GraphCodeBERT：基于数据流的代码预训练模型

摘要

针对编程语言的预训练模型在各种代码相关任务中取得了显著的经验改进。然而，现有的预训练模型忽略了代码的内在结构，代码的内在结构提供了关键的代码语义，能增强代码理解过程。GraphCodeBERT 是一种考虑到代码固有结构的编程语言预训练模型。GraphCodeBERT 在 Transformer 的基础上开发，除了使用掩码语言建模任务外，还引入了两个结构感知预训练任务。一个是预测代码结构边缘，另一个是对齐源代码和代码结构之间的表征，并利用图引导的掩码注意函数将代码结构纳入其中。结果表明，代码结构和新引入的预训练任务可以改善 GraphCodeBERT 的性能，并在四个下游任务中达到最先进的性能。本文在代码翻译任务中做出改进，加入学习任务，并探究不同句子表征提取方法对模型性能的影响。温度系数能够缩放相似度的尺度，影响 $softmax$ 输出的分布，进而影响模型性能。为探索更适合于本任务的温度系数，将其设置为模型可优化的参数。实验结果表明改进方案效果有限，但是为未来的研究工作提供了可参考的方向。

关键词：预训练模型；代码生成；数据流；对比学习

1 引言

ELMo [22]、GPT [24] 和 BERT [7] 等预训练模型已经在许多自然语言处理 (NLP) 任务上带来了显着改进。这些预训练模型首先在大型无监督文本语料库上进行预训练，然后对下游任务进行微调。NLP 预训练模型的成功也促进了编程语言预训练模型的发展。现有的工作 [5,8,14,15,28] 将源代码视为一系列标记和预训练在源代码上建模以支持代码搜索、代码完成、代码摘要等与代码相关的任务。然而，以前的工作仅利用源代码进行预训练，而忽略了代码的固有结构。这种代码结构提供了有用的代码语义信息，这将有利于代码理解过程。以表达式 $v = max\_value - min\_value$ 为例，v 由 max_value 和 max_value 计算得出。程序员并不总是遵循命名约定，因此很难仅从变量 v 的名称来理解其语义。代码的语义结构提供了一种利用变量之间的依赖关系来理解变量 v 的语义的方法。

在这样的背景下，作者提出了一种考虑代码内在结构的编程语言预训练模型 GraphCodeBERT。与抽象语法树 (AbstractSyntaxTree，AST) 一样，GraphCodeBERT 不采用代码的语法级结构，而是利用代码的语义级信息，即数据流，进行预训练。数据流是一个图，其中节点表示变量，边表示变量之间的”值从何而来”的关系。与 AST 相比，数据流的复杂程度较低，并没有带来不必要的深层层次，其属性使得模型更加高效。为了从源代码和代码结构中学习代码表示，作者还引入了两个新的结构感知预训练任务。一种是数据流边缘预测，用于从代码结构中学习表示；另一种是跨源代码和数据流的变量对齐，用于在源代码和代码结构之间

对齐表示。GraphCodeBERT 是基于 Transformer 的神经网络架构 [30]，作者通过引入图引导的掩码注意力函数对其进行扩展，以融入代码结构。

GraphCodeBERT 在 CodeSearchNet 数据集 [13] 上进行预训练，该数据集包含六种编程语言的 2.3M 函数与自然语言文档配对。GraphCodeBERT 在四个下游任务上评估：自然语言代码搜索、克隆检测、代码翻译和代码精化。实验表明，模型在四个任务上达到了最先进的性能。进一步的分析表明，代码结构和新引入的两个预训练任务可以改进 GraphCodeBERT，并且模型对数据流的参与具有前后一致的偏好。

综上所述，GraphCodeBERT 的贡献在于：(1) 它是第一个利用代码语义结构学习代码表示的预训练模型。(2) 提出了两个新的结构感知的预训练任务，用于从源代码和数据流中学习表示。(3) 模型在代码搜索、克隆检测、代码翻译和代码纠错 4 个下游任务上有显著提升。

2 相关工作

本部分讨论了关于编程语言模型和结合代码结构的神经网络的一些相关工作。在编程语言模型方面，讨论了现有的研究所使用的各种预训练模型和框架在不同的编程语言和任务上进行预训练的效果。在结合代码结构的神经网络方面，有研究人员已经提出了利用抽象语法树 (AST) 的模型或者结合了图结构来表示程序并使用图神经网络进行推理的方法等，在不同的代码相关任务中都取得了较好的性能。

2.1 预训练的编程语言模型

受预训练在 NLP 领域大获成功的启发 [7, 19, 25, 35]，针对编程语言的预训练模型也促进了代码智能的发展。Kanade 等人 [5,8,14,15,28] 通过掩码语言建模和下一句预测目标，在大量 Python 源代码语料库上预训练了一个 BERT 模型。Feng 等人 [8] 提出了 CodeBERT，这是一种通过屏蔽语言建模和替换标记检测对编程语言和自然语言进行预训练的双峰模型，可支持代码搜索等文本代码任务。Karampatsis 和 Sutton [15] 使用 ELMo 框架在 JavaScript 语料库上对上下文嵌入进行预训练，以完成程序修复任务。Svyatkovskiy 等人 [28] 提出了 GPT-C，它是在源代码数据上从头开始训练的 GPT-2 的变体，用于支持代码补全等生成任务。Buratti 等人 [5] 提出了 C-BERT，这是一种基于转换器的语言模型，在 C 语言编写的资源库集合上进行了预训练，在抽象语法树 (AST) 标记任务中取得了很高的准确率。

与之前的研究不同，GraphCodeBERT 是首个利用代码结构学习代码表示以提高代码理解能力的预训练模型，它进一步引入了图形引导的屏蔽关注函数，将代码结构纳入转换器，并引入了两个新的结构感知预训练任务，以从源代码和代码结构中学习表示。

2.2 结合代码结构的神经网络

近年来，一些利用 AST 等代码结构的神经网络被提出，并在代码相关任务中取得了很强的性能，如代码完成 [3,16,18]、代码生成 [4,23,36]、代码克隆检测 [32,33,37]、代码总结 [2,12] 等。Nguyen&Nguyen [20] 提出了一种基于 AST 的语言模型，支持在当前编辑位置检测和建议语法模板。Allamanis 等人 [1] 用图来表示程序，用图神经网络来推理程序结构。Hellendoorn 等人 [11] 提出了两种不同的架构，使用门控图神经网络和 Transformers 结合局部和全局信息

来利用代码的丰富结构表示。目前，很多研究都是利用代码结构来从头开始学习特定任务，而不是使用预训练模型。在这项工作中，作者研究了利用代码结构进行预训练代码表示的方法。

3 相关技术

在此部分中，我们将介绍数据流和对比学习的基本概念。后文中，我们将介绍如何使用数据流进行预训练，以及如何利用对比学习方法对代码翻译任务进行改进。

3.1 数据流

数据流是一种表示变量之间依赖关系的图，其中节点表示变量，边表示每个变量的值来自哪里。与 AST 不同，同一源代码在不同抽象语法下的数据流是相同的。这种代码结构为理解代码提供了重要的代码语义信息。以 $v = max\_value - min\_value$ 为例，程序员并不总是遵循命名规则，因此很难理解变量的语义。数据流在一定程度上提供了理解变量 v 语义的途径，即 v 的值来自数据流中的最大值和最小值。此外，数据流还支持模型考虑在不同位置使用同一变量或函数所引起的远距离依赖关系。以图 1为例，有四个变量名称相同 (即 x^3、x^7、x^9 和 x^{11})，但语义不同。图中的图形显示了这些变量之间的依赖关系，并支持 x^{11} 更多地关注 x^7 和 x^9，而不是 x^3。接下来，我们将介绍如何从源代码中提取数据流。如图 1所示：

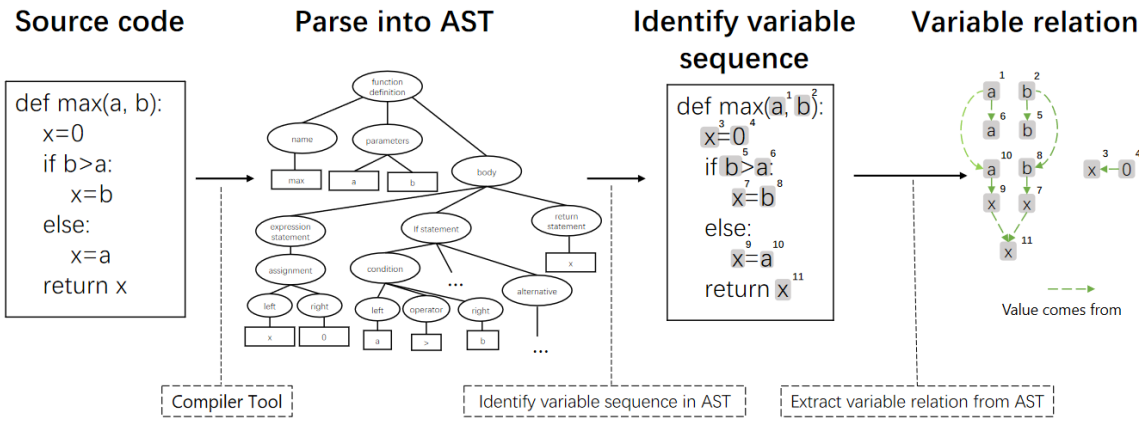

图 1. 根据源代码提取数据流的过程

图 1显示了通过源代码提取数据流的过程。给定源代码 $C = \{c_1, c_2, \ldots, c_n\}$，我们首先使用标准编译工具将代码解析为抽象语法树 (AST)。AST 包含代码的语法信息，终端 (叶子) 用于识别变量序列，表示为 $V = \{v_1, v_2, \ldots, v_k\}$。我们将每个变量视为图的一个节点，从 v_i 到 v_j 的直接边 $\varepsilon = \langle v_i, v_j \rangle$ 表示第 j 个变量的值来自第 i 个变量。以 $x = expr$ 为例，从 $expr$ 中的所有变量到 x 的边都被添加到图中。我们将有向边的集合记为 $E = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_l\}$，图关系 $G(C) = (V, E)$ 是用来表示源代码 C 中变量间依赖关系的数据流。

3.2 对比学习

对比学习的思想是使相似的样本更相近，不相似的样本互相远离 [10]，目标是要从样本中学习到一个好的语义表示空间 [9]。对于一对样本 x_i x_i^+ ，它们的特征分别为 h_i 和 h_i^+。在

一个训练簇中对样本对 x_i 和 x_i^+ 的训练目标为：

$$l_i = -log\frac{e^{sim\left(h_i,h_i^+\right)/\tau}}{\sum_{j=1}^{N} e^{sim\left(h_i,h_j^+\right)/\tau}} \tag{1}$$

其中 τ 是一个温度超参数，$sim\left(h_1,h_2\right)$ 是余弦相似度 $\frac{h_1^T \cdot h_2}{||h1|| \cdot ||h2||}$。在本文的工作中，我们利用下文中介绍的基于数据流的预训练代码模型 GraphCodeBERT 对输入代码语句进行编码 $h = f_\theta\left(x\right)$，然后结合对比学习目标 (如公式1所示) 进行微调。

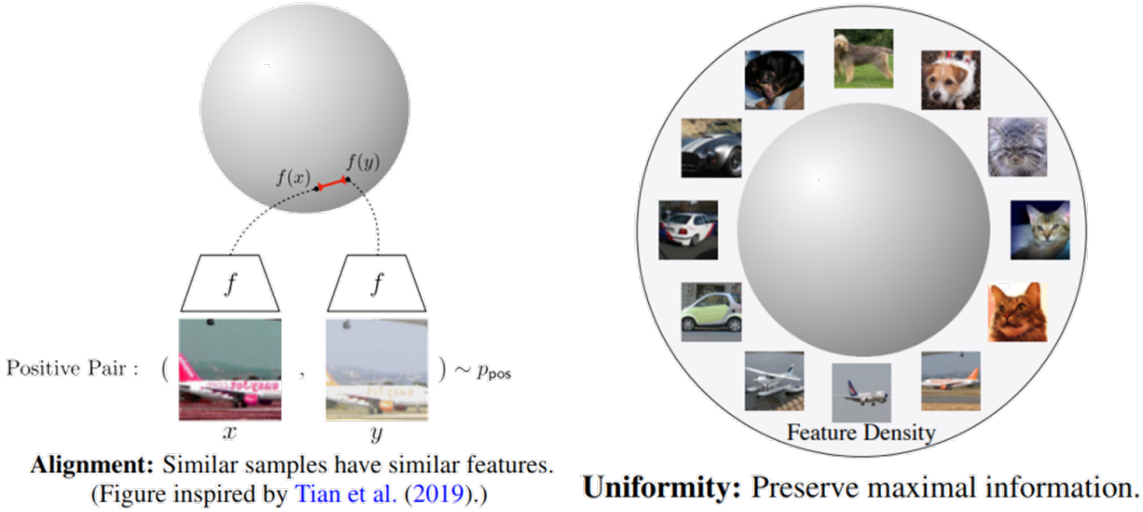

图 2. 输出单元超球体上特征分布的对齐性和均匀性的说明

下面将结合 Wang 等人 [31] 对嵌入向量分布的性质的讨论，解释对比学习的概念。Wang 等人认为对比学习一个重要的特点是它得到的特征向量具有对齐性和均匀性 (如图2所示)。对齐性，给定正样本对的分布 p_{pos}，对齐性计算配对实例嵌入之间的期望距离 (如公式2所示)，考察具有相似性的样本在空间中是否处在相近位置。指的是相似的样本拥有类似的特征向量，它应该对无关的噪声保持不变。

$$l_{align} \triangleq E_{(x,x^+)\ p_{pos}} ||f_x - f_x^+||^2 \tag{2}$$

均匀性顾名思义度量了嵌入是如何均匀分布的 (如公式3所示)，指的是系统应该倾向在特征里保留尽可能多的信息，这等价于使得映射到单位超球面的特征，尽可能均匀地分布在球面上，分布得越均匀，意味着保留的信息越充分。分布均匀意味着两两有差异，也意味着各自保有独有信息，这代表信息保留充分。

$$l_{uniform} \triangleq logE_{x,y\sim p_{pos}} e^{-2||f_x - f_y||^2} \tag{3}$$

4 GraphCodeBERT

在本节中，我们将介绍有关于 GraphCodeBERT 的具体内容。GraphCodeBERT，是一种基于 Transformer 的编程语言预训练模型。我们将从模型架构、图引导的掩码注意力和预训练任务，包括标准的掩码语言模型和新引入的预训练任务等方面描述。

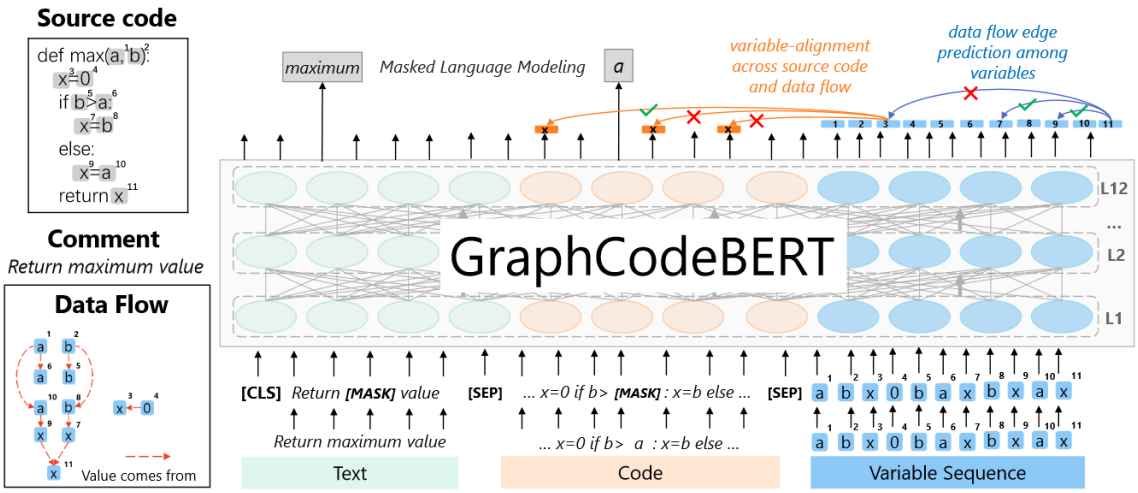

图 3. GraphCodeBERT 预训练框架

4.1 模型架构

图 3为 GraphCodeBERT 的模型架构。模型遵循 BERT [7] 结构，使用多层双向 Transformer [30] 作为模型主干。与仅使用源代码不同，作者还利用代码配对的注释来对模型进行预训练，以支持更多涉及自然语言的代码相关任务，如自然语言代码搜索 [8]。我们进一步将数据流，即图数据结构，作为模型输入的一部分。

给定一个源代码 $C = \{c_1, c_2, ..., c_n\}$ 和它的注释 $W = \{w_1, w_2, ..., w_m\}$，我们可以得到相应的数据流 $G(C) = (V, E)$，如第 3 节所述，其中 $V = \{v_1, v_2, ..., v_k\}$ 是变量的集合，$E = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_l\}$ 是表示每个变量的值来自何处的直接边的集合。这样，就可以将注释、源代码和变量集连接起来作为序列输入 $X = \{[CLS], W, [SEP], C, [SEP], V\}$，其中 $[CLS]$ 是位于三个片段前面的特殊标记，$[SEP]$ 是用于区分两种数据类型的特殊符号。

GraphCodeBERT 将序列 X 作为输入，然后将序列转换为输入向量 H^0。对于每个令牌，其输入向量是通过对相应的令牌和位置嵌入求和构造的。对所有变量，会使用一个特殊的位置嵌入来表示它们是数据流的节点。该模型在输入向量上应用 N 个 Transformer 层来产生上下文表示 $H^n = transformer_n(H^{n-1}), n \in [1, N]$。每个 Transformer 层包含一个架构相同的 Transformer，在第 n 层的输入 H^{n-1} 上执行多头自注意力操作 [30]，然后执行前馈层。

$$G^n = LN(MultiAttn(H^{n-1}) + H^{n-1}) \tag{4}$$

$$H^n = LN(FFN(G^n) + G^n) \tag{5}$$

其中 $MultiAttn$ 为多头自注意力机制，FFN 为两层前馈网络，LN 表示层归一化操作。对于第 n 个 Transformer 层，计算多头自注意力的输出 $\hat{G}^n$：

$$Q_i = H^{n-1}W_i^Q, K_i = H^{n-1}W_i^K, V_i = H^{n-1}W_i^V \tag{6}$$

$$head_i = softmax(\frac{Q_i K_i^T}{d_k} + M)V_i \tag{7}$$

$$\hat{G}^n = [head^1; ...; head^u]W_n^O \tag{8}$$

其中，前一层的输出 $H^{n-1} \in R^{|X|\times d_h}$ 分别使用模型参数 $W_i^Q, W_i^K, W_i^V \in R^{d_h \times d_k}$ 线性投影到查询、键和值的三元组，u 为头数，d_k 为头的维度，$W_n^O \in R^{d_h \times d_h}$ 为模型参数。$M \in R^{|X|\times|X|}$ 是一个掩码矩阵，如果允许第 i 个令牌与第 j 个令牌进行注意力计算，则 M_{ij} 是 0，否则 $-\infty$。

4.2 基于图的掩码注意力机制

为了将图结构纳入 Transformer，作者定义了一个图引导的屏蔽注意力函数来过滤掉不相关的信号。注意力掩蔽函数通过为注意力得分 $q_j k_i^T$ 加上一个无穷大的负值，使得使用 $softmax$ 函数后注意力权重变为零，从而避免了 q_j 关注关键字 k_i。为了表示变量之间的依赖关系，如果节点 v_j 与节点 v_i 之间有直接边 (即 $\langle v_j, v_i\rangle \in E$)，或者它们是同一个节点 (即 $i = j$)，则允许节点 q_{v_i} 关注节点关键字 k_{v_j}。否则，将通过在注意力得分中加入一个无限负值来屏蔽关注。为了表示源代码标记和数据流节点之间的关系，作者首先定义了一个集合 E'，如果变量 v_i 是从源代码 token c_j 中识别出来的，那么就存在 $\langle v_i, c_j\rangle$ / $\langle c_j, v_i\rangle \in E'$。在训练中，当且仅当 v_i, c_j 满足条件 $\langle v_i, c_j\rangle$ / $\langle c_j, v_i\rangle \in E'$ 时，就允许节点 q_{v_i} 和代码 k_{c_j} 相互关联。总而言之，作者使用图引导掩码注意矩阵 (如公式9) 作为等式7中的掩码矩阵 M:

$$M_{ij} = \begin{cases} 0, if\ q_i \in \{[CLS], [SEP]\}\ or\ q_i, k_j \in W \cup C\ or \langle q_i, k_j\rangle \in E \cup E' \\ -\infty, othrtwise \end{cases} \tag{9}$$

4.3 预训练任务

我们将在本节介绍用于预训练 GraphCodeBERT 的三个预训练任务。第一个任务是屏蔽语言建模 [7]，用于从源代码中学习表示。第二个任务是数据流边缘预测，用于从数据流中学习表征，我们首先屏蔽一些变量的数据流边缘，然后让 GraphCodeBERT 预测这些边缘。最后一项任务是跨源代码和数据流的变量对齐，用于对齐源代码和数据流之间的表示，预测变量是从哪里识别出来的。

屏蔽语言模型 作者效仿 Devlin 等人 [7] 的做法，在模型中应用掩码语言建模 (MLM) 预训练任务。具体来说，研究人员从源代码和配对注释中随机抽取 15% 的标记。在 80% 的情况下用 $[MASK]$ 标记替换它们，在 10% 的情况下用随机标记替换它们，在 10% 的情况下保持不变。MLM 的目标是预测这些采样标记的原始标记，这在之前的工作中已被证明是有效的 [7,8,19]。特别是，如果源代码上下文不足以推断出屏蔽的代码标记，模型可以利用注释上下文，鼓励模型将自然语言和编程语言表征统一起来。

边缘预测 为了从数据流中学习表示，作者引入了数据流边缘预测的预训练任务。这样做的目的是鼓励模型学习结构感知表征，这种表征可以编码 “值从哪里来” 的关系，从而更好地理解代码。具体来说，在训练过程中，随机抽样数据流中 20% 的节点 V_s，通过在掩码矩阵中添加无限负值来掩码连接这些抽样节点的直接边缘，然后预测这些被掩码的边缘 E_{mask}。以图2中的变量 x^{11} 为例，首先屏蔽图中的边 $\langle x^7, x^{11}\rangle$ 和 $\langle x^9, x^{11}\rangle$，然后让模型预测这些边。其中，$E_c = V_s \times V \cup V \times V_s$ 是边预测的候选集，$\delta(e_{ij} \in E)$ 在 $\langle v_i, v_j\rangle \in E$ 时为 1，否则为 0，而从 i 节点到 j 节点之间存在边的概率 $p_{e_{ij}}$ 是通过使用 GraphCodeBERT 中两个节点的表示，

按照一个 $sigmoid$ 函数进行点乘计算得出的。为了平衡正负样本的比例，对 E_c 的负样本和正样本进行了相同数量的采样。

$$loss_{EdgePred} = -\sum_{e_{ij} \in E_c} [\delta(e_{ij} \in E_{mask}) \log p_{e_{ij}} + (1 - \delta(e_{ij} \in E_{mask}))(1 - \log p_{e_{ij}})] \tag{10}$$

节点对齐 为了对齐源代码和数据流之间的表示，作者引入了源代码和数据流之间节点对齐的预训练任务，这与数据流边缘预测类似。模型不预测节点之间的边缘，而是预测代码标记和节点之间的边缘。这样做的目的是鼓励模型根据数据流对齐变量和源代码。以图4为例，先屏蔽数据流中的变量 x^{11} 与代码标记之间的边缘，然后预测数据流中的变量 x^{11} 是由哪个代码标记识别出来的。我们可以看到，根据数据流信息，模型可以预测变量 x^{11} 与表达式 “return x” 中的变量 x 一致 (即 x^{11} 的值来自 x^7 或 x^9)。

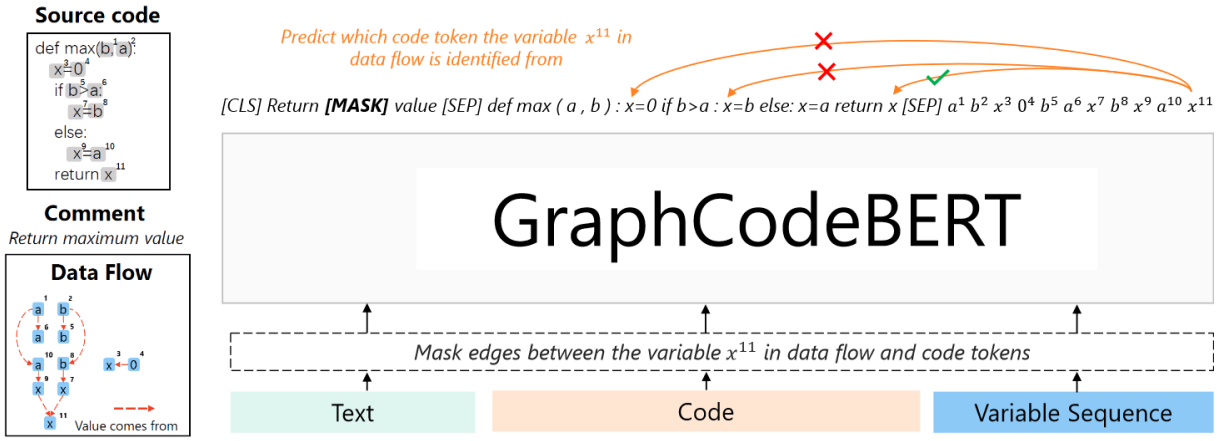

图 4. 节点对齐任务示例

具体来说，模型在图中随机抽样 20% 的节点 V_s'，屏蔽代码标记和抽样节点之间的边，然后预测屏蔽边 E'_{mask}。这项任务的预训练目标与公式10类似，其中 $E'_c = E'_s \times C$ 是节点对齐的候选集。同样的，也会对 E'_c 进行相同数量的负样本和正样本采样。

$$loss_{EdgePred} = -\sum_{e_{ij} \in E'_c} [\delta(e_{ij} \in E'_{mask}) \log p_{e_{ij}} + (1 - \delta(e_{ij} \in E'_{mask}))(1 - \log p_{e_{ij}})] \tag{11}$$

5 复现细节

5.1 与已有开源代码对比

本次复现工作选择的论文源代码及预训练模型已开源，根据作者发布的代码能够很好的复现出论文中四个下游任务的效果。本文在原论文的基础上对代码翻译进行改进，加入了对比学习任务。为探究更适合于代码翻译任务的句子级征，本文分别在 $[CLS]$、模型最后一层隐藏层输出的均值等多种池化情况下进行实验。相比较于常规的对比学习任务，本文将对比学习中的温度系数调整为模型可以优化的形式，以选择更有利于本任务的温度系数。原论文开源代码地址为：https://github.com/microsoft/CodeBERT。

5.2 复现任务

代码搜索 给定一个自然语言作为输入，任务旨在从候选代码集合中找到与语义最相关的代码。我们在包含六种编程语言的 CodeSearchNet 代码语料库 [13] 上进行实验。与 Husain 等人 [13] 使用的数据集和设置不同，作者通过手工制定的规则过滤低质量查询，并将 1000 个候选扩展到整个代码语料库，更接近现实生活场景。我们将使用 Mean Reciprocal Rank(MRR) 作为评价指标，并在下文中报告与原文结果的对比。

代码克隆检测 代码克隆是在给定相同输入时输出相似结果的多个代码片段。该任务旨在度量两个代码片段之间的相似性，这有助于降低软件维护的成本并防止 bug。我们在 BigCloneBench 数据集 [27] 上进行实验，选择精确度、召回率和 F1 得分作为评价指标。

代码翻译 代码翻译旨在将遗留软件从平台中的一种编程语言迁移到另一种编程语言。继 Nguyen 等之后 [21]，作者在与他们相同的几个开源项目中爬取数据集，这些项目都有 Java 和 C# 实现，根据这两种语言的文件名和方法名可以将它们配对。在去除重复和带有空函数体的方法后，构造出实验数据集。在实验阶段，选择用于衡量翻译文本质量的指标 BLEU 以及正确率作为评价指标，评估模型翻译的结果与标准答案之间的差异。

代码纠错 代码纠错旨在自动修复代码中的 bug，有助于降低 bug 修复的成本。使用了 Tufano 等 [29] 发布的数据集。源代码是存在错误的 Java 函数，目标代码是经过改进的函数。所有函数和变量名都已规范化。数据集包含两个基于函数长度的子集 (即小型和中型)，我们分别在两个数据集中进行实验，与代码翻译任务相同，选择 LEU 以及正确率作为评价指标。

5.3 改进方法

本文在原论文的基础上，在代码翻译这个下游任务中加入了对比学习目标，希望利用对比学习任务对齐代码语义，能够帮助模型更好的学习生成目标代码。通过改变句子级初始表征的提取方法，本文探究了不同提取方法对模型能力的影响。在计算对比学习损失时，加入温度系数的作用是调节对负样本的关注程度。一般的，越小的温度系数越关注于将正样本和最相似的负样本分开，以得到更均匀的表示。本文将温度系数作为模型可优化的参数，以找到更适合于本任务的温度系数值。

句子级表征的提取方法 本文探究了不同提取方法对实验结果的影响。在 BERT 最初的实现中，会在序列前加入 $[CLS]$ 将其作为后续文本分类的表示。而 $[CLS]$ 位本身没有语义，经过 12 层，得到的是 $self-attention$ 后所有词的加权平均，相比其他正常词，可以更好的表征句子语义。加上 BERT 在多个下游任务中的优秀的表现，我们通常使用 $[CLS]$ 表示整个句子嵌入的方法。Reimers 和 Gurevych [26] 及 Li 等人 [17] 的研究表明，采用预训练模型 (尤其是第一层和最后一层的隐藏层) 的平均嵌入度会比 $[CLS]$ 性能更好。因此本文选取 $[CLS]$、最后一层隐藏层输出的均值以及第一层和最后一层的隐藏层输出的均值，三种方法作为对比，探究不同提取方法对模型性能的影响。

构造样本对 目前对比学习的一个重要问题是如何构建正负样本对。在计算机视觉的对比学习框架 SimCLR [6] 中，它将一张图像经过两种不同的扩充方式得到一个正样本对，batch 中的其它样本作为负样本对。在 NLP 中，CLEAR [34] 等算法提出使用随机删除，替换，调整顺序等数据扩充方式进行数据扩充。但是因为文本的离散性，通过数据扩充得到的文本在文本流畅性，语法正确性上都要打些折扣。本文对代码翻译任务进行改进，对于一对翻译代

码，源代码与目标代码具有极高的语义相似度，在样本空间中应该处在相近的位置中，可以将一对翻译代码作为对比学习中的一对正样本对，将同一批中的其它目标代码与当前源代码构成一对负样本对，以此进行对比学习任务优化训练目标, 如公式12：

$$loss_i = -log\frac{e^{sim\left(h_i^{source},h_i^{target}\right)/\tau}}{\sum_{j=1}^{N} e^{sim\left(h_i^{source},h_j^{target}\right)/\tau}} \tag{12}$$

我们基于预训练编程语言模型 GraphCodeBERT 获得输入代码表征，将句子级的表征表示为 $h_i = Pooling(f_\theta(x_i))$，其中 $Pooling$ 表示提取句子表征的不同池化策略

可优化的温度系数 通过对比学习的优化目标 (如公式1) 不难看出，温度系数的作用是缩放相似度的尺度，从而影响 $softmax$ 输出的分布。在对比学习中，那些难以与正样本区分开的负样本其实可能是潜在的正样本。过分强迫与困难样本分开会破坏学到的潜在语义结构。因此，温度系数不能过于考虑两个极端情况。温度系数趋向于 0 时，对比损失退化为只关注最相似的负样本的损失函数；当温度系数趋向于无穷大时，对比损失对所有样本都一视同仁，失去了困难样本关注的特性。因此，找到一个合适的温度系数，对于更好的实现对比学习目标也很重要。在本文中，改变了以往人工调整温度系数的方法，将它设置为在模型训练过程中的可调整参数，以寻找更适合于当前任务的温度系数值。

6 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

6.1 实验环境

本文的实验环境如下表1所示。

表 1. 实验环境

因素	配置
操作系统	Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-91-generic x86_64)
CPU	Intel(R) Xeon(R) Platinum 8383C CPU @ 2.70GHz
GPU	四张 NVIDIA A100 40GB PCIe
语言	Python 3.7

6.2 论文复现实验

代码搜索的目的是将自然语言作为输入，从候选代码集合中找出语义最相关的代码。在实验中，模型需要从 1000 个候选答案中检索出一个查询答案。我们使用 GraphCodeBERT 分别对查询文本和源代码进行数据流编码，并计算它们对特殊标记 [CLS] 表示的内积作为相关性得分，从而对候选代码进行排序。在微调步骤中，我们设置学习率为 2e-5，批量大小为 32，查询语句和代码的最大序列长度分别为 128 和 256，最大节点数为 64。我们选择使用 Adam 优化器更新模型参数，并在验证集上执行早期停止。实验结果如表2所示：

表 2. Code Search 复现结果

Method	Ruby	JavaScript	Go	Python	Java	Php	Overall
GraphCodeBERT	0.703	0.644	0.897	0.692	0.691	0.649	0.713
Reproduction	0.709	0.647	0.896	0.693	0.692	0.647	0.714

代码克隆检测旨在度量两个代码片段之间的相似性。作者使用 BigCloneBench 数据集[27]，该数据集包含超过 600 万个来自 10 个不同功能的真实克隆对和 26 万个错误克隆对。实验中，将任务视为二分类来微调 GraphCodeBERT，其中我们使用源代码和数据流作为输入。由 [CLS] 的表示式通过点积计算出真克隆的概率。在微调步骤中，设置学习率为 2e - 5，批大小为 16，最大序列长度为 512，最大节点数为 128。我们使用 Adam 优化器来更新模型参数和调整超参数，复现结果如表3所示：

表 3. Code Clone 复现结果

Method	Precision	Recall	F1
GraphCodeBERT	0.948	0.952	0.95
Reproduction	0.965	0.937	0.95

代码翻译的目的是将遗留软件从平台中的一种编程语言迁移到另一种编程语言。我们将利用 Transformer 进行实验，使用与预训练模型相同的层数和隐藏大小，包括 12 层 Transformer，具有 768 维的隐藏状态和 12 个注意力头。为了利用预训练模型进行翻译，我们使用预训练模型初始化编码器，并随机初始化解码器的参数和源到目标注意力。我们将学习率设为 1e-4，批量大小设为 32，最大序列长度设为 256，最大节点数设为 64，同时使用 Adam 优化器更新模型参数和调整超参数，并在验证集上执行早期停止。复现结果如表4所示：

表 4. Code Translation 复现结果

Method	JAVA $\longrightarrow$ C#		C# $\longrightarrow$ JAVA	
	BLEU(%)	ACC(%)	BLEU(%)	ACC(%)
GraphCodeBERT	80.58	59.4	72.64	58.8
Reproduction	81.19	61.2	76.22	61.9

代码纠错旨在自动修复代码中的错误。与代码翻译任务类似，我们还使用序列到序列的 Transformer 模型进行实验。在微调步骤中，我们采用预训练的 GraphCodeBERT 作为编码器。设置学习率为 1e - 4，批处理大小为 32，最大序列长度为 256，最大节点数为 64。同样的，我们采用 Adam 优化器来更新模型参数。复现结果如表5所示:

表 5. Code Refinement 复现结果

Method	Small		Medium	
	BLEU(%)	ACC(%)	BLEU(%)	ACC(%)
GraphCodeBERT	80.02	17.3	91.31	9.1
Reproduction	79.76	17.9	90.95	10.9

本次的复现任务是参考了论文作者开源的代码完成的，根据源代码结合当前实验环境、版本配置做出微小调整，便可以很好的运行。从复现的实验结果来看，已经基本达到了原文的实验结果，值得注意的是在代码翻译中，本文复现的效果有着明显优于原文的效果，考虑到实验环境的细微不同，认为本次复现的结果是合理的、可以接受的。

6.3 论文改进实验

实验一：对比学习任务 在本文的改进实验中同样使用了 Transformer 结构。为了利用预训练模型进行表征提取以及翻译任务，我们使用 GraphCodeBERT 模型初始化编码器，并随机初始化解码器的参数和源到目标的注意力，用于训练一个生成式的模型。为与原文结果进行直接比较，在训练时，我们遵循了原文的实验设置 (包括层数和隐藏大小)，将学习率设为 1e-4，批量大小设为 32，最大序列长度设为 256，最大节点数设为 64，同时使用 Adam 优化器更新模型参数和调整超参数。我们首先讨论加入对比学习任务后，使用不同的池化方式对模型能力产生的影响。

表 6. 不同池化方法的比较

Method	JAVA $\longrightarrow$ C#		C# $\longrightarrow$ JAVA	
	BLEU(%)	ACC(%)	BLEU(%)	ACC(%)
GraphCodeBERT(LM)	80.58	59.4	72.64	58.8
Reproduction(LM)	81.19	61.2	76.22	61.9
LM_contrast_CLS	80.37	60.4	75.97	61.8
LM_contrast_AVG_Last	80.93	60.8	75.92	61.8
LM_contrast_AVG_First_Last	80.41	60.3	75.0	61.0

在实验结果6中，“LM” 表示当前加入了最大化语言模型目标 (如公式13所示) 进行优化。其中，$target = \{t_1, \ldots, t_n\}$ 表示目标代码序列，与目标序列对应的有待翻译序列 $source$，z_i^{source} 为序列表征，条件概率 P 的模型即为带有参数 Θ 的解码器。“contrast” 表示加入了对比学习任务。CLS、AVG_Last 及 AVG_First_Last 分别代表使用最后一层 [CLS] 令牌对应的 embedding 作为句子级表征的情况，AVG_Last 代表取最后隐藏层输出的均值作为句子级表征的情况，类似的有 AVG_First_Last。

$$Loss\_lm_i(target) = \sum_j logP(t_j \mid t_1, \ldots, t_{j-1}; z_i^{source}; \Theta) \tag{13}$$

结果表明，使用模型隐藏层最后一层的输出作为句子级表征的模型即 (第四组) 优于将 [CLS] 或隐藏层第一层和最后一层输出的均值作为表征的结果。其中，改进方法展现了比原文结果更优的性能，这证明了我们加入对比学习任务在代码翻译方面的有效性。

实验二：加入可调的温度系数 通过将温度系数设置为在模型训练过程中的可调整参数，以寻找更适合于当前任务的温度系数值，本文在对比学习的第二、三组 (对应表6的最后两行) 实验的基础上进行，其他实验设置与上文保持一致。实验结果如下表7所示：

表 7. 加入可优化温度系数后的实验结果

Method	JAVA $\longrightarrow$ C#		C# $\longrightarrow$ JAVA	
	BLEU(%)	ACC(%)	BLEU(%)	ACC(%)
GraphCodeBERT(LM)	80.58	59.4	72.64	58.8
Reproduction(LM)	81.19	61.2	76.22	61.9
LM_contrast_AVG_Last	80.93	60.8	75.92	61.8
LM_contrast_AVG_Last_temper	79.28	58.6	74.82	59.8
LM_contrast_AVG_First_Last	80.41	60.3	75.0	61.0
LM_contrast_AVG_First_Last_temper	79.39	59.2	75.48	60.8

根据实验结果我们不难看出将温度系数设置为模型可调整的参数后，模型性能不增反降。在实验一中我们将温度系数设置为 0.02，而在实验二中温度系数为随机初始化。在实验二的两个实验中，随机初始化值在 0.5 左右，经过模型的优化温度系数持续下降，而直到训练结束才下降到 0.03 左右，未达到实验一的水平。所以我们相信如果在实验二中，将温度系数初始化为一个相对较小的值会对实验结果产生一定的积极影响。

在改进实验中，没有实现优于复现实验的效果，我们将原因归咎为以下两点：(1) 时间有限，未对模型进行调参优化。(2) 实验二中温度系数的初始化值太大。(3) 未选择适合的评价指标。传统的 NMT 评估依赖于 BLEU 等基于 n-gram 重叠的指标。然而，在处理编程语言时，尽管代码稍有改动，但语法，特别是编译和计算输出可能会有很大差异。相反，在语义上等同的代码，如果仅在变量名或操作顺序上存在差异，其 BLEU 分数也会很低。

7 总结与展望

在本文中，我们介绍了利用数据流来学习代码表示的 GraphCodeBERT，以及两个结构感知的预训练任务，并复现了 GraphCodeBERT 的四个代码相关下游任务（包括代码搜索、克隆检测、代码翻译和代码精炼）达到了原文展示的性能。在代码翻译的任务上，本文尝试通过加入对比学习任务并引入可优化的温度系数的方法做出改进，实验效果差强人意，但为我们后续的研究打下来良好的基础，有充分的提升空间。在未来，本文尝试通过对模型进行参数调整，修改温度系数初始化值以及寻找更适合的评价指标的方法做出进一步的改进。

参考文献

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *CoRR*, abs/1711.00740, 2017.

[2] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *CoRR*, abs/1808.01400, 2018.

[3] Uri Alon, Roy Sadaka, Omer Levy, and Eran Yahav. Structural language models of code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 245–256, 2020.

[4] Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Oleksandr Polozov. Generative code modeling with graphs. *CoRR*, abs/1805.08490, 2018.

[5] Luca Buratti, Saurabh Pujar, Mihaela A. Bornea, J. Scott McCarley, Yunhui Zheng, Gaetano Rossiello, Alessandro Morari, Jim Laredo, Veronika Thost, Yufan Zhuang, and Giacomo Domeniconi. Exploring software naturalness through neural language models. *CoRR*, abs/2006.12641, 2020.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 1597–1607, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, pages 1536–1547, 2020.

[9] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6894–6910, 2021.

[10] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*, pages 1735–1742, 2006.

[11] Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. Global relational models of source code. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[12] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, pages 200–210, 2018.

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

[14] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Pre-trained contextual embedding of source code. *CoRR*, abs/2001.00059, 2020.

[15] Rafael-Michael Karampatsis and Charles Sutton. Scelmo: Source code embeddings from language models. *CoRR*, abs/2004.13214, 2020.

[16] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 150–162, 2021.

[17] Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 9119–9130, 2020.

[18] Jian Li, Yue Wang, Michael R. Lyu, and Irwin King. Code completion with neural attention and pointer networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4159–4165, 2018.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[20] Anh Tuan Nguyen and Tien N. Nguyen. Graph-based statistical language model for code. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 858–868, 2015.

[21] Xuan-Phi Nguyen, Shafiq R. Joty, Steven C. H. Hoi, and Richard Socher. Tree-structured attention with hierarchical accumulation. *CoRR*, abs/2002.08046, 2020.

[22] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings*

of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), pages 2227–2237, 2018.

[23] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1139–1149, 2017.

[24] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[26] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990, 2019.

[27] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal Kumar Roy, and Mohammad Mamun Mia. Towards a big data curated benchmark of inter-project code clones. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 476–480, 2014.

[28] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: code generation using transformer. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 1433–1443, 2020.

[29] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Trans. Softw. Eng. Methodol.*, 28(4):19:1–19:29, 2019.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[31] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 9929–9939, 2020.

[32] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pages 261–271, 2020.

[33] Huihui Wei and Ming Li. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3034–3040, 2017.

[34] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. CLEAR: contrastive learning for sentence representation. *CoRR*, abs/2012.15466, 2020.

[35] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[36] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 440–450, 2017.

[37] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 783–794, 2019.