

可追溯性转型：使用预训练的 BERT 模型生成更准确的链接

林金峰、刘亚林、曾庆凯、蒋孟、Jane Cleland-Huang

计算机科学与工程

圣母大学

美国印第安纳州圣母院

jlin6, yliu26, qzeng, mjian2, jane Huang@nd.edu

抽象的—软件可追溯性建立并利用不同开发工件之间的关联。研究人员建议使用深度学习跟踪模型将自然语言工件（例如需求和问题描述）链接到源代码；然而，它们的有效性受到标记数据的可用性和运行时效率的限制。在这项研究中，我们提出了一种称为 Trace BERT (T-BERT) 的新颖框架，用于生成源代码和自然语言工件之间的跟踪链接。为了解决数据稀疏问题，我们利用三步训练策略，使跟踪模型能够从密切相关的软件工程挑战（具有丰富的数据集）中转移知识，以比以前实现的精度更高的方式生成跟踪链接。然后，我们应用 T-BERT 框架来恢复开源项目中问题和提交之间的链接。我们比较评估了三种 BERT 架构的准确性和效率。结果表明，Single-BERT 架构生成了最准确的链接，而 Siamese-BERT 架构则在执行时间显著减少的情况下生成了类似的结果。此外，通过学习和转移知识，框架中的所有三个模型都优于经典的 IR 跟踪模型。在三个评估的真实 OSS 项目中，最好的 T-BERT 稳定优于 VSM 模型，使用平均精度 (MAP) 测量的平均改进为 60.31%。由于训练数据不足，RNN 在这些项目上表现严重不佳，而 T-BERT 通过使用预训练语言模型和迁移学习克服了这个问题。

因此，在过去的几十年里，研究人员探索了各种自动生成和发展链接的自动化方法。技术包括概率技术 [3]、向量空间模型 (VSM)、[4]、潜在语义索引 [5]、[6]、潜在狄利克雷分配 (LDA) [7]、[8]、AI 群体技术 [9]、循环神经网络 [10] 集成语义、启发式方法 [11]–[13]、技术组合 [7]、[14]、[15] 以及决策树和支持向量机的使用 [16] 将时间依赖性和其他与流程相关的信息集成到跟踪任务中。尽管做出了所有这些努力，生成的跟踪链接的准确性仍然低得令人无法接受，因此业界一直不愿将自动跟踪解决方案集成到其开发生命周期中。主要阻抗是语义阻抗，因为大多数现有技术依赖于单词匹配——直接匹配（例如，VSM）、基于主题的匹配（例如，使用 LSI 或 LDA）或基于构建特定领域本体的间接匹配弥合术语差距 [17]。结果好坏参半，特别是当应用于工业规模的数据集时，可接受的高于 90% 的召回率通常只能在极低的精度水平下实现 [18]。

软件可追溯性链接是指在软件开发中建立一种追踪关系，以便追溯软件各个组成部分之间的关联和依赖关系。这种链接通常通过文档、工具或系统来实现，目的是记录软件项目中不同部分之间的关系，以便更好地理解和管理软件的开发、维护和演化过程。

具体来说，软件可追溯性链接通常包括以下几个方面：

需求追溯性链接：将软件需求与设计、实现和测试阶段的相应部分建立链接，确保每个需求都得到了满足，并能够追踪需求的变更影响。

设计追溯性链接：将软件设计文档与实际的编码和测试部分相关联，以确保设计的正确实施，并追踪设计变更对其他部分的影响。

编码追溯性链接：建立源代码与需求、设计和测试之间的关系，以追踪代码变更的原因和影响，同时支持代码审查和版本控制。

测试追溯性链接：将测试用例与需求和设计关联，以确保每个需求都经过了相应的测试，同时追踪测试结果与实际实现的一致性。

变更管理追溯性链接：跟踪软件项目中的变更，包括需求变更、设计变更和代码变更，以便了解每个变更的原因和影响。

通过建立这些追溯性链接，软件开发团队能够更好地理解和管理软件项目，提高开发过程的透明度，降低错误引入的风险，并支持有效的变更管理。

索引/术语——软件溯源、深度学习、语言模型

我、我简介

软件和系统可追溯性是创建和维护软件工件之间的关系并利用生成的链接网络来支持有关产品及其开发过程的查询的能力。在美国联邦航空管理局 (FAA)、美国食品和药物管理局 (FDA) 等认证机构规定的安全关键系统中，可追溯性被认为是至关重要的 [1]。如果存在，跟踪链接支持各种软件工程活动，例如影响分析、合规性验证和安全保证。不幸的是，在实践中，手动创建和维护跟踪链接的成本和精力可能会受到抑制，因此跟踪链接通常不完整且不准确 [2]。因此，可追溯性数据通常不被开发人员信任，并且常常未得到充分利用。

自动化方法表现不佳的主要原因之一是相关工件之间经常存在语义差距 [10]。无法推理语义关联并弥合这一差距的技术无法建立准确且相对完整的跟踪链接。最近的工作提出了用于可追溯性的深度学习 (DL) 技术 [19]、[20]，但没有提供有效的解决方案。例如，郭等人。[10] 提出了一种基于循环神经网络 (RNN) 的架构，并评估了两种类型的 RNN 跟踪模型 (LSTM [21] 和 GRU [22])，用于针对来自小型数据集的子系统需求和设计定义之间生成链接。工业项目。虽然他们的结果表明准确性随着训练集大小的增加而提高，但他们的方法没有在大型训练集上进行训练，因此无法在更大或更多样化的项目中推广。我们将 LSTM 和 GRU 方法纳入其中以进行比较，并在本文中将它们统称为 TraceNN (TNN)。

有两个主要因素阻碍了深度学习追溯解决方案的进步。首先是训练数据的稀疏性，因为深度学习技术需要大量的训练

软件工件，例如需求、设计定义、代码和测试用例都包含自然语言文本，并且

荷兰国际集团的数据。单个软件项目中可用的手动创建的跟踪链接（即黄金答案集）通常不足以训练深度学习模型。第二个障碍是在大型工业项目中应用深度神经网络的实用性，因为训练和利用深度神经网络比更传统的信息检索或机器学习技术要慢得多。


Commit Message	<div>✖ fixup! fixup! Handle a multi-line query on Enter key-press (fixes #1031)</div> <div>🔗 master (#1109) 📦 v3.0.0 v2.2.0</div>
Code Change set	<div>from prompt_toolkit.filters import HasFocus, IsDone, Condition</div> <div>from prompt_toolkit.filters import HasFocus, IsDone</div> <div>multiline=Condition(lambda: self.multiline),</div> <div># N.b. pgcli's multi-line mode controls submit-on-Enter (which</div> <div># overrides the default behaviour of prompt toolkit) and is</div> <div># distinct from prompt toolkit's multiline mode here, which</div> <div># controls layout/display of the prompt/buffer</div> <div>multiline=True,</div>
Issue Summary	<div>Labels</div> <div>🔴 History search regressions in v2 #1031</div>
Issue Description	<div> cws commented on 6 Apr 2019</div> <div>Description</div> <div>I use pgcli in vi and multi-line mode. It seems that history search with an empty query buffer has regressed in v2, possibly due to the <code>prompt_toolkit</code> upgrade.</div> <div>In v1.11.0 if I use history search with <code>ctrl-r</code>, having previously executed multi-line queries, I am shown the full query that matches the pattern. If I press <code>ctrl-r</code> again, I'm shown the full next query that matches. If I press <code>ctrl-c</code> the buffer is set to the first matching query.</div>

图 1：提交消息和代码更改集示例，其中添加了绿线并删除了红线。该提交由提交者标记为所描述的问题。

本文报告的工作解决了这两个关键阻抗，以便提供快速、准确的自动化追溯解决方案来解决工业问题。更具体地说，我们提出的可追溯性语言模型（LM）方法旨在（1）提供准确且因此更值得信赖的跟踪链接，（2）适用于训练数据有限的项目，以及（3）扩大规模以支持大型项目时间复杂度低的工业项目。我们的方法利用 BERT（乙单向乙编码器右发言来自 时间ransformers）作为其底层语言模型。谷歌于 2018 年推出的 BERT [23]，在各种 NLP 任务中取得了显著的改进，主要是因为它的双向方法提供了比单向语言模型更深入的上下文信息。在本文中，我们探索了 BERT 在可追溯性领域的使用——介绍了我们所说的跟踪 BERT（T-BERT）。

T-BERT 是一个用于训练基于 BERT 的关系分类器以生成跟踪链接的框架。三种类型的关系分类架构特别适合可追溯性。这些是单架构、双架构和连体架构，我们将在本文后面更深入地描述它们。我们比较了这三种架构在自然语言工件（NLA）和编程语言工件（PLA）之间生成跟踪链接的有效性。NLA 是功能请求、错误报告、需求和设计定义等工件，它们主要使用自然语言编写，但也可能包括代码片段。相比之下，PLA 主要是编程语言工件，例如代码文件、代码片段、函数定义和代码更改集，其中还包含自然语言注释

和描述符。我们通过生成从问题到代码（由变更集表示）的跟踪链接来评估 T-BERT，我们将其称为 NLA-PLA 追溯挑战。

本文的其余部分安排如下。秒。II 概述了我们在本文中讨论的具体研究问题。秒。III 和 Sec. IV 详细描述了我們实现 NLA-PLA 可追溯性的方法，而 Sec. V 描述了我们为评估我们方法的有效性而进行的实验。根据这些实验获得的结果，我们得出了第二节中研究问题的答案。六. 最后，第二节。第七至第二节。IX 讨论相关工作、有效性威胁和结论。

二. 磷罗布莱姆S声明

研究人员通过使用预训练的深度学习模型来解决各种 NLP 问题，解决了数据稀疏问题和训练大型模型的性能问题。这种方法将训练阶段分为预训练和微调阶段。在预训练阶段，使用大量未标记数据和自监督训练任务构建深度学习模型。然后在微调阶段，模型在较小的标记数据集上进行训练，以便执行更专业的“下游”任务。基本概念是，在更大、更通用的数据集上预训练模型所学到的知识可以有效地转移到监督训练标签有限的下游任务中。此外，预训练模型为模型优化提供了比随机种子模型更好的起点。因此，它减少了局部优化陷阱的可能性并提高了整体性能。在较小的数据集上微调预训练模型比从头开始训练深度学习模型所需的时间要少得多。虽然预训练通用模型非常昂贵，但预训练阶段只需要执行一次，然后就可以重复用于各种下游任务。

基于 BERT 的语言模型利用 Transformer [24]在预训练阶段从语料库中学习上下文信息，然后将学到的知识转移到下游 NLP 任务，例如问答、文档分类和情感识别 [23]、[25]。据我们所知，这是第一个将 BERT 或其他基于转换器的方法应用于软件可追溯性任务的研究。我们提出了一系列研究问题来评估 T-BERT 是否能够有效解决可追溯性问题。我们的第一个问题定义如下：

问题一：给定基于单 BERT 关系分类器、双 BERT 关系分类器和连体 BERT 关系分类器的 T-BERT 模型的三种变体，哪种架构是在准确性和效率方面解决 NLA-PLA 可追溯性的最佳架构？

除了研究深度学习模型架构之外，我们还探索了提高模型准确性的不同训练技术。正如郭等人所讨论的。[10]，深度学习跟踪模型可能会遇到“性能玻璃天花板”并以相对较低的精度收敛。因此，我们将第二个研究问题定义为：

问题2：哪种训练技术可以提高准确性，而不会受到之前观察到的玻璃天花板的影响？

Gururangan 等人在他们的领域自适应预训练 (DAPT) 研究中声称，使用领域语料库进行预训练的第二阶段可以带来性能提升。这一发现促使我们探索第三个也是最重要的研究问题：

问题3：T-BERT 能否从资源丰富的检索任务中转移知识，以提高下游 NLA-PLA 跟踪挑战的准确性和性能？

冯等人。[26]证明，使用大量函数定义进行预训练的 BERT 语言模型可以有效解决下游代码搜索问题。在该研究中，研究人员提供了文档字符串（即 Python 注释）作为用户查询，并利用 BERT 模型来检索相关函数。由于文档字符串和函数在代码库中总是成对出现的，因此可以提供用于代码搜索问题的充足训练数据。我们的 RQ3 探讨了是否可以利用代码搜索问题作为训练任务来改进 T-BERT 以应对软件可追溯性挑战。由于此步骤发生在预训练和微调之间，因此我们将其称为 *中级培训*

三. A普罗奇

跟踪检索算法动态生成工件之间的跟踪链接[27]，例如，通过将源工件（例如，Python 文件）链接到目标工件（例如，问题或需求）。可追溯性算法计算源和目标工件对之间的相关性，并提出最相关的对作为跟踪链接。在本节中，我们首先介绍基于 BERT 的模型的基本架构及其方差，然后介绍具有三个非常适合解决此可追溯性问题的特定关系分类器的 T-BERT。

A. BERT 和语言模型简介

语言模型表示单词序列[28]上的概率分布，通过适当的训练可以根据周围的上下文有效地捕获单个单词的语义。考虑到一般上下文的重要性，基于预训练语言模型构建的深度学习模型通常比直接在特定任务数据集上训练的模型取得更好的结果。基于BERT的模型的架构是基于变压器的，其中模型中的每一层都是一个变压器层。转换器层允许 BERT 模型专注于句子中任何位置的术语，并且训练 BERT 模型是通过一种称为掩码语言建模 (MLM) 的新技术来完成的。在 MLM 训练任务中，BERT 随机屏蔽输入文本中的单词，然后优化自身以根据上下文信息预测被屏蔽的术语。在这个预训练步骤中，大量的语料库被输入到基于 BERT 的模型中，并利用生成的模型通过对特定任务的数据集进行微调来解决不同的下游任务。BERT 的一个显著特点是其跨领域的统一架构

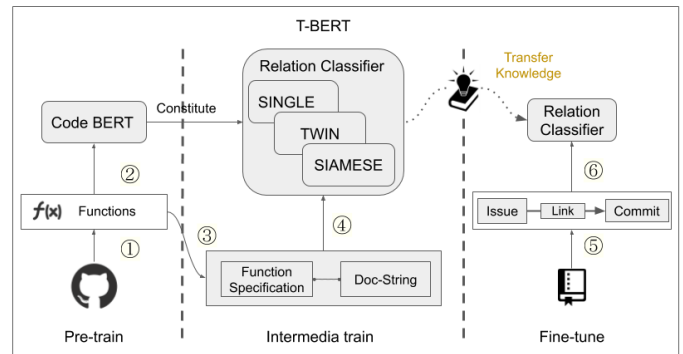


图 2：将 T-BERT 应用于 NLA-PLA 可追溯性的三步工作流程。1) 预训练数据是从 Github 项目收集的函数 2) BERT 被训练为具有这些函数的代码的语言模型，并与关系分类器组成作为 T-BERT 模型 3) 函数被拆分为规范和文档字符串并用作中间训练数据 4) T-BERT 模型使用代码搜索数据进行中间训练 5) OSS 数据集从 Github 存储库收集 6) T-BERT 模型使用转移的知识作为跟踪模型进行微调

不同的任务[23]，因为LM预训练和特定于任务的微调的架构几乎相同，只有模型的最后一层根据目标下游任务进行定制。该层通常被称为 *任务标题* 在基于 BERT 的模型中。

B. 用于软件可追溯性的 BERT

我们提出的解决方案代表了预训练、中间训练和微调的三重过程，如图 2 所示。在预训练阶段，专用语言模型在源代码上进行训练，然后用于构建 T-BERT 模型。在中间训练阶段，T-BERT 被训练来解决代码搜索问题。在此阶段，我们为 T-BERT 提供足够的标记训练示例，并期望它能够学习一般的 NL-PL 分类知识，这些知识最终可以转移到可追溯性挑战中。最后，在微调阶段，将中间训练的 T-BERT 模型应用于现实开源项目中的问题提交跟踪挑战。

C. T-BERT 架构

我们为软件可追溯性而研究的 T-BERT 架构的三个变体之前已应用于类似的基于文本的问题。这些变体是：

- 双胞胎：Twin BERT 架构如图 3a 所示。它利用两个 BERT 模型分别对 NL 和 PL 工件进行编码。然后，这两个工件被转换为两个独立的隐藏状态矩阵，其中标记由固定长度向量表示。我们对这些隐藏状态矩阵应用池化技术来制定表示伪影的特征向量。最后，我们将这两个特征向量连接起来以进行分类任务。

• 暹罗：连体 BERT 架构如图 3b 所示。它是单架构和双架构的混合体。它只使用一种 BERT 模型；然而，它不是像单个 BERT 那样为 NL-PL 对创建串联标记序列，而是将每个工件按顺序传递到 BERT 模型，并为两种工件类型（即 NL 和 PL）中的每一个创建单独的隐藏状态矩阵。然后，将生成的两个隐藏状态矩阵进行池化和连接，以生成联合特征向量，如 Twin BERT 架构中一样。然后将该联合特征向量发送到分类标头以完成预测任务。

Siamese 和 Twin T-BERT 架构都连接伪像特征向量以创建联合特征向量。尼尔斯等人。[29] 探讨了不同串联方法对 siamese BERT 架构的影响，并表明给定两个池化特征向量 u 和 v ，具有联合特征向量的连体 BERT ($u, v, |u - v|$) 在句子分类任务上取得了最佳性能。因此，我们应用这种类型的串联方法来融合 NL 和 PL 特征向量以创建联合特征向量。

• 单身的：单个 BERT 架构如图 3c 所示。NL 和 PL 文本使用特殊标记进行注释，然后连接成单个序列。例如，标记 [CLS]/[SEP] 用于注释句子的开头/结尾。带有标记的句子 $S_{s1}, S_2, S_3, \dots, S_{sN}$ 和一段带令牌的代码 $C_{C1}, C_2, C_3, \dots, C_{CN}$ 将被转换为输入格式 $[CLS]S_{s1}, S_2, S_3, \dots, S_{sN}[SEP]C_{C1}, C_2, C_3, \dots, C_{CN}[SEP]$ 。带注释和连接的序列被馈送到单个 BERT 以生成单个隐藏状态矩阵。随后的池化层会减少矩阵的维数以创建融合特征向量，该特征向量是孪生和孪生中联合特征向量的对应物。分类标头使用此特征向量来预测输入的 NL-PL 对是否相关。

四. 中号奥德尔时间下雨

在本节中，我们描述用于预训练、中间训练和微调阶段的训练策略。支持预训练和中间训练阶段的数据集由 Hamel 等人提供。[30] 来自他们对代码搜索问题的研究。它包括从众多 Github 项目中抓取的函数定义及其相关文档字符串，并且包括 Go、Java、JavaScript、PHP、Python 和 Ruby 编程语言。

微调阶段使用的数据集是我们团队从 OSS 检索的。我们通过 Github 的 API 提取问题和提交，并从提交消息中挖掘真实的跟踪链接。我们在图 1 中显示了数据格式，并在第 2 节中解释了数据收集过程的细节。弗吉尼亚州。在本研究中，由于大量活跃项目，我们选择 Python 作为训练和评估的目标语言；然而，我们的方法不依赖于语言。如果有足够的时间，相同的训练后过程可以应用于其他编程语言。

A. 三步训练

• 预训练代码语言模型：在预训练步骤中，我们利用 BERT 模型来学习 NL 和 PL 文档之间的单词分布，并将该 BERT 模型称为“代码 BERT”，以区别于仅处理 NL 文本的普通 BERT 模型。在普通 BERT 模型中，使用 masked LM (MLM) 任务将 BERT 预训练为语言模型。如前所述，在 MLM 任务中，选择并屏蔽 15% 的令牌，然后训练 BERT 根据其周围上下文恢复屏蔽的令牌。

鉴于预训练语言模型的成本非常昂贵，三个商业组织发布了各自的预训练代码 BERT 模型（Hugging Face [31]、CodistAI [32] 和 Microsoft [26]），它们都在 Code 上进行了训练-SearchNet 数据集。其中，我们利用微软的模型（称为 MS-CodeBert）作为我们的源代码语言模型，直接用于图 3 所示的 T-BERT 关系分类模型，因为它已被证明可以为各种下游软件工程提供改进的语言理解任务。MS-CodeBert 中的这些改进可归因于其“替换令牌检测”训练任务，该任务已被证明是训练 LM 的更有效方法 [33]。该训练任务用随机 token 替换语料库中的一小部分 token，然后需要 BERT 模型识别语料库中哪些 token 已被替换。

• 中级训练：对于中间训练，我们训练 T-BERT 模型来执行代码搜索问题，因为该问题本质上类似于 NLA-PLA 可追溯性挑战。在这两种情况下，我们都使用 T-BERT 根据代码功能的 NL 描述来检索相关源代码。CodeSearchNet 数据集¹为代码搜索问题提供了基准，因为数据集中的每个函数都与文档字符串配对。对于 Python，该数据集包括 824,342 个用于训练的函数、46,213 个用于开发的函数和 22,176 个用于测试的函数。该数据集非常适合中级训练，因为 1) 它的大小很大，因此 T-BERT 模型有足够的标记数据来学习识别 NL-PL 相关性的一般规则，2) 文档字符串和函数之间的关系是明确的，这意味着事实真相中的噪音最小，并且 3) 函数定义仅使用 Python 语法的一部分，这使得该任务比 NLA-PLA 可追溯性更容易处理。

我们将中间训练制定为二元分类任务，其中要求 T-BERT 识别给定的文档字符串是否正确描述了其配对函数。中间训练步骤中使用的损失函数是交叉熵损失，Adam Optimizer [34] 用于更新参数并优化损失函数。

代码搜索问题导致正负文档字符串到代码对的分布不平衡，因此我们创建了一个具有相同数量正负样本的平衡训练数据集。郭等人。[10] 采用了一种

¹CodeSearchNet 数据集 <https://github.com/github/CodeSearchNet>

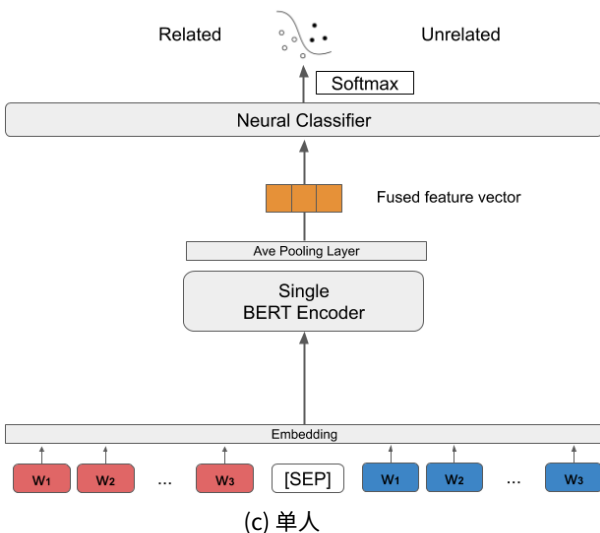
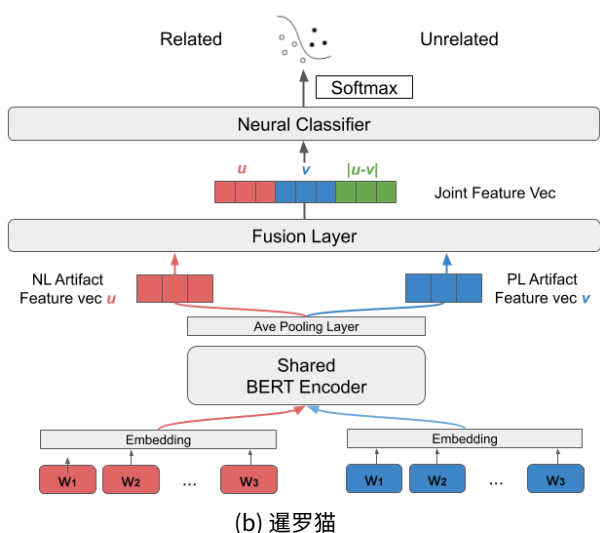
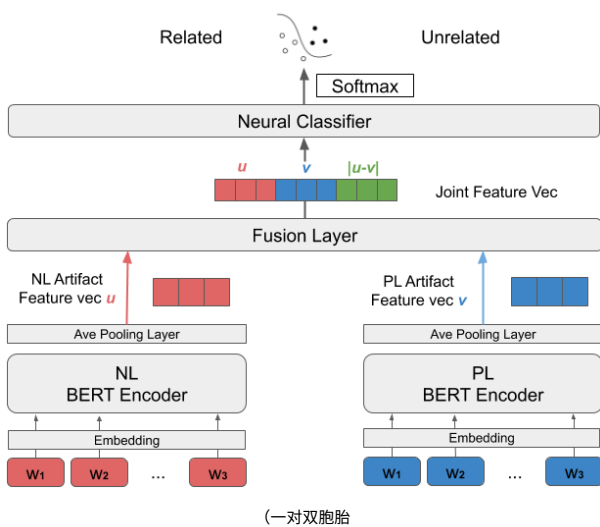


图 3：我们在实验中提出和评估的三个 T-BERT 模型的架构。

纳米欠采样策略，以避免训练数据大小膨胀，同时不断将模型暴露于以前未见过的负样本。我们采用类似的技术来构建训练样本。在每个时期，通过包含 CodeSearchNet 数据集中的所有函数和文档字符串对以及随机选择的相等数量的非相关对来构建平衡的训练集。我们在每个 epoch 开始时更新训练集，以便 T-BERT 模型可以从以前未见过的负例中学习。我们将此训练策略称为动态随机负采样（DRNS），并将其与第 2 节中描述的其他训练策略进行比较。IV-B。

• 微调：在微调中，我们使用了与上一步中讨论的类似的训练技术，但解决了使用真实世界 OSS 数据集跟踪代码提交问题的可追溯性挑战。尽管输入数据的格式与中间训练格式不同，但 T-BERT 对这两个任务使用相同的架构。如图 1 所示，问题由简短的问题摘要和长的问题描述组成，而提交由提交消息和代码更改集组成。对于每种类型的工件，我们连接文本以制定 T-BERT 模型的输入序列（即问题摘要 + 问题描述，然后提交消息 + 代码更改集。）与中间训练步骤相反，使用的数据集这一步是有限的、模糊的。据 Rath 等人报道，从 OSS 项目中挖掘的链接集不太可能完整且完全准确，因为工程师可能忘记标记问题，或者可能针对多个标签提交多个更改。[16]。此外，特定于项目的微调阶段的链接数量明显小于训练后使用的函数到文档字符串对的数量。如表所示。I，与 824,342 对函数和文档字符串相比，我们有大约 600 个用于微调的真实链接。此外，提交中的代码更改集比简短的函数定义具有更灵活和复杂的格式。

B. 负采样

郭等人观察到，在所实现的跟踪链接性能方面存在玻璃天花板，其中神经跟踪模型的准确性随着开始时训练周期的增加而增加，但随后达到峰值并随着进一步的训练而开始下降 [10]。我们对这种现象的假设如下。在训练的早期阶段，跟踪模型可以有效地学习区分正例和负例的规则，并且神经跟踪模型能够轻松排除许多不相关的示例（即源工件和目标工件几乎没有共同词汇的情况）。由于这些类型的负例构成了负例的大多数，随机负采样策略遇到的挑战性示例很少，因此开始基于朴素规则进行过度拟合，因为这些朴素规则适用于大多数简单情况。这种过度拟合会导致准确性在几个训练周期后下降。

我们的方法添加了高质量的负样本，通过我们提出的在线负样本来缓解这个问题

(ONS) 作为动态随机负采样 (DRNS) 的替代方案。ONS 的基本思想是，跟踪模型将在批次级别动态生成负例，而不是在每个 epoch 开始时创建训练数据集。为了便于说明，假设我们想要创建一批大小为 8 的包含 4 个正向链接的批次。如果我们 4 个 NL 工件和 4 个 PL 工件（即总共 16 个候选链接），我们包括 4 个正链接，然后从 12 个候选链接中选择 4 个负链接，以创建平衡批次。通过评估这些负面链接并根据预测分数对它们进行排名，我们可以识别更可能被错误分类的虚假链接。然后，通过将得分最高的负例合并到训练集中，我们提高负例的质量并避免早期过度拟合。这种方法的灵感来自于人脸识别领域的应用，其中人脸识别模型需要区分具有相似外观的人[35]。这种反例挖掘策略通常与对比学习[37]框架中的 Triplet Loss[36]相结合。在这里，我们采用它进行分类，并将其与广泛使用的交叉熵损失结合起来。

V.E实验乙估值

A. 数据收集

在本研究中，我们训练 T-BERT 并针对两种类型的数据集对其进行评估。第一个数据集是公开可用的 CodeSearchNet [30]。它包括六种不同编程语言的函数及其关联的文档字符串。如前所述，我们重点关注 python 函数。

其他数据集²我们利用的资源是从 Github 上的三个 OSS 项目中挖掘出来的，包括 Pgcli、Flask 和 Keras，如表中所述。I. 我们选择这些项目是因为它们是流行的、积极维护的 Python 项目，开发人员积极使用问题 ID 标记提交。我们检索问题及其讨论作为源工件，并提交作为目标工件。对于每个问题，我们都包含简短的问题摘要和较长的问题描述。如果在 markdown 中突出显示为代码块，我们会自动从问题讨论中删除堆栈跟踪，因为我们希望训练我们的方法来执行更艰巨的工作，即在问题和代码之间生成链接，而无需堆栈跟踪提供的更明确的信息。对于每次提交，我们都包含了提交消息和更改集。但是，我们从目标工件集中删除了非常小的提交（少于 5 个 LOC）。最后，由于 Github API 速率限制，我们每个项目最多检索 5000 个问题。

检索提交和问题后，我们通过使用嵌入到提交者添加的提交消息中的问题标签，从提交消息中挖掘“黄金链接集”。此外，我们还利用了拉取请求，因为接受的拉取请求会自动在 OSS 项目中创建问题和提交，并通过嵌入到提交消息中的问题 ID 将它们连接起来。使用正则表达式挖掘标签，以构建连接问题和提交的链接集。一风险

从提交消息中挖掘链接的一个问题是链接集可能不完整。刘等人。通过修剪数据集并仅保留链接集中出现的工件部分解决了这个问题[38]。我们采用这个过程来构建我们的数据集并在表 I 中报告结果

表 I: 可追溯性实验中利用的软件项目的规模。我们应用了第 2 节中描述的清洁程序。VA，清理工件和链接。

	提交	问题	链接	类型
PGCLI	原来的 2191 第531条	1197 第522条	第645条	数据库 命令行
烧瓶	原来的 4011 第752条	3711 第739条	第1159条	网络框架
喀拉斯	原来的 5348 第551条	4810 550	9375 第551条	神经网络 图书馆

B. 实验设置

我们在配备双十二核 2.2GHz Intel Xeon 处理器和 128GB RAM 的 Supermicro SYS-7048GR-TR SuperServer 上进行了实验。我们使用 1 个具有 10 GB 内存的 NVIDIA GeForce GTX 1080 Ti GPU 来训练和评估我们的模型。T-BERT 模型是使用 Py-Torch V.1.6.0 和 HuggingFace Transformer 库 V.2.8.0 实现的。我们在中间训练和微调中训练了 8 和 400 epoch 的模型。对于每个任务，我们使用批量大小为 8，批量累积步长为 8。我们将初始学习率设置为 1E-05，并应用线性调度器来控制运行时的学习率。关于模型选择，我们将数据集分为训练集（train）、开发集（dev）和测试集。我们使用训练数据集训练模型，并在开发数据集上测试其性能。然后，我们根据开发数据集选择性能最佳的模型并创建输出模型。我们最终评估并比较了测试数据集上输出模型的性能。在中间训练阶段，数据集已经被数据提供者分割。在微调阶段，我们将数据集分成十份，其中八份用于训练，一份用于开发，一份用于测试。

C. 评估指标

我们实验的指标包括 F 分数、平均精度 (MAP@3)、平均倒数排名 (MRR) 和精度@K。

• F 分数：F 分数是根据精确率和召回率计算得出的复合指标，经常用于评估可追溯性结果。F-1 分数对精确率和召回率赋予相同的权重，而 F-2 分数则更倾向于召回率而不是精确率。尽管精确率和召回率都很重要，但 F-2 通常更适合评估跟踪结果，因为召回率被认为比精确率更重要。我们通过枚举阈值来报告实验中的最佳 F 分数。

$$F_{\beta} = \frac{(1 + \beta_2) \cdot \text{精确} \cdot \text{记起}}{\beta_2 \cdot \text{精确} + \text{记起}} \quad (1)$$

²OSS数据集 <https://zenodo.org/record/4511291#.YB3tjyj0mbg>

• 平均精度：MAP 评估相关工件相对于检索工件的排名。每个源工件被视为用于检索工件的查询Q。对检索到的目标工件进行排名后，根据排名中所有相关目标工件的位置获得平均精度（AveP 或 AP）分数。然后计算 AveP 分数的平均值以返回 MAP。在我们的研究中，我们应用了一种更严格的指标，称为 MAP@3，其中只有排名前 3 位的工件才会对平均P 分数。该指标的公式如下所示，k 表示查询中相关目标工件的总数，秩我参考目标神器的排名：

$$\text{平均}P@3 = \frac{\sum_k \text{秩我} X}{k}, X = \begin{cases} \text{秩@我} & \text{如果秩我} \leq 3 \\ 0, & \text{否则} \end{cases}$$

$$\text{地图@3} = \frac{\sum_{\text{问}} \text{平均}P_j}{j} @3$$

• 平均倒数排名：MRR 是结果排名的另一种衡量标准。在每个查询中，排名为 N 的第一个相关目标工件将提供 1/N 的倒数排名。MRR 通过对所有查询 Q 的倒数排名进行平均来累积。这侧重于查询的第一个有效结果，而忽略总体排名。这是用于 CodeSearchNet 基准测试的标准指标。虽然 MAP 对于跟踪检索任务更为典型，但我们包含此指标是为了将我们的中间训练模型与其他代码搜索问题研究中的方法进行比较。

$$MRR = \frac{1 \sum_{\text{问}} \frac{1}{\text{等级我}}}{|\text{我}=1|}$$

• 精度@K：Precision@K 评估有多少相关工件被检索并排列在前 K 中。该指标的公式如公式 1 所示。5. 我们提供 K 值为 1 到 3 的结果。具有高 Precision@K 的跟踪模型意味着用户更有可能在前 K 个结果中找到至少一个相关的目标工件。

$$\text{精确@K} = \frac{\sum_{\text{问}} \frac{\text{我}/\text{相对值我@K}}{|\text{相对}|}}{|\text{相对}|}$$

我们可以看到，MRR和Precision@K忽略了召回率，而是专注于评估搜索结果是否能为用户找到感兴趣的结果。它们非常适合代码搜索问题，但不适用于召回特别重要的可追溯性。因此，我们仅应用 F 分数和 MAP@3 来评估可追溯性结果。由于我们的大多数查询的正确链接少于三个，因此完美的 MAP@3 分数代表接近 100% 的召回率。

六. 右结果ANDD讨论

我们报告了代码搜索问题和 NLA-PLAs 可追溯性问题的 T-BERT 模型的性能结果，并解决了第 2 节中定义的 RQ。我。

A. 评估代码搜索问题

我们的第一个评估探讨了当数据集有足够的标记示例时 T-BERT 模型的表现如何。我们为第 2 节中介绍的三种架构训练了 T-BERT 模型。III-C，使用 CodeSearchNet 数据集的训练部分。对于使用 ONS 技术训练的 T-BERT 模型，我们在模型名称后添加一个星号，以将其与使用 DRNS 训练的 T-BERT 模型区分开来。例如，SINGLE*指的是具有单一BERT 架构并通过在线负采样训练的模型。这六种模型的性能如表所示。二. 此外，我们还将 T-BERT 模型与 VSM、LDA、LSI 等三种经典跟踪技术进行了比较

(2) 该数据集的深度学习跟踪模型 TraceNN [10]。

Hamel 等其他研究人员。[30]和Feng [26]利用相同的数据集进行代码搜索研究，因此我们选择在他们研究中获得最佳MRR分数的方法作为与T-BERT的比较，并在同样的方式。对于每个文档字符串，我们将相关函数与 999 个不相关函数组合起来，并要求检索模型在 1000 个候选函数中找到正确的函数。然而，SINGLE 模型无法有效地处理整个数据集，因此在本例中，我们仅评估了 22,176 个查询中的 100 个。MRR评分比较如表所示。二. 观察每个人的学习过程

表 II：T-BERT 模型在 CodeSearch-Net Challenge 数据集上的评估

	F1	F2	地图	MRR	Pr@1	Pr@2	Pr@3
双胞胎	0.497	0.563	0.735	0.646	0.787	0.842	0.825
暹罗	0.604	0.668	0.814	0.866	0.915	0.839	0.730
单身的	0.482	0.572	0.825	0.930			0.900
双胞胎*	0.559	0.626	0.794	0.809	0.712	0.846	0.890
暹罗*	0.594	0.655	0.817	0.738	0.866	0.910	0.851
单身的*	0.612	0.678	0.837	0.910	0.930		
向量空间模型	0.219	0.255	0.314	0.351	0.251	0.341	0.397
LDA	0.005	0.010	0.012	0.008	0.013	0.017	0.025
大型集成电路	0.003	0.007	0.014	0.009	0.015	0.020	
神经网络LSTM	0.179	0.245	0.351	0.400	0.269	0.386	0.457
TNN-BiGRU	0.221	0.29	0.392	0.304	0.432	0.504	
合资biRNN				* 0.321			
合资-SelfAtt				* 0.692			
MSC				* 0.860			

JV=联合向量[30]；MSC=MS-CodeBERT [26]；TNN=TraceNN[10]；

* 之前针对相同 CodeSearch-Net 挑战数据集报告的结果。

表 III：T-BERT 模型在代码搜索问题上的训练和测试时间。记录具有 100 个查询的测试集的测试时间。

	战略	双胞胎	暹罗	单身的
火车(小时)	DRNS	156小时	138小时	164小时
测试 (秒)	DRNS	3254s	3264s	183357s
火车(小时)	英国国家统计局	146小时	142小时	283小时
测试 (秒)	英国国家统计局	3211s	3265s	193667s

模型中，我们在图 4 中可视化了第一个学习曲线

35,000 步优化。我们在训练期间以 1000 个步骤为间隔评估每个模型的性能，将中间模型应用于由 200 个开发示例组成的小型测试集。

B. 评估 NLA-PLA 可追溯性

然后，我们评估了 NLA-PLA 可追溯性问题上的 T-BERT 模型。如前所述，我们使用 8 倍跟踪链接进行训练，1 倍用于开发，1 倍用于测试。为了探索 RQ3，我们进行了一项对照实验，训练了两组 T-BERT 模型。在第一组中，我们继续训练在之前的实验中经过中间训练的 T-BERT 模型。在第二组中，我们在不应用从中间训练中学到的迁移知识的情况下训练 T-BERT 模型。当我们进行模型训练时，我们对两组应用了相同的训练数据集和 ONS 技术。为了保持缩写的一致性，我们将第一组中的模型命名为例如 SINGLE*+T；第二组模型为 SINGLE*。本实验的结果如表所示。IV，而显示可追溯性任务微调的 T-BERT 模型的学习曲线如图 5 所示。由于空间限制，我们仅显示 Pgcli 数据的学习曲线。

C. RQ2: ONS 如何缓解玻璃天花板问题？

在本节中，我们将讨论 ONS 对 T-BERT 模型的有效性如何缓解玻璃天花板问题。这个问题帮助我们确定训练 T-BERT 模型的最佳方法。为了回答这个问题，我们将 ONS 和 DRNS 应用于相同的 T-BERT 架构来创建测试组和控制组。图 4 显示了 CodeSearchNet 数据集上 T-BERT 模型训练期间的学习曲线。橙色线代表使用 ONS 训练的模型，而蓝色线代表使用 DRNS 训练的模型。我们发现，对于 TWIN 和 SINGLE 模型，橙色线始终位于蓝色线上方，这意味着 ONS 可以加速 T-BERT 模型的学习，同时也让它收敛于更高的值。而对于 SIAMESE，我们发现橙色线在早期步骤中高于蓝色线，但很快就收敛到相似的水平。这一结果表明 ONS 通过引入更难的负例有利于 T-BERT 模型训练。评价结果如表前六行所示。II 也支持这一发现，因为从所有指标的角度来看，TWIN* 和 SINGLE* 模型（第 4,6 行）比 T-BERT 模型（第 1,3 行）取得了更好的结果。SIAMESE*（第 5 行）和 SIAMESE（第 2 行）的结果非常接近，除了 F2 分数 (1.3%) 之外，所有指标的差异都在 0.5% 以内。

我们在表 III 中报告了 DRNS 和 ONS 的训练时间。ONS 在构建每个批次时引入了初始开销，但随后需要评估和排序的候选者较少。相比之下，DRNS 没有前期建设成本，但必须从大列表中采样数据，造成性能瓶颈。使用 ONS 仅显著增加了训练时间，但评估速度特别慢的 SINGLE 模型除外。

我们得出的结论是，ONS 比 DRNS 提供更好的准确性，并且只会增加评估过程缓慢的模型（例如 SINGLE）的训练时间。

D. RQ1: 哪种 T-BERT 架构更好？

此 RQ 重点是比较 T-BERT 在用于踪迹检索任务时的准确性和效率。性能比较是在 T-BERT* 模型上进行的，正如我们在 RQ2 中所示，T-BERT* 模型比 T-BERT 模型返回了更好的精度。为了回答这个问题，我们进一步将问题分为以下三个子 RQ。

• RQ1.1: T-BERT 模型能够解决 CodeSearchNet 和 NLA-PLA 溯源问题吗？

桌子。II 显示 SINGLE*、TWIN* 和 SIAMESE*（第 4-6 行）可以获得 0.6 左右的 F 分数和 0.8 左右的 MAP 分数。三个模型的 Precision@3 得分约为 0.9，这意味着 T-BERT* 模型可以在 10 个用户查询中的大约 9 个中返回相关函数。在 75% 到 80% 的情况下，正确答案定义排在第一位。该结果表明所有三种模型对于 CodeSearchNet 挑战均有效。在这三个模型中，SINGLE* 在所有指标方面均实现了最佳性能。然而，这三个模型之间的差距很小，并且所有三个模型都明显优于 VSM、LSI 和 LDA 三个 IR 模型创建的基线。

在表中。IV，前三行显示，应用于可追溯性挑战并在没有转移知识的情况下进行训练的 T-BERT* 模型的排名方式与代码搜索问题相同。然而，这三种模型之间的性能差距越来越大。这表明训练数据的大小对三种类型的架构有不同的影响。由于 TWIN* 模型包含两个内部 BERT 模型，因此该架构中的参数加倍，因此需要更多的训练示例来调整模型。尽管如此，所有 T-BERT* 模型都取得了比 IR 模型基线更好的结果。特别是在 Keras 数据集中，SIAMESE* 和 SINGLE*（第 2 行和第 3 行）的 F 分数高于 0.95，MAP 为 0.99，表明 T-BERT 在某些场景下可以提供完美的跟踪结果。

• RQ1.2: 哪种 T-BERT 模型最有效解决数据稀疏性和性能两个问题？

在选择用于生产的模型时，我们需要同时考虑准确性和效率。正如 RQ1.1 中所讨论的，SINGLE* 模型实现了最佳性能；然而，处理大规模数据集的速度非常慢。如表所示。III，TWIN 和 SIAMESE 需要大约 3000 秒来评估 100 个查询，而 SINGLE 需要大约 20000 秒。我们估计，使用我们当前的实验设置，SINGLE 需要大约 6000 小时来评估整个 CodeSearchNet 测试。但对于 TWIN 和 SIAMESE，我们只花了大约 20 个小时就可以在实践上评估整个测试集。在可追溯性挑战中，测试集相对较小。以 Pgcli 为例，它包含 2704 个候选链接，由 52 个源工件和 52 个目标工件组成。TWIN 和 SIAMESE 都需要大约 160 秒才能完成任务，而

表 IV: NLA-PLAs 可追溯性模型的评估

		F1	PGCLI F2	地图	F1	烧瓶 F2	地图	F1	喀拉斯 F2	地图
双胞胎*		0.450	0.491	0.574	0.524	0.577	0.683	0.450	0.491	0.574
暹罗*		0.621	0.654	0.728	0.681	0.731	0.801	0.962	0.962	0.990
单身的*		0.707	0.745	0.785	0.841	0.873	0.952	0.931	0.925	0.971
双胞胎*	时间	0.686	0.709	0.766	0.750	0.781	0.869	0.953	0.970	0.978
暹罗*	时间	0.729	0.748	0.779	0.820	0.830	0.920	0.971	0.977	0.990
单身的*	时间	0.730	0.789	0.859	0.884	0.862	0.92	0.972	0.989	0.990
向量空间模型		0.376	0.424	0.506	0.509	0.474	0.540	0.532	0.512	0.703
LDA		0.121	0.226	0.208	0.182	0.241	0.227	0.290	0.367	0.333
大型集成电路		0.085	0.145	0.147	0.127	0.164	0.142	0.072	0.126	0.109
神经网络LSTM		0.138	0.179	0.128	0.106	0.126	0.080	0.053	0.087	0.034
TNN-BiGRU		0.062	0.116	0.006	0.066	0.100	0.044	0.063	0.119	0.073

T=迁移学习, TNN=TraceNN[10];

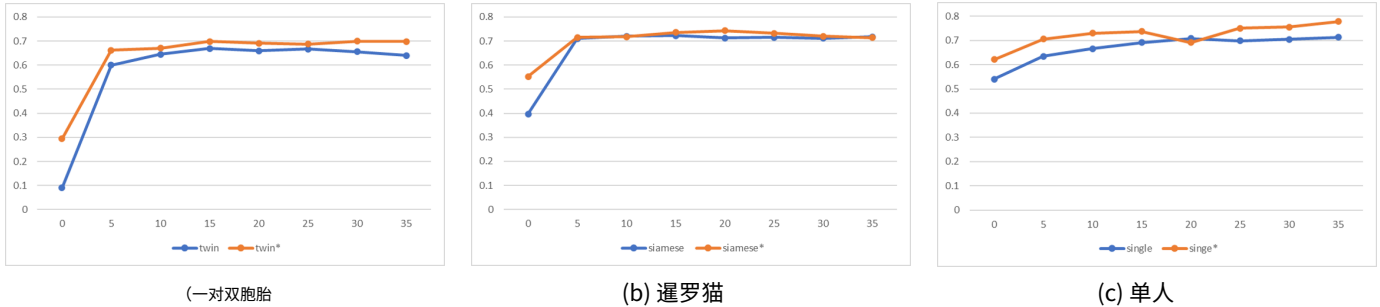


图 4: T-BERT 和 T-BERT* 模型在代码搜索挑战中的学习曲线。该图显示了前 35K (X 轴) Adam 优化步骤的 MAP 分数 (Y 轴)。

SINGLE 模型大约需要一小时。SIAMESE 和 TWIN 架构通过解耦特征向量创建步骤来加速该过程。在 SINGLE 架构中, NL 和 PL 文档对被馈送到 BERT 以创建联合特征向量。此步骤极其昂贵, 并且造成了 SINGLE 模型的主要性能瓶颈。假设我们有 N 个源工件和 N 个目标工件。SINGLE 的时间复杂度为 $O(N^2 * K)$ 用于为所有候选链接创建特征向量, 其中 K 指的是 BERT 模型将输入标记序列转换为特征向量所消耗的时间。TWIN 和连体只需要 $O(N * K)$ 将工件转换为特征向量, 然后 $O(N^2)$ 是时候将特征向量连接在一起了。TWIN 和 SIAMESE 的时间复杂度比 SINGLE 模型低一个数量级, 因此对于具有大量工件的项目更具可扩展性。我们认为, 考虑到准确性和效率, SIAMESE 是解决 NLA-PLA 可追溯性的最合适模型, 因为它可以在应对可追溯性挑战时达到接近 SINGLE 架构的精度, 同时保持 TWIN 架构的低时间复杂度。然而, 在准确性是首要考虑的情况下, 例如安全的可追溯性

对于关键项目, 用户应采用高性能硬件支持的 SINGLE 模型。

• RQ1.3: T-BERT 模型与其他方法相比如何?

对于 CodeSearchNet 挑战, 我们将 T-BERT 模型的性能与联合向量嵌入 (JVE) 和 MS-CodeBERT 进行了比较。JVE 的架构与 TWIN 类似, 利用两个编码器为分类网络创建特征向量。之前的研究报告称, JVE 在同一数据集上实现的最高 MRR 为 60%, 低于 T-BERT 模型。Microsoft 提供的 MS-CodeBERT 在我们的实验中使用了与 SINGLE 相同的架构。然而, MS-CodeBERT 在具有 16 个 Tesla GPU 的集群上以 256 的批量大小进行训练, 并且在训练过程中没有应用特殊技术。由于内存限制, 我们的机器只允许一小批, 但 SINGLE* 模型的 MRR 结果仅比 MS-CodeBERT 低 0.9%, 这表明我们的训练技术部分缓解了功能较弱的硬件带来的限制。

TNN[10]是Guo等人提出的基于RNN的跟踪模型。并设计用于生成 NLA-NLA 链接。我们根据作者的说明重建了模型, 并将其应用于我们的 NLA-PLA 问题以进行比较。TNN 利用 Word2Vec 嵌入将标记转换为向量。它使用两个备用 RNN 网络、LSTM 或双向 GRU (BiGRU) 来生成 NLA 和 PLA 的语义表示, 并将这些语义隐藏状态馈送到集成层以生成新的隐藏状态表示

表 V: NLA-PLA Pgcli 数据集上的模型性能。

	双胞胎*	暹罗*	单身的*
火车(小时)	12小时	12小时	13小时
测试 (秒)	170年代	163秒	5395s

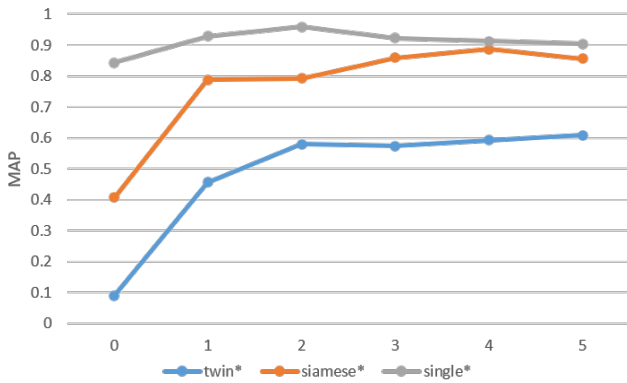


图 5：用于 NLA-PLA 跟踪挑战的 Pgcli 数据集上的 T-BERT* 模型的学习曲线。该图显示了 5k Adam 优化步骤（X 轴）上的 MAP 分数（Y 轴）

生成链接的 NLA 和 PLA 之间的相关性。我们的嵌入层是通过使用表 I 中报告的三个 OSS 的工件对 Skip-gram Word2Vec 模型进行无监督训练来构建的。在本研究中，我们评估了 RNN 层的 LSTM 和 BiGRU。TNN 结果显示在表底部。IV，并表明它在所有三个 OSS 项目上的表现均低于所有 BERT 模型和 VSM。我们在图 6 中提供了一个示例，显示了提交-问题对的 T-BERT 和 VSM 结果，由提交者标记为相关。

Guo 等人报告了其 NLA-NLA 数据集的 VSM 模型的改进，但我们无法针对我们的 NLA-PLA 问题复制这些改进。对 TNN 学习曲线的检查表明，TNN 有效降低了训练数据集中所有三个训练数据集的损失并提高了链接预测精度，但在验证数据集中较早收敛，然后精度下降 - 表明存在过拟合问题。对于这些结果有几种可能的解释。首先，Guo 等人使用的数据集包含 1,387 个正链接，而我们的链接为 530-739 个，这可能不足以进行 RNN 训练。其次，编程语言具有开放词汇表，可以在其中创建新术语作为变量和函数名称，因此与 NLA-NLA 链接相比，TNN 可能需要更大的训练集来生成 NLA-PLA 链接。我们的假设得到了观察的支持，即当应用于提供大量训练示例的 CodeSearchNet 时，TNN 不会过度拟合。T-BERT 模型利用来自预训练语言模型和相邻问题的转移知识来减少训练数据集大小的要求，因此能够应对经典深度学习跟踪模型无法轻松解决的跟踪挑战。这一特性使得 T-BERT 在工业应用中更加实用。

E. RQ3: T-BERT 在多大程度上可以利用从代码搜索到软件可追溯性的知识转移

表 IV 使用和不使用来自预训练后模型的转移知识进行训练的 T-BERT 模型。这

结果表明，代码搜索任务上的中间训练 T-BERT 模型可以显著提高其在可追溯性问题上的性能。以 Pgcli 上的 SIAMESE 为例，F2 分数从 0.654 增加到 0.748，而 MAP 分数从 0.728 增加到 0.779。使用不同 T-BERT 模型的其他数据集也观察到类似的结果。这表明从文本到结构代码（函数定义）学到的知识可以有效地转移到以下情况：1) 代码格式更加模糊，2) 训练数据的标签有限。中级培训改进

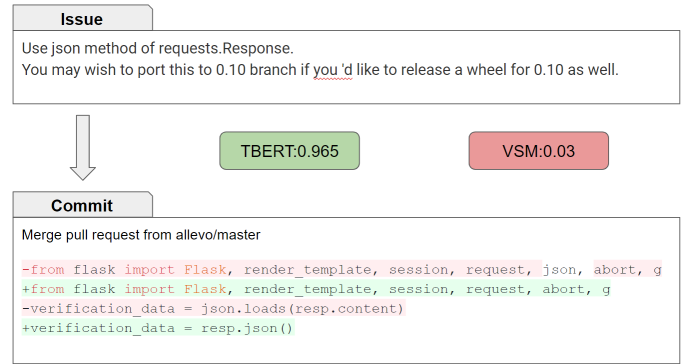


图 6：在此示例中，链接由开发人员标记，由 T-BERT 模型检索（由于语义相似性和上下文，得分高达 0.965），并被 VSM 错过，因为关键词“请求”和“json”是常用术语。

在三种架构类型中观察到不同程度的差异。如图 5 所示，蓝线 (SINGLE) 在很早的阶段就收敛了，这表明 SINGLE 在较小的特定任务数据集上只需要相对较少的 epoch 来本地化其转移的知识。SIAMESE 收敛速度较慢，而 TWIN 收敛速度最慢，这表明每种架构具有不同的知识传输能力。

七. 右兴高采烈瓦奥克

我们的研究使用三种不同的架构构建了 T-BERT 模型，所有这些架构之前都已用于解决其他领域的相关问题。卢等人。利用名为 TwinBERT 的类似 TWIN 的模型作为搜索引擎，在有机搜索结果旁边提供广告 [39]。他们使用强化训练技术，发现 TwinBERT 可以以高精度和低延迟返回结果。赖默斯等人。提出了一种 SIAMESE 架构来解决语义文本相似性等问题 [29]。他们训练模型来确定两个句子是否通过矛盾或蕴涵相关，并利用 SNLI [40] 和 Multi-Genre NLI [41] 数据集进行训练和评估。他们的结果表明 SIAMESE BERT 可以获得 0.76 左右的高斯皮尔曼等级相关性分数。他们还发现，使用平均池化比最大池化和第一个令牌 ([CLS] 令牌) 池化更有效；并将源隐藏状态和目标隐藏状态连接为（紫外线， $\|u - v\|$ ）取得了最好的成绩。我们采用了

我们为这项研究构建 T-BERT 模型时得到了这些发现。然而，目前还没有对 TWIN、SIAMESE 和 SINGLE 架构进行比较的研究。

为了解决 NLA-PLA 挑战，我们采用代码搜索问题作为中间解决方案。一些研究已经使用循环神经网络 (RNN) 解决了代码搜索问题。我们已经与 Feng 等人的工作进行了比较。[26] 和惠安等人。[30]见表。二、然而，在另一项研究中，Gu 等人。[42]将方法规范转换为API调用序列，然后用RNN处理该序列。他们报告在包含 100 个查询的测试集中实现了 0.6 MRR。然而，我们不能直接使这种方法适应可追溯性挑战，因为与 API 调用不同，代码更改集中的语句不是结构化的。解决 NLA-PLA 问题的一个相关领域是源代码嵌入。通过将源代码和文档转换为分布式表示，可以通过余弦距离和欧几里得距离等距离度量有效地计算这两类工件之间的相关性。Code2Vec [43]属于此类方法。T-BERT 模型可以通过在分类标头中集成 Cosine Embedding Loss 来适应这种类型的训练。我们将这种探索留给未来的工作。

八. 时间威胁到V活力

这项研究的有效性存在一些威胁。首先，我们目前的实验仅在Python项目上进行，当应用于其他编程语言时，结果可能会有所不同。此外，由于时间限制，我们仅在三个 OSS 项目上微调并评估了 T-BERT 模型性能，这可能不足以得出概括性结论。其次，我们通过挖掘 ID 被项目维护者明确标记为相关的问题和提交，从 OSS 项目构建实验数据集。尽管这是利用 OSS 项目进行可追溯性的传统方式，但可能会错过真正的链接。例如，错误报告可能隐藏了对几个其他问题（例如功能请求或其他错误报告）的依赖关系，即使解决父错误报告的提交未标记为“相关”。我们通过采用 Liu 等人建议的数据处理来减轻这种现象的影响。[38]。另一个重要的威胁是，虽然针对代码搜索问题进行训练的 SINGLE 架构的性能并不优于 CodeBERT，但可以使用超参数优化来实现进一步的改进。我们的实验受到硬件可用性的限制，无法进行过多的超参数调整。然而，T-BERT 模型之间的性能比较应该仍然有效，因为所有实验都是使用相同的参数进行的。最后，由于处理时间限制，我们在 100 个查询上评估了 SINGLE 模型，同时使用其他模型的整个测试集（表 II）。尽管没有报告，但我们还在 100 个查询上评估了 TWIN 和 SIAMESE，并观察到它们获得的结果与从整个测试集获得的结果几乎相同，这表明 100 个查询对于 SINGLE 模型来说是合理的样本大小。

九. C结论和F优图瓦奥克

本研究探索了几种不同的 BERT 架构，用于生成自然语言工件和编程语言工件之间的跟踪链接。我们的实验结果表明，SINGLE 架构实现了最佳精度，但执行时间较长，而 SIAMSE 架构实现了类似的精度，但执行时间更快。其次，我们表明，与 DRNS 相比，ONS 训练（基于负采样）提高了性能和模型收敛速度，且不会产生显著的性能开销。第三，我们发现 T-BERT 能够有效将从代码搜索问题中学到的知识转移到 NLA-PLA 可追溯性上，这意味着经过中间训练的 T-BERT 模型可以有效地应用于训练样例有限的软件工程项目，从而缓解了深度神经跟踪模型的数据稀疏问题。关于训练时间，我们表明相同的中间训练 T-BERT 可以应用于三个不同领域的 OSS 项目。通过避免对每个单独项目进行中间培训的需要，我们的方法能够有效地适应新领域。总之，我们的结果表明，T-BERT 生成的跟踪链接的准确度远远高于现有的信息检索和 RNN 技术，让我们更接近实现实用且值得信赖的可追溯性的愿景。

为了支持复制和再现性，我们在本文中提供了我们使用的数据集的链接，并在 github 上提供了 T-BERT 的完整实现和执行指令³。

在未来的工作中，我们将在更多样化的项目领域和编程语言中评估我们的方法，并将探索其在更多样化类型的软件工件（例如需求、设计和测试用例）中的应用。

A致谢

这项工作得到了美国国家科学基金会拨款 SHF:1901059 的部分资助。

右参考文献

- [1] L.里尔森, 开发安全关键软件: 航空软件和 DO-178C 合规性实用指南。CRC出版社, 2013年。
- [2] A. Mahmoud、N. Niu 和 S. Xu, “用于追溯链接恢复的语义相关性方法”, 载于 2012 年第 20 届 IEEE 程序理解国际会议 (ICPC)。IEEE, 2012 年, 第 183-192 页。
- [3] G. Antoniol、G. Canfora、G. Casazza、A. De Lucia 和 E. Merlo, “恢复代码和文档之间的可追溯性链接” IEEE 软件工程汇刊, 卷。28、没有。10, 第 970-983 页, 2002 年。
- [4] J. Huffman Hayes、A. Dekhtyar 和 SK Sundaram, “推进需求跟踪的候选链接生成: 方法研究”, IEEE 软件工程汇刊, 卷。32、没有。1, 第 4-19 页, 2006 年。
- [5] A. De Lucia、F. Fasano、R. Oliveto 和 G. Tortora, “通过可追溯性恢复功能增强人工制品管理系统”, 载于第 20 届 IEEE 软件维护国际会议 (ICSM), 2004 年, 第 306-315 页。

- [6] P. Rempel, P. Mäder 和 T. Kuschke, “走向特征感知的细化痕迹检索”, 载于 *第七届新兴软件工程形式可追溯性国际研讨会, TEFSE 2013*, 2013 年 5 月 19 日, 美国加利福尼亚州旧金山, N. Niu 和 P. Mäder, 编辑。IEEE 计算机协会, 2013 年, 第 100–104 页。[在线的]。可用: <https://doi.org/10.1109/TEFSE.2013.6620163>
- [7] A. Dekhtyar, JH Hayes, SK Sundaram, EA Holbrook 和 O. Dekhtyar, “需求评估的技术集成”, 载于 *第 15 届 IEEE 国际需求工程会议, RE 2007*, 2007 年 10 月 15 日至 19 日, 印度新德里。IEEE 计算机协会, 2007 年, 第 141–150 页。[在线的]。可用: <https://doi.org/10.1109/RE.2007.17>
- [8] HU Asuncion, AU Asuncion 和 RN Taylor, “具有主题建模的软件可追溯性”, 载于 *第 32 届 ACM/IEEE 软件工程国际会议论文集 - 第 1 卷, ICSE 2010*, 南非开普敦, 2010 年 5 月 1-8 日, J. 克萊默, J. Bishop, PT Devanbu 和 S. Uchitel, 编辑。ACM, 2010 年, 第 95-104 页。[在线的]。可用: <http://doi.acm.org/10.1145/1806799.1806817>
- [9] H. Sultanov, J. Huffman Hayes 和 W.-K. Kong, “群体技术在需求跟踪中的应用”, *需求工程*, 卷。16、没有。3, 第 209-226 页, 2011 年。
- [10] J. Guo, J. Cheng 和 J. Cleland-Huang, “使用深度学习技术在语义上增强软件可追溯性”, 载于 *2017 年 IEEE/ACM 第 39 届国际软件工程会议 (ICSE)*。IEEE, 2017 年, 第 3-14 页。
- [11] G. Spanoudakis, A. Zisman, E. Pérez-Miñana 和 P. Krause, “基于规则的需求追踪关系生成”, *系统与软件杂志*, 卷。72、没有。2, 第 105-127 页, 2004 年。
- [12] J. Guo, J. Cleland-Huang 和 B. Berenbach, “特定领域可追溯性专家系统的基础”, 载于 *第 21 届 IEEE 国际需求工程会议 (RE)*, 2013 年, 第 42-51 页。
- [13] J. Cleland-Huang, P. Mäder, M. Mirakhorli 和 S. Amornborvornwong, “打破可追溯性的爆炸实践: 向项目利益相关者推送及时的跟踪建议”, 载于 *2012 第 20 届 IEEE 国际需求工程会议 (RE)*, 美国伊利诺伊州芝加哥, 2012 年 9 月 24-28 日, MPE Heimdahl 和 P. Sawyer, 编辑。IEEE 计算机协会, 2012 年, 第 231-240 页。[在线的]。可用: <https://doi.org/10.1109/RE.2012.6345809>
- [14] S. Lohar, S. Amornborvornwong, A. Zisman 和 J. Cleland-Huang, “通过数据驱动的配置和跟踪特征组合提高跟踪准确性”, 载于 *2013 年第九届软件工程基础联席会议论文集*。ACM, 2013 年, 第 378–388 页。
- [15] M. Gethers, R. Oliveto, D. Poshyvanyk 和 A. De Lucia, “关于集成正交信息检索方法以改进可追溯性链接恢复”, 载于 *第 27 届 IEEE 软件维护国际会议 (ICSM)*, 2011 年, 第 133–142 页。
- [16] M. Rath, J. Rendall, JL Mao, J. Cleland-Huang 和 P. Mäder, “野外可追溯性: 自动增强不完整的跟踪链接”, 载于 *第 40 届国际软件工程会议论文集*, 2018 年, 第 834–845 页。
- [17] Y. Liu, J. Lin, Q. Zeng, M. Jiang 和 J. Cleland-Huang, “走向语义引导的可追溯性”, 载于 *国际需求工程会议*, 卷。2020 年, 2020 年。
- [18] S. Lohar, S. Amornborvornwong, A. Zisman 和 J. Cleland-Huang, “通过数据驱动的配置和跟踪特征组合提高跟踪准确性”, 载于 *第九届软件工程基础联席会 (ESEC/FSE)*, 2013 年, 第 378–388 页。
- [19] M. Borg, C. Englund 和 B. Duran, “可追溯性和深度学习安全关键系统, 其痕迹以深度神经网络结束”, 在 *过程中。可追溯性的重大挑战: 未来十年*, 第 48-49 页, 2017 年。
- [20] Y. Zhu, TS Zaman, T. Yu 和 JH Hayes, “利用深度学习提高代码可追溯性要求的准确性”, *可追溯性的巨大挑战: 未来十年*, p. 2017 年 22 日。
- [21] S. Hochreiter 和 J. Schmidhuber, “长短期记忆”, *神经计算*, 卷。9、不。8, 第 1735–1780 页, 1997 年。
- [22] J. Chung, C. Gulcehre, K. Cho 和 Y. Bengio, “序列建模中门控循环神经网络的实证评估” *arXiv 预印本 arXiv:1412.3555*, 2014 年。
- [23] J. 德夫林, M.-W. Chang, K. Lee 和 K. Toutanova, “Bert: 用于语言理解的深度双向转换器的预训练” *arXiv 预印本 arXiv: 1810.04805*, 2018。
- [24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, AN Gomez, Ł. Kaiser 和 I. Polosukhin, “注意力就是你所需要的”, 载于 *神经信息处理系统的进展*, 2017 年, 第 5998–6008 页。
- [25] J. Liu, Y. Lin, Z. Liu 和 M. Sun, “Xqa: 跨语言开放域问答数据集”, 载于 *计算语言学协会第 57 届年会论文集*, 2019 年, 第 2358–2368 页。
- [26] 冯正、郭大、唐大、段宁、冯小、龚明、寿禄、秦本、刘涛、江东等人。 , “Codebert: 用于编程和自然语言的预训练模型”, *arXiv 预印本 arXiv:2002.08155*, 2020。
- [27] J. Cleland-Huang, OC Gotel, J. Huffman Hayes, P. Mäder 和 A. Zisman, “软件可追溯性: 趋势和未来方向”, 载于 *软件工程的未来论文集*。ACM, 2014 年, 第 55-69 页。
- [28] 维基百科, “语言模型——维基百科, 免费百科全书”, 2020, [在线; 2020 年 7 月 22 日访问]。[在线的]。可用: <https://en.wikipedia.org/wiki/语言模型双向>
- [29] N. Reimers 和 I. Gurevych, “Sentence-bert: 使用 3 层 bert 网络进行句子嵌入”, *arXiv 预印本 arXiv:1908.10084*, 2019。
- [30] H. 侯赛因, H.-H. Wu, T. Gazit, M. Allamanis 和 M. Brockschmidt, “CodeSearchNet 挑战: 评估语义代码搜索的状态” *arXiv:1909.09436 [cs, 统计]*, 2019 年 9 月, arXiv: 1909.09436。[在线的]。可用: <http://arxiv.org/abs/1909.09436>
- [31][在线]。 可用的: <https://huggingface.co/huggingface/codeBERTa-small-v1>
- [32][在线]。可用: <https://huggingface.co/codistai/codeBERT-small-v2>
- [33] K. 克拉克, M.-T. Luong, QV Le 和 CD Manning, “Electra: 将文本编码器预训练为判别器而不是生成器” *arXiv 预印本 arXiv:2003.10555*, 2020。
- [34] DP Kingma 和 J. Ba, “Adam: 一种随机优化方法”, *arXiv 预印本 arXiv:1412.6980*, 2014 年。
- [35] F. Schroff, D. Kalenichenko 和 J. Philbin, “Facenet: 用于人脸识别和聚类的统一嵌入”, 载于 *IEEE 计算机视觉和模式识别会议论文集*, 2015 年, 第 815–823 页。
- [36] 维基百科贡献者, “Triplet loss”, 202-, [在线; 2020 年 8 月 22 日访问]。[在线的]。可用: https://en.wikipedia.org/wiki/Triplet_loss
- [37] T. Chen, S. Kornblith, M. Norouzi 和 G. Hinton, “视觉表示对比学习的简单框架”, *arXiv 预印本 arXiv:2002.05709*, 2020。
- [38] Y. Liu, J. Lin 和 J. Cleland-Huang, “多语言软件项目的可追溯性支持”, *arXiv 预印本 arXiv:2006.16940*, 2020。
- [39] W. Lu, J. Jiao 和 R. Zhang, “Twinbert: 将知识提炼为双结构 bert 模型以实现高效检索”, *arXiv 预印本 arXiv:2002.06275*, 2020。
- [40] SR Bowman, G. Angeli, C. Potts 和 CD Manning, “用于学习自然语言推理的大型注释语料库”, 载于 *2015 年自然语言处理经验方法会议 (EMNLP) 会议论文集*。计算语言学协会, 2015。
- [41] A. Williams, N. Nangia 和 S. Bowman, “通过推理理解句子的广泛覆盖挑战语料库”, 载于 *计算语言学协会北美分会 2018 年会议论文集: 人类语言技术, 第 1 卷 (长论文)*。计算语言学协会, 2018 年, 第 1112–1122 页。[在线的]。可用: <http://aclweb.org/anthology/N18-1101>
- [42] X. Gu, H. Zhang 和 S. Kim, “深度代码搜索”, 载于 *第 40 届国际软件工程会议论文集, ICSE 2018*, 瑞典哥德堡, 2018 年 5 月 27 日至 6 月 3 日, M. Chaudron, I. Crnkovic, M. Chechik 和 M. Harman, 编辑。ACM, 2018 年, 第 933–944 页。[在线的]。可用: <https://doi.org/10.1145/3180155.3180167>
- [43] U. Alon, M. Zilberstein, O. Levy 和 E. Yahav, “code2vec: 学习代码的分布式表示”, *ACM 编程语言会议录*, 卷。3、没有。POPL, 第 1-29 页, 2019 年。