描述PyPl中的深度学习包供应链：域、集群和脱离

北京大学软件与微电子学院，中国高中重点实验室

信心软件技术有限公司，中国教育部
何RUNZHI，宾熙，计算机科学学院，周明辉。北京大学中国，以及中国教育部高可信度软件技术重点实验室

深度学习（DL）框架已经成为快速发展的DL领域的基石。通过在分发元数据中指定的安装依赖关系，许多包直接或过渡地依赖于DL框架、一层又一层，形成DL包供应链（SCs），这对于DL框架保持竞争力至关重要。然而，关于如何培养和维持DL包sc的重要知识仍然缺乏。实现这些知识可能有助于DL.框架制定有效措施，以加强其SC，以保持竞争力，并为研究人员和从业者阐明DL SC中的依赖问题和实践。在本文中，我们探讨了两个具有代表性的PyPI DL包SCs中的包的域、集群和脱离参与，以弥补这一知识差距。我们分析了近600万个PyPI包分布的元数据，并为两个流行的DL框架构建了版本敏感的SCs；张量流和PyTorch。我们发现，在这两个SCs中的流行软件包（以每月下载量衡量）覆盖了34个域，属于8个类别。应用程序、基础设施和科学类别占SC和TensorFlow中流行包的85%以上，和PyTorch SC分别开发了基础设施和应用程序包的专门化。我们采用莱顿社区检测，在两个SCs中检测131和100个簇。集群主要呈现四种形状：箭头、星形、树形和森林，依赖复杂性越来越增加。大多数集群是箭头或星形，而树和森林集群占大多数包（张量流SC：70.7%，PyTorch SC：92.9%）。我们确定了包脱离SC的三组原因（即，从DL框架及其安装依赖项中删除）；依赖性问题、功能改进和安装方便性。在TensorFlow SC中，最常见的原因是依赖性不兼容性，而在PyTorch SC中，则是为了简化功能和减少安装大小。我们的研究为DL框架供应商、研究人员和从业者就PyPI DL SCs的维护和依赖性管理实践提供了丰富的启示。

中国化学会概念：• 软件及其工程学→软件的演变；维护软件；• 以人为本计算→开源软件。

附加的关键词和短语：软件供应链、PyPI生态系统、深度学习、软件结构和进化

1介绍

在过去的十年中，深度学习（DL）技术被广泛应用于各种任务中，如人脸识别[34]、机器翻译[22]和代码生成[64]，性能优异。爆发的

作者地址：北京大学软件与微电子学院Kai Gao，gaokai19@pku.edu.cn，北京市海淀区一四合院路No. 5号，邮编：100871。中国和教育部高信心软件技术重点实验室。中国；龙志和。rzhe(apku.edu.cn；，冰Xie，xiebing@pku.edu.cn；明辉Zhou，zhmh@pku.edu.cn，北京大学计算机科学学院，北京市海淀区一四合院路No. 5号，北京。100871，中国，中国教育部高可信度软件技术重点实验室。

ACM1049-331x/2024/1-艺术
https://doi.org/10.1145/3640336

DL技术在DL框架中是必不可少的，它提供了一组api，允许开发人员更容易、更快速地设计和训练DL模型，并已成为DL领域的基石。许多DL框架已经由各种组织推出，其中TensorFlow[14]和PyTorch[11]是最流行的[13]。

研究表明，开发人员在使用DL框架执行不同的任务[37]时有不同的需求。为了满足开发人员的不同需求，DL爱好者在Python社区[28]中发布了大量的软件包。这些包基于DL框架直接或传递提供的api提供了各种专门的功能。为了允许像pip这样的包管理工具在安装时自动安装DL框架，DL框架在这些包的分发元数据[6]中被指定为直接或传递的安装依赖项。“逐渐地，从DL框架开始，通过逐层安装依赖，大量的包形成了DL包供应链（SCs）。”

随着DL的蓬勃发展，出现了许多DL框架。然而，对于DL框架来说，在竞争对手中保持竞争并不容易。近年来，一些DL.框架的发展已经停止，如CNTK[3]和Chainer[4]，由于它们缺乏竞争力。考虑到DL技术在不同任务中的广泛采用以及第三方包[5]的重用，我们认为一个繁荣的DLpackSC对于一个竞争的DL框架至关重要。首先，SC中的包为开发人员提供了各种现成的专门功能，这可以帮助DL框架更好地满足开发人员的不同需求[37]。其次，从属包的数量被从业者视为该包的受欢迎程度的一个指标，并影响到第三方包[53]的选择。从这个意义上说，一个繁荣的SC有助于DL框架来吸引用户。此外，DL frameworks，e.g..the PyTorch网站的PyTorch有一个专门的页面，根据PyTorch[16]提供包。因此，DL框架的培养和维持其软件包sc是很重要的。

以往的工作主要是在整个包装生态系统层面的SCs，而不是像DL这样的特定领域层面。这些研究将包装生态系统中的SC构建为一个有向图，并设计了度量标准来衡量其结构和进化[31-33,50,55,73]。然而，这些度量主要是根据图的角度构建的，不能很好地反映SC的性质，比如SC中的包域。一些研究调查了在清单文件中声明对DL框架的依赖性的代码存储库。安装依赖关系)[39]或在导入语句中（例如，导入dependencies）[36,65]，e.g.，应用程序域[39,65]，更新行为[36,39]，和流行因素[65]。然而，代码存储库可能o1可能不会作为PyP1包发布，而且导入依赖不同于安装依赖，这使得如何培养和维持DL包sc的知识仍然分散和不完整。实现这些知识的主要意义是帮助DL框架供应商制定有效的措施，以加强他们的SCs，以保持竞争力（如上所述）。此外，这些知识还可以为研究人员和DL从业者阐明DL SC中的依赖管理问题和实践。

为了进一步填补这一知识空白，本文研究了PyPI包装生态系统中两个具有代表性的DL包SC，它们分别来自TensorFlow和PyTorch（简称张流SC和PyTorchSC）。紧张流和PyTorch是最广泛采用的两个DL框架[37]，PyPI是Python的官方第三方包注册表[9]，是DL字段[28]中的主流编程语言，也是紧张流和PyPorch提供API[15,20]的主要语言。考虑到一个繁荣的DL SC应该有助于满足DL开发人员的不同需求，并且需要许多软件包的参与，我们制定了以下研究问题：

RQ1（域）：张力流SC和Pylorch SC中的包覆盖哪些域？包的域是指包可以解决的任务类型。研究SC中的包域可以确定DL SC应该提供的支持域，以帮助满足开发人员的需求。此外，它也能帮助我们获得a

根据PyPA[7]的说法，这个发行版是一个软件包的一个特殊版本。

更好地理解SC的capability，e.g.，它的优缺点，和常见的使用场景。这样的理解可以帮助DL框架供应商加强他们的SCs，以更好地满足开发人员的需求，并帮助DL从业者在为他们的任务选择DL框架（SC）时做出更明智的决策。之前的工作已经研究了直接依赖于DL框架[39]的代码存储库的应用程序域和基于导入依赖项[65]的DL存储库SC中的包的域。然而，它们只涵盖了DL包SC的一小部分（更多细节请在第3节中），并且仍然缺乏对DL包SC中的域分布的更完整的理解。

RQ2（集群）：在两个SCs中形成了什么类型的包装除尘器？DL包SC是一个复杂的图，包含许多相互依赖的包，这使得其结构变得神秘。DL包SC分解成包集群包紧密连接在同一集群但稀疏连接集群之间，是一个很好的方法来了解包如何参与SC和识别关键包吸引其他包SC，从而帮助DL框架采取有效的行动来吸引包。例如，每个集群中其他包所依赖的包会显著影响SC的正常功能，并应对安全措施进行优先排序。此外，同一集群中的包可能共享类似的功能，因此包集群可以作为从业者在SC中搜索合适的包的有价值的来源。

RQ3（脱离）：包装在多大程度上脱离两个SC？软件在[41]的演化过程中删除一些依赖关系是很常见的。如果一个包从某个版本中完全删除了对DL框架及其依赖项的依赖，我们引用该从DL包SC中分离的包。软件包的脱离可能会影响SC的可持续性和稳定性，并导致用户流失，从而降低DL框架的竞争力。此外，包的脱离也可能预示着SC中的包开发人员所面临的潜在依赖管理问题。因此，了解为什么包脱离sc有助于DL框架采取对策以保持包的参与。

然而，对于DL包SC中包脱离的流行程度和理由知之甚少。

为了回答这些问题，我们首先通过分析近600万个PyPI包发行版的元数据，构建了一个对生态系统和版本敏感的安装依赖数据库。在数据库的基础上，我们通过迭代检索声明对张量流或PyTorch的直接或传递安装依赖的包来构造张量流和PyTorch SC。我们对两个SCs中流行的包的描述（以每月下载的数量来衡量）进行了主题分析，并确定了34个包域

跨越八个类别。超过85%的流行软件包属于应用程序、基础设施、

和科学类别。基础设施和应用程序类别占最流行的包

TensorFlow SC和PyTorch SC分别揭示了两种SC的不同特化。我们采用Leiden社区检测算法，在两个sc中检测了131和100个包集群。集群主要呈现四种形状：箭头。星，树和森林，增加依赖的复杂性。尽管大多数集群是箭头或星形，但树和森林集群占了包的大多数（70.7%和92.9%respectively），这表明少数包吸引了大多数其他包到SC。我们发现，脱离任何一个SC的软件包的数量呈上升趋势。我们确定了与三个方面相关的七个脱离原因：依赖性问题、功能改进和安装的方便性。两种SCs最常见的脱离原因不同，这表明它们存在不同的依赖管理问题。

简而言之，本文的贡献如下：

- 全面了解PyPI封装生态系统中两个具有代表性的DL包sc中的包的域、集群和脱离参与。

4 • K.高等人。

• 一种构建对版本敏感的PyPI包SCs和公共数据集和scripts²的自动化方法

以促进未来的研究。
• 为DL框架供应商、研究人员和DL从业人员提供维护和丰富建议

PyPI DL包sc的依赖性管理实践。

2、背景及相关工作

2.1背景

软件SC的定义。在传统生产中，SC是指在将原材料转化为最终产品的过程中涉及的公司网络，公司之间的供应关系是通过产品[59]的流动来建立的。软件[21]之间也存在类似的供应关系。具体地说，如果一个软件使用了另一个软件（如包api）提供的功能，则假定它们之间建立了供应关系（即依赖性）。随着软件开发越来越依赖于外部软件，软件之间的供应关系变得越来越复杂，逐渐形成一个由依赖项连接的软件网络。，软件SC。

代码存储库vs。包分布。*之前的工作[65]研究了依赖于*

通过将存储库的URL与PyPI包的分发元数据中声明的代码存储库URL进行匹配，并分析了将包发布给PyPI的代码存储库的域。然而，大量（大约1/3）的PyPI包并没有在分发元数据3中声明它们的代码存储库信息。因此，从代码存储库中识别PyPI包将会省略大量的PyPI包。在本文中，我们选择在PyPI中发布的包分布，因为我们可以很容易地获得PyPI包的完整列表和它们之间的依赖关系，这允许我们构建为完整的包

尽可能地在PyPl中的SCs。

安装依赖关系vs。导入依赖关系。现有的文献主要调查两种依赖性-

：安装依赖关系和导入依赖关系。安装依赖关系在软件的清单文件中指定，如setup.py、requirements.txt和软件包分发元数据[72]。导入依赖关系

是在代码文件的导入语句中指定的，而不是[65]，e.g.，导入火炬。

在本文中，我们选择通过分析包分布之间的安装依赖关系来构建DL包sc，有四个原因。首先，安装依赖是之前分析包SCs（如[33,50,73,76]）的工作中使用最广泛的。其次，Python提供了一个官方的包，即打包[19]，它可以准确地解析在包分发元数据中指定的安装依赖关系。第三，准确识别导入依赖项（即在导入语句中导入包的导入名称映射到包名称）具有挑战性，因为Python包的导入名称不一定与包名称相同，而且不同的包可能共享相同的导入名称[38]。第四，安装依赖项保留了依赖项版本信息，这使我们能够分析软件包的脱离情况。但是，安装依赖存在[24]的依赖问题，即一些安装依赖软件不使用安装依赖。为了评估这个问题的影响，我们手动检查了一些软件包，如第3节所述。**PyPIDL软件包SC的定义。**我们将PyPI包SC概念化为一个有向图G=（P，E），其中P中的顶点是PyPI包，E中的边表示包之间的安装依赖关系。形式上，安装依赖关系可以定义为有序对（P[u]，Pa[d；]），表示应首先安装版本d；包Pa，包P的版本u；。我们把P称为Pa的依赖，把Pa称为Pu的依赖。我们将PyPI DL包SC称为PyPI包SC的一个子图，它从一个DL框架开始，并根据框架直接或传递地包含所有包，详见第3节。

2https:/github.com/gaokai320/PyPI-DLSC
在复制包中提供了计算比率的脚本。

ACM跨软。雕刻方法。

2.2相关工作

我们的工作在于软件SC和DL的交集，因此有两组相关的工作：包装生态系统中SC的研究和DL软件工程（SE）的研究。

包装生态系统中SCs的2.2.1研究。由于重用的流行第三方包软件开发[41]，包在各种包装生态系统是一个流行的话题在SE社区威特恩etal.[73]调查NPM的演变，发现包的数量和包之间的依赖关系迅速增加。Decan等人[30,31,33]和Kikas等人[50]分析了多个包装生态系统中包装SCs中直接和传递依赖的分布。他们还发现，这些SCs中的包和依赖关系不断增长，传递依赖导致这些SCs的脆弱性。

已经开展了大量工作，以识别和减轻一揽子SCs中的风险。Valiev等人[68]发现，PyPI软件包在SC中的相对位置对其可持续性有显著影响。Vu等人[70]研究了PyPI包中的代码与其相应的代码存储库之间的差异，以识别潜在的恶意代码注入。Liu等[55]研究了脆弱性传播和传播演化。Zahan等人[78]提出了6个安全弱点指标，以识别NPM中潜在的SC攻击。Wang等人研究了PyPI[72]和NuGet [54]中由传递依赖关系引入的依赖关系。

尽管做出了这些努力，但关于如何培养和维持DL包sc的知识仍然有限。

针对DL的SE的2.2.2研究。利用GitHub、堆栈溢出（SO）、访谈等数据，为DL[37]进行了大量的开发工作。具体来说，DL错误得到了广泛的研究。Zhang等人[82]探讨了张量流应用程序中bug的症状、根本原因和修复挑战。Islam等人[46,47和Humbatova等人[45]研究了几种DL框架应用中的bug。本文还探讨了与DL作业[79]、DL框架[48,49,77]、DL依赖堆栈[44]、培训[81]和部署[27]相关联的错误。一些研究确定了开发[37,40,80]和部署[26]DL应用程序的挑战。这项工作为SE社区提供了解决DL问题的见解，但从SC的角度来看，没有项工作能触及问题。

很少有人研究依赖于DL框架的代码存储库的特性。Han等人。[39]研究了代码存储库的项目目的、应用程序域、更新行为和依赖性版本分布。Dilhara等人[36]研究了在代码中导入它们的代码存储库中的DL框架的使用和更新。这两项研究有两个主要的局限性。首先，他们只关注直接依赖于DL框架的代码存储库，而忽略了SC的网络结构。其次，他们收集的代码存储库可能会作为PyPI包发布，也可能不会发布。因此，这两项研究只覆盖了DL软件包SC的一小部分。Tan等人[65]通过重复检索直接或传递导入张流或PyTorch的代码库，构建了紧张流和PyTorch存储库sc。他们在SC中定位了作为PyPI包发布的代码存储库，并研究了这两个SCs的结构、应用程序域和流行因素。然而，它们的SCs与软件包SCs有很大的不同，原因有两个原因。首先，他们的SCs错过了第2.1节中讨论的大量PyPI包。我们的sc中大约70%的包裹不属于他们的包裹范围（更多细节见第3节）。其次，包的代码存储库之间的导入依赖关系可能会过度表示包分发版之间的安装依赖关系。具体来说，并不是存储库中的每个文件（例如测试文件和示例文件）都将被打包到分发版中，因此在这些文件中声明的导入依赖关系不是分发版的安装依赖关系。手动检查[65]的SCs中的100个软件包表明，导入依赖性

在27个包的代码存储库中声明的并不是它们的发行版的安装依赖关系。这两个差异导致了他们的研究和我们的研究之间的领域分布结果的差异，如第4.2节所述。

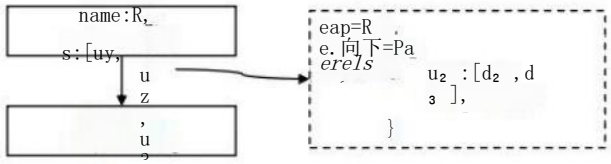


图1.SC示例。

总之，早期的研究只覆盖了PyPI DL包SC的一小部分，他们的结果不能充分揭示如何培养和维持PyPI DL包SC。4

3、供应链建设

Python打包管理机构（PyPA）是一个负责Python打包的工作组，它在BigQuery上托管一个公共数据集，其中包含在PyPI[10]上发布的所有发行版的元数据，这使我们能够尽可能构建完整的PyPI DL包SCs。我们在2021年11月4日下载了数据转储文件，其中包含354,636个软件包和5,743,721个发行版。

我们首先构建一个依赖数据库D来支持构建PyPI DL SCs。该数据库是基于生态系统的，因为它覆盖了所有的PyPI软件包。与之前工作[30,68]提出的分析PyPI包SCs的方法相比，它还考虑了依赖关系的版本信息，以获得PyPI包发布之间的依赖关系（即版本敏感），这使我们能够构建更细粒度的DL包SCs，并研究包的脱离情况。具体来说，PyPI包分发在其元数据的reak_dist字段中指定其安装依赖关系。该字段中的每个项都包含一个包名Pa和一个可选的版本约束c。一个版本约束为一个依赖项指定了一个PEP508版本范围[2]。我们在打包[19]包之上实现了一个版本约束解析器，这是PyPA发布的一个用于解析依赖项规范的工具，以定位满足要求的依赖项版本。具体来说，对于在PyPI中发布的P的每个版本u；（通过查询PyPI分布元数据转储），我们检查它是否满足 C. 如果满足，我们将插入一个记录（Puj、Pad、） D. 最后，D包含了281,443,215条记录。

基于D，我们构建了PyPI DL SCs如下。为了对版本敏感，我们向SC中的包和依赖关系添加属性。对于每个包，我们维护一个属性我们，它是一个列表，存储在SC中出现的所有包的关联。对于每个依赖关系，我们维护一个属性，该属性使用映射来存储包Pu与其依赖Pa之间的版本依赖关系。键中的每个键代表P版本，该值代表依赖于键对应的Pu的所有版本的Pa版本。以图1为例，假设在SC中有一个包Pu和一个包Pa，Pi，uj，uz，u3versions和dj，dz，dg版本出现在PC中，如us属性所示。Pu与Pa之间的依赖关系e的关系属性表明，Pa只依赖于Pu的uz和u3versions。具体来说，Pa的dz和d3versions依赖于Pu的uz版本，所有版本的Pa都依赖于美国版本的Pu。

基于第2.1节中的定义，我们按照算法1中描述的过程迭代地构建对版本敏感的PyPI DL SC。考虑到一个DL框架可能发布多个packages，e.g..Tensorflow框架发布三个包：紧张流、紧张流cpu和紧张流gpu，算法以包名的列表N作为输入。未访问是未探索的包队列。我们的算法在更新中检索包的直接依赖项，并在每次迭代中使用未探索的直接依赖项进行更新。未访问被初始化为PyPI中的所有发布版本（第2行~6）。对于未访问的每个包的每个版本，我们从D中查询直接依赖于它的包的所有版本

*如果未指定，请参加以下章节。”SC”指Py Pl包SC，“依赖”指安装依赖。

算法1：构造PyPI DL SC

输入：一个软件包名称的列表：N
输出：供应链：G
P←顶点)。E←边缘（），未访问-顶点（）
//输入包作为初始未访问的包
2 **观察∈N做**
p-Vertex（名称=n，我们=列表（））
p.更新_版本（GetAllpy p.版本(n)）
5 P.merge (p)
太监。合并(p)
7而不叫=（）做
8 从属-顶点（）
9 预见u∈未被访问
10 **预览版本∈u.我们做**
11 dps-获取依赖项（美国名称，版本
12 从属成员。合并（dps）
13 **预测∈dps做**
//将新的依赖项合并到E
14 e←边缘(up=u.name，rels=(版本：
d.us})
15 E.合并(e)
//获取新的未访问过的软件包和版本
16 未访问的受抚养人
11将新找到的软件包和版本合并到P中
17 P←P.合并（从属）
18 G←DiGraph (P,E)
19返回g

（第9行~12），并将它们之间的依赖关系合并到&（第13行~15）。合并采用两种形式：添加新边或更新现有边的记录（添加新的键值对或更新现有值）。依赖项表示直接依赖于未访问的包的包的所有版本。我们通过将从属包与P（第16行）进行比较，并将新找到的包（从属包）合并到P（第17行），从而获得新的未访问的包。合并操作要么添加新的包，要么向我们现有的包添加新的版本。重复上述过程，直到未访问的过程为空（第7行）。

我们分别输入张流、张流cpu、张流和火炬的输入，构建了张流和PyTorch SC。最后，TensorFlow SC包含2567个包，19116个版本和3232条边；PyTorch SC包含3278个包，26563个版本和5440条边。我们还在2021年11月之前构建了张量流SC和PyTorch SC，并与[65]进行了比较。在这两个SCs中有1,087个独特的软件包和772（71.0%）不被[65]覆盖。我们调查了未覆盖包的月下载和家属数量，发现：1)185个包的月下载高于平均水平，占（233）包的月下载高于平均水平的79.4%；2)85个包有依赖，如130个家属的火炬视，21个dependents.pytorch-transformers有13个依赖，12个依赖的紧张流概率。因此，我们的SCs比[65]覆盖了更受欢迎和关键的包，并提供了在PyPI中的DL包SCs的更完整的图片。

8 • K.高等人。

如第2.1节中所讨论的，包可能不会使用其安装依赖（即膨胀的依赖），从而影响构建的DL包SC的准确性。为了评估这个问题的流行程度，我们分别从TensorFlow SC和PyTorch SC中随机选择了334个和344个软件包（95%置信水平，5%置信区间）。对于每个采样包，我们下载一个出现在SC中的最新版本的发行版，并检查该发行版是否在SC中导入其安装依赖项。在取样包装中，7个（334个）和6个（344个）包装在检查时已从PyPI中取出。在剩下的327个和338个packages，305（93.3%）和334（98.8%）包中导入它们的安装依赖关系，这表明在我们构建的两个SCs中，膨胀的依赖关系问题是罕见的。此外，膨胀的依赖问题似乎在拉伸流SC中比在PyTorch SC中更常见，这与第6节中的发现相一致。2.

4rq1：在紧张流sc和火炬sc中覆盖哪些领域？

4.1方法

虽然手动检查是识别包的域的一种可行的方法，但由于包的数量很大，几乎无法通过所有的包。因此，我们选择关注两个SCs中的流行包，以每月下载的数量来衡量，这是一个广泛使用的流行度量[35,69,70,76]，原因有三。首先，直观地看，每月下载的次数越多，说明包或包的依赖者越多，这可以更好地反映SC的核心功能。其次，之前的工作[65]也选择了流行的软件包来分析域分布。第三，许多包很少是used，e.g.、玩具或虚拟包[65,71]，这不能很好地反映SC的核心功能。考虑到新创建的包，我们在获得分发元数据转储一个月后，从公共PyPI下载统计数据集中检索所有Py PI包的下载数量。即，在2021年11月4日至2021年12月4日之间。然后，我们在两个SCs中选择每月下载量超过3,825次的软件包，即PyPI软件包的平均每月下载量。我们使用每月的平均下载量作为选择指标，以平衡所选软件包的代表性和进行人工检查的时间成本。这样，我们在TensorFlow和PyTorch SC中分别获得了219个和259个包，可用于人工标记，且高于之前工作[65]中调查的包的数量。也许并不奇怪的是，两个sc中流行的包与41个重叠的包不相交，留下437个独特的包。

我们遵循一般的主题分析[29]程序来确定软件包域。首先，两个检查员（第一和第二作者）检查每个包的摘要、描述和存储库自述文件，以收集有关其域的相关文本。然后，它们独立地读取和重读所有材料，生成描述每个包的域的初始代码，并将类似的代码合并到主题中。如果一个包由于缺乏其域上的信息而无法被标记，则将其标记为不清楚。接下来，他们比较代码和主题列表，直到达成协议，并为每个标识的代码开发一个包含定义和示例的编码指南。在这个过程中，我们参考了NeurIPS论文呼吁中列出的主题，这是DL领域的顶级会议之一，以确保生成的领域和类别的合理性。之后，检查人员独立使用编码指南对所有材料进行标签，由Cohen的Kappa[58]测量的评分者间信度为0.953。这些分歧通过一次讨论得到了解决。最后，张量流SC中的201个包和Py Torch SC中的247个包被正确地标记。28个软件包被标记为不清楚，并在以下讨论中被排除在外。整个过程大约需要200个工时，并在复制包中提供了相关材料。

5https://www.surveysy stem.com/sscalcehtm
https://neurips.cc/Conferences/2023/CallForPapers

ACM跨。软。雕刻方法。

描述PyPl：领域中的深度学习软件包供应链。集群和脱离

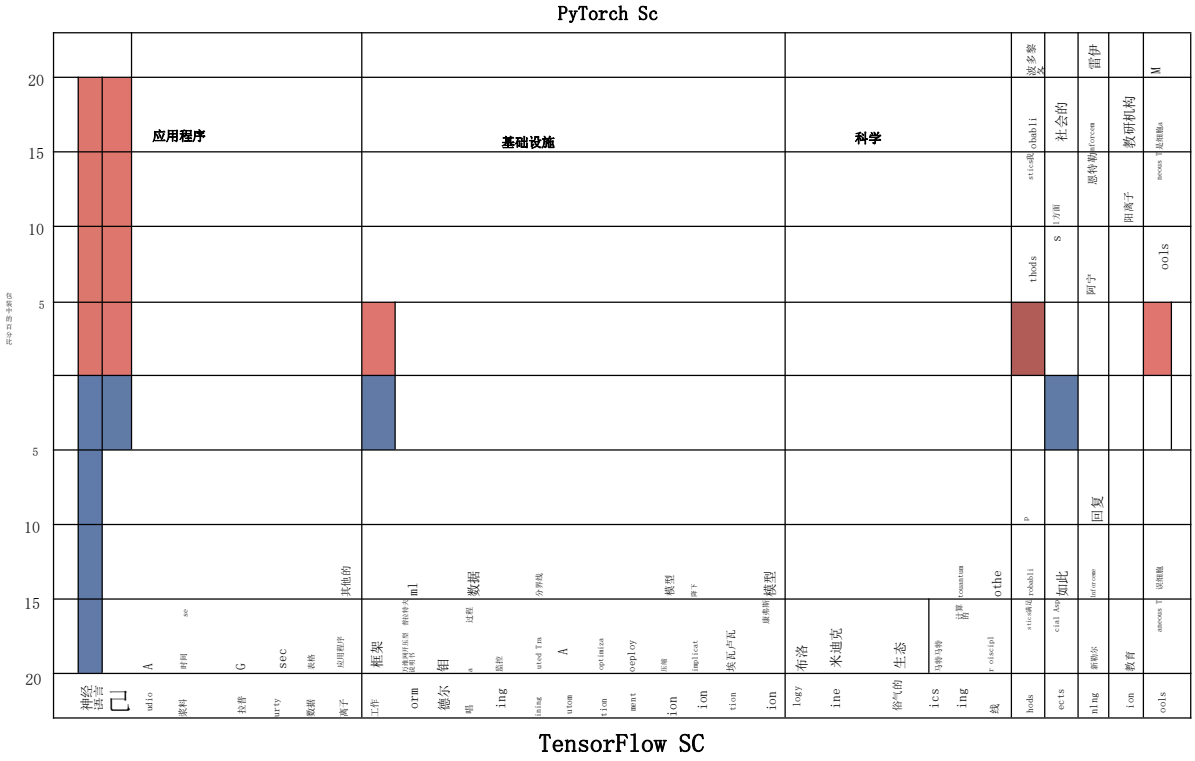


Fig.2.Distribution中的34个软件包域和8个类别。上部（红色条）对应PyTorch SC，下部（蓝色条）对应TensorFlow SC。

4.2结果

图2显示了在任何一个SC中流行的包所覆盖的域分布。我们总共确定了34个人

软件包域分为八个类别。

应用类别包括在计算机科学中长期研究的9个领域，如自然语言处理（NLP）和计算机视觉（CV）。这类包提供了促进DL技术在特定类型的数据（如图像、文本和音频）上的应用的功能，例如数据预处理和模型架构。这一类在PyTorch SC中占流行包数量最高（52.6%），在TensorFlow SC中占第二高（36.3%），其中NLP和CV占优势。我们还注意到了这两个SCs之间的差异。一般来说，Py Torch SC中应用包的比例明显高于TensorFlow SC，通过单边比例z检验（p<0.001）证实。众所周知，PyTorch提供了比张量流更好的编程能力，我们的发现进一步表明，PyTorch SC在许多流行包的应用领域发展了竞争优势。具体来说，PyTorch SC在NLP、CV、时间序列和表格预测方面提供了更高比例的包，而TensorFrowSC提供了更高比例的IR（信息检索）包。此外，NLP包的数量略少于PyTorch SC中CV包的数量，但在TensorFlow SC中，前者是后者的两倍。这种差异可能归因于相关工具的可用性。例如，

火炬视觉是PyTorch SC中最受欢迎的软件包。它提供了流行的数据集、模型架构和

ACM跨。软。雕刻方法。

计算机视觉[12]的常见图像转换，并依赖于PyTorch SC中的943个软件包。这就是说。

火炬愿景为PyTorchSCCV包的繁荣奠定了基础。但是在张流SC中缺少类似的CV包。

基础设施是PyTorch SC的第二大类别（24.7%），也是TensorFrowSC的最大类别（37.8%）。此类别中的包提供了适用于其他类别中的多个域的功能。它包括13个域。这类域在两个sc之间的分布差异很大。第一与PyTorchSC（$p<0.01$）相比，紧张流SC提供的流行基础设施包的比例明显高得多，这表明紧张流SC在基础设施领域具有竞争优势。其次，一个显著的区别是，张量流SC中流行的MLOps平台包的比例要高得多。这些软件包由各种生产平台供应商发布，如Azure机器学习？和TensorFlow Extended[8]and提供的功能可以促进这些平台中DL模型的开发、部署和管理。人们认识到，张量流为生产化提供了更多的特性，而我们的
研究结果进一步证实了张量流SC中生产化工具的繁荣性。第三、所占的比例
数据预处理、监控、分布式培训。张量流中的部署和模型转换软件包
SC高于PyTorch SC，但PyTorch SC提供了更高的百分比的优化，模型
压缩和训练简化软件包。这些差异不仅表明了工具的丰富性
但也反映了该框架的功能缺陷。例如，我们承认，张量流SC提供了一个强大的部署tools，e.g.TensorFlow服务和张量流Lite [26]的集合。然而，TensorFlow并不为ONNX[8]提供本地支持，这是一种开放的ONNX模型标准格式，支持框架之间的互操作性，而Py Torch则支持。因此，在张流SC中出现了三个流行的模型Con版本包，使张流和ONNX模型之间的转换。另一个例子是PyTorch SC中的训练简化包。尽管使用PyTorch构建复杂的模型很方便，但开发人员还是必须编写大量的样板代码来训练DL模型。为了解决这些问题，许多训练简化包，如火把点燃被提出，以简化复杂的训练配置。总的来说，这些差异表明张量流SC在生产DL模型方面提供了更强的能力。

令人惊讶的是，科学类拥有第三多的软件包（拉伸or流SC占13.4%，PyTorchSC占9.3%）。这类软件包促进了DL技术在许多科学学科中的应用，特别是生物学、医学和物理学。接下来，它们将转向概率方法、强化学习和社会方面。概率方法包提供了多种贝叶斯推理方法和贝叶斯优化模型，特别是高斯过程。PyTorch SC提供的相关软件包的比例（3.6%）高于TensorFlow SC（2.0%）。强化学习（RL）软件包提供了各种RL环境和模型。TensorFrowSC中3.0%的流行软件包和PyTorch SC中的2.0%都属于这个领域。社会方面是一个新兴的热门研究课题，旨在减少DL带来的歧视和不可靠决策等不良影响。它涉及到许多方面，如健壮性、可解释性、公平性和隐私性。似乎TensorFlow SC比PyTorchSC（2.0%）提供更多的（4.0%）社交包，这可能是由于在生产DL模型时，社会方面非常值得关注，而TensorFlow SC更用于将DL模型部署到生产环境中。在that，3.0%and之后，张量流SC和Py Torch SC中4.9%的流行软件包是杂项工具，它们提供了常用的实用程序和包装器。最后，我们注意到这两个SCs中的三个流行的包是对DL书籍的支持材料和教授开发人员DL techniques，e.g.nlpia的课程，这是一个包含动作手册中自然语言处理的代码示例的包。

7https//azure.microsoftcom/en-us/produets/machine-learning
Shttps//www.tensorflow.org/tfx

我们还与[65]报告的结果进行了比较，[65]基于代码存储库之间的导入依赖关系构建了DL SCs，并分析了已经发布了PyPI包的存储库的应用程序域。为了进行比较，我们的操作如下：1)保持包在2019年11月之前发布，其数据收集时间；2)将应用程序、科学、概率方法和RL包映射到域相关（DR）包，其余映射到非域相关（NDR）包。我们的研究证实了[65]的一些发现。张量流SC提供了更多的RL包；CV和NLP是最流行的两种DR包类型。然而，也有一些值得注意的不同。首先，与他们发现的在两个SC中DR和NDR包的数量几乎相同不同，我们发现在PyTorch SC中DR包的数量约是NDR包的1.6倍，这表明两个SC的专门化不同。其次，他们发现有更多的CV包比NLP包SC，相比之下，我们发现NLP包的数量是相当的和大约两次CV包PyTorch SC和紧张SC，表明相对更关注NLPSC。第三，我们发现Py Torch SC没有提供更多的实验结果分析包，而是Py Torch提供了更高比例的监测包，可能是因为实验结果分析非常重要，而PyTorTorch没有提供本地支持(例如，拉伸板可视化工具）。这种差异可以归因于在第2.2.2节中讨论的存储库SCs和软件包SCs之间的差异。

在41个重叠的软件包中，18个是关于应用程序的，12个是关于基础设施的。这些重叠的包依赖于来自两个SC的包，要么提供支持两个DL框架的功能，要么使用来自一个SC的包来补充另一个SC。例如，tempeh是一个跟踪模型的内存使用情况和运行时的框架。它依赖于张量流和Py Torch来支持用张量流和Py Torch编写的模型。作为后一种情况的一个例子，memenn是建立在PyTorch上的框架，使用张流SC的拉伸板进行记录，因为当时PyTorch SC并没有提供相关工具。

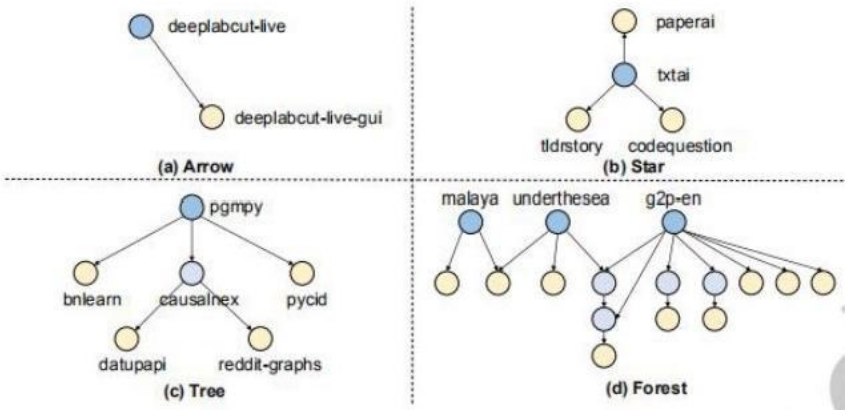
RQ1的总结：以两个SCs的每月下载量来衡量的流行软件包涵盖了34个域，属于8个类别。这两种SC中超过85%的软件包属于应用程序、基础设施和科学类别。Py Torch SC提供了更高比例的应用程序包，如NLP和CV。TensorFlow SC支持MLOps平台等基础设施包和部署，
这表明张量流SC和PyTorch SC在应用中有了竞争优势
基础设施域。

5rq2：在两个SCS中形成了什么类型的包集群？

5.1方法

从概念上讲，SC是一个有向图，使我们能够使用图社区检测技术将SC的复杂结构分解为包集群。首先，在这个RQ中，我们希望评估SC中派生包的影响。因此，根节点（即拉伸流和PyTorch框架）及其连接边从进一步分析中删除，使我们在拉伸流SC中有892个节点938条边，在PyTorch SC中有2044个节点2.661条边。相当数量的节点（TensorFlowSC中65.2%，PyTorchSC中37.6%）是孤立的点，这表明它们在SC中唯一依赖的是DL框架。接下来，我们在修剪后的SCs上运行莱顿社区检测算法。选择该算法是因为其效率、稳定性和形成致密团簇的能力，从而捕获了SC井[67]的结构特征。

我们的目的是了解包裹如何参与SC，并确定吸引其他包裹到SC的关键包裹。为此，我们关注形状和复杂性（由两个广泛使用的度量标准[30,33]来衡量：集群size，i.e.、节点数量和平均degree，i.e.，即节点数量上的边数）



四个簇形状的Fig.3.Example。

的集群。它们展示了整个包的分布，这提供了对SC的特征的基本理解。具体来说，集群的形状描述了集群内的包总体上是如何相互依赖的，集群的平均程度表示集群中包的依赖关系的平均数量。它们可以很好地反映软件包如何通过依赖关系参与SC。集群大小指的是集群中的包的数量，并有助于识别增加SC中的包参与度的包。

为了按形状对集群进行分类，我们首先将所有集群可视化为图°，这是一个提供通用图形分析工具[17]的包。接下来，与第4.1节相同的检查员进行开放编码[63]，以确定集群形状。具体来说，他们观察包在集群的可视化图形中是如何相互依赖的。然后，一个检查器将每个集群分配给一个初始代码，该初始代码描述了集群中包之间的依赖关系结构，并与另一个检查器讨论迭代地改进代码，直到达成协议。在那之后，他们一起开发了一个编码指南。为了验证可靠性，他们使用编码指南对所有集群进行独立编码，Cohen‘s Kappa值为0.907。这些冲突已通过一次会议得到了解决。上述过程约需要150个工时，并在复制包中提供了相关材料。

5.2结果

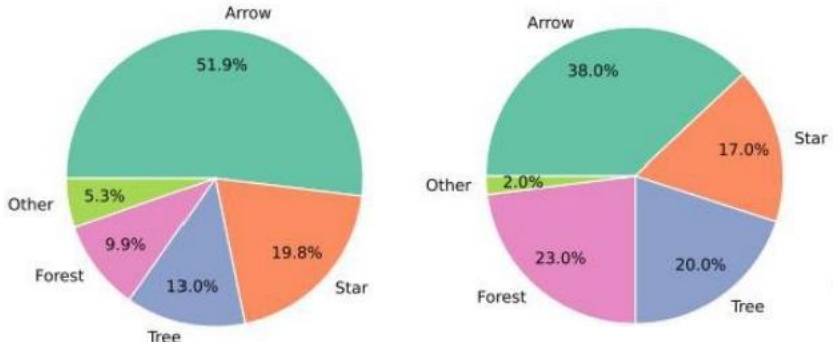
集群形状。Leiden算法分别在修剪后的张量流和PyTorch SC中检测131个和100个包簇。星团主要呈现四种形状：箭头、星形、树形和森林。图3显示了

四种形状，图4显示了每个簇形状在两个SCs中的比例。

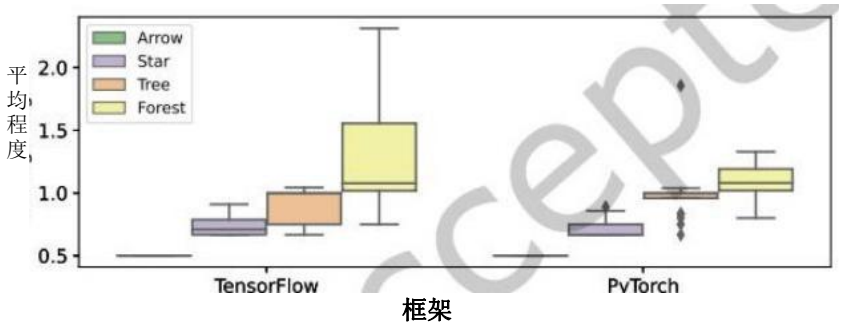
- 箭头集群只包含两个软件包，其中一个取决于另一个。例如，在张流SC的箭头集群中，包数据-gui依赖于包，并为其提供GUI（图3(a)）.68（51.9%）和38（38.0%）集群处于这种形状，分别占包的15.2%和3.7%。值得注意的是，这些集群中的包并不依赖于其他集群中的包。
- 星团呈现一个集中的形状，包含至少三个包，其中一个包依赖于其余的包。在图3(b)中，包txtai直接依赖于其余三个包。26个（19.8%）和17（17.0%）集群处于这种形状，分别占两个SCs中软件包的12.1%和3.3%。大多数（23和14个）星团包含不超过5个星包。

° 请参考复制包中的图形/拉伸流信息和图形/PyTorch文件夹。

ACM跨。软。雕刻方法。



TensorFlowSC（左）和PyTorchSC（右）中簇形状的Fig.4.Distribution。



平均度的Fig.5.Distribution。

- 树集群呈现具有根包的层次结构，集群中的其他包直接或传递依赖于根包。与星团相比，树簇中的包之间存在传递依赖关系。在图3(c)中，pgmpy是集群的根包，它直接或传递地依赖于其余五个包。17（13.0%）和20（20.0%）集群也处于这种形状，分别占26.8%和38.6%

  包分别。具体来说，两个SC中最大的集群都是这种形状。
- 森林集群显示了更复杂的层次结构，包中的其他包依赖于多个根包。图3(d)显示了张力流SC中的一个森林集群。这个集群包含三个根包，即。马拉亚、地下和g2p-en，依赖于集群中剩下的13个包。13（9.9%）和23（23.0%）集群也处于这种形状，分别占43.9%和54.3%的软件包。超过75%的森林集群在两个SCs中包含超过10个的包。

除了上述四个形状外，很少有（17和2）包在其他形状中形成簇。具体来说，TensorFlow SC中的三个集群和Py Torch SC中的两个集群只包含一个包，其后期版本依赖于其早期版本。张力流SC中的一个集群包含两个相互依赖的软件包。张量流SC中的三个集群包含一个包，这直接依赖于其他包的不同。由于其他集群的数量很小，而且它们包含的包很少，下面我们将关注四个主要形状。

集群复杂性。为了了解不同集群形状的复杂性，我们比较了它们的大小和平均度。首先，我们应用Mann-惠特尼U检验[57]来比较箭头的大小。星、树和

森林集群。由于我们执行了多个测试，我们使用Holm-Bonferroni方法来调整p值来控制家庭错误率[43]。结果表明，在两个SCs中，森林集群比树集群包含更多的包（调整后的p-value：0.009和1.40e-4），星集群比调整后的包（p-value：6.57e-21和1.32e-12）。然而，树星团和星星团的大小没有显著差异（调整后的p-value：0.188和0.112）。图5显示了不同形状的集群的平均度的分布（每个数据点代表一个集群的平均度）。我们可以观察到，张量流SC和0.50,0.67,1.00的箭头、星形、树形和森林簇的平均度逐渐增加，随着中值为0.50,0.71,1.00,1.08而增加，PyTorch SC的值为1.08。Holm-Bonferroni法的Mann-Whitney U检验也有显著差异。特别是，大多数森林集群（13个中的11个和23个中的20个）的平均度不低于1，这表明森林集群中的一个包平均与至少有两个包有依赖关系。我们还研究了树和森林集群的深度，考虑到它们的复杂性。集群的深度被定义为根包和集群中其他包之间的最长路径的长度，并反映了根包吸引传递依赖项的能力。总体而言，森林集群的平均深度高于SC中的树集群（TensorFlow SC：2.46 vs.2.06，PyTorch SC：2.39 vs.2.30），表明森林集群中的根包更有可能吸引传递依赖。总之，箭头、星形、树形和森林集群显示出越来越复杂的依赖关系。PyTorchSC（43.0%，92.9%）中树木和森林集群的比例以及树木和森林集群中软件包的比例远远高于紧张流SC（22.9%，70.7%）。这表明PyTorch SC在包之间具有更复杂的依赖关系，结构更加集中。有两个原因可以解释这一点。首先，PyTorch SC比TensorFlow SC（276,10.8%）具有更高的具有依赖项的软件包的百分比（429,13.1%）。第二，正如Mann-Whitney U测试（p-value：2.17e-99）所建议的那样，PyTorch SC中的包比TensorFlow SC中往往依赖于更多的包。

为了更深入地了解SC的结构特征，我们进一步检查了每个SC中包含超过10个包的大型集群，从而在TensorFlow SC中得到15个集群，在PyTorch SC中得到20个集群。我们在TensorFlow SC和Py Torch SC中分别发现了8个（15个）和18个（20个）呈森林形状的大簇。我们还检查了这些大型集群中最依赖的包（简称核心包）。值得注意的是，TensorFrowSC中1/3的大型集群和Py Torch SC中1/4的大型集群的核心包是官方的packages，i.e.，由DL框架团队发布的包。它表明，官方的一揽子计划在形成SC中发挥着重要作用。我们还发现，这些核心包可以很好地反映RQ1中揭示的两个SCs的域分布的差异。首先，Py Torch SC中的5个（20个）核心包是关于CV的，而TensorFlow SC中的15个核心包中没有一个是关于CV的，这与两个SC中CV包比例的巨大差异相一致。其次，张流SC的15个核心包中有40%属于基础设施类别，PyTorch SC的百分比为25%，这与张流SC中基础设施包的较高百分比相一致。

RQ2的总结：莱顿算法在Tensor流和PyTorch SC中分别检测131个和100个簇。集群主要呈现四种形状：箭形、星形、树形和森林形，依赖的复杂性不断增加。树和森林集群约占集群的22.9%和43.0%，但包含了大部分（70.7%和92.9%）的包，这表明少数包吸引了大多数包到SC。PyTorch SC中树和森林集群的比例较高，这表明PyTorch SC在包之间比张流SC具有更复杂的依赖关系。大型集群的核心包往往是任何一个SC中的官方包，并展示了每个SC的（目标）专长。

6rq3：在多大程度上，为什么软件包要脱离两个SCS？

6.1方法

我们认为一个包脱离PyPIDLSCifit在SC（Vse）中的最新版本比PyPI（Vpgpi）中的最新版本要早。换句话说，它在版本Vs..之后从安装依赖中删除DL框架及其依赖项通过这种方式，我们发现364个包和201个包分别脱离了张量流和PyTorch SC。我们从包的代码存储库中确定了脱离的原因，因为代码存储库文档记录了丰富的开发活动data（e.g..commits、问题、拉请求和发布说明[74,75]），这些原因已被用于识别之前工作中各种软件开发场景的原因，例如，库迁移[41]和依赖数据库弃用[42]。具体来说，我们采取了以下三个步骤：

0找到包的代码存储库对于每个脱离SC的包，我们首先根据之前的工作[61,69,70]，从其PyPI页面上的存储库链接中确定它的代码存储库。如果链接中断或不存在，我们将它的名称和描述作为其代码存储库。大约75%的软件包链接到SC中的存储库（TensorFlow SC：275/364，PyTorch SC：150/201）。该比率与以前的工作[68]相当。

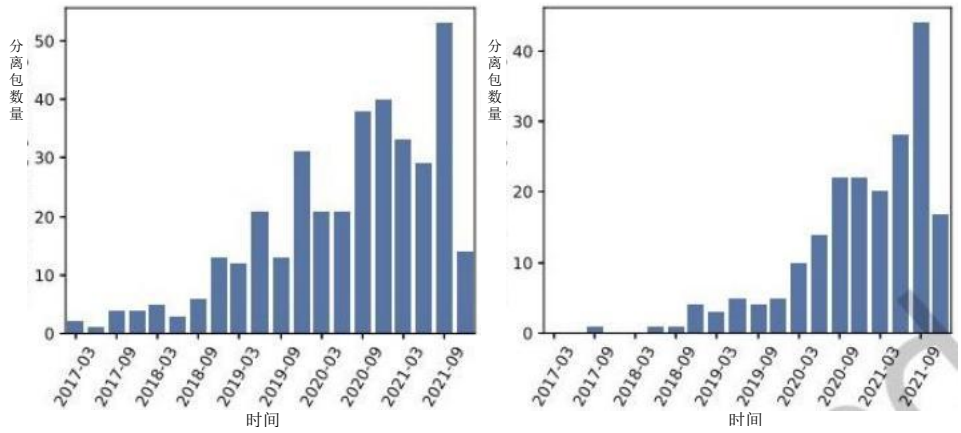
②提取与脱离相关的文本。对于每个定位的代码存储库，我们首先确定从setup.py和requirements.txt等依赖规范文件中删除DL框架及其依赖项的提交。如果提交消息证明脱离（例如，删除依赖项），我们记录相关文本。否则，我们将查看问题、提取请求和包含依赖项名称的发布注释，并选择相关文本。最后，我们提取了两个SCs中约30%的软件包的脱离相关文本（TensorFlow SC：83/275，Py Torch SC：44/150）。这个比例很低，因为开发人员没有记录他们工作背后的基本原理，正如之前的工作[66]所揭示的那样。尽管如此，该比率与之前的工作相当，这些工作确定了迁移原因[41]和代码存储库中的[42]。

③进行专题分析。前两位作者对提取的文本进行了主题分析，以挖掘软件包脱离的原因。每个检查员独立地读取和重读提取的文本，生成描述脱离原因的初始代码，并将类似的代码合并到主题中。然后，他们讨论生成的代码和主题，直到他们达成协议，并开发一个编码指南。接下来。每个检查员都独立地使用编码指南对所有提取的文本进行编码。请注意，多个代码可以分配给一个文本片段。在这种情况下，我们使用MASI距离[60]来测量两个检查员的代码和克里彭多夫的alpha[51]之间的距离来测量评估者之间的可靠性。克里彭多夫的alpha值是0.927，高于推荐的阈值0.8 [52]。召开一次会议以解决冲突并分配最终的代码。这个过程大约需要200个工时，相关的材料可以在复制包中找到。

6.2结果

图6显示了两个SCs中脱离接触的软件包的趋势。我们可以看到，每个季度脱离任何一个SC的软件包的数量总体上呈增长趋势（请注意，我们只有2021年最后一个季度的部分数据）。在2021年第三季度，张量流和PyTorch SC都经历了脱离软件包的急剧增长。这些发现表明，张力流和Py火炬SC都面临着日益严重的包装脱离问题。

我们分别从张量流和PyTorch SC中的83和44个脱离软件包中确定了与三个方面相关的7个脱离原因。表1显示了在SC中提到每个原因的脱离包的数量和比例。请注意，比例之和超过100%，因为一个包可能因多种原因脱离SC。在本节的其余部分中，我们将详细说明每个原因。**依赖关系。**38（45.78%）和13（29.55%）在张量流和PyTorch SC中的包脱离是由于依赖问题。这个类别包括两个原因：被删除的依赖关系会导致不兼容性或过于臃肿。



在张力流SC（左）和PyTorchSC（右）中每季度的分离包的Fig.6.Number。

不兼容性。一些包脱离SC，因为它们的依赖关系导致不兼容

与其他必需的软件包、硬件或操作系统有关的问题或错误。这是23(27.71%提到的拉伸流SC中最常见的原因。最主要的原因是张力流软件包之间的冲突。如第3节中提到的，张量流框架发布了三个软件包：张量流。张流cpu和张流流gpu。这三个包共享相同的API名称，这意味着安装其中一个包会覆盖另一个已经安装的包。io开发人员通常采用两种策略来解决这个问题。第一种方法是从依赖关系中删除紧张或流包，并指导最终用户为他们的环境安装正确的包。另一种方法是检查setup.py中的环境（例如，gpu的存在），并安装相应的张量流软件包。此策略的缺点是，必须从源代码处安装该软件包。

膨胀。15（18.07%）和四个（9.09%）软件包脱离了两个SCs，因为它们不使用

完全删除了依赖项，或者包开发人员假设依赖项已经安装在
环境。例如，一个已经将Py Torch从其依赖项中删除的软件包解释说
设置开发环境在PyTorch上工作，你可能已经安装了：我膨胀了

依赖关系增加了程序包的安装大小和导入时间。一旦找到，开发人员将从依赖项规范文件中删除它们。而膨胀的依赖性问题似乎在拉伸流SC中的软件包中更为常见，这与第3节中的手动检查相一致。

功能。27（32.53%）和24（54.55%）软件包分别脱离张量流和PyTorch SC以改进功能。

性能16个和7个软件包脱离张力流和PyTorch SC以获得更好的性能，

eg..simpler模型，更快的导入时间和执行时间。有些包只是删除了依赖关系，例如，用中置文件替换MTCNN来删除张量流+Keras依赖关系。！2.一些人选择迁移到其他人那里
包，例如。如果我们去掉PyTorch，然后（也许），代码可能会变得更干净、更快、更灵活
使用并发期货。进程池执行程序：！

t0https://github.com/lensorflow/tensorfow/issues/7166
1thttps://github.com/Quansight/pytest-pytorchfissues/13#issue-861367114
i2https//github.con/apangasa/bestof/issues/21
t3https//github.con/openclimatefix/nowcasting_dataset/ssuesy/86

ACM跨。软。雕刻方法。

表1.软件包脱离的原因。

原因	TensorFlow SC[1]	PyTorch SC[2]
依赖性	38(45.78%)	13(29.55%)
不兼容性	23(27.71%)	10(22.73%)
膨胀的	15(18.07%)	4 (9.09%)
功能	27(32.53%)	24(54.55%)
表演	16(19.28%)	7 (15.91%)
简化	9(10.84%)	13(29.55%
独立于框架	3 (3.61%)	5(11.36%)
安装	25(30.12%)	14(31.82%)
灵活性	15(18.07%)	1(2.27%)
尺寸调整	10(12.05%)	13 (29.55%)

张量流SC共183个脱离的包
我的行程

简化。一些包根据各种包提供了繁琐的功能。

它们脱离了SC，以简化其功能，以便开发人员更易于使用。在拉伸流和PyTorch SC中，有9个（10.84%）和13个（29.55%）脱离工作的包提到了这个原因。这一原因和尺寸修剪是PyTorch SC中最常见的两个脱离原因。软件包通常采用三种方法来简化其功能。有些包将某些功能转换为可选特性，并将这些功能所需的依赖项转换为额外的依赖项[2]。例如，这些……如果用户需要解释sk学习模型，则可能不需要。如果需要，它们可以使用pip作为可选的依赖项安装。

一些软件包将它们的功能分成多个软件包。例如，
我们最近将管道深度学习-准备数据迁移到我们的姐妹项目ClinicaDL中，该项目处理与深度学习相关的一切。I[5]Some软件包采用了一种相当激进的完全删除的方法

需要这些依赖关系的功能。例如，我们将删除所有依赖项和代码相关项
模型训练和实验，以保持数据的洞察力集中在io和统计学上.16

独立于框架。一些软件包脱离了一个特定的DL SC，以支持更多的DL框架。

例如，Tonic现在没有直接的PyTorch和PyTorch视觉依赖，这意味着如果某人
想要将它与另一个管道一起使用。紧张)，他们将能够做到这一点。！7三、五人脱离

这两个SCs中的软件包都提到了这个原因。

安装。25（30.12%）和14（31.82%）软件包与张量流和PyTorch SC分离。

灵活性。包脱离SC，以简化安装过程以适应

各种环境。灵活性和兼容性的区别在于，后者是被动触发的
而前者被主动执行，以放松安装要求。例如，有
主要的张量流软件包在大多数情况下都可以工作.….

https//github.com/interpretml/DiCE/pull74
i5https//github.com/aramis-lab/clinica/issues/458
t6https://github.com/Unity-Technologies/datasetinsights/ssuey/143
t7https://tonicreadthedocs io/en/latest/about/release_noteshtmleid4

18.K.Gao等人。

但最终我们应该允许用户在紧张流SC中安装一个独立的包.1815

在PyTorch SC中只有一个脱离的包提到了这个原因。如上所述，应变流为不同的硬件平台提供了单独的包。而PyTorch只提供一个包。因此
张量流SC中的软件包更关心硬件需求。

*尺寸修剪。张流或PyTorchis很大，它们的构建发行版占用数百兆的空间，导致SC中的包的安装规模也很大。因此，一些软件包脱离SC，以减少其安装尺寸。与包因为不使用依赖关系而脱离SC的情况不同，在大小修剪的情况下，脱离依赖关系的包使用依赖关系。除了删除或将依赖项转化为额外的依赖项之外，我们还发现了这些包
选择迁移到任何一个SC中的ONNX。例如，1)对pytoich的依赖性很大……让我们尝试替换
它由ONNX，希望更小。92)从张量流切换到onnx运行时，更小的二进制大小20.10

（12.05%）和13（29.55%）包从两个SC中分离出来以削减其安装尺寸，这表明PyTorch SC中的包更有可能遭受大的安装尺寸。

我们进一步分析了七个原因的趋势，以确定为什么在两个SCs中脱离参与的软件包的数量增加了（如图6所示）。我们发现，简化功能和调整安装大小是PyTorch SC中脱离包增加的主要原因，可能是由于PyTorch框架的规模不断增加，以及这些脱离包提供的越来越繁琐的功能。我们还发现，由于不兼容性和性能问题，增加的包脱离紧张流SC，可能是因为越来越多的包遇到三个紧张流包之间的冲突，而紧张流不能满足脱离包的性能要求，如导入时间和最先进的模型。

RQ3的摘要：任何一个SC中脱离的包的数量都呈上升趋势。包
脱离SCC有七个原因：依赖性问题、功能改进和易于安装。TensorFlow SC中最常见的脱离原因是依赖不兼容性问题，而在Py Torch SC中是为了简化功能和减少安装大小，这表明不同DLSC中的包面临不同的依赖管理问题。

7个含义

我们的研究为DL框架供应商、研究人员和利益相关者就PyPI DL SCs的维护和依赖性管理实践提供了丰富的建议。为了征求从业者对我们建议的意见，我们采访了5人，其中包括两名桨桨DL框架的贡献者，一名MindSpore DL框架的贡献者，以及来自两家大型IT公司的两名DL研究人员。总的来说，他们认识到我们的建议是明确的和信息丰富的，而且一些建议与他们的做法相一致。它们还提供了关于一些建议的额外信息。我们将在本节中详细阐述这些建议。

7.1 DL SC维护

框架改进。一个像样的DL框架是一个繁荣的DL SC的基础。我们的发现揭示了DL框架应该考虑的两个问题。

- 包装尺寸。DL框架是非常大的包，其预构建的发行版占用数百兆字节的空间。因此，安装直接或过渡地依赖于它们的DL包会消耗大量的磁盘空间和网络流量。特别是，许多软件包都脱离了DL SCs

i8https://github.com/OpenNMT/OpenNMT-tf/pull/802
tPhttps:/github.con/raymon-ai/raymon/ssuey/50
20https:/github.com/paradigm/ultrastar_pitch/issues/5

ACM跨。软。雕刻方法。

PyTorch SC，以减少其安装尺寸（RQ3）。因此，DL框架可以考虑控制其包的大小，以保留包。我们设想DL框架可以分裂成一个包含许多插件组件的基本包。基本包包含常用的功能，如CNN层，而插件组件包含针对特定的tasks，e.g.、可视化的功能。桨的贡献者指出，包含多个版本的CUDA二进制文件是DL框架包规模过大的主要原因。他们进一步透露，他们控制着桨桨的包装
通过设置支持的CUDA版本的范围。

- *多硬件支持。张流SC的一个主要原因是不同因素之间的冲突*

分别为CPU和GPU发布的流包（RQ3）。目前，有两种方法可以支持多个硬件平台。首先是为不同的硬件平台构建单独的软件包。大多数DLframeworks。张力流、MxNet和桨桨采用这种方法。第二种是提供一个支持广泛硬件平台的单一包，就像PyTorch所做的那样。这两种方法都有其局限性：前者导致不同包之间的冲突，而后者增加了包的大小。桨和头脑精灵贡献者证明他们选择第一种方法是因为PyPI为包分发设置了大小限制（60MB），他们必须向PyPI维护者要求增加分发大小限制。1这种困境很大程度上是由于PyPI缺乏支持上传针对不同硬件平台的预构建发行版。22 PyTorch滚轮存储库利用本地版本标识符[1]来区分不同硬件平台的发行版。3然而，在PyPI上不允许使用本地版本标识符，更糟糕的是，打包包最近删除了[18]对本地版本标识符的支持。总之，当前的Python打包规范对DL软件包维护者带来了大量的挑战，因为它们无法优雅地支持多个硬件平台。因此，Python社区可能会建议对打包规范进行调整(例如，重新引入本地版本标识符或具有
pip理解软件包之间的冲突)。

晶包多样性。DL在我们的社会中被广泛采用，激发了不同的应用程序和工具，从RQ1中确定的34个包域可以证明，包的多样性反过来促进了SC的发展
繁荣我们建议DL框架供应商从两个方面使链多样化。

- 竞争优势值得注意的是，TensorFlow和Py Torch SC分别开发了基础设施和应用软件包（RQ1）。我们还发现，官方的软件包在形成SC（RQ2）中起着重要的作用。因此，对于新兴的DL框架，它们可以从为NLP和CV等两个SCs中占软件包最多的域提供官方包开始。这些官方软件包可以提供通用的数据集和模型架构。然而，我们注意到并不是所有的DL框架都优先考虑CV和nlp相关的包。例如，MindSpore直到2023年才会提供这些软件包（即MindCV和MindNLP）。然后，他们可能会花精力在特定的领域（e.g..Social方面和强化学习）上，以增加他们在DL社区中的受欢迎程度和独特的竞争力。这一建议与心灵贡献者证实的心灵精灵所做的相一致。具体来说，MindSpore提供了许多科学软件包，如头脑地球和MindSPONGE，以在科学领域获得竞争优势。
- 新兴领域。DL框架供应商可以跟踪采用DL技术会产生影响的新兴领域和有前途的领域。例如，作为SE研究人员，我们对SC中缺乏与SE相关的流行包感到惊讶。鉴于DL技术越来越多地应用于不同的SE任务(例如。代码搜索[25]和代码生成[64])，并显示了良好的结果，我们预计在不久的将来，DL和SE社区可以一起构建工具来促进SE，e.g..the的DL的开发

2 ‘https://www.dampfkraftcom/code/distributing-large-fles-with-pypi html
22https://讨论蟒蛇org/t what-to-do-about-gpus-and-the-buill-distributions-that-support-them/7125
23https://download pytorch org/whl/torch_stable.html

SE的“ImageNet”、公共数据（如抽象语法树）预处理工具，以及用于各种SE任务的大型预训练模型。桨桨和心灵的贡献者认识到A14SE包的缺陷，并考虑在这一领域投入更多的努力。

协作。首先，RQ1揭示了DL在各种科学学科中被广泛采用。将DL技术应用到这些领域通常需要领域专家知识。因此，DL框架供应商可以与来自其他学科的研究人员和从业者建立协作。其次，DL模型的规模正在急剧增长，涉及数十亿个参数、数百gb的训练数据和大型GPU集群。培训这种模式几乎不是任何单一组织都负担不起的，因此需要与其他组织合作。最后，我们发现一些包依赖于来自两个SC（RQ1）的包，这反映了在一个特定的SC中缺乏工具支持。因此，我们也强烈鼓励与其他DL SCs中的软件包进行协作，以补充工具支持。桨贡献者还强调DL框架供应商和硬件供应商之间的协作，以促进硬件和软件的深度融合，并充分释放硬件的计算潜力。

风险管理。树和森林集群通过它们的根包（RQ2）组装了大约70%的~90%的包，这表明大多数DL包之间存在密集的依赖关系。这种关系放大了SC risks，e.g.malicious代码注入，而流行的根包中的漏洞引发了大量依赖包的维护者的恐慌。因此，对于DL SCs和SCs在其他领域和包装生态系统。应该更加注意这些根软件包，以避免和减少风险。一方面，根包中的漏洞将沿着SC传播，并影响同一集群中的其他包。因此，可以设计漏洞通知和补丁传播工具来减轻根包的安全问题的级联影响。这些工具可以与软件包管理器（例如，npm audit[2]）或代码托管平台（e.g.Dependabot[2]）集成，从而为下游开发人员和最终用户提供及时的更新通知。另一方面，如果根包脱离SC，即，删除对DL框架及其依赖项的依赖项，它们的依赖项也将脱离SC。因此，保留根包对SC的稳定性和可持续性也至关重要。

7.2 DL SC依赖性管理

互操作性。我们发现TensorFlow和PyTorch SC在软件包域（RQ1）的分布上存在很大差异，说明DL社区的资源分散在不同的DL SCs中。这可能会给DL开发人员增加额外的负担。例如，如果开发人员使用Py Torch SC中的包A进行原型开发，使用TensorFlow SC中的包B进行部署，她/他必须熟悉这两个框架，并可能编写冗余代码。因此，我们考虑框架之间的互操作性可以帮助开发人员利用不同的DL SCs [56]的优势。目前的互操作性工具，如ONNX和MMdnn [56]，主要关注于通过定义一组共同的操作符，在不同的DL框架之间转换模型。然而，由于不同DL框架支持的操作人员之间的差异，工程工作是强大的，应用场景仍然有限。桨的贡献者补充说，不同框架的同一操作员的不同和不断发展的实现也对现有的互操作性工具带来了挑战。理想情况下，DL框架可以符合一个通用的互操作性工具。但是，由于不同的激励机制，DL框架供应商并不容易达成共识。因此，DL从业者可以共同补充DL框架对现有工具的支持。此外，还可以探索实现DL框架之间的互操作性的新技术。

依赖部门。DL包通常引入许多依赖关系，这增加了依赖关系问题的风险，如依赖关系不兼容、繁琐的功能和大的安装大小（RQ3）。

2https://文档npmjs.com/cl/v9/commands/np-audit
25https://github.com/依赖

为了简化它们的依赖关系树并解决上述问题，一些包将需要依赖关系的非核心功能划分为“额外功能”。具体来说，这些包选择将其功能分割为一个具有主依赖项的基本包和多个具有它们所需要的相应额外依赖的可选组件。默认情况下，只安装基本程序包所需的依赖项。只有当用户显式地声明使用可选组件时，才会安装额外的依赖项。为了划分依赖关系，一个良好的模块化架构是一个先决条件。也就是说，与基包中提供的功能相对应的Python代码文件不会导入额外的依赖项，以便基包在没有额外依赖的情况下正常工作。因此，可以探索如何正确指定额外依赖项和相关自动化工具的实践。这两位DL从业者认为这个建议非常有用，因为他们开发了支持不同框架的包，这个建议可以帮助他们更好地组织依赖关系，并免除用户下载不必要的依赖关系。我们认为类似的做法可以应用于其他包装生态系统中的sc。

依赖性去博客化。我们发现15个和4个脱离TensorFlow SC和PyTorch SC的包提到它们根本没有使用元数据中声明的依赖关系（RQ3），这表明在PyPI包中可能是常见的。膨胀的依赖关系增加了程序包的安装大小，并减缓了导入过程。尽管一些工作已经研究了其他生态系统中的臃肿依赖，如Java/Maven，e.g.、膨胀工具[23]和进化[61,62]，但Python/PyPI和其他打包生态系统中的臃肿依赖仍未得到充分研究，并且缺乏自动膨胀Python依赖的工具。我们建议研究人员在未来弥合这些知识差距。

除了DL框架之外，我们认为控制包大小、风险管理、依赖划分和依赖扩展的影响也适用于其他打包生态系统中的包，如NPM。以控制包大小的含义为例，NPM开发人员经常遭受
在将大型包发布到NPM时失败，建议大型包“应该避免或
分解成更小的packages“.26

8对有效性的威胁

内部效度涉及到我们如何得出我们的发现。主要的威胁来自于手动标记包域、集群形状和脱离原因所引入的作者偏见。为了减轻作者的偏见，两个检查员独立地执行标签，通过一系列的讨论开发编码指南，并在整个过程中衡量评估者之间的可靠性。评分者之间的可靠性结果都表明很高的一致性。然后，从包的分布元数据中提取包之间的依赖关系是最近许多研究中的一种标准实践（例如，[33,55]）。但是，并不是所有的运行时依赖项都列在分发元数据中。为了评估这种情况的严重程度，我们在两个SCs中随机抽取100个包，下载它们的最新发行版，并将在代码中导入的包与元数据中声明的依赖项进行比较。我们发现有6个包导入没有在元数据中声明的包，但所有6个包都没有依赖包，每月下载量少于900个。因此，我们认为这种情况的影响应该是很小的，不会使我们的结果无效。未来的研究可能会分析软件包的源代码，以获得更精确的依赖性解析。对域分布分析的流行包的选择也对内部有效性构成了威胁。为了评估所选包的代表性，我们进一步在TensorFlow和PyTorch SC中分别抽取334和344个包（95%置信水平和5%置信区间）。我们使用派生的域来标记抽样的包，并获得类似的结果：超过85%（张量流SC：89.6%，PyTorch SC：87.4%）的包属于应用、基础设施和科学类别；两个sc分别支持基础设施（张量流SC：32.9%，PyTorch SC：22.2%）和应用程序（张量流SC：39.4%，Py Torch SC：51.2%）。因此，我们认为所选择的流行的软件包

20https://github.com/npm/npm/issue/12750

是代表。只有大约30%的脱离包被用于识别脱离的原因，这构成了另一个威胁。虽然我们不知道这些包有多代表性，但代码存储库记录脱离背后基本原理的包可能比没有脱离的包质量更好。因此，他们的实践也应该为一揽子计划脱离行动提供有价值的见解。最后，使用不那么近期的数据可能会威胁到我们的发现，因为我们的发现可能不能反映DL的最新进展，如大型语言模型（LLMs）和扩散模型。如果仔细考虑，我们可以看到使用不那么近期的数据主要威胁着RQ1的发现。然而，我们认为它们对我们的发现（尤其是RQ1）的影响是有限的，原因有三：1)进展（e.g..LLMs）主要是通过拥抱Face平台的预训练模型权重而不是软件包。2)它们背后的底层技术在2021年之前就已经被提出了，并由我们的数据集中的软件包提供。例如，变压器架构被llm广泛使用，并于2017年提出。我们的数据集包含许多提供变压器架构的包，如火炬变压器。3)研究进展主要集中在NLP和CV域，这是我们数据集中最流行的两个领域。这表明，即使考虑到进展情况，域的分布也可能不会发生很大的变化。因此，尽管使用了不那么近期的数据，我们相信我们的发现仍然为DL包SCs提供了有价值的见解。

外部效度涉及到我们的研究结果的推广。本文研究了两种最流行和最具代表性的DL框架——张量流和PyTorch的sc。作为比较，我们还为另外三个DL frameworks，i.e.、MindSpore、桨桨和MxNet构建SC，其中分别包括6、33和87个包，远少于张流SC（2567）和PyTorchSC（3278）中的包数量。因此，我们认为从两个繁荣的DL SCs中获得的结果也应该为其他DL框架提供有价值的见解。尽管如此，考虑到DL领域（例如不同的硬件平台）和PyPI包装生态系统的独特特点，研究人员在将我们的发现应用于其他领域（例如web框架）或其他包装ecosystems（e.g.Maven或NPM）之前必须谨慎。然而，我们的SC构建方法和分析框架应该可以推广到其他领域和其他包装生态系统。进一步的研究可以了解不同领域和不同生态系统之间的相似性和细节。然后，PyPI并不覆盖所有的Python包，因为并不是每个代码存储库都会向PyPI发布包。然而，考虑到PyPlis是开发人员找到、安装和发布Python包的最大和事实上的包注册表，并且主机超过48.8万个包，我们相信PyPIare中的包SCs具有代表性。把包发布在其他平台（例如，GitHub代码库）考虑将导致一个更完整的包SC，但它需要重要的努力，超出了当前study，e.g.how收集尽可能完整的Python代码库，如何准确地识别包发布的代码库和如何识别这些依赖包。我们把它作为未来的工作

构造效度是我们的集群大小和平均度量衡量集群复杂性的程度。这两个指标在之前分析包装生态系统中的SCs的工作中被广泛使用，例如（[30,33]）。我们相信，这两个指标有很高的潜力来反映集群的复杂性。

9结论

通过分析了近600万个PyPIpac包分布，我们构建了紧张流和PyTorch的版本敏感的SCs，并分析了它们的包域、集群和脱离。我们发现两个SCs中流行的软件包覆盖了跨越8个类别的34个域，两个SCs分别开发了基础设施和应用程序的专门化。我们发现两个SCs中的包簇主要是箭头的形状。星，树和森林，增加依赖的复杂性。大多数软件包都集中在树木和森林集群中。我们还发现，包脱离SC有七个原因：依赖问题、功能改进和易于安装。最常见的脱离接触

张力流SC和PyTorch SC的原因是不同的。总的来说，这两个sc共享相似的包域和集群形状，但在专门的包和依赖管理问题上有所不同。我们的发现为DL框架供应商、研究人员和从业者在PyPI DL SCs的维护和依赖管理实践提供了丰富的启示。

确认

国家自然科学基金资助项目：61825201和62332001。

参考文献

[1]2013.PEP440-版本标识和依赖性规范。https://peps.python.org/pep-0440/.（已于2023年2月2日访问）。[2]2015.PEP508-针对Py thon软件包的依赖性规范。https://peps.python.org/pep-0508.（已于2022年08月04日访问）。[3]2019.microsoft/CNTK：微软认知工具包（CNTK），一个开源的深度学习工具包。https://github.com/microsoft/CNT

所有的免责声明。（已于2022年12月15日访问）。

[4]2019.首选网络将其深度学习研究平台迁移到pytorch-首选网络，Inc.https://www.preferred。

jp/en/news/pr20191205/.（已于2022年12月15日访问）。

[5]2021.太阳风猎户座安全漏洞：软件供应链范式的转变。https://snyk.io/blog/solarwinds-orion-

安全漏洞，软件供应链的转变。（于2002年08月2日访问）。

[6]2022.核心元数据规范-Python打包用户指南。https://packaging.python.org/en/latest/specifications/core-

metadata/.（已于2022年2月8日访问）。

[7]2022.术语表-Python打包用户Guide.https://packaging.pythonorg/en/latesvglossary/.（已于2022年3月8日访问）。

[8]2022.onnx.ai.https//onnx.ai/.（已于2022年9月9日访问）。

[9]2022.•Python软件包Index.https://pypiorg/。（已于2022年12月22日通过）。

[10]2022.Python软件包索引（PyPI）-市场-谷歌云控制台。https://console.cloudgoogle.com/marketplace/product/gep-

公共数据系统。（已于2022年08月04日访问）。

[11]2022.PyTorch.https://pytorch.org/.（已于2022年12月27日访问）。

[12]2022.pytorch/vision：计算机视觉特有的数据集、转换和模型。https://github.com/pytorch/vision.（访问时间08/14/2022).

[13]2022.堆栈溢出开发人员调查，2022年https://survey.stackoverflow.co/2022/*最受欢迎的技术学家的部分。其他的部分

框架和库。（已于2023年1月31日访问）。

[14]2022.TensorFlow.https://www.tensorflow.org/.（已于2022年12月27日访问）。

[15]2023.API文档|张量流v2.12.0。https://www.tensorflow.org/api_文档。（已于2023年6月13日访问）。

[16]2023.Ecosystem|PyTorch.https://pytorch.org/ecosystem/.（已于2023年3月2日访问）。

[17]2023.igraph-Network分析软件。https://igraph.org/.（已于2023年6月27日访问）。

[18]2023.让我们允许‘-local。版本说明符中的版本标签-包装-允许本地版本标签

版本内的说明符/22781。（已于2023年2月2日访问）。

[19]2023.Packaging.https://packaging pypa.io/en/stable/.（已于2023年3月2日访问）。

[20]2023.PyTorch C++API-PyTorch主documentation.https://pytorch组织/cpp文档/。（已于2023年6月13日访问）。

[21]Sadika Amreen，博格丹•比切斯库，兰迪•布拉德利。Tapajt Dey.马宇星，奥德里斯莫库斯，萨拉穆萨维和罗素扎雷茨基2019。一种测量牙线生态系统的方法。新加坡施普林格，新加坡，1-29。https://doi.org/10.1007/978-981-13-7099-1_ 1

[22]Dzmitry巴赫，和Yoshua Bengio.2015.Neural机器翻译联合学习对齐和翻译。*在第三届学习代表国际会议上，ICLR 2015。圣地亚哥美国加州，2015年5月7-9日，会议跟踪记录，*

Yoshua Bengio和YannLeCun（电子版）。

[23]Bobby R. Bruce，张天一。阿罗拉、徐国庆和Kim.2020.JShrink：In-Depth调查

破坏现代Java应用程序。在第28届ACM欧洲软件工程会议和联合联席会议的会议记录中

软件工程基础研讨会（虚拟事件，美国）（ESEC/FSE 2020）。美国计算机协会

纽约纽约，美国.135-146.

[24]Yulu曹。林陈，马万英。周玉明、林章Wang.2022.Towards更好的依赖管理：

首先来看看Python项目中的依赖性气味。IEEE软件工程学报（2022）.1-26。

[25]陈清英和明辉Zhou.2018.A神经框架，用于源代码的检索和总结。2018年第33届IEEE/ACM

自动化软件工程国际会议（ASE）.826-831。

陈振鹏，曹彦斌。刘元强，王浩宇。谢涛、刘玄哲，2020年。关于基于深度学习的软件部署所面临的挑战的综合研究。在第28届ACM欧洲软件工程会议联席会议的会议记录中

和软件工程基础研讨会（虚拟事件，美国）（ESEC/FSE 2020）计算机械协会，
纽约NY.USA.750-762.

[27]珍鹏陈。姚慧汉。楼一玲、曹彦斌、刘元强、王浩宇、玄哲Liu.2021.An对基于深度学习的移动应用部署缺陷的实证研究。在第43届软件工程国际会议的会议记录中（马德里，西班牙）（ICSE 21）。IEEE出版社，674-685。

[28]Fronsows Chollet.2017.Deep学习与Python（第一版）。美国曼宁出版有限公司。

[29]Daniela S. Cruzes和软件工程主题合成的Dyba.2011.Recommended步骤。2011年国际
经验软件工程与测量研讨会-284。.275

[30]亚历山大·德肯，汤姆·犯罪分子。和Maelick Claes.2016.On的《软件包依赖网络的拓扑结构：三种编程语言生态系统的比较》。在第十届欧洲软件架构研讨会上(哥本哈根。丹麦)（ECSAW 16）计算机机械协会。纽约，纽约，美国，第21条，第4页。

[31]亚历山大Decan。Tom Mens和Maelick Claes.2017.An对OSS包装生态系统中依赖问题的实证比较。在
2017年IEEE第24届软件分析国际会议。进化和再造（SANER）.2-12。

[32]亚历山大德肯，汤姆犯罪，和Eleni Constantinou.2018.On的安全漏洞的影响在Npm包依赖网络。在第十五届采矿软件资料库国际会议的会议记录上(哥德堡。瑞典）（MSR78）协会
计算机，纽约，美国，181-191。

[33]亚历山大Decan。汤姆男人和菲利普Grosjean.2019.An七软件依赖网络进化的经验比较
包装生态系统。经验性索菲威工程公司24,1（2019年2月1日），381-416。

[34]建康邓。贾郭。雪南南和斯特法诺斯Zafeiriou.2019.ArcFace：Additive角缘损失的深人脸识别
2019年IEEE/CVF计算机视觉和模式识别会议（CVPR）4685-4694。https://doi.org/10.1109/CVPR.2019.00482

[35]Tapajit Dey和Audris Mockus.2018.Are软件依赖性供应链指标对预测NPM软件包流行程度的变化很有用？。在第14届软件工程预测模型和数据分析国际会议（奥卢，芬兰）（承诺18）。美国计算机协会纽约纽约，美国，66-69。

[36]Malinda Dilhara，Ameya Ketkar，和Danny Dig.2021.Understanding Software-2.0：A研究机器学习图书馆的使用和
演化ACM跨。软的。雕刻方法的。第55条，第30、4条（2021年7月），第42条，pages.https:/doiorg/10.1145/3453478

[37]Kai Gao，王志行。他和明辉Zhou.2023.On研究了深度学习中软件工程需求的可变性：阶段、趋势和应用程序类型。IEEE《软件工程学报》49,2（2023），760-776。https://doi.org/10.1109/TSE.2022.3163576

[38]海桥谷、何浩浩、明辉Zhou.2023.Self-Admitted图书馆在爪哇岛的迁移。包装舒适的茎：一项比较研究。2023年IEEE软件分析、进化和再造国际会议（SANER）627-638。https://doi org/10.1109/SANER56733.2023.00064

韩[39]，邓四光，罗伟，陈志，尹建伟，新Xia.2020.An依赖网络的实证研究
深度学习图书馆。在2020年IEEE软件维护和进化国际会议（ICSME）.868-878。

[40]俊晓韩、伊马德·世哈布、万志远、邓四光、辛霞2020。程序员如何讨论深度学习
框架。经验软件工程25.4（017月2020）.2694-2747。

[41]Hao He、何龙志、古海桥、明辉Zhou.2021.A的Java图书馆迁移的大规模实证研究：流行率。
趋势和理由。在第29届ACM欧洲软件工程联合会议和研讨会的会议记录中
软件工程基础（雅典，希腊）（ESEC/FSE 2021）。计算机机械协会，纽约。美国纽约，
478-490.

何龙志[42]、何浩、张玉霞、明辉Zhou.2023.Automating依赖更新的实践：探索性研究
GitHub依赖。IEEE《软件工程学报》49,8（2023），4004-4022。https://doi.org/10.1109/TSE.2023.3278129

[43]Sture Holm.1979.A简单的顺序拒绝多重测试程序。斯堪的纳维亚统计杂志6,2（1979），.65-70。
http://www.jstor org/stable/4615733

[44]开封黄。陈比环、吴素生、曹俊明、马磊、深度学习区的新Peng.2023.Demystifying依赖虫。在第31届ACM联合欧洲软件工程会议和软件工程基础研讨会（<conf-loc>，<city>San Francisco</city>，<state>CA</state>，<country>USA</country>，</conf-loc>）（ESEC/FSE 2023）计算机协会，纽约，纽约，美国，450-462。https://doi.org/10.1145/3611643.3616325

[45]Nargiz Humbatova.古内尔贾汉吉罗瓦，加布里埃尔巴沃塔，文森佐里乔。Andrea Stocco和Paolo Tonella.2020.Taxonomy研究了深度学习系统的真实故障。在ACM/IEEE第42届软件工程国际会议会议记录上(首尔。韩国)（ICSE 20）。计算机机械协会，纽约，纽约，美国，1110-1121。

[46]Md Johirul lslam.张阮。潘和拉扬。2019.对深度学习错误特征的综合研究。
在2019年第27届ACM欧洲软件工程联合会议和欧洲软件工程基础研讨会上
软件工程(塔林工程公司。爱沙尼亚)（ESEC/FSE 2019）计算机机械协会，纽约。纽约，美国。510-520.

[47]Md Johirul lslam.阮阮和Rajan.2020.Repairing深度神经网络：修复模式和挑战。
在ACM/IEEE第42届软件工程国际会议会议记录上(首尔。（ICSE‘20）协会

计算机械。纽约.1135-1NY, USA146.

[48]LiJi，HaoChong。王小音。黄林鹏和宣生Lu.2020.An对张力流内虫子的实证研究。在针对高级应用程序的数据库系统中。Yunmook Nah.本崔。李宗伟。徐羽。月亮和万（版）。施普林格国际出版公司。查姆，604-620。

[49]LiJi，HaoChong。王小音。黄林鹏。宣生Lu.2021.The症状、原因和修复内的虫子
学习图书馆。系统与软件杂志177（2021），110935。

[50]Riivo Kikas，乔治斯 • 古西奥斯，马龙 • 杜马斯，和Dietmar Pfahl.2017.Structure和软件包依赖网络的演变。
在采矿软件存储库国际会议论文集(布宜诺斯艾利斯。阿根廷）（MSR '17）。IEEE出版社，
102-112.

[51]Klaus Krippendorff.2011.Computing克里彭多夫的阿尔法-可靠性。(2011).

[52]Klaus Krippendorff.2018.Content分析：其方法的介绍。Sage出版物。

[53] • 恩里克 • 拉里奥斯 • 巴尔加斯。毛里西奥 • 安妮切。克里斯托夫 • 特鲁德，地方法官布伦丁克，和乔治奥斯 • Gousios.2020.Selecting的第三方
图书馆：从业者的观点。第28届ACM欧洲软件工程会议联席会议会议记录
软件工程基础研讨会（虚拟事件，美国）（ESEC/FSE 2020）。美国计算机协会
纽约纽约，美国。245-256.https:/doi.org/10.1145/3368089.3409711

[54]李振明、王振明、林泽奇、成志张、建光Lou.2022.Nufix：Escape。在
第44届软件工程国际会议的会议记录（匹兹堡，宾夕法尼亚州）（ICSE "22）。计算协会
机械公司，纽约，纽约，美国，1545-1557。

刘[55]成伟、陈森、范玲玲、陈比环、刘杨和新Peng.2022.Demystifying在NPM生态系统中的脆弱性传播及其通过依赖树的进化。在第44届软件工程国际会议的论文集上
（宾夕法尼亚州匹兹堡）（ICSE 22）。计算机机械协会，纽约，纽约，美国，672-684。

[56]于刘，陈程、张、秦婷、项吉、林浩翔和毛Yang.2020.Enhancing之间的互操作性
深度学习框架通过模型转换。第28届ACM欧洲软件工程联席会议会议记录
软件工程基础会议和研讨会（虚拟事件，美国）（2020年的会议）。计算协会
机械公司，纽约，纽约，美国，1320-1330。

[57]H.B.Mann和D.R.Whitney.1947。在检验两个随机变量中的一个是否随机地大于另一个时。《数理统计学年鉴》18,1（1947）.50-60。

[58]Mary L McHugh.2012.Interrater的可靠性：kappa统计量。生化医学22、3（2012），276-282。

[59]约翰 • 门泽，威廉 • 德威特。金伯勒，闵孙宏。南希W. Nix。卡洛D.史密斯，和扎克G。Zacharia.2001.DEFINING
供应链管理。《商业物流杂志》22,2（2001），1-25。https://doi.org/10.1002/j.2158-1592.2001.b00001.x
arXiv: https//onlinelibrary.wiley.com/doi/pdf/10.1002/j.2158-1592.2001.tb00001.x

[60]Rebeca Passonneau.2006.Measuring关于语义和语法注释的集值项目（MASI）的协议。在诉讼程序
第五届语言恢复和评价国际会议（LREC '06）。欧洲语言资源协会（ELRA），
意大利热那亚。

[61]塞萨尔 • 索托-瓦莱罗、托马斯 • 杜里欧和Benoit Baudry.2021.A对爪哇岛依赖性膨胀的纵向分析。在第29届ACM欧洲软件工程会议和软件工程基础研讨会（雅典，希腊）（ESEC/FSE 2021）。计算机机械协会，纽约，纽约，美国，1021-1031。

[62]塞萨尔索托-瓦莱罗，尼古拉斯哈兰德，马丁蒙珀罗斯，和Benoit Baudry.2021.A全面研究膨胀的依赖关系
麦文生态系统。实证软件工程26.3（2021年3月25日），45。

[63]Klaas-Jan Stol，Paul Ralph和Brian Fitzgerald.2016.Grounded软件工程研究理论：批判性评论和
指导方针。在第38届软件工程国际会议（奥斯汀，德克萨斯州）的论文集上（ICSE '16）。的协会的
计算机械。纽约，美国，120-131。

[64]Sun泽宇，朱奇浩。熊英菲。孙，LiliMou，Lu Zhang.2020.TreeGen：A基于树的变压器架构进行代码生成。行政情报会议记录34,05(4月。2020),8984-8991.https://doi.org/10.1609/aaai.v34i05.6430

谭[65]鑫、高凯、周明辉、李Zhang.2022.An对深度学习供应链的探索性研究。在第四届会议中
国际软件工程会议（匹兹堡，宾夕法尼亚州）（ICSE '22）计算机机械协会。新的
约克郡NY.USA.86-98.

[66]英晨天、张玉霞、林江先生、惠Liu.2022.What发出了一个很好的承诺信息？。在诉讼程序中
第44届软件工程国际会议（匹兹堡，宾夕法尼亚州）（ICSE '22）。美国计算机协会新的
约克郡纽约。美国，2389-2401。

[67]V.A.Traag,L.沃尔特曼和新泽西州van Eck.2019.From鲁万到莱顿保证联系良好的社区。科学家报告
9.1（3月26日2019）.5233。

[68]马拉瓦利耶夫。和James Herbsleb.2018.Ecosystem-Level在开源项目中的持续活动的决定因素：PyPI生态系统的一个案例研究。在2018年第26届ACM欧洲软件工程联席会议的会议记录中

ACM跨。软。雕刻方法。

和软件工程基础研讨会(布埃纳维斯塔湖，佛罗里达。USA)(ESEC/FSE 2018).计算协会
机械。纽约NY.USA.644-655.

[69]Duc-Ly Vu.2021.py2src：Towards自动（和可靠的）识别源的Vu.2021.py2src：Towards PI包。2021年第36届IEEE/ACM *自动化软件工程国际会议（ASE）1394-1396。*

[70]Duc-Ly Vu.法比奥先生。伊万帕什琴科。亨里克板，和安东尼诺Sabetta.2021.LastPyMile：Identifying的差异
在源和包之间。第29届ACM欧洲软件工程联合会议和研讨会
关于软件工程的基础(雅典。希腊）（ESEC/FSE2021）计算机机械协会，纽约。NY,
美利坚合众国780-792.

[71]公爵先生，比如帕什琴科。法比奥·马萨奇，亨里克板，安东尼诺Sabetta.2020.Typosquatting和抢抢注攻击蟒蛇生态系统。2020年IEEE欧洲安全和隐私研讨会（EuraS&PW）.509-514. https://doi.org/10.1109/EuroSPW51379.2020.00074

王[72]英、明文、刘叶邦、王一波、李振明、王超。海宇，致志张。张旭，朱志良。2020.守望者：监控Python库的依赖冲突。第42届软件工程国际会议（韩国首尔）（ICSE '20）。计算机机械协会，纽约，纽约，美国，125-135。

[73]Erik·威特恩，菲利普·苏特，和Shriram Rajagopalan.2016.A看看犹太软件包生态系统的动态。在诉讼中
第13届采矿软件存储库国际会议（奥斯汀，德克萨斯州）（MSR '16）。美国计算机协会
纽约纽约。美利坚合众国.351-361.

吴[74]建宇、郝和、肖文新、高凯、明辉Zhou.2022.Demystifying GitHub软件发布说明问题。在
第30届IEEE/ACM国际项目理解会议（虚拟事件）会议论文集（ICPC '22）。协会
计算机械。纽约，美国，602-613。https://doi.org/10.1145/3524610.3527919

吴[75]吴建宇、徐伟伟、高凯、李景月、明辉Zhou.2023.Characterize《GitHub项目软件发布说明：结构、写作风格、内容》。2023年IEEE索菲威分析、进化和再造国际会议（SANER）.473-484。https://doi org/10.1109/SANER56733.2023.00051

徐[76]万伟伟、何浩、高凯、明辉Zhou.2023.Understanding以及修复PyPI生态系统中的开源许可不兼容性。2023年第38届IEEE/ACM自动化软件工程国际会议（ASE）.178-190.https: //doi.org/10.1109/ASE56229.2023.00175

[77]Yilin Yang。何天星、夏志龙、杨Feng.2022.A对深度学习中虫子特征的全面实证研究
构架信息和软件技术151（2022），107004。

[78]Nusrat·扎汉，托马斯·齐默尔曼，帕特里斯·戈德弗罗伊德，布伦丹·墨菲。钱德拉·马德拉和Laurie Williams.2022.What是npm供应链中的薄弱环节？。2022年，第44届软件工程国际会议：软件工程在实践中（ICSE-SEIP）.331-340。

张[79]如，小文聪，张宏宇。刘于、林浩翔、毛Yang.2020.An对深度学习工作项目失败的实证研究。在ACM/EEE第42届软件工程国际会议（韩国首尔）（ICSE 20）的会议记录中
美国计算机协会纽约，美国，1159-1170。

[80]天一张。高翠云、马磊、吕迈、Kim.2019.An深度学习应用开发中常见挑战的实证研究。2019年IEEE第30届软件可靠性工程国际研讨会（ISSRE）104-115。

[81]张晓宇。翟娟、马世清、赵Shen.2021.AutoTrainer：An自动DNN训练问题检测与修复
体系在第43届软件工程国际会议的会议记录中(马德里。Spain)(ICSE 21).IEEE出版社，359-371。

[82]Hewho张，陈一凡。张成志。熊英飞和陆Zhang.2018.An对张量流程序错误的实证研究。在
第27届ACM签名软件软件测试与分析国际研讨会论文集（阿姆斯特丹，荷兰）（ISSTA 2018）
计算机机械协会，纽约，纽约，美国，129-140。

张玉霞，周明辉。Audris Mockus和Zhi Jin.2021.Companies参与OSS开发——开放堆栈IEEE软件工程交易的实证研究47,10（2021），2242-2259。