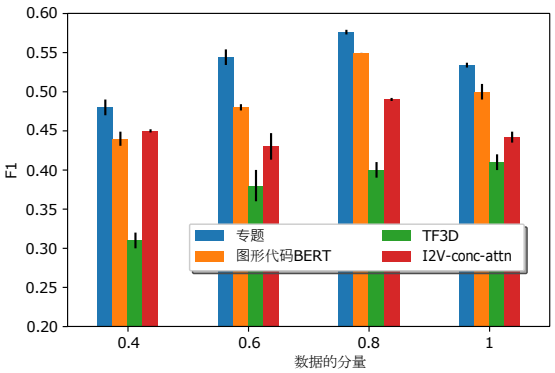
图5：(a)不同基线对5、10、15和20个主题的自动标签的F1分数比较和(b)LRAP分数比较。

A. 不同的数据大小的影响

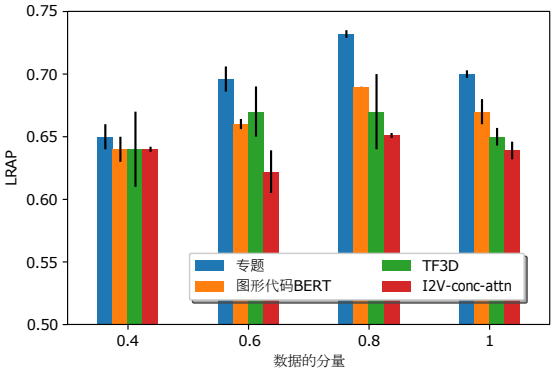
我们的目标是提供一个工具，在与资源库嵌入相关的各种任务上容易训练。因此，我们将我们的模型与各种规模的训练集的基线进行比较评估。图6显示了F1分数和LRAP作为

表一:报告实验的主题和数据集的大小（存储库的数量）。使用70-30%的训练-
测试比例来生成训练和测试数据。图中报告/显示的指标是根据测试数据计算的。

专题数量	数量 储存库	选定的主题
5	760机器	学习(ML), 深度学习(DL), 数据库, Django,Rein-Learning. 强化学习（RL）。
10	1200	ML, DL, 数据库, Django, RL, Tensorflow(TF), Ethereum, Computer Vision (CV), Bot, Hacktoberfest
15	1376	ML, DL, 数据库, Django, RL, TF, Ethereum, CV, Bot, Hacktoberfest, 自然语言处理(NLP), 算法, Bitcoin, Cryptocurrency, Flask
20	1586	ML, DL, Database, Django, RL, TF, Ethereum, CV, Bot, Hacktoberfest, NLP, Algorithm, Bitcoin, Cryptocurrency, Flask, Security, Docker, Linux, API, Covid-19



(a)



(b)

图6：（a）在40%、60%、80%和完整的数据集上，20个主题多标签分类的F1分数比较（b）在40%、60%、80%和完整的数据集上，20个主题多标签分类的LRAP分数比较。

20个主题分类的训练集部分的函数。我们观察到，即使我们使用深度嵌入注意力模型，但只要有少量的

储存库以获得高性能。这意味着训练和抓取成为一项即时和低成本的任务，即使是训练和抓取新主题的数据集或其他分类任务。TF3D，正如预期的那样，在小的数据体系中是有用的，但随着训练集的增加，总体来说，依靠资源库的深度嵌入的注意力模型，提供了更高的分数。随着数据集规模的增加，各种模型的性能也在稳步上升，但也慢慢趋于饱和。在未来，随着我们收集更多的数据，我们打算增加Topical使用的参数数量，希望能获得更好的性能。在更大的数据集中，可以进行进一步的参数优化，以进一步提高性能，如调整降维大小和注意力大小。

B. 讨论

结果部分回答了我们的主要研究问题，该问题可进一步细分为以下问题。

- 我们怎样才能从代码库的不同信息源中提取有用的表示？为了回答这个问题，我们提出了Topical，它提取并结合了方法级表示和依赖关系图来计算版本库级表示。我们的结果表明，Topical优于其他基线，包括仅基于源代码信息的表示，以及在自动标记任务中导入的库。
- 我们如何结合方法层面的表述来生成资源库层面的嵌入？我们的结果表明，使用Topical可以从现有的方法级表示法（import2vec）中生成资源库级表示法。不仅Topical与这些现有的表征兼容，而且通过Topical计算的资源库级表征在集合方法级表征方面优于其他方法。

表二描述了三个基于注意力的基线的额外指标，强调了它们之间的质的区别。在表中，精度（测试库的平均值）[37]是由模型正确预测的主题数量与Topical预测的与库相关的总主题的比率。召回率[37]（测试库的平均值）是指模型正确预测的主题与与库相关的总主题的比率。优化的阈值是用于将模型的概率输出转换为二进制预测的阈值。一般来说，对于20类多标签分类任务，这三种算法都有很好的召回率值，然而，决定整体性能的是精度。尽管Topical没有最高的召回率（但与其他基线相当接近），但它的精确性明显更高。将这些与LRAP分数结合起来，表明Topical分配的主题具有较高的精确性，并将它们置于高等级。另一方面，基于GraphCodeBERT和Import2Vec的基线似乎挑选了尽可能多的主题，但将它们放在较低/最底层的位置，从而导致低精度和LRAP得分，但召回率高。另外。请注意，即使是20个主题，Topical（以及所有基于注意力的基线）提供的LRAP得分也大于0.6。笼统地说，这表明这些基线将相关的主题放在排名前50%的位置，因此，它们糟糕的精确度可能会因为不考虑排名靠后的主题而得到改善。

C. 对有效性的威胁

我们可以发现论文中的一些结果的有效性受到了一些威胁。前几名算法的性能差异不大，但我们通过重复实验对数据集重新取样，以确保标准差足够低，并低于测得的差异。性能的变化也可以作为我们收集的数据集大小和主题数量的函数而产生。我们的研究显示了性能是如何被一系列的数据大小和主题数量所影响的。结果（图5）表明，标签在大量话题的应用中也是有效的。然而，我们注意到，即使在标记更多的主题时性能下降，我们预计本文的主要贡献和见解仍将保持，即一个生成资源库级嵌入的框架。对有效性的威胁也可能来自保持低资源（硬件、计算时间）的实际决定。例如，超参数（填充长度）在一部分数据集的实验中被优化，并在所有实验中使用。同样，我们提出的框架是一个轻量级的框架，不需要在GPU/TPU上进行训练，因此，引入专门的硬件可能会因为进一步优化或处理更大的数据库而影响结果。最后，自动标记是一种多标签分类，在某些情况下，多标签分类的专门方法/损失会影响我们的观察。同样，也可以有一种说法，即更好地选择/分配主题，甚至推理主题间的子主题。

D. 限制条件

在未来，我们希望利用基于GPU的训练将Topical的应用扩展到更大的数据集。更大、更复杂的神经网络架构受益于更大的数据集，并且通常更擅长于难度更高的任务，如总结。请注意，即使没有这些设置，我们的研究中提出的方法和见解仍然有效。我们想在未来解决的另一个限制是对我们的方法进行测试，用于自动标记以外的任务，例如总结或寻找类似的存储库。最后，我们希望在未来解决本文中提出的一些方法相对较低的精度和召回率。

E. 相关工作

以前对源代码主题建模的研究[40]。[41]发现成功地调整了以前用于信息检索（IR）和自然语言处理（NLP）任务的文档主题建模的统计方法[42]-[44]。这些技术将源代码

表二：Topical模型与mean-GraphCodeBERT和TF3D基线的比较表。

模型	精度	召回率	优化阈值 局部
	.485 (0.017)	.63 (0.032)	.217 (0.025)
GraphCodeBERT0	.41 (0.031)	0.67 (0.0)	0.14 (0.01)
I2V-conc-attn0	.35 (0.03)	0.63 (0.03)	0.187 (0.01)

作为独立标记的集合，使用术语频率分析学习代码的统计模型，如TF-IDF或Latent Dirichlet Annotation（LDA）[40], [41], [43], [45]-[47]。Ascuncion等人展示了LDA对于发现软件工件上的主题的有用性，以及这些主题如何在可追溯性任务中发挥作用，该任务旨在发现相关软件工件链接之间的联系[48]。在一个不同的方法中，使用GitHub存储库中的 "README "文件，实现了对代码的自动标记与主题[45]。为了评估*Topical*与统计方法（如LDA或TF-IDF）相比的性能，我们开发了一个指定的术语频率模型。我们在此称之为TF3D，因为与Topical类似，它利用了来自三个资源库领域的术语分布，即源代码、软件包进口（依赖关系）和文档串。我们将在本工作的后面详细介绍该模型和结果。

基于深度神经网络的方法已经用大型参数化语言模型取代了许多术语分布模型，例如，BERT[13]，[26]。预先训练好的BERT模型最近被Guo等人扩展到了亲子语言模型。CodeBERT[9]是一个在几种主要编程语言上训练的多语言模型。CodeBERT在GraphCodeBERT变体中得到了进一步的改进，它将代码结构信息纳入了模型。GraphCodeBERT [10] 在与 CodeSearchNet 数据集 [21] 相关的任务上取得了最先进的结果。最近，Theeten等人开发了 "CodeCompass"，它使用他们提出的Import2Vec[8]工具来表示嵌入空间中的脚本依赖关系，其中类似的依赖关系在空间中被聚在一起。每个软件包都与一个数字向量相关联，CodeCompass利用相似性测量来推荐相关的软件包。Import2Vec[8]是我们研究的基线之一。我们表明，Import2vec与Topical的注意力机制相结合，会产生一个更有效的资源库级别的表示。这与现有的结合脚本级信息的方法相反，例如，脚本级信息的平均值或串联。[7].

现有的解决方案[49]-[52]提出的版本库级标签通常依赖于文本信息，如文件名、READMEs和典型代码库中的进一步文档。例如，Izadi等人[1]最近的工作是通过收集文件名、READMEs和Wiki数据来对资源库进行主题标记。通过使用DistilBERT对标记化的单词进行处理，他们为版本库制作了一个单一的嵌入，并添加了一个由sigmoid激活函数完成的全连接层，以实现多标签标记。我们与这些方法不同的是，Topical只使用脚本级别的信息（代码），而不使用README或Wiki数据中的文本信息。从README中提取有用的信息是一项NLP任务，它把导致软件开发者创建的原始README的质量负担。此外，它使学习到的回复容易在README中产生误导信息。最近，Rokon等人提出了Repo2vec[7]，它提供了版本库级别的嵌入，但它将README文件和元数据等文档纳入嵌入输入。更详细地说，Repo2Vec结合了来自源代码、元数据（包括READMEs）和版本库结构的信息。它通过将这三个信息源串联成一个单一的向量，将其聚合成一个嵌入。然后，该架构在一个数据集上被训练，该数据集由人类注释的标签组成，即两个给定的资源库是否相似。在这项工作中，我们在一个我们提供的自动生成的数据集上训练topical。我们的爬虫已经提取了与资源库相关的GitHub策划的主题，因此不需要人类注释，而人类注释可能难以扩展和评估。最重要的是，Topical结合了 "纯内容 "的信息，*即*源代码、仓库结构和源代码中的相关文档串。在这里，通过使用注意力机制，Topical产生的分类结果明显好于嵌入向量，而嵌入向量是通过平均值或串联来聚合信息的，正如我们的实验所示。因此，在此推测，注意力机制有可能在相似性任务中胜过repo2vec。最后，我们专注于那些不一定有足够或可靠的README文件（或任何文件）的版本库，主要是将源代码和版本库结构编码为密集嵌入，这是一个高度可靠和稳健的版本库表示。请注意，我们为我们的数据库自动抓取软件库，这并不局限于有文档的软件库。

IV. 结论

在本文中，我们介绍了Topical，一个直接从版本库源代码生成版本库级别嵌入的工具。Topical使用来自GitHub的策划主题，这些主题与软件库一起被抓取，以获得软件库级别的嵌入，可用于进一步的下游任务。Topical优于一个有竞争力的统计模型TF3D和Import2Vec基线，我们在本文中也介绍了我们的部分贡献。Topical的嵌入，同时也衔接了从函数到脚本级的不同尺度的代码表示，也在不同的代码表示领域、源代码、文档串和依赖关系中运作。我们的Github爬虫和储存库的策划数据集也被分享给社区，以促进进一步的研究，以及我们提出的模型的代码。

我们的一个关键见解是，使用注意力来结合脚本级别的嵌入可以提高以下方面的性能

下游任务。值得注意的是，资源库嵌入矢量使用预先训练好的BERT模型，这导致了一个简单和可转移的包，可以用低成本的方式使用，并且不需要特殊的硬件来完成其他的下游任务，或额外的主题。未来的研究将调查基于主题嵌入的其他下游任务，如根据资源库的内容用流畅的自然语言句子总结资源库，并确定为资源库作出贡献的开发者的技能范围。

参考文献

[1] M.Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms, " 2021.

[2] S.W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457-479, 2014.

[3] M.Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness, " *ACM Comput.Surv.*, vol. 51, no.4, Jul. 2018.[在线]。Available: https://doi.org/10.1145/3212695

[4] D.Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp.523-527.

[5] H.Washizaki, H. Uchida, F. Khomh, and Y.-G.Gue´he´neuc，"研究设计机器学习系统的软件工程模式，"在*2019年第十届实证软件工程国际研讨会上 in Practice（IWESEP）*。IEEE，2019年，第49-495页。

[6] T.H. Le, H. Chen, and M. A. Babar, "深度学习用于源代码建模和生成。Model, applications, and challenges," *ACM Computing Surveys (CSUR)*, vol. 53, no.3, pp. 1-38, 2020.

[7] M.O. F. Rokon, P. Yan, R. Islam, and M. Faloutsos, "Repo2vec:确定软件库相似性的综合嵌入方法，"在*2021年IEEE国际软件维护会议上 和进化（ICSME）*。IEEE，2021年，第355-365页。

[8] B.Theeten, F. Vandeputte, and T. Van Cutsem, "Import2vec:Learning embeddings for software libraries," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories（MSR）*.IEEE，2019年。pp.18-28.

[9] Z.Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T.Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," 2020.

[10] D.Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan。A.Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N.Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow, " 2021.

[11] R.Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi。V. Zolotov, J. Dolby, J. Chen, M. R. Choudhury, L. Decker, V.Thost, L. Buratti, S. Pujar, and U. Finkler, "Project codenet: A large-scale AI for code dataset for learning a diversity of coding tasks," *CoRR*, vol. abs/2105.12655, 2021.[在线]。Available: https://arxiv.org/abs/2105.12655

[12] I.Goodfellow, Y. Bengio, and A. Courville, *Deep learning*.MIT press, 2016.

[13] J.Devlin, M.-W.Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics:人类语言技术，第一卷（长篇和短篇论文）*。明尼苏达州，明尼阿波利斯。Association for Computational Linguistics, Jun. 2019, pp. 4171-4186.[在线]。Available: https://aclanthology.org/N19-1423

[14] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics:人类语言技术，第一卷（长篇论文）*。新奥尔良，路易斯安那州。计算语言学协会，2018年6月，第2227-2237页。[在线]。Available: https://aclanthology.org/N18-1202

[15] A.Radford和K. Narasimhan，"通过生成性预训练提高语言理解能力"，2018年。

[16] C.Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y.Zhou, W. Li, and P. J. Liu, "Exploring limits of transfer learning with a unified text to text transformer," 2020.

[17] A.Kanade, P. Maniatis, G. Balakrishnan, and K. Shi, "学习和评估源代码的上下文嵌入，" 2020.

[18] A.Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundation of Software Engineering*, ser.ESEC/FSE 2020。美国纽约：计算机协会，2020，第1433-1443页。[在线]。Available: https://doi.org/10.1145/3368089.3417058

[19] L.Buratti, S. Pujar, M. A. Bornea, J. S. McCarley, Y. Zheng, G. Rossiello, A. Morari, J. Laredo, V. Thost, Y. Zhuang, and G. Domeniconi, "通过神经语言模型探索软件的自然性，" *CoRR*, vol. abs/2006.12641, 2020.[在线]。可用：https://arxiv.org/abs/2006.12641

[20] R.M. Karampatsis和C. Sutton, "Scelmo: Source code embeddings from language models," 2020.

[21] H.Husain, H.-H.Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet挑战：评估语义代码搜索的状况，" 2020。

[22] Y.Zhang, F. F. Xu, S. Li, Y. Meng, X. Wang, Q. Li, and J. Han, "Higit-class:关键词驱动的github资源库分层分类，"在*2019年IEEE国际数据挖掘会议（ICDM）*。IEEE，2019，第876-885页。

[23] A.Cohen，"Fuzzywuzzy:Fuzzy string matching in python, july 2011," *URL http://chairnerd. seatgeek. com/fuzzywuzzy-fuzzy-string-matching-in-python/*, vol. 1, 2014.

[24] G. Navarro, "A guided tour to approximate string matching," *ACM Comput.Surv.*，第33卷，第1期，第31-88页，2001年3月。[在线]。Available: https://doi.org/10.1145/375360.375365

[25] L.Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091-1095, 2007.

[26] V.Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, " 2020.

[27] V.Salis, T. Sotiropoulos, P. Louridas, D. Spinellis, and D. Mitropoulos, "Pycg:python中的实用调用图生成，" 2021年。

[28] A.Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L.Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser.NIPS'17。Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000-6010.

[29] J.Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[30] S.Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study:以Yelp评论数据集为例，"在*2020年电子通信和人工智能国际研讨会上*，2020，第98-101页。

[31] D.Azcona, P. Arora, I.-H.Hsiao, and A. Smeaton, "user2code2vec:基于源代码分布式表征的学生剖析嵌入，"《*第九届学习分析与知识国际会议论文集*》，2019年，第86-95页。

[32] D.Fu, Y. Xu, H. Yu, and B. Yang, "Wastk: a weighted abstract syntax tree kernel method for source code plagiarism detection," *Scientific Programming*, vol. 2017, 2017。

[33] M.M. Islam和R. Iqbal, "Socer:一种新的代码重用的源代码推荐技术，"在*2020年IEEE第44届年度计算机、软件和应用会议（COMPSAC）*。IEEE，2020，pp. 1552-1557.

[34] K.S. Jones，"术语特异性的统计解释及其在检索中的应用"，《*文献杂志*》，1972。

[35] S.Cronen-Townsend, Y. Zhou, and W. B. Croft, "Predicting query performance," in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser.SIGIR '02。 New York, NY, USA: Association for Computing Machinery, 2002, p. 299-306.[在线]。Available: https://doi.org/10.1145/564376.564429

[36] Z.Chu, K. Stratos, and K. Gimpel, "Natcat:使用自然注释资源的弱监督文本分类"，*arXiv预印本 arXiv:2009.14335*，2020。

[37] H.Daume´，*机器学习的课程*。Hal Daume´ III, 2017.

[38] I.Loshchilov和F. Hutter, "解耦权重衰减正则化" *arXiv预印本 arXiv:1711.05101*, 2017.
[39] D.P. Kingma和J. Ba, "Adam: A method for stochastic optimization," 。*arXiv预印本arXiv:1412.6980*，2014。
[40] A.Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A.De Lucia，"如何有效地将主题模型用于软件工程任务？一种基于遗传算法的方法，"在*2013年第35届国际软件工程会议（ICSE）*。IEEE, 2013, pp.522-531.
[41] T.-H.Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories, " *Empirical Software Engineering*, vol. 21, no.5, pp. 1843-1919, 2016.
[42] T.K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259-284, 1998.
[43] D.M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation, " *J. Mach.Learn.Res.*, vol. 3, no. null, p. 993-1022, Mar. 2003.
[44] X.Yi and J. Allan, "A comparative study of utilizing topic models for information retrieval," in *European conference on information retrieval*.Springer，2009，第29-41页。
[45] A.Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging github repositories," ser.EASE'17。New York, NY, USA: Association for Computing Machinery, 2017, p. 314-319.[在线]。Available: https://doi.org/10.1145/3084226.3084287
[46] A.Hindle, M. W. Godfrey, and R. C. Holt, "什么是热点，什么不是。窗口化的开发者主题分析"，在*2009年IEEE国际软件维护会议*。IEEE, 2009, pp.339-348.
[47] A.De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "用信息检索方法标记源代码：一项实证研究，" *Empirical Software Engineering*, vol. 19, no.5, pp. 1383-1420, 2014.
[48] H.U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1.IEEE, 2010, pp.95-104.
[49] F.Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *2012 28th IEEE International Conference on Software Maintenance（ICSM）*.IEEE，2012年，第600-603页。
[50] F.Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *2013 20th Working conference on reverse engineering (WCRE)*.IEEE，2013年，第182-191页。
[51] N.Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp:一个通过在线内核学习检测类似移动应用的框架，"在*第八届ACM国际网络搜索 和数据挖掘会议论文集*，2015年，第305-314页。
[52] Y.Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering（SANER）*.IEEE，2017，pp.13-23。