南京大学

研 究 生 毕 业 论 文

（申请工学硕士学位）

论 文 题 目	面向相似度量的源代码理解技术
作 者 姓 名	邹智鹏
学科、专业名称	工学硕士 (软件工程领域)
研 究 方 向	软件工程
指 导 教 师	陈振宇 教授

2021 年 5 月 20 日

学　　号：**MG1832015**

论文答辩日期：**2021 年 5 月 20 日**

指 导 教 师：　　（签字）

Source Code Representation Technology for Similarity Measurement

By

Zhipeng Zou

Supervised by

Professor **Zhenyu Chen**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：面向相似度量的源代码理解技术

工学硕士(软件工程领域) 专业 2018 级硕士生姓名：邹智鹏

指导教师（姓名、职称）：陈振宇 教授

摘 要

伴随着大数据时代的到来和大量数据知识的积累，现在的软件系统已经逐渐从信息化朝智能化方向发展，智能化软件工程便是如此。精准理解代码含义，能够用于代码分类、缺陷检测、克隆检测、代码检索等众多智能化软件工程任务。但现有的代码理解方法无法全面地捕捉代码文本层次的语义特征，并且处理语法结构特征的方法具有较高的复杂度和较差的鲁棒性，制约甚至忽略了语法特征的提取。此外，现有的抽象语法树处理方法，引入了大量的噪声，显著降低了代码理解的准确性。

源代码理解客观上即是能用文本刻画其实现的功能，并寻找一个映射，用一固定维度的低维稠密向量来刻画代码含义，并在文本语义空间中使用相似性度量方法进行评估的过程。源代码将预先被处理为语言无关的 AST 形态，根据相应的定义和算法构造路径对序列，从中提取语法、语义等特征。综合现有的问题，本文提出了混合编码器，即用于获取语义信息的代码子序列编码器，和用于获取语法信息的路径对编码器。其中，代码子序列编码处理源代码中的自然语言片段，丰富了语义特征的建模。本文使用更简单的静态路径对编码器，处理代码的语法结构，实现了更鲁棒和更准确的语法理解。特别地，本文还提出基于自注意力机制的动态路径融合方法 (**S**elf-attention based **P**ath **F**usion, SPF)，将自注意力机制作为 RNN 的下游，实现更准确的结构特征融合，大幅降低了原始 AST 处理的噪声，增强了代码理解的准确性，降低了问题规模，提升了编码效率。

本文分别在方法名生成和代码同文本语义相似度量两个任务上，对代码理解方法进行验证。实验结果显示，本文的方法在两个任务上的效果均明显超越了基准实验的表现。根据实验结果分析，子序列处理方法解决了 AST 叶子节点和方法名中存在的长尾问题。凭借 RNN 的稳定性和时序性，AST 节点间的依赖关系建模更加准确，提升了语法特征抽取效果。SPF 方法在 RNN 网络上的应用，抑制了噪声数据干扰，动态地得到了高质量的特征组合，最终在两个任务上分别取得了相对基准方法 20% 和 60% 的指标提升。此外，综合损失函数改进和代码 API 信息，使本文的 SPF 方法达到了更优的代码理解目标。

关键词：代码理解，智能化软件工程，语义匹配，自注意力，相似度量

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Source Code Representation Technology for Similarity Measurement

SPECIALIZATION: Software Engineering

POSTGRADUATE: Zhipeng Zou

MENTOR: Professor Zhenyu Chen

Abstract

With the development of big data and the accumulation of a large amount of knowledge in data, software systems have gradually transformed from informatization to intelligence, such as intelligent software engineering. Source code comprehension can be used for many intelligent software engineering tasks, including code classification, defect detection, clone detection, and code retrieval. However, the existing methods for comprehension cannot capture the code semantic from the literal aspect completely, and are complex and not robust for code syntax. It lows or even ignores the code syntax extraction. Futher more, the methods based on abstract syntax tree are disturbed by numerous noises, which significantly reduces the performance of code comprehension.

Objectively, source code comprehension is a procedure to make a text can express its function. And we can find a mapping, to represent the code as a fixed, low-dimensional and dense vector, which can be used to measure the similarity between codes and texts in the semantic space with the vector space model. We first convert the source code will into AST that independent with programming language, then construct a sequence of path pairs based on our definitions and algorithms to obtain features consist of syntax and semantics. In this thesis, we proposed a hybrid encoder, that is, a sub-token encoder for semantic, and a path pair encoder for syntax. Among them, the sub-token encoder encodes text fragments in the source code to a vector, which strengthens the semantic features. And we adopt a simpler static path encoder to handle the syntax of the code, which leads to a more robust and accurate syntax comprehension. In particular, we also proposed a dynamic path fusion method named *Self-attention based Path Fusion, i.e.* ***SPF***. So that, the syntax features fusion can be more effective, the noise in the existing AST encoding method can be greatly reduced, the accuracy

of code comprehension can be improved, the problem scale can be simplified, and the code encoding can be more efficient.

We conduct two tasks: ***1) method name generation*** and ***2) semantic matching between code and text*** to test our methods. The experimental results show that all our methods obviously outperform the benchmark experiment on both tasks. Relied on the robustness and efficiency of RNN, the sequential features between AST nodes are more precise, which improved syntax feature extraction. The proposed dynamic SPF method attached to the RNN network suppressed the interference of noise in data and obtained a high-quality fused feature automatically. Finally, compared with baseline, both two tasks improved the metrics about 20% and 60%. In addition, based on the variant of loss and API corpora, we further optimized the proposed SPF to a better code comprehension method.

Keywords: Code Comprehension, Intelligent Software Engineering, Semantic Matching, Similarity Measurement

目录

表 目 录

插图

第一章　引言

1.1　项目背景

在软件工程的发展过程中，随着项目的迭代以及需求的变更，已经积累了海量的数据，包括文档、缺陷、源代码、二进制资源等类型。软件文档通常大致包含可行性报告、开发计划、需求文档、数据要求说明、测试计划、设计文档、用户手册、操作手册、测试分析文档等。面向开发者的工具类或框架类项目，还需包含应用程序接口文档（API 文档）。软件开发过程中，始终伴随着缺陷的产生，提出缺陷再修复缺陷是软件迭代过程中至关重要的一步。缺陷报告即是用于描述软件项目中已有问题的一类报告，用于同开发者指出当前版本所存在的各类问题和复现过程。据统计，截至 2013 年 12 月，Apache 与 Ecplise 项目已经分别有超过 39,300 篇和 422,800 篇缺陷报告，且在 Bugzilla[1]系统中 Eclipse 项目还保持着平均每天 29 篇的增长速度 [62]。以 Linux 内核项目为例，截至 2019 年 12 月，Linux 5.4 版代码行数业已达到 18,000,000 行 [45] 的量级，此外开源社区还源源不断地在贡献更多地优质项目。诸如 Apache 基金会下，已有 341 个活跃项目[2]，海量的项目带来了海量的源码以及海量的文档信息，同时也预示着其中蕴含着大量的软件工程领域知识与信息。这从正面和侧面说明当前软工领域存在大量源代码数据的现状，同样也说明对源代码理解挖掘的重大意义。杨芙清指出，软件工程实质上是对现实世界的认知，实现了由现实世界抽象、分解到工程领域的一个规范过程，预示着软件工程数据实质上是对现实世界知识的一类工程化总结。对此类数据的挖掘过程，可视为透过软件本身获取对应领域知识的过程。代码理解即是从软件的代码中，获取到其中的知识，再应用于自动化软工任务。

大数据时代，在信息化的推进下，各行各业各领域均已积累了海量级的数据，为了从海量的数据中快速获得有效的知识，数据挖掘技术在此背景下大放异彩，取得了巨大成功。源代码本身也是一种数据模态，其具有同文本相似的特性，用于向计算机说明程序的执行逻辑，是一种人可理解和计算机可以理解的中间高级语言。如软件设计借助使用程序框图来转化为源代码，作为一段业务的具体执行逻辑。源代码知识也可以通过数据挖掘的方式，进一步将只可被开发人员理解的代码语言，转化为可供普通人士阅读的知识形态。数据挖掘实际

[1] https://www.bugzilla.org/
[2] https://projects.apache.org/

上是从已有的数据集中发现模式，利用模式获得决策的过程 [56]。常见的如线性回归算法，能够从一系列的实数值中，找到一个可近似覆盖绝大部分点的线或平面，使用该线或平面可以对新样本给出所需值的实数预测。对于一些有限离散集的预测问题而言，通常使用分类算法来处理，如垃圾邮件分类问题。常见的分类算法，决策树、支持向量机（SVM）等算法，能够实现对数据的归纳，并分类为特定的类别。源代码也可以结合使用分类算法，来实现代码分类任务。郎大鹏等提出使用随机森林算法，来识别恶意代码。但是这些算法都需要明确的标签，支撑算法的学习，该过程被称为有监督学习。相反的，无需对样本明确标签便能够在数据中识别模式的一类算法，称之为无监督学习。典型的无监督学习方法，如关联规则挖掘能够利用共现统计信息，挖掘出样本之间的潜在联系，而聚类算法能够实现将相似样本归纳为若干簇类。通过代码理解算法，按照空间距离使用聚类算法可以获得到大量具有一定相似性的代码集合，对各个集合索引，可在代码检索任务中大幅提升效率。

使用机器学习实现代码理解是有效途径之一，但问题在于机器学习严重依赖于特征工程。数据挖掘技术的应用克服了海量数据中获取知识的难点，在软件工程领域也同样有巨大的应用价值。文本挖掘方法被广泛用于该领域，并且取得了一定的成就 [67]。数据挖掘技术和机器学习算法很大程度地依赖于特征工程，即把每个样本使用具有代表性、相对独立的若干属性以及属性值表示，才能被机器学习算法所识别。特征工程对于经验与感觉的要求很苛刻：特征与特征之间应尽可能相对独立，离散值与连续值应根据不同算法区别处理，异常值、缺失值对模型的影响程度也不尽相同。特征工程同挖掘算法处于一种强耦合状态。使用机器学习算法处理代码，便同样存在上述问题。随着以神经网络为核心的深度学习技术的发展，机器学习方式也变得更加端到端。传统机器学习算法具有较好的可解释性，而深度学习算法则降低其可解释性，却能大幅减少特征工程的影响，从而提升算法的性能。

在软件工程挖掘领域，同样也遵循着这种转变过程。在深度学习兴起之前，大部分研究通过构建包括 Ohe-Hot、词频、词性、n 元共现信息、主题等一系列特征，实现缺陷报告挖掘。直至 Word2Vec[30] 出现，文本领域便纷纷转向神经网络语言模型（Neural Network Language Model, NNLM），用低维稠密向量替代高维稀疏向量，带来了突破性变革。在此之后，NNLM 愈发走向成熟，循环神经网络（Recurrent Neural Network, RNN）、文本卷积（TextCNN）[23]、ELMo[36]、GPT1(/2)[38, 39]、Xformer[3]、Berts[4]等使得深度文本挖掘逐步深入。将代码理解

[3]指以 Transformer[54] 为基础结构的变体

[4]指以 Bert 为基础的各类变体，如 Albert[25]、RoBerta[28] 等

同深度学习结合起来，也逐渐成为研究趋势之一，借助深度学习强大的特征处理能力，能够避免代码理解所需的特征工程，而是自动化地从代码中获取所需特征。

综上所述，源代码知识在自动化软件工程任务中发挥着重要作用，是实现自动化软工的不可或缺的一部分。基于数据挖掘和深度学习算法的背景，本文提出从深度学习的角度，对源代码进行理解，探索源代码到自然语言的映射建模方法。可以明确的是，源代码虽然是由自然语言片段所构成的序列，但其具有更加严格的语法、句法、控制流限制，因此对源代码的理解不仅需要考虑字面文本，还需对其内部蕴含的结构理解，方可实现准确建模。

1.2　研究现状

代码理解通常是众多软件工程任务中的重要一环，如缺陷报告的处理，一般集中于重复报告识别、优先级预测、开发者推荐等任务。在处理缺陷报告挖掘时，通常将其视为文本建模任务，采用机器学习算法实现具体目标。Tian 等使用缺陷报告的标题和详情文本，使用词袋模型衡量文本间的相似度，作为评价缺陷报告相似性的特征之一，实现重复报告识别。类似的，Sun 等采用 TF-IDF 表示文档信息，再使用支持向量机（Support Vector Machine, SVM）判别模型识别重复报告。已有的方法，均旨在针对已有的缺陷报告通过特征工程，获得该报告的表示，从而能够借助算法实现判别。但是缺陷报告中不仅存在缺陷的文本描述，还附带了同缺陷相关的代码片段，现有的研究中却通常忽略该部分数据，导致无法完整地提取到缺陷报告中的信息 [48, 52]。

随着机器学习技术的发展，越来越多的研究开始将深度学习方法应用到软件工程领域中。同样地，在这些应用中，现有的研究仍然忽略了代码理解的过程。Ramay 等同样将缺陷报告视为由文本构成的样本，利用自然语言处理 (NLP) 技术获得缺陷报告中的文本向量，借助分类模型实现了缺陷优先级预测。在开发者推荐领域，Guo 等通过使用 NLP 技术学习缺陷报告中的文本特征，并使用卷积神经网络（CNN）获取到整篇报告的特征，识别出相似的报告来实现开发者推荐。从已有的研究中可以发现，包括重复报告识别、优先级预测、开发者推荐等任务，从根本上来讲都是通过相似的报告从而推断出所需的目标。实际上，在开发者推荐任务中，同开发者关联最紧密的应是由其编写的源代码。但已有的研究仍然只是从缺陷报告中的文本描述角度出发，将缺陷信息同开发者关联起来，导致了开发者特征抽取的不足。

从自动化软工任务的视角分析，大部分的任务都可以使用相似性度量解决。

如开发者推荐任务中，可以认为，相似性高的代码可以推荐给同一个开发者修复。同样地，在代码检索任务中，相似性度量也必不可少。通过评估文本同代码的相似性，可以检索到同文本高度匹配的代码片段。相似性实现的方式有多种，如Sun 等使用 SVM 判别方法来实现重复识别，但 SVM 核心也是拟合一个超平面，使得正负样本同分割超平面的距离和足够远，然而距离在某种程度意义上也可用于表示两个样本间的相似度，即距离大的样本其更不相似，反之相似。于世英等在做相似代码检测时，提出将代码视为序列片段，并使用权重分配的方式，最终将代码按照相似度聚类。

在大数据领域，一个重要的场景是信息检索 [68]，鉴于代码数据的量级，代码检索任务同样价值重大。对文本检索而言，传统的方法通常通过词的 n 元匹配实现，或通过统计方法，将文本视为由词组成的文档，并统计词的共现频率，作为对文档的理解。另外，主题模型也从概率的角度完成了由离散形式的字符到概率模型的映射，使其能够使用无监督方式获得到文档所蕴含的主要含义。代码同样可视为一种序列数据 [66]，但除序列特征外，代码还包含有严格的语法特征，因此文本检索的方法无法直接应用到代码检索任务中。所有的统计模型均难以突破上下文语义的瓶颈，统计模型武断地摒弃了字与词的先后顺序，限制了其对实际场景的建模，导致仅凭传统统计方法的搜索引擎在处理复杂检索任务时捉襟见肘。该不足也极大地制约了代码检索的构建，代码序列间的顺序关系体现了其严格的语法特性，基于文本的处理方法必然导致语法信息的缺失。

从宏观上讲，搜索场景主要包含：内容理解、内容召回、内容排序（粗排、精排）等三个阶段，代码检索也可大致分为上述阶段。在软件维护过程中，开发者时常需要从海量的数据中准确地搜索到指定的内容，如根据文本搜索到代码示例，这体现了代码检索的重要作用。Chaparro 等也尝试将检索应用在缺陷定位场景，利用缺陷报告中的文本信息实现检索，但由于缺陷定位在于找到代码中问题的具体位置，使用文本检索的方法并不能将缺陷同具体代码构成强关联。因此Liang 等通过结合源代码与缺陷报告中的文本信息，使用深度学习方法实现了更精准的缺陷定位。为适应代码检索的需要，需要能够准确地理解代码每一处的内容。向量召回技术是检索任务中常用的技术之一，在代码检索任务中也可采用该方案实现。将代码使用深度学习方法，理解为包含代码语义、语法和模式等的知识向量，再结合使用相似性度量算法，召回相关代码。Jiang 等研究指出，代码理解仍有很大提升空间。在源代码知识的巨大背景和代码理解方法不佳 [21] 的趋势下，本文进一步研究了更高效的代码理解方法。

鉴于传统的文本方法大多忽略了词序特性，且纯文本信息无法准确建模源代码信息，本文提出了一种基于抽象语法树（Abstract Syntax Tree，AST）的代

码理解方法，并从相似度量的角度实现代码与文本间的跨模态度量。该方法实现了软件工程领域研究从文本粒度到代码粒度的下沉，并且实现了两者的相似性度量，能够更加友好地为以相似度为基础的软件工程领域任务提供增益。不仅如此，在检索场景下，源码理解实现了宏观搜索场景下的内容理解环节，能够为后续的内容召回提供一种基于向量召回的解决方案。

1.3　论文组织与主要贡献

本文采用的组织结构如下，第 一 章主要分析了当前软件工程的发展现状，受数据挖掘的启发，智能软件工程也具有重大价值，并且存在许多挑战。

第 二 章根据智能软件工程的动机，介绍了从语法、语义等层次理解代码以及具体应用任务的相关技术，根据现有方法的不足，确立本文采用的技术路线。

确立好所用的技术后，第 三 章详细讨论了本文在代码理解上面临的问题，定义了相关的任务和目标，根据具体的任务和代码形态详细阐述了语法、语义等特征提取模型。

根据模型结构和数据规格，第 四 章则详细介绍了实现该实验过程中的各个细节，涉及超参数设定、训练方法等，并对得出的实验结果、实验数据和样本案例，结合已有方法中的问题，具体分析了本文方法的效果。

第 五 章总结了本文的研究方案和研究成果，进一步说明了本文研究仍存在的不足，指出了后续工作的大致方向。

综合本文的研究，可以总结出本文的贡献如下：

(1) 提出代码子序列编码（Subtoken Encoder）方法，来处理代码对应 AST 的叶子节点，获取更加准确的代码语义特征；

(2) 提出基于 RNN 的路径编码器更好地建模时序依赖，处理代码对应 AST 的非叶子节点，更好地保留和建模 AST 中的文法信息，获取更加准确的代码语法特征；

(3) 提出自注意力机制实现源代码特征动态融合，抑制了过度的噪声，加快了模型计算速度；

(4) 提出基于相似性度量算法的语义匹配结构，构建了文本与代码表征间的相似性评估方法，实现了在自然语言空间内更好地理解代码含义的目标。

1.4　本章小结

本章主要简单介绍了当前大数据时代背景下的数据挖掘目的以及方法，数据挖掘目的在于从海量的数据中发现有效的知识和模式，完成对历史数据的归纳。结合软件工程领域的任务，可以发现现有的研究尚未充分利用代码中的知识，但代码理解在众多任务上具有巨大价值。在应用层面，本章认为从海量代码中得到并快速评估其语义信息能够支撑各项任务。综合而言，本章主要讨论现有的研究技术和领域，确立了本文将代码理解为一段固定维度的向量的研究目标，并且提出将代码知识向量映射到文本空间中对齐理解的主要研究方案，来支持自动化软件工程中的各项任务。

第二章　相关工作

2.1　深度语义提取模型

源代码本质上是一种特殊的序列数据，具有同文本相似的时序性，同样代码的书写、变量的命名等通常暗含了代码的功能含义，具有语义性，可使用语言模型处理。在传统机器学习工作中，对自然语言的处理通常使用 One-Hot 编码的形式表示词，但是 One-Hot 编码向量维度会随着词典的大小呈线性增长，且稀疏性过大，进而导致维度灾难。在此基础上，便涌现出了基于词频统计的词袋模型、TF-IDF[41]、LSI[19]、LDA[6] 等方法，且都可直接在无监督任务上构建[37, 64]。已有的统计方法都忽略了文本中的词序、语序等信息，导致其无法准确捕捉文本的语义。为了实现文本的低维表示，Mikolov 等提出使用神经网络方法，获得关于词上下文的表达。Word2Vec[30] 使用 CBoW 与 Skip-Gram 两种方法对词的表达进行训练，CBoW 在于使用一定窗口内的上下文单词预测中心词，而 Skip-Gram 则相反，使用一个词作为输入，预测其在一定窗口大小内的上下文单词。词袋模型无法很好地考虑上下文信息在于其将每篇文档视为独立存在，并忽略了共现特征。为此，Pennington 等设计了基于全局共现特征的词向量表示方法 GloVe。然而，虽然 Word2Vec 与 GloVe 方法均解决了词的上下文表示问题，却仍然无法实现歧义区分，即含有不同意义的相同词仅能使用一个向量表示[1]。

自然语言处理（Natural Language Processing, NLP）对应着诸多文本相关的任务，如文本分类、情感分析等。不同于传统机器学习，深度学习领域通常使用全连接神经网络 (Full Connected Network, FCN) 解决分类问题。全连接神经网络对向量进行线性或非线性变换，从而实现特征空间变换以及非线性拟合。为实现代码语义层面的提取，需要使用部分自然语言处理方法，获取代码文本层次的语义特征，但上述的方法均存在如歧义性问题、上下文关联等特征缺失的不足，因此需采用更准确的语义提取模型。

为了能够从代码序列中获得语义和上下文特征，可从自然语言模型的角度，深入讨论语义抽取方法。Word2Vec 与 GloVe 之所以产生歧义，是由于其利用语料中的上下文关系建模，因而词向量更偏向于更常见的模式，即 n-gram 特征或共现特征。为避免使用时的歧义，最好的方式在于加入上下文信息对词向量进行修正。对词向量求均值（平均池化，Mean Pooling）[35] 以及按权重求和（和

[1]如 “Apple Inc.” 与 “Apple tree” 中的 “Apple” 仅有一个表示。

池化，Weighted Sum Pooling）[3] 便可在一定程度上修正词向量的歧义。但无论是平均池化还是和池化，仍然忽略了上下文的词序关系：“Tom is chasing Jerry”与 “Jerry is chasing Tom”，语义相反但最终池化结果相同，仍然导致歧义。造成该问题的原因在于，池化虽然能够考虑上下文，但是忽略了位置关系。循环神经网络（Recurrent Neural Network，RNN）[13] 按照顺序依次处理每个词，天然地将位置信息融入到序列理解的过程中。在 RNN 中，每个词按照其自然语序依次输入到网络中，并针对每个时刻的输入都产生一个输出，表示当前输入同之前所有上下文语境构成的语义信息。然而，这并不能解决所有问题，如对于含有相同起始语境的语句：“I had lunch.” 与 “I had recieved.”，对于 “had” 而言，虽然起始上下文一致，但是语义完全不同。为解决该问题，可在处理序列时，同时理解前向与后向的上下文环境，即使用双向循环神经网络（Bidirectional Recurrent Neural Network，BiRNN）[44]。与 RNN 不同的是，BiRNN 在编码过程中，对序列进行正向、逆向编码，分别考虑了历史与未来的语义环境，使得词的语义理解更加准确。RNN 通过对词的位置与上下文特征的捕捉，解决了词向量的歧义问题，但是其存在并行计算效率低的问题。基于卷积神经网络 (Convolutional Neural Network, CNN) 的模型能够解决该问题。Kim提出使用 1 维 CNN 处理词向量构成的文档矩阵，并得到文档的语义向量，用于对文档分类。普通 RNN 存在短期记忆的问题，并且容易产生梯度消失，难以优化求解。Zaremba 等通过引入门控机制，提出了 LSTM 网络结构，解决了 RNN 的短期记忆，使得循环神经网络能够更好地处理长序列。在机器翻译 (Neural Machine Translation, NMT) 任务中，Cho 等提出了 GRU 结构，进一步简化了 LSTM 结构，在序列建模上取得了成功。为了增强模型对源码中时序性特征的建模，本文选择 RNN 网络体系中的 GRU 模型，该方法能够解决短期依赖问题并且具有更小的计算量。

基于深度语义模型，可以在代码理解过程中充分提取到语义层面的特征，该过程同获取文本语义相似。使用深度语义模型，可以在提取代码语义时解决词向量等方法的歧义和上下文缺失问题。

2.2　代码语法理解模型

语法特征是源代码数据同文本数据的显著区别之一，准确的代码理解不仅应提取到语义特征，还应同时获取语法结构特征。由于源代码可以由不同的程序语言编写，且不同程序语言存在不同的语法要求，在获取代码语法特征时，通常将代码转化为语言无关的其他形态再进行分析处理。现有的研究通常从模型结构的角度出发，使模型结构能够自然地适配代码对应的中间形态数据规格。本

节详细地分析了无中间形态、树形态和图形态三种形式的研究工作，并具体分析了其中的问题和挑战。

2.2.1　基于纯文本的代码理解

在软件工程领域，对代码的分析与理解能够借助海量项目中的源代码，获取到大量的知识，并用于广泛的任务中。像代码克隆检测、代码检索、代码推荐等任务，都需要借助代码理解来完成。在上述任务中，均可基于一个假设，即相似的代码能够表示为相似的一段向量，从而使用该向量完成这些任务。在 NLP 领域，通常使用 Word2Vec 方法 [30] 将词表示为低维稠密的向量，表示其语义信息。类似地，在计算机视觉领域则有 VGG[47] 等方法将图片表示为向量，作为其特征。而在软件工程领域，越来越多的方法也相继被提出，用于将源代码表示为一个固定维度的向量。由于源代码由一系列的文本序列所构成，因此可以将 NLP 领域中的方法用于处理源代码文档。Allamanis 和 Sutton使用 n-gram 语言模型实现了代码提示任务，并且Bavishi 等统计源代码中的词汇信息作为上下文，并使用自编码器 (Auto-Encoder, AE) 与 RNN 模型实现参数名称预测。Kamiya 等, Sajnani 等也都将源代码视为文本序列用于表征代码，并将该表征用于克隆检测任务中。由于 Bert 在 NLP 任务上的惊人效果，Feng 等使用 Bert 的预训练模型，在代码文本上进行预训练，获得了不俗的效果。基于文本序列的源代码理解方法，仅从序列顺序上间接地提取了代码的语法信息，忽略了代码的强语法特性。虽然其能够从字面层次对代码内容进行解析获取语义，但源代码与文本文档的区别在于，代码是一种具有严格语法规范的文本序列，其语法特征比文本更强，使用文本的处理方式直接应用于代码并不适用。传统的基于词序列的方法，粗暴地忽略了代码中包含的结构信息，从而降低了代码理解的准确性。

2.2.2　基于树结构的代码理解

为了能理解代码中的语法信息，Neamtiu 等将源码抽取成为抽象语法树 (Abstract Syntax Tree，AST)，用于源码理解。AST 能够将一段代码，按照其语言语法规则，表示为一棵具有层次结构的树。Mou 等, Tai 等提出的树结构 RNN 和树结构 CNN 进一步将神经网络结构的处理能力由线性数据结构扩展到了符合树结构的数据场景。AST 是一种典型的树结构数据，使用树结构神经网络，能够由底至上处理 AST 树中的各个节点，并得到树的特征表示。Sun 等结合 Tree-CNN 来提取 AST 的结构信息，作为语法层面的特征。但是树 RNN 与树 CNN 都受限于树的深度与宽度，即当树过于宽时，模型并行化能力大大降低，而当树过深

时，会导致严重的梯度消失。AST 路径节点间的依赖体现了强时序性，因此本文采用 RNN 模型对 AST 的非叶子节点进行编码，提取代码中的语法信息。相较于 Tree-RNN，普通的 RNN 更能够利用 GPU 的并行计算优势，且更能避免梯度消失问题。

2.2.3　基于图结构的代码理解

Kipf 和 Welling提出图神经网络，将卷积运算推广至图结构数据场景。除 AST 外，代码还可以被转化为控制流图（Control-Flow Graph, CFG）的形态，用于实现代码结构的理解。不同的是，控制流图表示了源代码对应的控制流程，在该结构中可能含有环结构，如循环结构和递归结构等。Scarselli 等提出的图神经网络，在基于 CFG 的模型中也被广泛采用，Tufano 等便使用图嵌入方法 (Graph Embedding Strategy) 对 Java 字节码对应的 CFG 进行编码，用于评估代码间的相似性。但是图神经网络受限于采样策略，且对于时序性较强的结构会造成信息缺失。由于图神经网络具有较高的复杂度，因而在处理代码的 CFG 时，还需要保证代码 CFG 具有较高的稠密性，否则稀疏图将导致模型严重地过拟合。事实上，代码由于其规模的限制，并不能构成复杂的图网络，故图神经网络不适用。

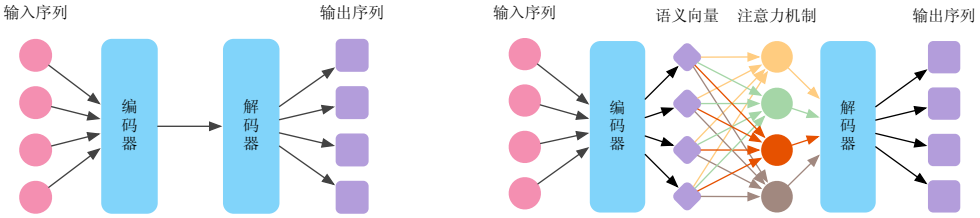
2.3　文本生成模型

代码知识可以应用于多个场景，根据代码内容预测对应的方法名称便是其中之一。该场景是实现代码理解后，将代码知识转化为文本的过程。方法名描述了代码逻辑的客观含义，因此该任务可以用于评估代码理解的效果，且生成文本可帮助更好地阅读和理解代码。该任务为生成问题，故需采用生成模型实现。

2.3.1　序列到序列生成

序列到序列（Sequence to Sequence, Seq2Seq）结构也被称为编码器-解码器（Encoder-Decoder）结构。该结构将网络分为两个子网络：编码器网络和解码器网络。编码器网络用于将输入的序列，编码解析成一个固定的语义向量，而解码器网络则相反根据编码器给定的语义向量，从中解码出目标序列。在该结构下，编码器与解码器可以是任何形式的网络结构，如图2.1(a)所示。机器翻译通常采用 Encoder-Decoder 结构 [11]，该结构使用一个网络 $f(\cdot)$ 作为编码器，将源序列 $x_1, x_2 \cdots, x_m$ 编码为一个语义向量 $\boldsymbol{c}$，并使用 $g(\cdot)$ 作为解码器，将语义向量解码为目标序列 $y_1, y_2, \cdots, y_n$。**该结构说明若将源代码视为一段源序列，其也应该能**

够被编码为一个语义向量，而代码理解的过程即是搜寻一个最佳 $f^*(\cdot)$ 的过程。Vinyals 等所提出的方法，也可视为一个 Seq2Seq 结构，其编码器为基于 CNN 的



(a) Seq2Seq 结构　　(b) Seq2Seq-Attention 结构

图 2.1: Seq2Seq 与 Attention 的结构与差异

预训练模型，作为图像编码器。而解码器为一个基于 RNN 结构的生成网络，用于从图像编码中，获得文本描述。Seq2Seq 结构能够解决一系列生成问题，其目标为拟合一个目标函数：$\arg\max_{\theta} \prod_{t=1}^{n} p\left(y_t|y_{t-1}, y_{t-2}, y_1, x; \boldsymbol{\theta}\right)$，$x$ 表示编码器提供的输入，而 y_t 为解码器自回归时的每个时刻的结果，$\boldsymbol{\theta}$ 为需要学习的参数。

2.3.2　基于注意力生成

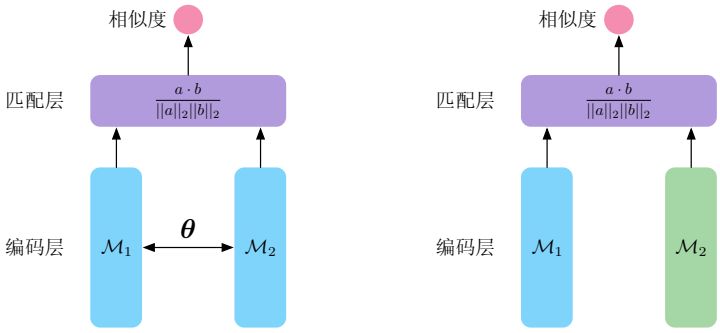
注意力机制在于，在编码器与解码器中间，添加一层注意力，如图2.1(b)所示，在编码器得到序列中每个词的语义向量后，注意力机制使用一个线性层，得到每个向量的权重，并对所有向量加权求和，得到解码器所需的上下文语义向量。解码器的每一次解码，都将进行该过程。在神经网络机器翻译任务中，注意力机制被证明能有效提高翻译水平 [4]。不仅如此，在 CV 任务中也使用通道注意力和空间注意力机制 [57] 达到了更好的图片处理效果。注意力机制核心思想在于，在解码的每个时刻，只关注 $\boldsymbol{c}$ 中的某些局部信息。每一次解码，都抽取语义编码的一个子集 $\boldsymbol{c}_t$ 作为编码器的输入。除此之外，Luong 等进一步扩展注意力机制，提出全局注意力机制 (Global Attention Mechanism) 与局部注意力 (Local Attention Mechanism) 机制，进一步提升了注意力机制在机器翻译任务上的表现。自从注意力机制在 NMT 上取得成功，注意力机制已经成为包括情感分析、阅读理解、序列标注、文本分类等任务在内的基本方法 [8, 59, 60, 65]。随着 NLP 技术的发展，基于自注意力机制的 Transformer 网络 [54] 凭借强大的特征表达能力而逐渐成为主流，本文基于自注意力机制提出了一种全新的路径融合方法。本文借助注意力机制的强大表现能力，将普通注意力机制、全局注意力机制、自注意力机制应用于代码理解任务，使模型能更多地关注重要的片段，从而提升代码理解的表现。本文基于注意力机制提出了一种全新的路径融合方法，用于代

码理解任务中，显著提升了代码理解的效果。

2.4　语义对比模型

考虑向量召回场景，可将代码理解得到的向量同文本向量执行语义匹配任务。当文本描述的功能和代码的客观含义一致，二者的向量的相似指标应该为相似。在检索任务中，亦可将文本向量作为 Query，于海量源码中找到符合的数据。语义对比模型能够处理匹配问题，同样也可处理代码-文本的语义匹配任务。

2.4.1　孪生网络相似性学习



(a) 孪生神经网络结构　　(b) 深度语义匹配模型结构

图 2.2: 孪生与伪孪生网络结构

孪生神经网络（Siamese Neural Network）最初被用于人脸验证 [12] 任务中，对任意两张图片经过同一个 “单体网络”$\mathcal{M}$，得到各自的特征向量，并使用余弦相似度作为两张图片是否匹配的衡量标准。同一个人脸的两张图片，其特征向量的余弦相似度应接近于 1，相反则接近于 −1。同 Seq2Seq 结构一样，孪生神经网络也是匹配任务中的一种通用框架。孪生神经网络结构通常一次性输入两个样本，使用共享参数的 $\mathcal{M}$ 对两个样本编码，如图2.2(a)所示。作为一种通用型结构，孪生网络中的 $\mathcal{M}$ 可以根据任务替换，在人脸验证任务中通常使用基于 CNN 的模型，而在 NLP 任务中，则可以使用基于 RNN 等的模型。Neculoiu 等便使用 LSTM 作为 $\mathcal{M}$，用于学习文本之间的相似性。孪生网络结构使用简单的向量空间模型（Vector Space Model，VSM），在相对低维向量上做点积运算，能够加快运算速度，并且将上游特征编码与下游相似性评估两个阶段解耦，在工程实践上具有更高的灵活性。基于该特性，可以将预训练模型与孪生网络结构结合，根据任务将 $\mathcal{M}$ 替换为对应的预训练模型继续微调即可。

2.4.2　双塔网络相似性学习

在孪生网络结构中，$\mathcal{M}$ 对两个样本权重共享，当两个样本不属于同类样本时，共享权重则无法适用。伪孪生网络沿用了孪生网络的双塔式结构，不同的是，对输入的两个样本分别使用 $\mathcal{M}_1$ 与 $\mathcal{M}_2$ 分别编码，且 $\mathcal{M}_1$ 与 $\mathcal{M}_2$ 不共享参数。深度语义匹配模型 [20]（Deep Structured Semantic Models，DSSM）是一种语义匹配任务的经典双塔结构，如图2.2(b)所示。DSSM 结构与伪孪生网络，都是经典的双塔式结构，该结构可以很好地用于两个相同或不同模态的样本间相似度学习。与孪生网络一样，双塔式结构可以根据实际任务和输入样本，替换上游的每一个分支网络，Palangi 等, Shen 等基于 CNN 与 LSTM 分支网络提出的 CNN-DSSM 与 LSTM-DSSM 方法都被用于语义理解任务中。本文认为，对源代码理解的过程即是将源代码理解为 “功能描述” 的过程。显然源代码属于具有强语法规则的文本序列，而功能描述则为自然语言序列，基于 DSSM 在语义学习上的成功，本文选择 DSSM 网络结构并结合伪孪生神经网络中的思想及训练技巧，提出了代码与文本的相似性学习方法。

2.5　本章小结

本章总结了文本、软工领域在理解任务上的应用与差异，基于这些任务对涉及的网络结构模型进行分析。从分析讨论中可知，若单纯地将源代码视为纯文本文档，将丢失语法结构的信息。基于该点提出了两个研究任务：1) 生成任务，用于根据代码生成方法名；2) 语义匹配任务，用于评估代码和注释文档之间的语义相似度。本文充分利用源代码自身含有的数据特征，包括方法名称和文档注释，构建了以 Seq2Seq 为基础骨架的方法名预测任务和基于 DSSM 双塔网络的文本与代码语义匹配任务。两个任务均需要使用一个网络获取代码的向量表示，称为 “代码编码器”。受注意力机制在 CV 与 NMT 上的启发，本文将注意力机制用于 AST 路径代码理解方法，实现了高效、准确的代码理解。

第三章　深度代码理解技术

3.1　代码理解定义

代码理解在软件工程知识挖掘中发挥着重要作用，在代码克隆检测、代码检索和代码推荐等诸多场景中都有重要作用。综合而言，上述任务均可归纳至一个统一的过程：将代码中的语法和语义信息表示为一个低维向量。得益于迁移学习思想在深度学习领域中取得的成功，使用该向量结合分类、回归和聚类等算法，可完成大量软工任务。本文聚焦于方法名生成和代码语义匹配两个任务，旨在得到准确高效的代码向量，用于广泛的下游任务。同 NLP 中的语言模型目标相似的是，相似的代码其向量表示应该在同一语义空间内具有更大的相似度，这为检索、推荐初期的召回提供了依据。使用代码向量，获得准确的功能描述的自然文本，说明代码向量已经准确包含了代码中的功能信息。本文的技术目标在于将代码表示为一个准确包含功能信息的低维向量，从该向量中获取方法名称，并令其与注释文档的语义保持较高的一致性。

源代码的主要表现形式为文本串，根据现有的技术研究分析，若对代码文本采用 NLP 方法处理，将需要繁重的数据清洗过程，保证代码文本的质量，但仍然无法全面地获取代码信息。AST 可视为一种介于源代码与二进制码的中间语言无关形态，比文本更具结构性且噪声更低，较二进制码具有更高的可读性。本文针对源代码的 AST 形态进行处理，拟从固定的模式和结构中，准确地提取其中的语法和语义信息。

AST 是一种多叉树的结构，源代码中的每一条语句都可被解析为一棵由根节点构成的子树。图3.1为计算 $sum = \sum_{i=a}^{b} i$ 对应的代码与 AST，每一条语句都可以从 AST 中找到一棵子树表示。其中，／(①)表示方法的参数列表，／(②)表示初始化累加变量 sum 值为 0，／(③)表示循环语句中循环初始条件的定义，／(④)表示循环终止条件的判定，／(⑤)表示累加运算，／(⑥)表示返回语句。基于这个特性，对 AST 的理解则被转化为对上述子树的理解。代码理解的目标在于将图3.1中的 AST 结构，表示为一段低维语义向量。

对于 AST 结构，树结构的神经网络模型 [51] 能够自底向上地将每个子树进行合并，得到最终的树编码。树结构网络性能受限于树的规模，当树的深度过深时，自底向上合并的操作越多，模型被串行操作限制。为增强模型的并行性，本文采用纯 RNN 结构处理 AST，无需再沿着树的深度合并子树。为适应 RNN

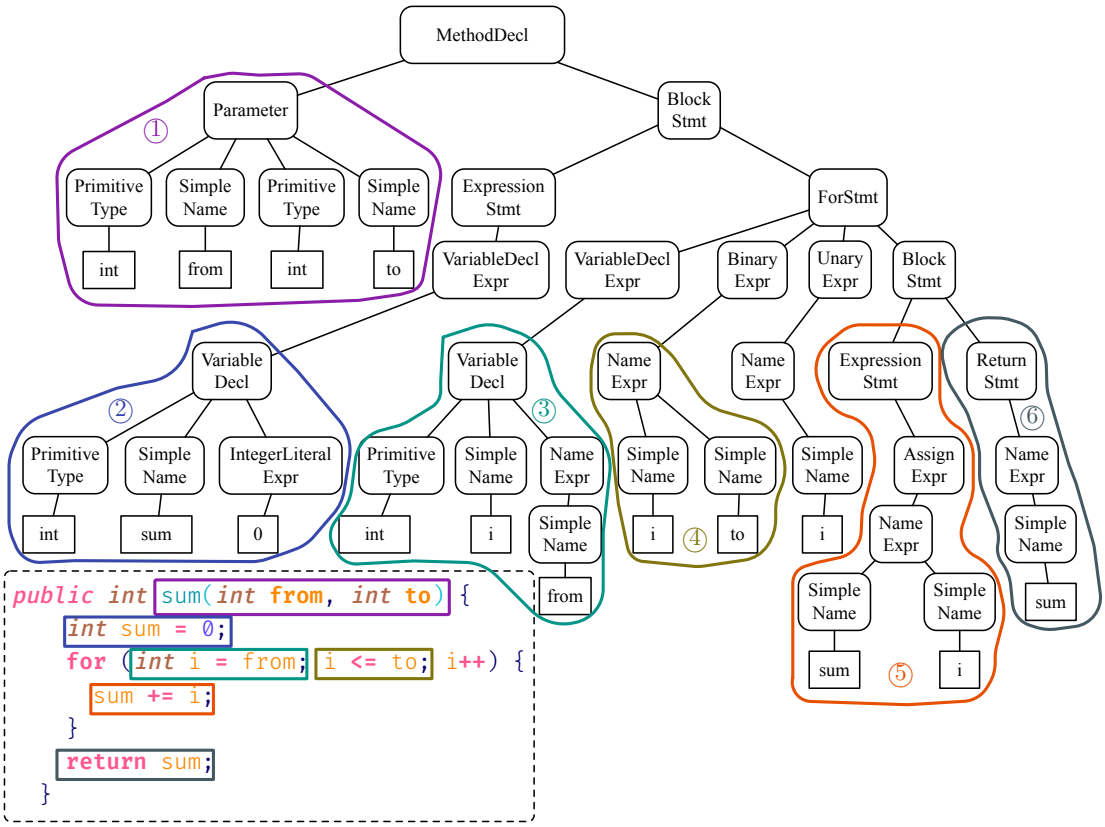


图 3.1: 源代码与对应 AST 路径（部分）组合

的序列建模方式，本文将子树以根节点为中心构成路径对，以多个路径对表示子树中的所有路径。如图3.1中子树，其有效路径有：1) `int` → `PrimitiveType` → `Parameter` ← `SimpleName` ← `from`; 2) `int` → `PrimitiveType` → `Parameter` ← `SimpleName` ← `to`。这两条路径对应着方法的两个参数。据此可以推断，源代码中的程序语句可由 AST 中若干条有效路径对构成。编程语言本质上是在解决一类问题，并可以解释为 $b = f(p_1, p_2, \cdots, p_n)$ 的过程，而 $f(\cdot)$ 为方法的具体处理逻辑，而 p_n 表示所需的若干输入，b 为输出。分析 AST 树可知，p_n 通常为树上的叶子节点（终止符），而控制流则为非叶子节点（非终止符），由此可给出定义3.1与定义3.2。

定义 3.1. *AST 的子树结构可以由任意两个满足先序遍历顺序的两个叶子节点间的路径集合表示，描述为：*

$$T = \{Path_{s,e}\}, s \in L, e \in L$$

其中 L 为叶子节点集合，s, e 为 AST 满足先序遍历先后顺序的两个叶子节点，即 $ID(s) < ID(e) < N$，N 表示子树的宽度（叶子节点数），$ID(x) : x \to x$的索引。

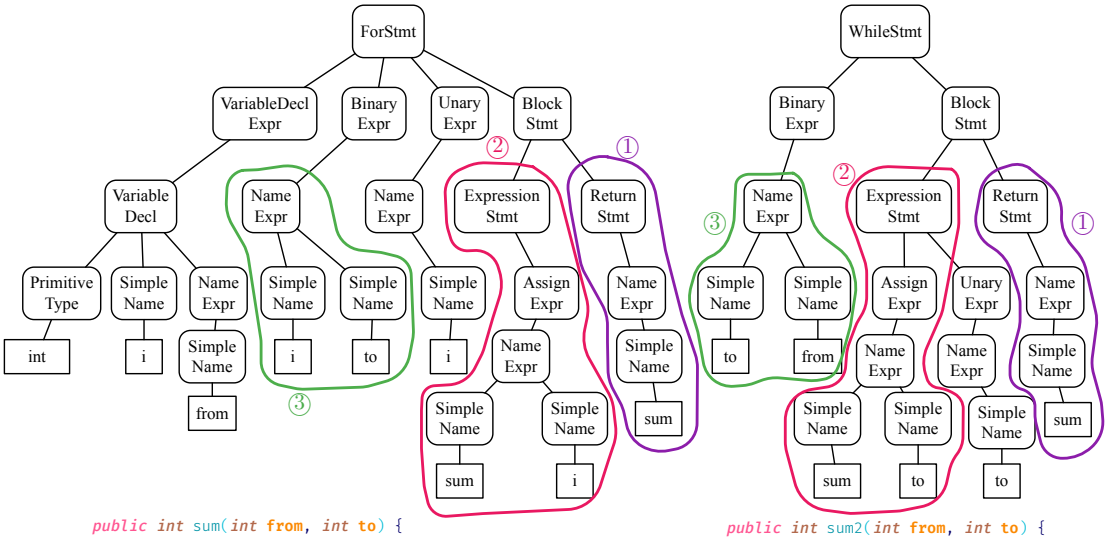
定义 3.2. *AST* 中的路径对 $Path_{s,e}$ 由左子树叶子节点 s 到公共根节点 p 和 p 到右子树叶子节点 e 途径的所有节点构成，描述为：

$$Path_{s,e} = \{s, t_1, t_2, \cdots, t_l, p, t_{l+1}, t_{l+2}, \cdots, t_r, e\}$$
$$t_k = \langle d_k, n_k \rangle,\ k \in \{m | 1 < m < l \vee 1 < m < r, m \in \mathbb{N}\} \wedge d_k \in \{\rightarrow, \leftarrow\}$$

其中每个中间节点为二元组，包含与上一个节点连接的方向 d_i 与途经的 *AST* 节点 n_i，l, r 分别表示节点所在左子树与右子树的深度。

定义3.2描述了 AST 中路径对的抽取规则，基于该规则得到树中的路径集合。代码理解即是理解路径集合，发现路径对中的模式与语义信息。在 AST 中，模式是源代码中语法信息的体现，语义对应着源代码中的功能业务。本文将使用 AST 中的路径对，实现从代码的语义与语法两个角度，获取准确的特征向量。

图3.2为同一个功能的两段不同实现代码及对应的 AST（部分）结构，如图3.2中①所表示的 “返回语句子树”，以及②所表示的 “累加语句子树”。从代码文本上看，两段代码不同，但从实现的功能上看，两段代码的目标一致。对比①、②、③中的结构可以看出，这两段代码核心的 AST 结构上具有一定的相似性。据此可以总结归纳出性质3.1，即相同的代码虽写法不同，但 AST 却具有相似结构。




```
public int sum(int from, int to) {
    int sum = 0;
    for (int i = from; i <= to; i++) {
      sum += i;
    }
    return sum;
  }
```

```
public int sum2(int from, int to) {
    int sum = 0;
    while (to >= from) {
      sum += to;
      to--;
    }
    return sum;
  }
```

图 3.2: 相似代码 AST（部分）对比

性质 3.1. 相似性：相同的代码必存在相同或相似的 *AST* 子结构，以 T_1, T_2 表示两段代码的 *AST*，则可描述为：

$$T_1 \cap T_2 \to T_1 \wedge T_1 \cap T_2 \to T_2.$$

相反，若两棵 *AST* 具有相同或相似的子结构，那么这两段代码存在一定的相似性，可描述为：

$$T_1 \cap T_2 \to T_1 \wedge T_1 \cap T_2 \to T_2 \Rightarrow \exists\ x_1 = f(T_1), x_2 = f(T_2) \text{ 使得 } \|x_1 - x_2\| \to 0.$$

x_1, x_2 表示两段代码，若 x_1, x_2 相似则两段代码的范数趋近于 *0(同 word2vec[30] 结论)*.

根据性质 3.1的相似性质，论文的目标在于寻找满足该性质约束的最优编码器 $f(\cdot)$，且 $f(\cdot) \in \mathbb{R}$。

3.2　源代码构成与特性

一份完整的 Java 源代码通常由：1) 环境依赖语句，即`import`语句；2) 声明语句，即类声明；3) 方法列表，即实例与静态方法等。但值得注意的是，`import`之间会随着深度而不断迭代，即一个项目内的各个`import`语句，会链式地`import`直到 JDK。而类可能还有匿名内部类、内部类，且类方法之间会存在相互调用。在方法内部，除去方法的调用栈外，则还有基本控制语句，即循环、条件等。

经过对 Java 源码的解析，将会产生如下的部分：1) `PackageDeclaration`，类文件的包声明，按照包的形式组织，长度不一；2) `ImportDeclaration`，该类所依赖的其他各个类，由并列的若干个包及类构成；3) `ClassOrInterfaceDeclaration`，类声明，代表某个类或其匿名内部类的声明；4) `FieldDeclaration`，类的属性声明，用于描述一个类所含的属性；5) `MethodDeclaration`，某个类所含有的方法的根节点，类的每个方法都以此类型作为根节点。在一个方法中，根据 Java 编程规范的约定：单一职责原则，特定的方法中仅含有单一任务的实现，这为代码特征到文本空间的投影提供了事实依据。

为了能够准确地得到代码片段的特征，需要首先对代码进行 AST 解析，从而将离散的程序语言，转换为具有特定流程及结构的序列 [70]。同时，方法在执行过程中，不可避免地会需要声明各类变量、常量。而高质量的代码，其变量命名将尽力保留变量的用途含义，因此变量名将有可能包含能够说明代码功能或语义的信息。在数据处理中，将代码中所有的变量名，进行分词并得到 *subtokens*。

树的先序遍历能够符合程序执行的顺序关系，用该遍历算法处理 AST 语法树，可在保留语句执行顺序的前提下，得到整个方法所执行的各个子任务。为此，基于 [2] 的思路，对 AST 的路径进行单独编码。

本文三节中提出从语法与语义两个角度理解源代码。从图3.1中可以看出，叶子节点一般是特定情况下的值（常量或各类名称等），而非叶子节点为某一个固定的 AST 节点。

对现有研究 [2] 的分析，可以总结出以下问题：

(A) 在处理代码的 AST 时，叶子节点没有额外处理，导致提取语义时词典爆炸，且对低频词不友好；

(B) 全连接神经网络没有充分考虑代码 AST 节点间的时序依赖特性，导致无法充分提取代码语法特征；

(C) 处理代码结构时，静态地将 AST 的任意叶子节点两两组合，导致大量路径对，同时引入大量噪声，导致模型无法正确拟合有效数据。

本文将使用两个任务来验证方法的有效性，首先使用一种 Seq2Seq 结构构建了一个方法名预测模型，其次构建了一个双塔式的向量召回模型，实现了文本与代码向量的相似度评估。从任务上看，我们均实现了从代码语义空间到文本语义空间的转换，能够实现对代码片段的语义理解，并且能够从自然语言的角度对其语义进行验证。

3.3　源码 AST 编码技术

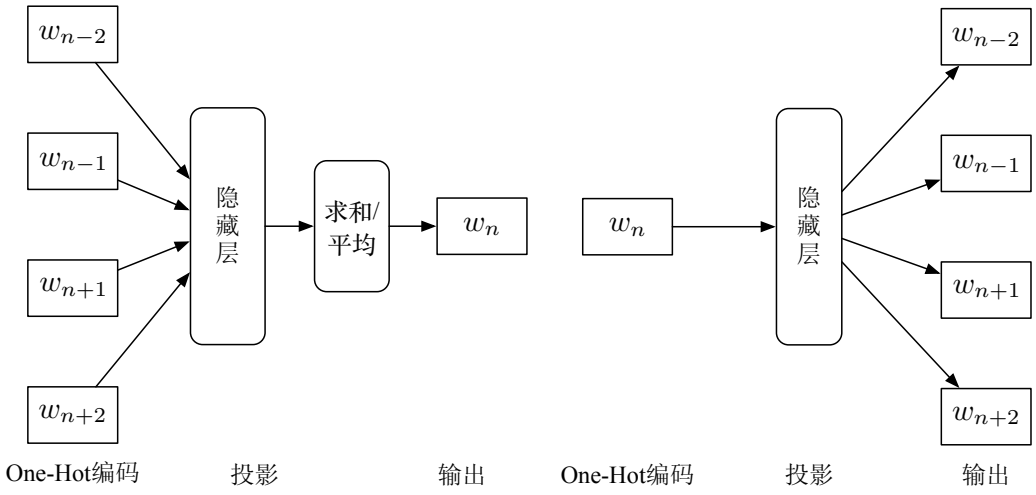
3.3.1　代码子序列编码器

(1) 预训练代码域词向量　基于 AST 叶子节点都是有意义的文本这一特性，本文提出子序列文本编码器用于编码叶子节点中的 “词”，从而提取代码中的部分语义信息。使用子序列信息同时可以解决问题(A)，将一个长词拆分为若干个短单词构成的序列，如 “`findAnItemInList`” 拆分为 5 个高频词 “find an item in list”，一定程度地解决了低频词问题。本文基于超过 40 个开源项目的 API 文档，构建了代码理解任务的领域词典，包含了常用的软件工程领域的专有名词、缩写词等。使用 API 文档构建的领域词典，较开放语料得到的词典能获得更好的性能。使用既有的开放词典，通常会忽略软件工程领域中的特有词，如：jpa、ssl 等。

表 3.1: 代码子序列编码器符号对照表

符号	含义
w_i	子序列中各个片段 (Token)，i 表示位置
$Embedding(\cdot)$	嵌入层，将序列片段映射为向量
$softmax(\cdot)$	非线性激活函数：$\frac{\exp(x_i)}{\sum_{j=1}^{N}\exp(x_j)}$
r_t	GRU 单元中的重置门，t 表示时刻（序列位置）
$\widetilde{h}_t$	GRU 待更新的隐藏状态
i_t	GRU 中的更新门
h_t	GRU 中 t 时刻最终的隐藏状态
h	代码子序列编码器对文本序列编码的最终结果（若双向，则拼接）

词向量技术是 NLP 任务中必要的一步，这是文本映射到向量空间中的一种通用方法。本文采用 Word2Vec 模型，使用 gensim[1]在 API 文档上预训练获得词向量。Word2Vec 提供了 CBoW（如图 3.3(a) 所示）和 Skip-Gram（如图 3.3(b) 所示）两种训练词向量的方法，区别在于输入与输出的建模。CBoW 使用上下文词



(a) CBoW 模型　　(b) Skip-Gram 模型

图 3.3: Word2Vec 两种词向量模型结构

预测中间被掩盖的词，若词在整个语料出现的概率较低，则其对模型更新的贡献较低，导致 CBoW 对低频词不友好。相反，Skip-Gram 使用中间词预测上下文词，即便输入的词出现概率很低，但是通过该词的窗口内的上下文词，也可得

[1]https://radimrehurek.com/gensim/models/word2vec.html

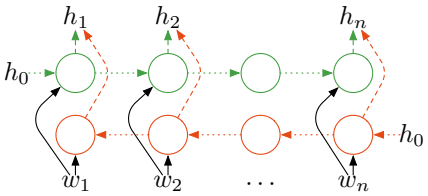


图 3.4: 双向 RNN 词编码器

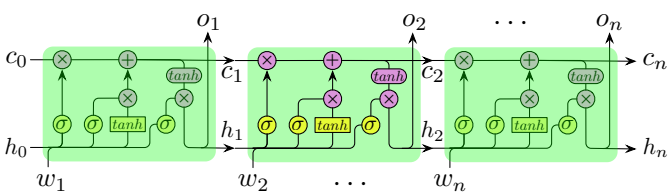


图 3.5: LSTM 编码器

到较好的表示。因此本文使用 Skip-Gram 模型，训练词向量，降低未登陆词的影响。将每个词用 w_i 表示，则 Skip-Gram 模型可以被表示为：

$$w_{i\pm n} = softmax\,(Embedding(w_i))$$

其中 $2n$ 表示上下文窗口大小，每个中心词 w_k，都取 $k \in [i-n, i+n)$ 作为上下文。

(2) 代码子序列拆分　在 Java 的编程规范中，对变量、类等的命名都遵循驼峰命名法，这使得定义 3.1 中的 s, e 都遵循该规则。为获取子序列，本文首先使用正则表达式将叶子节点的值拆解为若干个词的序列。

代码 3.1 正则表达式拆分子序列

```
import re
def convert_to_subtokens(text: str) ->  str:
  reg = '([A-Z]+)([A-Z])' # 专有名词
  text = re.sub(reg, r' \1 \2', text) # 拆解专有名词
  reg = '([^\\.])([A-Z][a-z]+)' # 按照驼峰命名拆解
  text = re.sub(reg, r'\1 \2 ', text)
  reg = '([-_\\?,:;\\$~\\(\\)\\{\\}!@#%\\^\\&\\*\\+<>])' # 按照标
      点符号拆解
  text = re.sub(reg, r' \1 ', text)
  return re.sub(' +', ' ', text)
```

代码 3.1 将叶子节点中的字符串按照专有名词的大写、驼峰命名规范和字面常量中的常用分隔符进行拆分，得到叶子节点的子序列。

(3) BiRNN 代码语义编码　为解决词向量的歧义问题，本文使用 BiRNN 来获取最终的语义向量，如图 3.4 所示。为降低模型复杂度，本文选择 GRU 作为 RNN 单元。根据预训练词向量，每个子序列可以获取一个词向量 $w_i \in \mathbb{R}^{d_w}$。GRU 结构同图 3.5 的结构相似，不同在于 GRU 的门控机制更加简单。鉴于每个叶子节点都是变量名称、类名或常量，将其按照代码 3.1 拆分成子序列后，仍只有较短的长度，本文取 RNN 最后一个时刻的隐藏状态 h_n 作为叶子节点的语义向量。

叶子节点的语义向量，采用同 RNN 处理普通文本相同的方法获得：

$$\begin{aligned} r_t &= \sigma\left(W_r w_t + V_r h_{t-1} + b_r\right) \\ \widetilde{h_t} &= \tanh\left(W w_t + r_t \odot V h_{t-1} + b\right) \\ i_t &= \sigma\left(W_i w_t + V_i h_{t-1} + b_i\right) \\ h_t &= \left(1 - i_t\right) \odot h_{t-1} + i_t \odot \widetilde{h_t} \end{aligned} \tag{3.1}$$

其中 $\sigma(\cdot)$ 为 sigmoid 激活函数，w_t 为每个时刻的词向量，r_t 表示重置门，i_t 表示更新门，$\odot$ 表示哈达玛积[2](Hadamard Product)，$W_* \in \mathbb{R}^{d_h \times d_w}, V_* \in \mathbb{R}^{d_h \times d_h}, b_* \in \mathbb{R}$ 表示 GRU 单元的待学习参数，而 d_h 表示 RNN 的隐藏状态维度，d_w 为输入向量的维度。GRU 的门控机制思想是通过重置门过滤上一时刻隐藏状态中的无效部分，得到当前时刻的隐藏状态，并通过更新门决定当前时刻与上一时刻的信息的影响程度。更新门实质上是一种应用于时间序列的残差连接思想。该策略可以有效地避免梯度消失，实现长期依赖。

公式 (3.1) 描述的是 GRU 单元中单个方向（仅正向或逆向）的运算过程，但本文采用了双向 RNN，因此每个方向都将得到一个隐藏状态。对于输入 w_n 的两个隐藏状态为 $\overrightarrow{h_n}, \overleftarrow{h_n}$。后续网络使用时，将两个方向的隐藏状态拼接作为完整的语义编码 $h = \left[\overrightarrow{h_n}; \overleftarrow{h_n}\right]$。从水平方向上看，RNN 随着序列长度扩展了模型的 “宽度”[3]，但 RNN 还可在 “宽度” 的垂直方向上堆叠多个 RNN 单元，称之为深度 RNN。Peters 等通过堆叠 RNN 单元提出了 ELMo 深度词向量模型，受此启发，本文在垂直方向上同样堆叠了一定的深度。考虑到模型整体的复杂程度，本文未采用Peters 等中拼接多层隐藏状态的做法，而是仅用最后一个输入的最后一层的隐藏状态 $h_n^{(n_l)} = \left[\overleftarrow{h}_n^{(n_l)}; \overrightarrow{h}_n^{(n_l)}\right]$，$n_l$ 表示代码子序列编码器的深度。

3.3.2　静态路径对编码器

根据定义 3.1 ，可以将语句表示为路径对的集合：

$$statement = \left\{ \left\langle s, Path_{s,e}, e \right\rangle \middle| \, s, e \in L \right\}$$

其中 L 表示叶子节点的集合，根据 3.3.1 节中的代码子序列编码器，每个叶子节点将根据规则拆解为子序列，即 $s = \{w_1, w_2, \cdots, w_l\}, e = \{w_1, w_2, \cdots, w_r\}$，$l(r)$ 表示左（右）子树叶子节点的子序列长度。根据公式 (3.1) 的过程，得到节点 s, e 的

[2]向量对应元素的乘积

[3]神经网络的宽度是指神经元的数量，这里的宽度是指 RNN 单元横向展开的广度

表 3.2: 静态路径对编码器符号对照表

符号	含义
statement	程序代码中的语句 (块)
h_s, h_e	代码子序列编码器的编码结果，表示起始和结束叶子节点的编码，对应表 3.1 中的 h
$Emb_t(\cdot), Emb_q(\cdot)$	表示两个编码层：节点编码与节点类型编码
e_{t_i}, e_{q_i}	节点与类型的嵌入值，$t_i(q_i)$ 为第 i 条路径的第 $t(q)$ 个节点 (类型)
e_i	第 i 个节点和类型向量的拼接
r_i	GRU 的重置门，i 表示节点位置，同表 3.1 的 r_t
d_p	路径对的序列长度
n_t	路径对的数量（索引）
$\boldsymbol{r}_p$	路径对的编码集合（经过编码器后）
r_{p_i}	第 i 条路径的完整编码向量（经过编码器后）
LayerNorm(·)	层归一化方法，即公式 (3.3)

语义向量为 $h_s = \left[\overrightarrow{h_l}; \overleftarrow{h_l}\right]$, $h_e = \left[\overrightarrow{h_r}; \overleftarrow{h_r}\right]$[4]。路径对编码器则作用于每条 $Path_{s,e}$ 记录，即处理定义 3.2 中的每个节点。

每一条路径记录都是按照节点所在树的深度顺序拼接，具有时序性。静态（先序）路径是指，在满足定义 3.1 的情况下，先按照深度遍历获取所有的叶子节点到根节点的路径，再按照先序遍历顺序，构造任意两个叶子节点间的路径（详见算法 3.1）。路径编码器是一个深度 RNN 的网络结构，当获取到所有的路径对后，便使用双向 GRU 进行编码，即 $h_p = \text{GRU}(\{Path_{s,e}\})$。同 NLP 方法一样，每个节点都使用 Embedding 层表示，得到节点的初始向量。为丰富节点的特征，本文根据不同的节点归纳出 19 个节点类型，作为每个节点的附加特征。对于每个节点 t 都有 $q = type(t)$ 表示其类型。根据节点的两类特征，本文给定两个 Embedding 层：Emb_t, Emb_q 用户获取节点与对应类型的向量，即有：

$$e_{t_i} = Emb_t(t_i),$$
$$e_{q_i} = Emb_q(q_i).$$

本文取 $e_{t_i} \in \mathbb{R}^{d_t}, e_{q_i} \in \mathbb{R}^{d_q}$，在进行特征融合时，直接将两个向量拼接，得到 $e_i = \left[e_{t_i}; e_{q_i}\right]$ 表示节点 t_i 的向量。其中，; 表示向量拼接。

[4]为便于讨论，本节将最后一层的隐藏状态仍用 h 表示

算法 3.1: 静态路径对构造算法

输入: 长度为 l 的根节点到所有叶子节点的路径 *paths*
输出: *pairs*, 满足定义 3.2 的路径对

```
if 1 ≥ l then
    return Join(paths);
end
声明结果的空数组: pairs ;
for i ← 0 to l − 2 do
    from ← paths_i;
    for j ← i + 1 to l − 1 do
        to ← paths_j;
        k ← 0;
        while from_k = to_k and k < |from| and k < |to| do
            k ← k + 2 ;
        end
        for pos ← 0 to |to| do
            to_pos ←↑;
        end
        pair ← Concat(Join(Reversed(from)), Join(to));
        将 pair 放入 pairs;
    end
end
return pairs
```

将 e_i 作为路径编码器的输入，在正向与逆向过程中执行

$$
\begin{aligned}
r_i &= \sigma\left(W_r e_i + V_r h_{i-1} + b_r\right), \\
\widetilde{h_i} &= \tanh\left(W w_t + r_i \odot V h_{i-1} + b\right), \\
z_i &= \sigma\left(W_z e_i + V_z h_{i-1} + b_z\right), \\
h_i &= (1 - z_i) \odot h_{i-1} + z_i \odot \widetilde{h_i}.
\end{aligned}
$$

在深度双向网络中，取最后一层最后一个节点对应的隐藏状态 $h_{d_p}^{(n_l)}$ 作为 AST 路径对的表示 $h_{d_p}^{(n_l)} = \left[\overrightarrow{h}_{d_p}^{(n_l)}; \overleftarrow{h}_{d_p}^{(n_l)}\right]$。标准化 (Normalization) 是一种有效的加快收敛和减小过拟合的技术，公式 (3.2) 对最终 AST 路径的特征向量作层标准化，得到每条路径的最终向量。

$$
r_{w_i} = \mathrm{LayerNorm}\left(h_{d_p}\right) \tag{3.2}
$$

$\boldsymbol{r_w} = \left[r_{w_1}, r_{w_2}, \cdots, r_{w_{n_t}}\right]$，$n_t$ 表示路径对的最大数量。标准化技术通过改变向量的均值、方差，将神经网络的输出调整至同族分布的不同区间，使梯度更稳定。

$$
\begin{aligned}
\mu &= \frac{1}{D}\sum_{i=1}^{D} a_i \\
\sigma &= \sqrt{\frac{1}{D}\sum_{i=1}^{D}(a_i - \mu)^2} \\
a &:= \frac{a - \mu}{\sigma + \epsilon} * \gamma + \beta
\end{aligned} \tag{3.3}
$$

如公式 (3.3)为标准化的具体过程，a 为神经网络的输出，γ,β 为标准化层需要学习的参数，ϵ 为分母的平滑因子。

3.3.3　动态路径对编码器

表 3.3: 基于自注意力机制的编码器符号对照表

符号	含义
E	上一层的向量，这里指经过特征融合后的路径对特征向量
Q	自注意力机制的 Query
K	自注意力机制的 Key
V	自注意力机制的 Value
V'	经过自注意力融合的新 Value
A	自注意力机制权重得分
$M^{\neg}$	Mask 上三角矩阵

3.3.2 节遵循定义 3.2 严格使用先序遍历顺序穷举地构造路径对来表示 AST。这种处理方式虽然能够覆盖所有的可能序列，但对于深度学习模型来说会引入过多的噪声，导致问题 (C)。

证明： 设 AST 的叶子节点数量为 n，则按照定义得到的路径对数量 n_t 为：

$$n_t = \sum_{i=1}^{n-1} i = \frac{1+n-1}{2}(n-1) = \frac{1}{2}n(n-1). \tag{3.4}$$

在 AST 中，叶子节点为程序语句的变量名、常量名等，因此

$$n'_t \propto k,$$

n'_t 表示有效路径数，k 表示代码语句数。显然 $k < n < \frac{1}{2}n(n-1)$。$n'_t < n_t$ 得证。算法 3.1 的复杂度为 $O(n^2)$，从中也可看出数据规模为 n^2 量级。本文的实验发现，使用动态路径对编码器可以有效地降低噪声带来的影响，提升模型性能。

动态路径对编码器是一个基于自注意力（Self-Attention）机制的路径编码网络，本文称之为 SPF w/SelfAtten（Self-Attention based Path Fusion，基于自注意力融合机制）网络。自注意力机制来源于 Transformer 模型中，用于捕获序列之间的关系所提出。基于该思想，本文将自注意力机制用于特征融合的过程，以此来满足定义 3.1 对于路径对的定义。在 Transformer 中，自注意力机制是一种点

积式运算，若 E 表示上一层的向量，可根据如下公式获得注意力得分：

$$\begin{aligned} Q &= W_q \times E \\ K &= W_k \times E \\ V &= W_v \times E \\ A_{i,j} &= \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) \\ V' &= A \times V \end{aligned} \tag{3.5}$$

其中 $W_q \in \mathbb{R}^{d_m \times d_k}, W_k \in \mathbb{R}^{d_m \times d_k}, W_v \in \mathbb{R}^{d_m \times d_v}$ 为学习的参数，d_m 表示输入向量的维度，d_k 表示注意力的投影空间维度，d_v 表示特征值的投影空间维度。

SPF 将实现自动从路径列表中发现有效的路径融合，不用根据原始数据预先构造。在此，可以将定义 3.2 加以调整得到定义 3.3 。从定义出发，编码器得到各路径的向量，通过自注意力机制对各个路径的向量进行融合，即可得到 $Path'$ 的特征向量。

定义 3.3. *AST* 的有效路径对可以由若干满足先序遍历顺序的叶子节点到根节点的完整路径的融合表示。将任意一个叶子节点到根节点的路径表示为 $Path_i = \left\{r, t_1, t_2, \cdots, t_{d_p-1}, i\right\}$，则有效路径对为：$Path' = Fusion\left(\{Path_i\}\right), \forall \{i | 1 \le i \le n_t, i \in \mathbb{N}\}$.

与 3.3.2 节的路径编码器一样，每条路径 $Path_i$ 经过 GRU 的处理得到语义向量 $h_{d_p} = \left[\overrightarrow{h}_{d_p}; \overleftarrow{h}_{d_p}\right]$，经过标准化后得到 $\boldsymbol{r}_{p_i}$。经过特征融合层（见 3.3.4 节）后得到含有语义和语法的特征 $\boldsymbol{r}$，根据公式 (3.5)加以改进，使用点积得到路径之间的注意力：

$$\begin{aligned} Q &= W_q \times \boldsymbol{r} \\ K &= W_k \times \boldsymbol{r} \\ A &= \text{softmax}\left(\boldsymbol{M}^\urcorner \odot \frac{QK^\top}{\sqrt{d_k}}\right) \end{aligned} \tag{3.6}$$

$$\boldsymbol{M}^\urcorner = \begin{bmatrix} -\infty & 1 & 1 & \cdots & 1 \\ -\infty & -\infty & 1 & \cdots & 1 \\ -\infty & -\infty & -\infty & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ -\infty & -\infty & -\infty & \cdots & -\infty \end{bmatrix} \tag{3.7}$$

所得的 $A \in \mathbb{R}^{n_t \times n_t}$ 便是路径 $Path$ 之间的注意力得分，其中 $\boldsymbol{M}^\urcorner$ 为满足定义 3.3 中的 “先序遍历顺序” 约束条件所用的 “上三角” 掩码矩阵，具有公式 (3.7)的值。使

用上三角可以保证 $A_{i,j} = \begin{cases} 0, & i \geq j \\ p, & i < j \end{cases}$，且 $p \in (0,1)$。当 $A_{i,j} = 0$ 时，注意力机制将忽略该位置的值，而只在两条路径满足先序遍历约束时才计算。公式 (3.6) 计算了任意两条路径间的特征融合权重，使用该权重可以得到融合后的特征 $\boldsymbol{r} := A\boldsymbol{r}, \boldsymbol{r}_i = \boldsymbol{r}_i + \sum_{j=1}^{n_t} A_{i,j}\boldsymbol{r}_j$。

3.3.4　语法与语义融合层

表 3.4: 融合层符号对照表

符号	含义
r_w	SPF 模型中的叶子节点文本编码器输出
r_s	静态路径对模型中起始叶子节点的编码器输出
r_e	静态路径对模型中终止叶子节点的编码器输出
$\widetilde{\boldsymbol{o}}$	语法语义向量拼接的降维结果
o_1	特征融合层过滤操作的输出
$\widetilde{o}_1$	特征融合层重建操作的输出
$\boldsymbol{o}$	特征融合层经过残差后的最终结果

路径编码器仅用于理解 AST 中的非终止符，主要提取源代码的语法结构和特定模式。特征编码层用于将 3.3.1 节的语义特征同 3.3.2 节或 3.3.3 节的语法特征融合为一个完整的语义向量。特征融合采用带残差的多层非线性变换实现，由于编码器的特征具有较高维度，含有一定的噪声，因此本文使用一种瓶颈结构用于过滤特征中的噪声，得到更有效的向量表示，具体结构见图 3.6 。

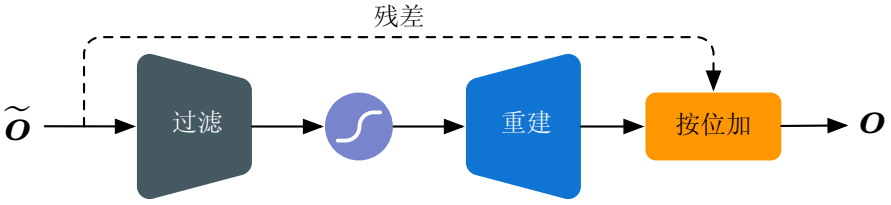


图 3.6: 特征融合层结构

为方便讨论，本节以 r_s, r_e 表示静态路径对模型中文本编码器得到的语义特征，r_w 表示动态路径对模型（SPF w/SelfAtten）编码的文本语义，$\boldsymbol{r}_p$ 表示路径编码器的路径特征。融合的过程采用多个线性层实现，对于静态路径对模型，首先对特征进行降维得到 $\widetilde{\boldsymbol{o}} = W_o\left[r_s; \boldsymbol{r}_p; r_e\right] + b_o$。类似地，对动态路径对的特征降维有 $\widetilde{\boldsymbol{o}} = W_o\left[\boldsymbol{r}_p; r_w\right] + b_o$。将降维后的结果，输入至融合层进行特征融合，获得复

合特征。公式 (3.8) 刻画了该融合过程，从公式中也可看出，在经过过滤、重建后，原始特征和充分融合后的特征使用残差结构进行连接。

$$\begin{aligned} o_1 &= \sigma\left(W_1\widetilde{\boldsymbol{o}} + b_1\right) \\ \widetilde{o_1} &= W_2\widetilde{\boldsymbol{o}} + b_2 \\ \boldsymbol{o} &= \widetilde{\boldsymbol{o}} + \widetilde{o_1} \end{aligned} \tag{3.8}$$

其中 $W_1 \in \mathbb{R}^{d_f \times (d_w+d_p)}, W_2 \in \mathbb{R}^{d_r \times (d_w+d_p)}$，且 $d_f < d_r$。从图 3.6 中也可看出，W_1 所表示的线性层用于将原始特征降维至低维空间，而 W_2 的线性层用于将特征重新投影至原始空间，从中实现对噪声信息的过滤。

3.4　方法名预测解码器

本文提出使用 “方法名预测” 和 “文本-代码语义匹配” 两个任务，验证编码器的效果。根据 3.3.1 节的讨论，代码子序列可以一定程度地解决代码域词典中低频词的负面影响。同样地，本节将方法名按照代码 3.1 的规则，将方法名拆分为子序列，通过生成序列预测方法名。本节主要讨论解码器的结构，以及解码时使用的注意力机制等解码策略。

表 3.5: 解码器符号对照表

符号	含义
$\boldsymbol{c}$	上下文特征，同表 3.4 中的 $\boldsymbol{o}$
$q(\cdot)$	编码器特征融合方法
s_t	解码器的隐藏状态，t 表示生成的第 t 个片段
v	解码器输出层的输出
$\boldsymbol{c}_t$	注意力解码中生成第 t 个片段所用的上下文
$\boldsymbol{\alpha}$	解码器注意力权重得分
$\alpha_{i,j}$	生成第 i 个片段时对第 j 条路径对的注意力得分

3.4.1　传统 RNN 方法名解码器

解码器被用于将编码的结果解析为具有可读性的表现形式。Alon 等把方法名预测视为一个多分类问题，每个方法名都是一个类别。从数据集的角度分析，方法的命名组合形式不一，导致了大量的类别数量，甚至出现一个类别只有一个样本的情况，使模型对此类低频样本建模能力下降[5]。从子序列的角度出发，方

[5]大量的样本对模型更新贡献更大，从而忽略低频样本。

法名也可视为一系列词构成的序列，可以使用序列生成方法获取该序列。自回归模型是用于时间序列的常用稳定模型之一，基于 RNN 的解码器便是一种成熟的自回归生成模型。解码器的目标在于最大化整个目标序列的联合概率：

$$P(Y|X) = \arg\max_{\theta} \prod_{t=1}^{N} p(y_t|y_{<t}, x; \theta) \tag{3.9}$$

根据公式 (3.9) 的描述，解码器将在每个时间 t 生成一个片段，最终构成的最大概率的序列便是目标序列。同时，也可发现解码器在解码阶段，只考虑历史生成结果 $y_{<t}$，说明只需单向处理即可。

基于 RNN 的解码器每个时刻接受两个输入：$y_{<t}$ 和 x，x 表示编码器表示的语义向量，$y_{<t}$ 表示已经生成的序列。不同于编码器的是，解码器仅使用单向 RNN，其大致结构如图 3.7 所示，每个生成过程，RNN 单元都使用了历史状态

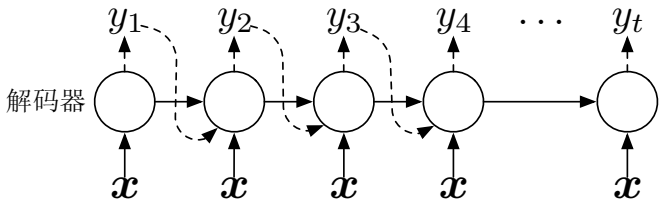


图 3.7: 基于 RNN 的解码器

s_{t-1}。每次生成都是一个分类问题，用于得到一个词，生成的词再根据词向量作为下一次解码的输入。可以设每一次解码得到的输出为 y_t，编码器的特征为 $\boldsymbol{c} = q(x_1, x_2, \cdots, x_{n_t})$，解码器将进行如下运算得到时刻 t 的隐藏状态：

$$s_t = f(y_{t-1}, \boldsymbol{c}, s_{t-1}) \tag{3.10}$$

$q(\cdot)$ 表示一种特征融合方法，如 $q(x) = \frac{1}{N}\sum_{i=1}^{N} x_i$，$f(\cdot)$ 为 GRU 单元的前向过程，得到隐藏状态后，作为特征用于分类得到目标词：

$$\begin{aligned} v &= W_o(\sigma(W_s s_t + b_s)) + b_o \\ y_t &= \frac{\exp(v_i)}{\sum_{j=1}^{N}\exp(v_j)} \end{aligned}, \tag{3.11}$$

其中 $\sigma(\cdot)$ 表示激活函数，作用于 RNN 隐藏状态的特征变换，W, b 均属于待学习参数。3.3.3 节提到静态路径对编码器需要处理 n^2 量级的样本，但公式 (3.10) 只用 $\boldsymbol{c}$ 表示这些样本的特征，显然，方法 $q(\cdot)$ 的过程会丢失一部分信息。

3.4.2　注意力机制方法名解码器

直观地理解，解码时每生成一个词只需要代码中的一部分语句即可，对应着 AST 中的一部分子树，这同人工阅读和命名的习惯一致。注意力机制即是模拟该人工操作，从所有的路径对中选出对当前操作最有贡献的路径对。同公式 (3.10) 不同，注意力机制每一次都使用新的 c_t 作为编码器提供的特征。图 3.8 刻

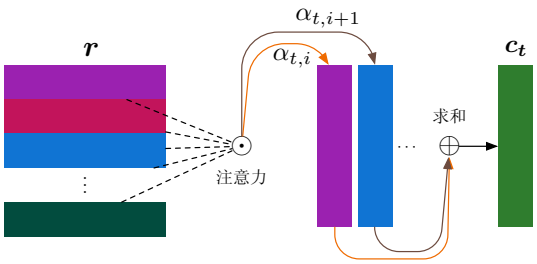


图 3.8: 注意力机制示意图

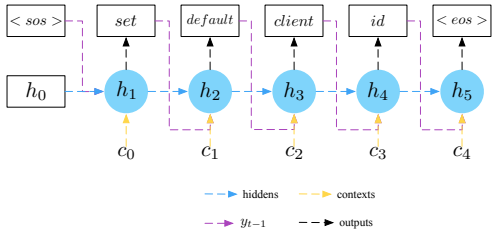


图 3.9: 基于注意力机制解码器

画了解码器中注意力机制实际的执行过程：1) 根据线性运算，对编码器特征打分；2) 按注意力得分，对编码器特征加权求和，获取融合特征 $\boldsymbol{c}_t$。同样，用 $\boldsymbol{r} \in \mathbb{R}^{n_t \times d_r}$ 表示编码器得到的特征，它按照注意力得分产生 t 时刻解码所需的上下文 $\boldsymbol{c}_t$：

$$\boldsymbol{c}_t = \sum_{i=1}^{N} \alpha_{t,i} \cdot \boldsymbol{r}_i \tag{3.12}$$

$\boldsymbol{\alpha}_t$ 表示注意力得分，它使用同公式 (3.5)相似的过程获得。当 $\boldsymbol{\alpha}_i = \frac{1}{n_t}$ 时，它将等价于 $q(\cdot)$，说明注意力只是一种动态的特征融合方法。

$$\begin{aligned} \alpha_{t,i} &= \frac{\exp(e_{t,i})}{\sum_{j=1}^{n_t} \exp(e_{t,j})} \\ \boldsymbol{e}_t &= \frac{Q K_t^{\top}}{\sqrt{d_k}} \\ Q &= W_q \boldsymbol{r} + b_q \\ K_t &= W_k s_t + b_k \end{aligned} \tag{3.13}$$

公式 (3.13)描述了注意力打分过程，Q, K_t 由两个前馈投影网络分别获得，再根据点积运算得到 $\boldsymbol{e}$，并最终得到打分。同样，d_k 表示投影空间维度。对照图 3.9 和公式 (3.10) ，可以给出解码过程：

$$\begin{aligned} s_t &= f\left(y_{t-1}, \boldsymbol{c}_t, s_{t-1}\right) \\ y_t &= g\left(s_t\right) \end{aligned} \tag{3.14}$$

结合上述讨论，可以分别给出静态路径对模型和动态路径对模型的整体框架图 3.10 和图 3.11 。图 3.10 对应了静态路径对模型，该图左侧为编码器，右侧为解码器。代码文本编码器先对路径的起始叶子节点和终止叶子节点编码，得

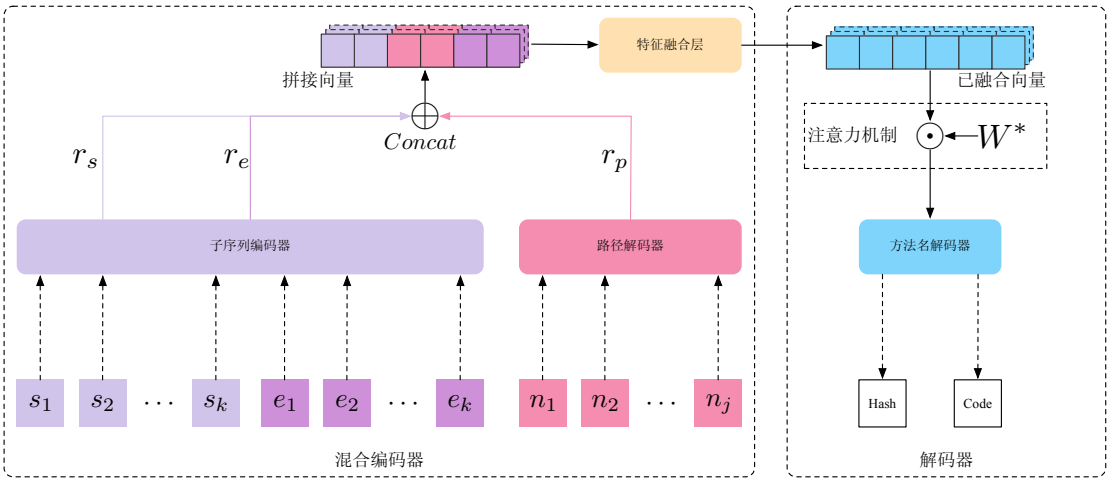


图 3.10: 基于 RNN 的静态路径对方法名生成模型

到两个语义向量 $\boldsymbol{r}_s, \boldsymbol{r}_e$。值得注意的是，这两个叶子节点属于同域文本序列，可以使用相同编码器处理。获取到语义信息后，路径编码器对去除了叶子节点的路径再次编码，获得路径向量 $\boldsymbol{r}_p$。为将两类特征融合，可首先拼接两类特征，即 $\boldsymbol{r} = \left[\boldsymbol{r}_s; \boldsymbol{r}_p; \boldsymbol{r}_e\right]$ 作为特征融合层的输入。右侧解码器使用融合后的特征，使用注意力机制得到方法名的文本序列。

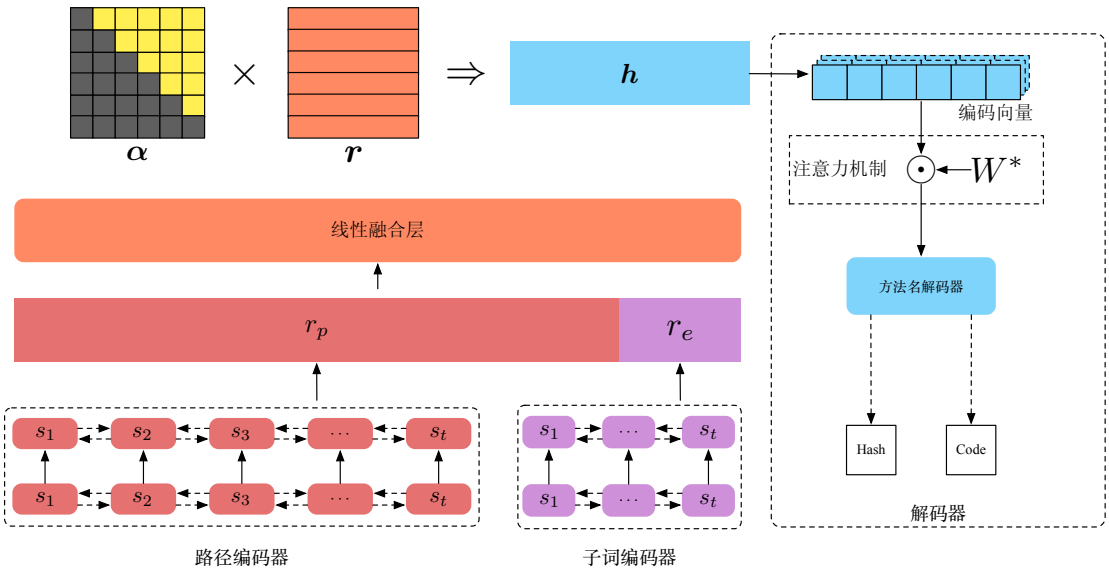


图 3.11: 基于自注意力自动路径融合的方法名生成模型

同样地，图 3.11 是动态路径对模型的框架图，它和静态模型的区别在于文

本编码只需一次，在语义-语法融合特征下游新增了一层自注意力机制，构造路径与其他可能路径的融合特征。从该图也可以看出，融合层的结果 $\widetilde{\boldsymbol{r}}$ 与经过上三角阵处理后的权重矩阵[6]相乘，得到路径对特征 $\boldsymbol{r}$ 用于解码。

这两个模型的主要区别体现在路径对的处理和编码上，SPF w/SelfAtten 模型仅需一次文本编码，处理的 AST 路径长度更短、数量更少，从而达到更快的处理速度。自注意力机制实现了一种动态融合策略。实验证明，自注意力机制不仅可以用于 Transformer 结构，同样也能推广到 RNN 等网络中。两种结构下都使用了特征融合层，特征融合可以达到特征降维，过滤噪声的效果，而且对特征降维后作非线性变换，可以增强模型的拟合能力。

从预训练的角度来看，编码器的设定和特征抽取能力至关重要，它是下游具体任务效果好坏的基础。

3.4.3　BeamSearch 搜索策略

Encoder-Decoder 框架存在难收敛的问题，因为解码的每一步都需要使用 $y_{<t}$ 作为输入，若历史输出有错误，会给后面的解码带来严重的错误累积，使得模型整体难收敛甚至不收敛。解决该问题的办法是使用 Teacher Forcing 机制，训练时使用正确标签，而非使用模型的输出。模型每个时刻的输出，仅用于计算损失和梯度。但是在预测阶段，每个样本的标签（方法名序列）对模型都不可见，只能使用每个时刻的输出产生所有的序列。简单地使用每个时刻的输出依次生成，是一种贪心策略[7]，每次选择最大概率的输出作为结果，因此最终结果很容易陷入局部最优解。

为获得全局最优解，可将每个时间步的可能结果都作为当前的候选结果，穷举所有可能的序列组合，再按最大联合概率作为目标序列。该搜索策略能够达到全局最优，但是复杂度过高。不妨设候选集的大小为 n，最大序列长度为 T，则共需要搜索 n^T 条序列。

Beam Search 是在穷举搜索和贪心搜索基础上的折中策略，在搜索过程中，它仅选择最可能的 b 个序列作为下一步搜索的候选集。b 为集束宽度，即 Beam Size。考虑每一步仍然产生 n 个候选结果，则将构成 $b\times n$ 个候选序列。鉴于 Beam Search 仅选择 b 个最大概率的序列，且 n 个候选结果中，仅有 b 个结果能与当前序列组合得到较大概率，因此可从 n 个序列中按照概率选择 b 个候选序列，构成 b^2 个组合，从中选择概率最大的 b 个组合作为当前过程的最大序列。该策略

[6]图中黄色部分表示有效部分，灰色部分表示被忽略的部分

[7]即 Greedy Search

将 n^T 的问题削减为 b^2 的搜索问题，规避了贪心搜索的局部最优，有更大概率获得到全局最优。如图 3.12 所示模拟了集束搜索的过程，当 $b = 3$ 时每一步都检

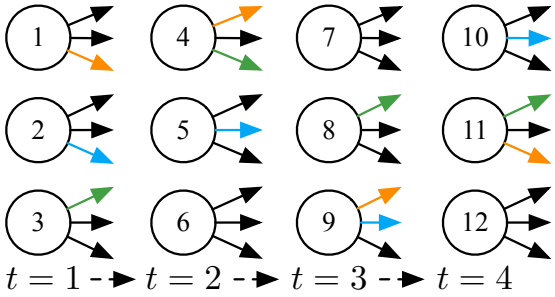


图 3.12: BeamSearch 搜索示意图

查 $b^2 = 9$ 个组合，并从中选择 $b = 3$ 个进行下一步，最终获得 $1 \to 4 \to 9 \to 11$、$2 \to 5 \to 9 \to 10$ 和 $3 \to 4 \to 8 \to 11$ 三个序列。

本文在方法名生成的预测阶段，均使用 Beam Search 搜索获得最佳序列。

3.5　代码文本语义匹配技术

语义匹配是 NLP 领域的常见任务，这个任务可以应用到诸多的场景，如智能问答、推荐系统、搜索引擎等。如问答系统中，对比问句和答案间的文本相关度，可以筛选出正确答案。

假设 3.1. 相似的文本一般具有相似的上下文场景，并且更倾向描述同一个概念。

将假设3.1推广至源代码，可以认为具有相同或相似内容的源代码更倾向于实现同一个功能，这从性质3.1和图 3.2 中也可看出。但根据性质3.1构建语义匹配模型，则每对代码必须要存在是否相似的标签，将会带来大量的标注工作。本文将该问题转化为代码与文本语义匹配，从而规避人工标注相似代码。

代码文本语义匹配基于一个基本事实：源代码与其文档注释的描述具有语义一致性，根据性质*3.1*的相似性，那么源代码的语义向量与对应文档注释的语义向量其广义的距离应趋近于 *0*。本文基于该事实，构建出代码 & 文本的语义匹配模型，并在本节详细讨论。

语义匹配的流程是，先使用代码编码器获取代码的特征向量，再对文本进行编码获取特征向量，对两个向量降维至同一语义空间后，使用余弦相似度[8]获取匹配得分。如图 3.13 所示是一个双塔结构，并且双塔间相互独立。该框架需要

[8]距离与余弦相似度均能评估相似性，当广义上的距离趋近于 0 时，表示相似，当余弦相似度趋近 1 时也相似。

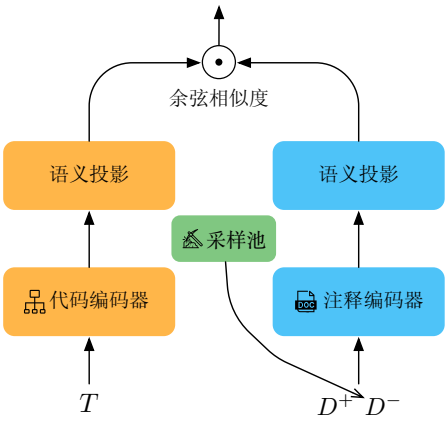


图 3.13: 语义匹配模型框架

配置两个不同的编码器，分别获取代码和注释的语义向量 $\boldsymbol{r}_t = Encoder_{ast}(T)$ 和 $\boldsymbol{r}_d = Encoder_{doc}(D)$。采样池的作用在于为模型提供负样本。本文使用向量空间模型中的余弦相似度，来评估文本与代码间的语义匹配得分。语义投影是一个线性前馈网络，得到最终代码语义向量 $\boldsymbol{u}_t = W_1\boldsymbol{r}_t + b_1$ 和注释向量 $\boldsymbol{u}_d = W_2\boldsymbol{r}_d + b_2$. 最终得到匹配得分 $score = \frac{\boldsymbol{u}_t \cdot \boldsymbol{u}_d}{\|\boldsymbol{u}_t\|_2 \|\boldsymbol{u}_d\|_2}$。本文在方法名生成任务中定义了两个编码器，结合迁移学习的思想，可以将这些任务中训练得到的编码器作为骨架网络[10][9]，作为图 3.13 的代码编码器。语义匹配的目标函数是最大、最小化正负样本间的相似度，即匹配样本间相似度趋近于 1，否则趋近于 −1，因此该任务的损失函数为：

$$\mathcal{L}(T, D) = \begin{cases} 1 - cosine(T, D), & D \in D^+ \\ \max(0, cosine(T, D) - margin), & D \in D^- \end{cases} \tag{3.15}$$

其中 $margin$ 为判定负样本的阈值，即认为当 $cosine(T, D) < margin$ 时都判定为满足不相似的条件。进一步将该公式合并，可以表示为：

$$\mathcal{L}(T, D) = \frac{1}{N}\sum_{i=1}^{N} y_i(1 - cosine(T, D)) + (1 - y_i)\max(0, cosine(T, D) - margin).$$

相似性度量的正样本，是从 Java 类文件中提取的方法代码、文档注释构成的序列对。但深度学习模型需要同时提供正、负样本才能保证模型正确收敛，否则模型将坍塌至失去泛化能力。考虑只有正样本参与训练，那么模型只需把所有的样本均预测为正样本即可。为保证模型有效，需要构造一定的负样本。因而，本文引入采样池结构，支持采样生成负样本。

[9]即 Backbone Network[10]

3.5.1　随机负采样

负样本的构造，是一个为代码添加不匹配的文档注释的过程。简单地，可以预先在数据集中随机地采样文档注释，构成一系列负样本对，作为训练样本，如算法 3.2 的描述。每加载一个正样本，都随机采样 n 个负样本。

算法 3.2: 负样本随机采样

输入: 代码集合 C，注释集合 D
输出: 负样本对 P
1 **foreach** c **in** C **do**
2　　$j \leftarrow \text{Random}()$;
3　　**while** $D_j = D_c$ **do**　　▷ D_c 表示代码 c 对应的文档注释
4　　　　$j \leftarrow \text{Random}()$;
5　　**end**
6　　Push $< c, D_j >$ **into** P;
7 **end**
8 **return** P

负样本构造可以在训练前执行，也可在训练时执行。训练前构造的方式，称为离线采样（Offline Sampling），而在训练时采样，称为在线采样（Online Sampling）。离线采样只能事先定义好采样比，每次训练采用相同的负样本对，这制约了负样本的数量。本文采用在线采样策略，即每一次加载数据，都重新使用算法 3.2 对每个正样本采样一个新的负样本。在线采样能够每次都重新采样，随机性更强。随着训练迭代次数的增加，构成的负样本对数量相对更多，使模型更具泛化性。

3.5.2　负采样增强

语义匹配的负样本挖掘，对模型的收敛和性能具有关键影响。若负样本过于简单，那么模型将轻松识别，从而无法分辨困难样本[10]，模型泛化能力减弱。虽然 3.5.1 节中的简单随机负采样，使用在线采样后能产生相对更多的采样结果，增大负样本的覆盖率，但是每次采样一个样本的策略仍然会受样本量和训练次数的影响。从应用的角度考虑，语义匹配能够应用于场景：在全量候选集上计算得分，得出召回排序结果。

本文基于召回思想，采用一种增强的采样策略获取负样本。在算法 3.2 基础上，增加一个超参数 n 表示采样比，可以得到更好的采样效果。算法 3.3 第 2 行根据采样比，每段代码循环采样 n 个负样本，构成更大的负样本集合，可以在一次迭代中获得更多可能的困难负样本。

[10]困难样本是指，看起来匹配但实际上并不匹配

算法 3.3: 负样本随机采样增强策略

```
输入: 代码集合 C，注释集合 D，采样比 n
输出: 负样本对 P
foreach c in C do
    for i ← 1 to n do
        j ← Random();
        while D_j = D_c do        ▷ D_c 表示代码 c 对应的文档注释
            j ← Random();
        end
        Push < c, D_j > into P;
    end
end
return P
```

经过增强后的在线采样策略可以保证模型在训练过程中更多地接触困难样本，从而稳定训练过程。该策略成功地避免了模型坍缩，增强了样本多样性。

3.5.3　路径对池化

表 3.6: 池化层符号对照表

符号	含义
$\boldsymbol{h}$	待池化的特征
$\boldsymbol{\alpha}$	自注意力机制权重得分
e_i	注意力机制能量值

定义 3.1 描述了路径对的定义，编码器的输出对应着每条路径对的语义向量。语义匹配任务，需评估一段代码与文本语义向量间的匹配度评分。由此可见，编码器的输出还需再次压缩方可用于语义匹配。本文采用均值和注意力两种池化方法，来合并编码器的所有路径向量，本节对这两种方法进行详细说明。

(1) 均值池化　均值池化是 CNN 中一种常用的池化手段，用于减小特征图的大小，融合相邻的特征点。但 CNN 中的池化是根据卷积核，在视野内缩小特征图规模。在 NLP 场景中，若采用序列模型处理输入序列，则 $N \propto T$，即每个特征图都是一维向量，向量个数与序列长度成正比，因此只能对向量进行融合。对 AST 而言，特征向量的数量也取决与路径对的数量。池化的目的便是将所有的路径对的特征，融合为一个特征。将路径对数量视为通道，则均值池化将基于通道 N，对所有向量取均值：

$$\boldsymbol{h} = \frac{1}{N}\sum_{i=1}^{N} h_{i,:}$$

其中 $\boldsymbol{h} \in \mathbb{R}^{N\times *}$。均值池化将变长特征转为了定长特征，相比向量拼接的融合方法，均值池化无需额外的参数，效率更高。

(2) 注意力池化　注意力池化可以视为均值池化的一个特例，均值池化视各个通道间均匀分布，相反，注意力机制认为各通道间不服从均匀分布。均值池化虽然减少了模型参数，但均匀的先验分布通常与实际的特征分布存在较大差异，导致模型性能下降。为建模特征的真实分布，本文使用全局注意力机制来对齐各特征，其过程如下：

$$\boldsymbol{h} = \sum_{i=1}^{N} \alpha_i \cdot h_{i,*}$$

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{N} \exp(e_j)}$$

$$e_i = W h_{i,*} + b$$

其中 $W \in \mathbb{R}^{d_c}$。这种全局注意力引入 $x \sim \boldsymbol{\alpha}$ 来替换 $x \sim \mathbb{U}(0, N)$，能更准确地表示各特征之间的重要程度。

3.6　本章小结

本章从定义、目标、技术三个角度详细阐述了本文的核心工作。本文将代码理解映射为特征抽取任务，旨在通过使用深度学习方法，获取到源代码准确的特征向量。首先，将代码文本解析为 AST 可以得到代码的结构化表示形式，从代码文本与 AST 的对照可以看出，AST 的路径对与源码中的语句有多对一的关系。总体上，本章详细描述了两个编码模型，将代码转化为语义向量。从模型角度来看，本文提出了一种基于自注意力机制的路径对自动融合方法，用于自动组合可用的路径。从任务角度来看，使用序列建模方法名生成任务，更加合理。

第四章　实验设计与结果分析

实验时首先将源代码处理得到对应的抽象语法树，再根据定义 3.1 和定义 3.2 可以得到 AST 中所有符合要求的由节点构成的序列。该序列可以作为程序代码语句解析后的另一种表现形态，即若干条序列可对应一条程序语句。静态路径对模型（Static Path Fusion without SelfAttention, SPF o/SelfAtten）将处理所有输入的序列，并将首尾两个叶子节点使用代码子序列编码器提取语义信息，再根据其他节点提取语法结构信息，最终融合得到完整序列的信息。得到的所有的序列信息，则表示为代码理解的结果。定义 3.3 进一步调整了序列构造策略，使用自注意力机制动态融合代码 AST 叶子节点对应的路径序列。与静态模型不同的是，动态模型（Self-attention based Path Fusion, SPF w/SelfAtten）只需要处理一个叶子节点即可。

4.1　研究问题简述

问题 (A)：代码语义遭遇的长尾问题；现有的研究在处理代码时，将 AST 中叶子节点表示的文本作为一个词，并直接使用词向量技术获取其语义。该方式面临严重的词典长尾问题，即每个叶子节点表示的文本同代码的项目、领域、语言、功能等存在强烈的依赖关系。变量命名通常组合简单单词，得到可读性更高的变量名。但高频词经过组合后，会成为低频词，导致叶子节点失去准确语义。本文采用子序列处理方法，将 AST 叶子节点按照 “驼峰命名”、“下划线” 等规则，将叶子节点文本拆分，得到更准确的语义信息。类似地，在方法名生成任务中，基准实验的将每个方法名作为一个完整的标签，将任务建模为分类问题。该建模方式会使长尾问题带来严重的标签不均衡，降低模型最终的泛化能力。

问题 (B)：不全面的代码语法结构特征；现有的研究在处理代码语法结构时，通常采用树、图等的网络，从网络结构上适配数据结构，但所选用的模型却很容易受到梯度消失的影响，使语法特征抽取不鲁棒。而基于全连接网络的网络，则更是直接忽略了序列中的强时序关系。本文采用定义 3.1 和定义 3.2 将 AST 的树结构，拆解为平行的序列结构，从而可使用朴素的 RNN 网络提取每个序列中的时序关系，将问题规模削减，得到更鲁棒的语法编码器。相较于基准实验的全连接网络，该编码器对语法结构的提取能力更强。

问题 (C)：静态代码路径对存在噪声。定义 3.2 的处理方式包含了大量不具

备实际含义的路径序列，但由于路径序列同程序语句存在对应关系，噪声过多将导致模型过度拟合噪声而无法识别有效数据。针对该问题，本文使用在线采样策略，降低噪声数据在每一次训练时的影响。此外，本文还使用定义 3.3 根本上调整了路径序列的构造方法，使模型每次仅需处理一条路径，并结合自注意力机制动态地组合“路径对”，抑制噪声干扰。

4.2　数据集介绍及构造

根据第三章的模型介绍，本文需提供三个任务的数据集：1) 抽取代码语义所需词向量的预训练语料；2) 静态代码路径对模型（SPF o/SelfAtten）的训练数据；3) 动态代码路径对模型（SPF w/SelfAtten）的训练数据。本节集中讨论数据集的构建过程和数据规模。

(1) 预训练语料　本文采用了子序列文本编码器来捕获 AST 中叶子节点的语义信息，但限于开放域语料和词汇同软件工程的语料存在一定的差异。为重新构造更适配源代码理解场景的词典和语料，在本文的实验中，共爬取了超过大于 30 个项目的 API 文档，如表 4.1 列举的 API 文档的来源。每个 API 文档页面规格都采用 Javadoc 标准页面排版，因此可以使用 XPath 选择器：

(a) `'//div[@class="block"]'`;

(b) `'//li[@class="blockList"]/dl/dd'`,

获取所有文本语句。粗略统计，API 文档中的类、方法、参数等的描述文本经过初步清洗后，量级约为 80M。从表中也可看出，API 文档所属的项目涵盖了语言、算法结构、分布式、多线程、JSON、网络、缓存和数据库等多个细分领域。

(2) 特殊字符清洗　网络爬虫解析的为 HTML 文档，通过 HTML 文档转化为文本文档，中间会掺杂含有特殊字符：`'\x01'`，`'\n'`，`'\t'`，HTML 标签等。在构建领域词典之前，还需进一步清洗文档中的特殊元素。清洗过程仍采用正则表达式进行，详细的各个规则可见表 4.2 。对于特殊的格式，如方法调用 `a.b(c, d)`将被清洗为 `a invoke b with parameters: c and d .`。

对清洗后的文本数据，使用 NLTK[1]和 wordninja[2]分词，构建软件工程领域下的词库。NLTK 主要使用英文常用标点分词，而 wordninjia 根据常用词，将连词分割为空格分隔的子序列，如`'httpcomponent'`分割为`'http component'`。使用上

[1] https://www.nltk.org

[2] https://github.com/keredson/wordninja

表 4.1: API 文档列表

项目	API 地址
OpenJDK	https://docs.oracle.com/javase/7/docs/api/allclasses-frame.html
SpringCore	https://docs.spring.io/spring-framework/docs/current/javadoc-api/allclasses-frame.html
SpringORM	https://www.javadoc.io/doc/org.springframework/spring-orm/4.3.14.RELEASE
Mybatis	https://mybatis.org/mybatis-3/apidocs/allclasses-frame.html
Guava	https://guava.dev/releases/18.0/api/docs/allclasses-frame.html
OpenJPA	http://openjpa.apache.org/builds/3.1.2/apidocs/allclasses-frame.html
JodaTime	https://www.joda.org/joda-time/apidocs/allclasses-frame.html
CommonsLang	https://commons.apache.org/proper/commons-lang/javadocs/api-3.4/allclasses-frame.html
CommonsCollections	http://commons.apache.org/proper/commons-collections/apidocs/allclasses-frame.html
CommonsMath	http://commons.apache.org/proper/commons-math/apidocs/allclasses-frame
CommonsEmail	https://javadoc.io/doc/org.apache.commons/commons-email/latest/allclasses-frame.html
CommonsCompress	https://javadoc.io/doc/org.apache.commons/commons-compress/latest/
CommonsPool2	https://javadoc.io/doc/org.apache.commons/commons-pool2/latest/allclasses-frame.html
CommonsText	https://javadoc.io/doc/org.apache.commons/commons-text/latest/allclasses-frame.html
DBCP2	https://javadoc.io/doc/org.apache.commons/commons-dbcp2/latest/allclasses-frame.html
Configuration	https://commons.apache.org/proper/commons-configuration/apidocs/allclasses-frame.html
Lucene	https://lucene.apache.org/core/5_2_1/core/allclasses-frame.html
Ehcache	https://www.ehcache.org/apidocs/3.2.0/allclasses-frame.html
Tomcat	https://tomcat.apache.org/tomcat-9.0-doc/api/allclasses-frame.html
Hibernate	https://docs.jboss.org/hibernate/orm/5.4/javadocs/allclasses-frame.html
JacksonDatabind	https://fasterxml.github.io/jackson-databind/javadoc/2.7/allclasses-frame.html
JacksonCore	https://fasterxml.github.io/jackson-core/javadoc/2.7/allclasses-frame.html
OkHttp	https://javadoc.io/static/com.squareup.okhttp3/okhttp/3.14.9/allclasses-frame.html
ActiveMQ	https://activemq.apache.org/maven/apidocs/allclasses-frame.html
Hadoop	https://hadoop.apache.org/docs/stable/api/allclasses-frame.html
HBase	https://hbase.apache.org/apidocs/allclasses-frame.html
JUnit	https://junit.org/junit5/docs/current/api/allclasses.html
ActiveJDBC	http://javalite.github.io/activejdbc/2.2/allclasses-frame.html
Cache	https://javadoc.io/doc/javax.cache/cache-api/latest/allclasses-frame.html
C3P0	https://www.mchange.com/projects/c3p0/apidocs/allclasses-frame.html
HugeCollections	http://openhft.github.io/Java-Lang/apidocs/allclasses-frame.html
HTML	https://javadoc.io/static/net.sourceforge.htmlunit/htmlunit/2.36.0/allclasses-frame.html
Json	https://www.javadoc.io/doc/org.json/json/latest/allclasses-frame.html
Jersey	https://eclipse-ee4j.github.io/jersey.github.io/apidocs/3.0.0/jersey/allclasses-frame.html

表 4.2: 数据清洗规则

说明	正则规则
常用标点	`\n|\t|\.*/[a-zA-Z\.]+|[0-9]+\.[0-9]|<[\s\S]+>[\s\S]*</[\s\S]+>|<[\s\S]+/>|<[^>]+>`
HTTP(S)	`(https?|ftp|file)://[-A-Za-z0-9+&@#/%?=~_|!:,.;]+[-A-Za-z0-9+&@#/%=~_]`
方法调用	`([A-Za-z]+)\.([a-zA-Z0-9]+)\(([\s\S]*)\)`
方法声明	`[a-zA-Z0-9]+)\(\)`
方法声明	`[a-zA-Z0-9]+)\(([\s\S]*)\)`
驼峰命名	`([A-Z][a-z]{1,})`
专有名词	`([A-Z]{2,})`
空白字符	`\s{2,}`

述词库和语料，使用 gensim[3]获取 word2vec 词向量。在实验中，word2vec 采用 Skip-Gram 模型，并取词向量维度为 300，即 $d_w = 300$。

(3) AST 数据集　本文实验所需 AST 数据全部基于定义 3.2 和定义 3.3 构建。首先，从 Github[4]中获取 Java 项目，对项目中的每个类使用 JavaParser[5]分析得到 AST。接着，再使用先序遍历算法，获取根节点到所有叶子节点的路径。对

[3]https://radimrehurek.com/gensim/

[4]https://github.com

[5]https://javaparser.org

于两个不同的模型，分别按照算法 3.1 和定义 3.3 构建路径对。对方法名生成任务，提取路径 `MethodDecl ↓ Name ↓ [Name]` 中的方法名称，再使用正则表达式等规则，将其拆分为单词序列，作为目标序列。相反，语义匹配任务，需在解析 AST 时，额外地获取每个方法的文档注释，经过清洗后，得到语义文本。

特殊地，在代码中还可能包含各类注解，如继承类中存在的 “Override”。在数据清洗时，将 JDK 自带的注解节点及相关路径忽略，避免由于`@Override`、`@Deprecated` 等注解导致采样到过多的噪声。其他自定义注解，则仅构造注解到方法名节点间的路径。不仅如此，数据清洗后还对各个节点，按照先验知识和表 4.3 中的规则归纳出每个节点对应的类型，如 IfStmt、BinaryExpr、BlockStmt 分别属于语句 (Statemnt)、表达式 (Expression)、语句 (Statemnt) 三类。其中，UNK 为解决 OOV 问题，PAD 则用于序列填充。节点类型特征是额外先验特征，更能

表 4.3: AST 节点类型对照表

类型	匹配规则	备注
unk	`'\[UNK\]'`	未知节点
cls	`'\[CLS\]'`	Trm 预留标记
sep	`'\[SEP\]'`	Trm 预留标记
mask	`'\[MASK\]'`	掩码
leaf	无	叶子节点
pad	`'\[PAD\]'`	填充节点
declaration	`'^[a-zA-Z]+Declaration$|^[a-zA-Z]*Declarator'`	声明，表示方法、变量声明等
expr	`'^[a-zA-Z]+Expr$'`	表达式，表示一元、二元运算等
constant	`'^[a-zA-Z]+LiteralExpr$'`	常量值，字面常量、`null`等
privilege	`'Modifier'`	权限值，对应访问权限
name	`'^[a-zA-Z]*Name$'`	名称节点，对应方法名、变量名等
statement	`'^[a-zA-Z]+Stmt$'`	基础语句，如条件、循环语句等
type	`'^[a-zA-Z]*Type$|^[a-zA-Z]*Pair'`	类型，如整型、字符串等
entry	`'^[a-zA-Z]*Entry'`	迭代器
creation	`'^[a-zA-Z]*Creation'`	创建语句，表示`new`语句
direction	`'\u2193|\u2191|\\|>|<\\|'`	方向，表示相邻节点的方向
parameter	`'^[a-zA-Z]*Parameter'`	参数列表
level	`'^[a-zA-Z]*Level'`	多维数组的维度
methodname	`'<mask>'`	方法名

丰富节点在网络中的表现。除此之外，由于 AST 节点规模有限[6]，则 AST 节点向量维度无法太高，否则容易导致过高的过拟合。加入节点类型特征，可以先验地将特征向量分割为两部分，能够一定地减小节点嵌入的过拟合问题。

综上，AST 数据集包含两种形式路径对：定义 3.2 和定义 3.3 ，每个路径对中的非终止符按照规则映射出对应的类型，构成模型的输入。数据标签同具体任务保持一致，方法名预测使用拆分后的单词序列作为标签，用于生成任务。语义匹配任务使用对应的文档注释的文本序列作为标签，用于相似度任务。

[6]在本数据集中，节点数大约为 70 个

根据 Java 类文件，抽取到路径对与任务标签，随后，对数据按项目拆分。将数据所属项目作为类别，使用 scikit-learn[7]分层采样，得到训练集、验证集、测试集，遵循 6:1:1 的比例，详细的数据分布情况见表 4.4 。验证集的主要作用是通过观察模型在验证集上的表现，确定模型所需超参、学习率，并判断收敛情况等。测试集用于最终评估模型的性能，确定它的泛化能力。根据项目分层抽样，可以确保数据之间不存在交集，否则无法说明模型有效性。

表 4.4: 数据集规模

项目	训练集 (16k)		验证集 (2k)		测试集 (2k)	
	类数	方法数	类数	方法数	类数	方法数
Java	3,505	40,109	508	6,092	509	6,021
Tomcat	905	9,480	132	1,045	129	1,427
Hibernate	1,727	12,520	262	2,124	241	1,929
Spring	632	4,994	92	901	94	721
Mybatis	182	1,326	26	187	26	137
Commons(+10)	1,294	11,067	184	1,827	184	1,909
Guava	383	6,166	56	1,050	55	1,031
Joda	119	2,327	17	333	18	195
Lucene	1,780	12,931	262	1,638	261	1,905
Solr	2,151	15,829	312	2,115	314	2,181
Hadoop	2,585	30,058	386	9,121	380	3,856
HBase	615	6,676	91	859	82	837
ActiveJDBC	42	523	6	52	9	34
ActiveMQ	1,404	10,948	207	1,779	199	1,572
C3P0	80	908	10	189	12	118
Jackson	389	4,401	57	601	56	815
FastJson & Json	126	1,133	19	116	17	512
HugeCollections	37	379	5	73	5	46
Jersey	245	2,673	38	529	34	311
OkHttp	83	770	12	128	12	138
OpenJPA	860	12,780	127	1,843	118	1,918
JUnit	163	11,318	21	97	25	168
总计	189,316		32,768		27,421	

训练阶段，每次加载一个批次的 AST 路径对作为输入样本，在方法名生成任务中，还加载方法名序列作为标签。在语义匹配任务中，额外地加载文档注释文本，设置标签为 1，并根据 3.5.2 节采样一批负样本，配置标签为 0。训练时保证，正样本对的相似度大于负样本对间的相似度，且正样本对相似度需近似 1.

4.3　实验方法及介绍

本节详细地介绍本文的实验配置，包括对比基准实验、验证性实验以及消融实验等。此外，还分析了训练使用的策略、超参数以及测试过程的区别。为准确衡量模型表现，本节根据具体任务及特性，选取对应的量化指标，并基于该指标给出解释性说明。

[7]https://scikit-learn.org/stable/

实验环境：Ubuntu 16.04，Python 3.6，Numpy v1.20，Pandas v1.2.2，NLTK v3.5，scikit-learn v0.24，PyTorch v1.7.1，CUDA 11.1，Nvidia RTX 3090 24G，Nvidia Driver 460.39.

(1) 超参数选择　超参数是模型必不可少的参数，它不从数据中学习而来，而是根据先验、统计、业务场景和问题规模而人为设定。从模型角度考虑，由于使用混合编码器，分别编码叶子节点（子序列）和路径对。根据第 三 章的模型设计，编码器均为基于 RNN 的网络，为便于训练时样本批量化，需给每个样本固定一个最大长度，并将不足最大长度的使用填充值补齐。在本文的实验中，将叶子节点（子序列）的最大长度 L_w 设置为 20，静态路径对模型（见 3.3.2 节）中路径对最大长度 n_t 设置为 40。类似地，根据对每个方法的路径对数据集统计可知，路径对数量绝大部分集中在 300 左右，但是其中含有一定的噪声。Zhang 等提出对路径对采样，可以达到正则化效果，由此本文设置路径对最大数量 n_p 为 256，对超过该数量的路径对使用随机采样。将 n_t 设置为 40 的理由可见图 4.1 ，从图

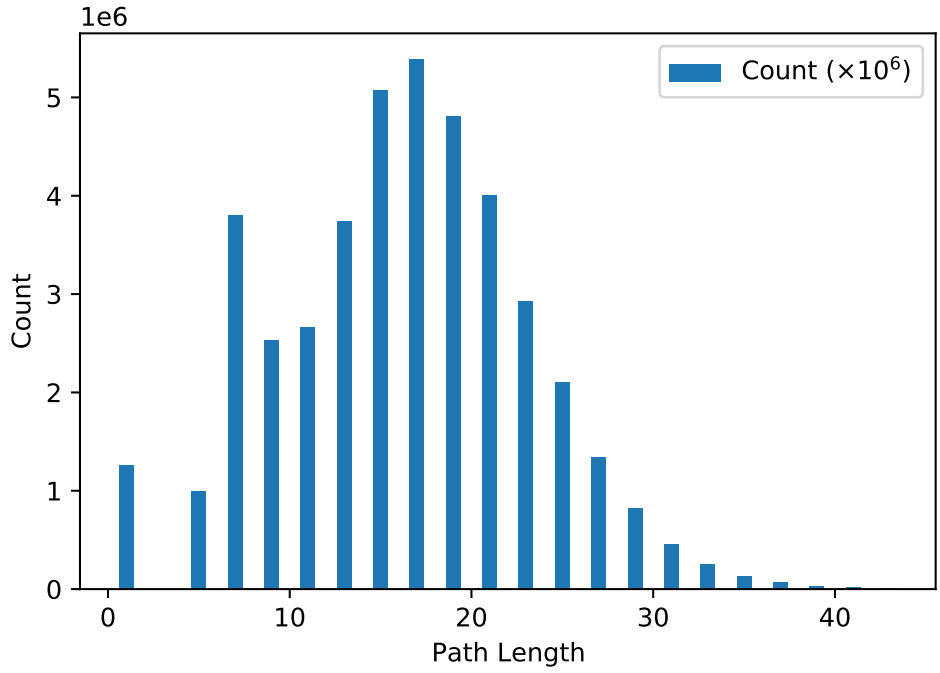


图 4.1: 静态路径对长度分布情况

中可知，SPF o/SelfAtten 路径对的长度分布近似地服从正态分布。大部分长度仍然集中于 20 ~ 30 之间，根据长度分布可以认为，设定参数为 40 符合数据场景。不同的是，SPF w/SelfAtten 模型由于只有单向路径，因此最大路径长度 n_t 仅设置为 20。根据公式 (3.4)，静态路径对模型路径对规模为 $\frac{1}{2}n(n-1)$，而 SPF

表 4.5: 超参数取值对照表

参数	参数说明	参数取值	
		SPF o/SelfAtten	SPF w/SelfAtten
L_w	叶子节点最大长度	20	20
n_t	路径对最大深度	20	40
n_p	路径对最大数量	24	256
p_{emb}	词向量层 Dropout	0.2	0.2
$p_{encoder}$	RNN 编码器 Dropout	0.3	0.3
p_{out}	输出层 Dropout	0.2	0.2
d_w	子序列文本词向量维度	300	300
d_n	AST 节点向量维度	256	256
d_t	AST 节点类型向量维度	128	128
d_h	RNN 隐藏层大小	256	256
l_{text}	文本编码器层数	2	2
l_{path}	路径编码器层数	3	3
d_k	注意力 K 的投影维度	-	128
d_q	注意力 Q 的投影维度	-	128
d_s	语义匹配空间维度	512	512
$l_{decoder}$	解码器序列长度	8	8
lr_{mng}	方法名生成学习率	10^{-3}	10^{-3}
lr'_{ast}	语义匹配下代码编码器学习率	5×10^{-6}	5×10^{-6}
lr_{doc}	语义匹配下文档编码器学习率	10^{-4}	10^{-4}
ξ	权重衰减系数	10^{-4}	10^{-4}
ψ	梯度裁剪范数	10	10

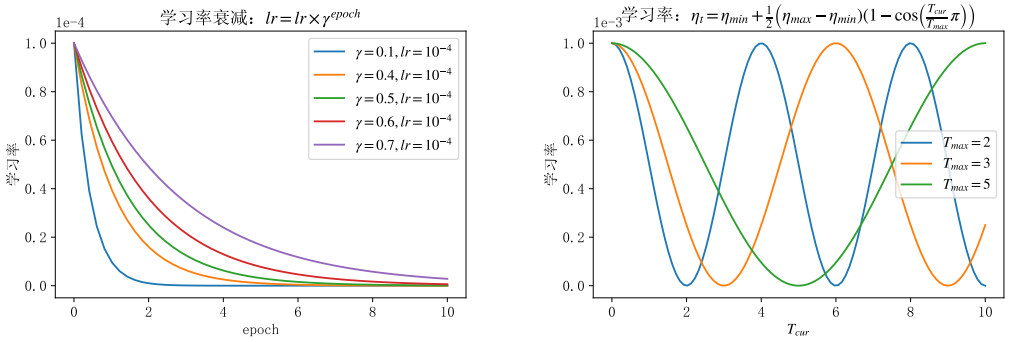
w/SelfAtten 路径对数量 $k \propto n$，由此设置 SPF w/SelfAtten 路径数量为 24[8]。为防止模型过拟合，训练时采用 Dropout 与权重衰减策略。对词向量的嵌入层，使用 $p_{emb} = 0.2$ 的 Dropout 概率，类似的，在 RNN 编码器层之间取 $p_{encoder} = 0.3$。在输出层，在线性变换前使用 $p_{out} = 0.2$ 的概率值。

另一个影响模型复杂的超参数为隐藏层神经元数量，在文本编码器中，取子序列词向量维度为 300，即 $d_w = 300$。词向量维度的确定需综合词表大小确定，不同于 Bert 采用隐藏层相同大小的维度，本文词向量规模较小因此采用更低维度的词向量。此外，较小的词向量，能够一定程度地减少模型参数[9]。同理，鉴于节点词典较小，从而选择更低的向量维度 $d_n = 256, d_t = 128$. RNN 编码器采用双向网络，设置隐藏层大小为 $d_h = 256$，层数为 $l_{text} = 2$。SPF w/SelfAtten 中的自注意力机制是其的核心结构，本文取 $d_k = d_q = 128$. 同时，在语义匹配任务中，将投影的语义空间设定为 512 维，且全局注意力的输出维度固定为 1。在解码器端，根据对数据集中方法名长度的统计，将解码器最大长度设为 8。具体地，超参数配置和相关说明可见表 4.5 。

模型训练过程中，需指定学习率以及优化算法。本文在实验中，方法名生

[8] $n = 24$ 时，$\frac{1}{2}n(n-1) = 276 > 256$

[9] 设词表大小为 V，维度大小为 M，则 Embedding 层大小为 $V \cdot M$，采用较小的维度 N，则词向量参数量为 $V \cdot N$，额外地空间投影参数为 $N \cdot M$，则 $V \cdot M - (V \cdot N + N \cdot M) = V(M-N) - NM$，若 $V(M-N) - NM \geq 0$，则 $V \geq \frac{NM}{M-N}$。当 $N = 300, M = 512$ 时，$V \geq 724$，显然成立，模型参数更少。



(a) 指数衰减策略　　(b) 余弦退火衰减策略

图 4.2: 学习率衰减策略

成任务从零开始训练，可采用较大学习率 10^{-3}。但是语义匹配任务直接使用方法名生成任务中的编码器作为骨架网络，仅改变下游，因而更适合用小学习率 10^{-4}。特别的是，在语义匹配任务中，由于网络结构存在两个分支网络，存在难以收敛的问题 [9]。(Chen 和 He, 2020) 提出使用停止梯度[10]的思想，在网络的一条分支切断梯度传播，避免了孪生神经网络的退化问题。不仅如此，迁移学习认为使用预训练好的模型在新的场景使用，只需要在新的数据上微调即可。基于这两点经验，在语义匹配实验中，将两个编码器设置为不同的学习率，其中文档编码器学习率更大为 10^{-4}，而代码编码器则较小为 5×10^{-6}。

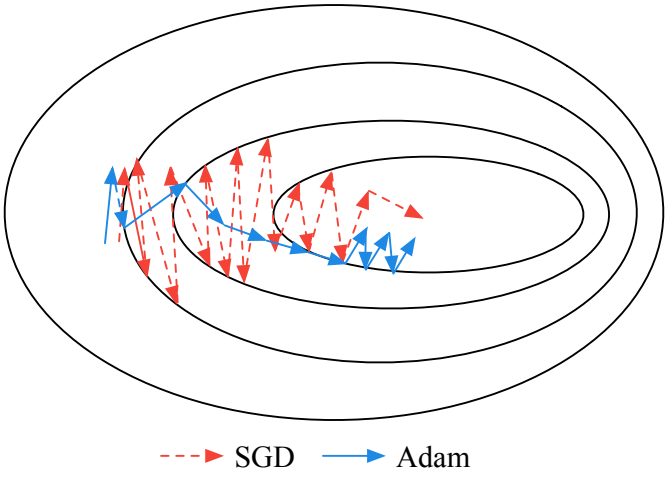


图 4.3: 优化算法对比

Adam 是目前深度学习中较为常用并且收敛速度快的优化算法之一，其使用二阶动量来动态地调整学习率，能够使模型前期快速收敛，如图 4.3 所示。

[10]stop-gradient 策略

AdamW 是应用权重衰减的代表算法之一，该算法在 Adam 的基础上，使用权重衰减来惩罚损失函数，能够降低模型过拟合风险，增强泛化性。在实践中，将权重衰减率设置为 10^{-4}。Adam 优化算法虽收敛速度快，但收敛效果往往并非最佳。本文通过两种手段来缓解该问题：1) 学习率衰减；2) SGD 收尾。如图 4.2 所示，学习率衰减能随着模型的训练迭代次数增加而减小，而当使用 Adam 开始震荡时，则更换 SGD 优化算法，使用小学习率在有限的空间内搜索最优收敛点。相较于 Adam，SGD 虽然收敛慢，但是却能得到更好的收敛结果。适时地衰减学习率，则能够避免在训练过程中，因学习率过大而跳过最佳收敛点而落入鞍点。本文的实验，采用指数下降及余弦退火的学习率调整策略，来控制学习率的变化。在前 5 个迭代中，使用指数下降（系数为 0.6，如图 4.2(a)），减小学习率，之后再根据余弦退火算法（如图 4.2(b)）时刻调整学习率，避免模型落入鞍点。

虽然，本文使用了 GRU 单元来替换 RNN 单元，解决了梯度消失问题，但却仍然存在梯度爆炸的风险。为了解决该隐患，本文将 RNN 的梯度使用梯度裁剪方法，按梯度的二范数为 10 的约束进行裁剪，确保模型稳定收敛。

(2) 评估指标选取　在训练时，为保证每次迭代效率，使用贪心搜索策略获得每个输出片段，避免搜索过程的巨大时耗。并且，由于训练时使用 Teacher Forcing 机制，这个过程产生的输出并不能代表最终模型的泛化性。因此在训练阶段，使用每个片段的 **TopK ACC** 指标作为评价，用于判断模型是否正常收敛，以及过拟合情况。**"Acc@1"** 指标表示 Top1 Acc，指的是在使用贪心搜索策略下，每一步输出时选择最大概率输出作为当前时刻输出所得出的指标，类似的也可得到 Top5 Acc 等，具体计算方式可见算法 4.1。Acc 指标的另一个缺点在于，无

算法 4.1: TopK ACC 指标

输入: 方法名序列 Y，预测序列 P，k 取值
输出: TopK 指标值 v
对预测序列，按照 k 切片：$p \leftarrow P[:k]$;
$i \leftarrow 0$;
$pos \leftarrow 0, neg \leftarrow 0$;
foreach y ***in*** Y **do**
    if $y \in p_i$ **then**
        $pos \leftarrow pos + 1$;
    else
        $neg \leftarrow neg + 1$;
    end
    $i \leftarrow i + 1$;
end
return $\frac{pos}{pos+neg}$;

法评估样本整体的质量，如连贯性等。Acc 高可能是在高频词下的预测能力更好，但是实际产生序列的整体效果很差。为此，本文选用 BLEU 指标评估生成质量。对生成的序列，BLEU 按照 n-gram 匹配，n-gram 匹配度越高则生成的序

列连贯性、合法性更好。换言之，Acc 衡量了生成结果在离散程度下的优劣，而 BLEU 指标是对生成结果的连续性评估。

在语义匹配任务中，实验中使用召回率来评估模型质量。可以考虑这样一个场景，在搜索或推荐系统的初步阶段，需要从海量的数据中找出一批与查询（Query）可能相关的目标样本，供后续精准排序使用。这便需要保证，算法能尽量找出准确的相关的样本，如从 100,000 个样本中，根据查询找出相关的 100 个样本进行排序，这个过程便是一个召回的过程。基于这个任务场景，**TopK Recall** 指标能够衡量模型的好坏，其中 K 便是上述过程的 100 个样本，显然随着 K 的增大，召回率会更高。当 K 相同时，召回率越高则模型更好。

(3) 实验方法　在实验中，首先训练方法名生成模型，因为相对而言，使用 Teacher Forcing 机制的网络较双塔网络更好收敛。生成模型中，每个时间点都生成一个片段，可视为一个分类过程。模型优化的目标为最小化交叉熵损失函数：

$$\mathcal{L}_{mng} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log p_i$$

其中 p_i 表示预测概率。在数据集中，每个词出现的概率存在巨大差异，这便出现了类别不均衡问题。在每个类别的交叉熵上赋予一个权重，可一定程度地解决该问题。该权重使得低频率样本获得较大权重，而高频率样本获得较低权重，使得模型对来自高频率和低频率样本的损失贡献相当：

$$\mathcal{L}_{mng} = -\frac{\sum_{i=1}^{N} y_i \cdot weight_{y_i} \log p_i}{\sum_{i=1}^{N} weight_{y_i}}.$$

weight 按照数据集方法名称中的单词频率而构造。交叉熵损失的目标将指导模型输出标签概率接近 1，这是一个极强的约束，容易降低模型泛化能力。除此之外，Li 等提出使用 Dice Loss（见公式 (4.1)）解决 NLP 中的类别不均衡问题，该损失对模型的输出约束更小，更加平滑。它的问题在于，损失函数收敛的点过于接近于决策边界。但该损失的好处在于能够根据模型的输出值，自动调节样本所占权重，故又被称为“自我调节的 Dice Loss”。

$$\boldsymbol{DSC} = 1 - \frac{(1-p_{i1})p_{i1} \cdot y_{i1} + \gamma}{(1-p_{i1})p_{i1} + y_{i1} + \gamma} \tag{4.1}$$

对公式 (4.1)求导可得损失函数的在 $2p_{i1} - 1 = 0$ 处，即 $p_{i1} = \frac{1}{2}$ 处收敛，详细推导见公式 (4.3)。为解决该问题，本文在原始的 DSC 基础上，增加额外的控制因子 α，将收敛边界放大，得到更健康的损失函数：

$$\boldsymbol{DSC}' = 1 - \frac{(1-p^{\alpha})py + \gamma}{(1-p^{\alpha})p + y + \gamma} \tag{4.2}$$

该损失函数能够当 $\alpha = 1$ 时等价于原始 DSC 损失，而当 $\alpha > 1$ 时，收敛点由 $p = \frac{1}{2}$ 变为 $p = (\alpha+1)^{\frac{-1}{\alpha}}$. 详细证明过程及推导过程可见公式 (4.4)。

$$\begin{aligned}
u' &= ((1-p_{i1})p_{i1} \cdot y_{i1} + \gamma)' \\
&= (1-2p_{i1}) \cdot y_{i1} \\
v' &= ((1-p_{i1})p_{i1} + y_{i1} + \gamma)' \\
&= 1 - 2p_{i1} \\
\Rightarrow \frac{\partial \mathcal{L}_{DSC}}{\partial p_{i1}} &= -\left(\frac{u'v - v'u}{v^2}\right) \\
&= -\frac{(1-2p_{i1}) \cdot y_{i1}\left((1-p_{i1})p_{i1} + y_{i1} + \gamma\right)}{v^2} \\
&\quad + \frac{(1-2p_{i1})\left((1-p_{i1})p_{i1} \cdot y_{i1} + \gamma\right)}{v^2} \\
&= (1-2p_{i1})\frac{-y_{i1}^2 + (1-y_i)\gamma}{v^2} \\
&= \frac{(2p_{i1}-1) \cdot (y_{i1}^2 - (1-y_{i1})\gamma)}{v^2}
\end{aligned} \tag{4.3}$$

证明. 对公式 (4.2)求导，可得：

$$\begin{aligned}
u' &= \left((1-p_{i1}^{\alpha})\, p_{i1} \cdot y_{i1} + \gamma\right)' \\
&= (1-(\alpha+1)p_{i1}^{\alpha}) \cdot y_{i1} \\
v' &= ((1-p_{i1}^{\alpha})p_{i1} + y_{i1} + \gamma)' \\
&= 1 - (\alpha+1)p_{i1}^{\alpha} \\
\frac{\partial \mathcal{L}_{DSC}}{\partial p_{i1}} &= -\frac{u'v - v'u}{v^2} \\
&= -\frac{1}{v^2}(1-(\alpha+1)p_{i1}^{\alpha}) \cdot y_{i1}((1-p_{i1}^{\alpha})p_{i1} + y_{i1} + \gamma) \\
&\quad + \frac{1}{v^2}(1-(\alpha+1)p_{i1}^{\alpha})\left((1-p_{i1}^{\alpha})\, p_{i1} \cdot y_{i1} + \gamma\right) \\
&= \frac{1}{v^2}((\alpha+1)p_{i1}^{\alpha} - 1)(y_{i1}^2 - (1-y_{i1})\gamma)
\end{aligned} \tag{4.4}$$

由于 $p_{i1} \in [0,1]$，则 $(\alpha+1)p_{i1} \in [0, \alpha+1]$，则可得 $((\alpha+1)p_{i1}^{\alpha} - 1) \in [-1, \alpha]$. 当 $\alpha = 1$ 时即为公式 (4.1)。损失函数收敛，即使得 $((\alpha+1)p_{i1}^{\alpha} - 1) = 0 \Rightarrow p_{i1} = (\alpha+1)^{\frac{-1}{\alpha}}$，要使得 $p_{i1} > \frac{1}{2}$，即 $\left(\frac{1}{\alpha+1}\right) > \left(\frac{1}{2}\right)^{\alpha} \Rightarrow \alpha < 2^{\alpha} - 1$，显然当 $\alpha > 1$ 时，不等式成立。 □

在语义匹配任务中，更关注应用相似性度量方法后，根据代码 AST 能找到多少匹配相关的文档注释。结合任务场景和模型结构，可得出损失函数为：

$$\begin{aligned}\mathcal{L}_{sm} &= \mathcal{L}_1 + \mathcal{L}_2 \\ &= \frac{1}{N}\sum_{i=1}^{N} y_i(1-\cos(T,D^+)) + (1-y_i)\max\left(0,\cos(T,D^-)-\beta\right) \\ &\quad + \frac{1}{N}\sum_{i=1}^{N}\log\left(1+\sum_{d\in D}\exp\left(-\gamma\Delta\left(T,D^+,d\right)\right)\right) \\ \Delta\left(T,D^+,D^-\right) &= \cos\left(T,D^+\right)-\cos\left(T,D^-\right)\end{aligned}, \tag{4.5}$$

该损失函数分为相似度损失和召回损失两部分，相似度损失采用孪生神经网络中类似的损失函数（公式 (3.15)），而召回损失实际上是一个基于所有负样本的对数损失，促使代码同正样本的相似度大于同采样的所有负样本的相似度。损失中的 β 表示负样本判定阈值，在实验中经过搜索与尝试，取 $\beta = 0.3$ 可得最佳效果。类似地，γ 为召回损失中的指数平滑因子。

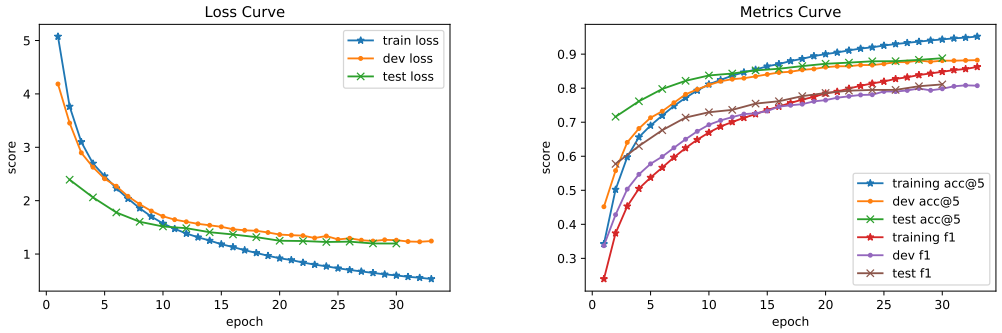
综合上述的数据集、超参数和损失函数，训练得到两个任务的模型，并基于 Code2Vec[2] 的编码器，在本文的测试数据集上得出基准数据。主要是根据已有的实验结果，以表格、统计图、可视化图等的视角，去剖析实验结果，并从**数据指标上、具体数据案例（Case 分析）、理论分析（原因及可解释性）**三个角度对实验结果进行分析。

图 4.4 所示为方法名生成任务训练过程中损失与指标曲线，从图 4.4(a) 曲线可以看出，模型在 epoch=20 前后，开始轻微震荡。通过观察验证集上的损失曲线，选择早停机制[11]停止训练，并换用 SGD 优化器在该迭代之后继续训练，直至曲线再次震荡。

4.4　实验结果分析

经过实验验证，得出表 4.6 的实验数据，从数据中可直观地看出，在方法名生成任务上，本文采用的基于静态路径对模型和动态路径对模型均能够取得高于基准模型的效果。同时也可看出，相较于 FCN 前馈网络，RNN 的编码器更能够处理 AST 路径中的时序特征，证明定义 3.2 、定义 3.1 和定义 3.3 正确有效。虽然 SPF w/SelfAtten 在 Top1 Acc 指标上相对于 SPF o/SelfAtten 表现略差，但 BLEU 指标仍有明显提升。鉴于 Acc 基于贪心搜索计算，这便说明贪心搜索策

[11] Early Stopping



(a) 方法名生成任务学习曲线　　(b) 方法名生成任务指标曲线

图 4.4: SPF o/SelfAtten 在方法名生成任务上的指标及损失

略相对更难找到最优序列。特别的，SPF w/SelfAtten 模型相较于基线模型相对

表 4.6: 方法名生成实验数据

模型	Acc@1	BLEU(%)
Code2Vec* (Alon 等, 2019)	0.477	28.910
SPF o/SelfAtten	**0.564**(↑ 18.24%)	**31.737**(↑ 9.78%)
SPF w/SelfAtten	0.498	**34.70**(↑ 20.03%)

提高了 20.03% 的 BLEU，从指标角度证明了 SPF w/SelfAtten 模型的有效性。从实验角度看，SPF w/SelfAtten 模型具有更高的时间和空间效率，每一次迭代仅为 SPF o/SelfAtten 的 $\frac{1}{7}$ 倍[12]。

把方法名生成任务作为预训练任务，得到较为稳定的编码器参数后，更换生成器的下游结构，重新训练语义匹配模型，得到表 4.7 的数据。召回指标从测试集中的全量数据计算，对每个代码 AST 同所有文档计算相似度，排序后选取 **TopK** 数据并根据匹配对计算得出 **TopK Recall** 值。除基准模型与 SPF w/SelfAtten 模型实验外，针对语义向量的池化方法还设计了消融实验，从数据可以看出，全局注意力机制在两个模型上均优于均值池化。并且可以得出同表 4.6 相同的实验结论，即本文的两个模型均能超越基线模型。

理论上分析，全局注意力机制能够得到更符合真实数据的分布，能够比先验的均匀分布表现更佳。详细地，从表 4.7 来看，SPF o/SelfAtten[13] 与 SPF w/SelfAt-

[12]在 “SPF o/SelfAtten” 尽可能占满显存，而 “SPF w/SelfAtten” 仅使用 $\frac{1}{2}$ 显存的条件下计算

[13]表示未使用自注意力机制

ten[14]相比，纵向来看，SPF w/SelfAtten 使用均值池化时比 SPF o/SelfAtten 更好，从中可以认为**自注意力机制能够具有更低的噪声**。而横向看，全局注意力机制与平均池化在 SPF o/SelfAtten 上相差不明显，这**表明无论是注意力机制还是平均池化，都将受到噪声干扰**，而不得不为一部分噪声也同样分配一些注意。即以均值池化为基准，SPF o/SelfAtten 模型中全局注意力与均值池化效果相差无几，这表示注意力得到的分布近似接近于均匀分布。与之相反，SPF w/SelfAtten 却有 2 个百分点左右的区分度，这表明实质上在噪声较少的情况下，全局注意力能比平均池化获得到更好的效果，同时也进一步说明了不使用自注意力机制存在噪声干扰问题。

表 4.7: 语义匹配实验数据

模型	Rec@5(%)		Rec@10(%)	
	GlobalAtten	Mean	GlobalAtten	Mean
Code2Vec* (Alon 等)	16.90	14.52	23.88	21.33
SPF o/SelfAtten	**21.72**	**21.68**	**29.61**	**29.45**
SPF w/SelfAtten	**27.05**	**25.56**	**36.24**	**34.79**

问题 (A)： 代码语义遭遇的长尾问题；

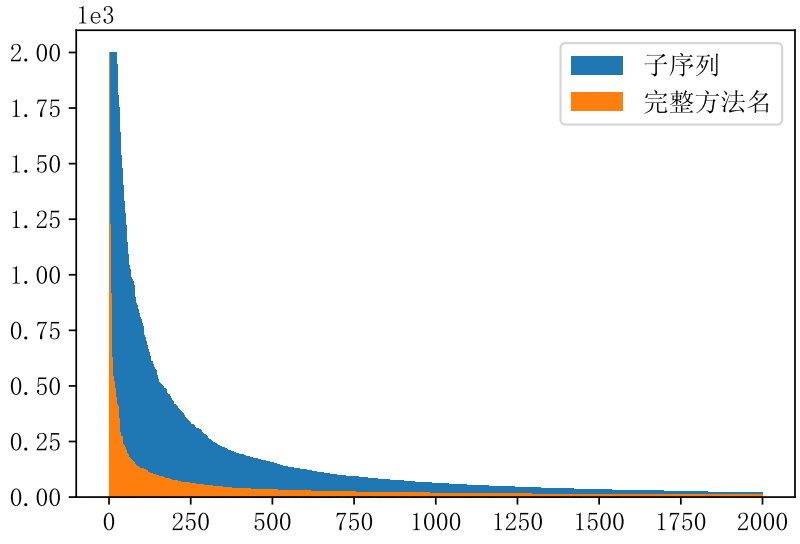


图 4.5: 原始方法名与子序列长尾效应（部分）

[14]表示使用了自注意力机制

显然，当不使用方法名子序列时，方法名称能够根据词库中的每个词任意组合，可设词库大小为 n，方法名平均长度为 k，则构成方法名词典最大为 $\binom{n}{k}$，并且有绝大部分都为低频词。经过统计，在训练数据中，方法名构成的词典高达 64,586，而经过子序列拆分处理的词典大小仅为 6,629。如图 4.5 所示为两种处理下方法名称的长尾效应示意图，从图中也可看出，经过子序列拆分后（与横轴面积更大部分），长尾效应相对原始方法名（与横轴面积更小部分）有较为显著的缓解。

问题 (B)：不全面的代码语法结构特征；

从实验数据上看，SPF o/SelfAtten 也取得了超越 Code2Vec 编码器的效果，获得了 9.78% 的相对指标的提升。从模型结构上分析，Code2Vec 编码器使用的全连接神经网络来编码 AST 路径，与 NLP 相似，全连接网络是平行结构，不具备先后关系的约束，无法提取 AST 中的模式特征。参考图 3.1 可知，代码的规则同 AST 路径节点的结构关系保持一致。从取得的实验结果来看，基于 RNN 的编码器确实能够提升传统基于全连接网络的特征抽取能力。

结合表4.6与表4.7综合来看，即便不使用自注意力机制对路径进行自动融合，也能够获得超越 Code2Vec 的表现，即在两个任务上，SPF o/SelfAtten 均获得了超越基线的表现。这表明在解决问题 (A)与问题 (B)后，模型性能得到有效提升。而在加入自注意力机制后，又更进一步地提升了模型的整体表现，这表明问题 (C)解决后，模型性能又能够明显提升。

问题 (C)：静态代码路径对存在噪声。

为了检验 SPF w/SelfAtten 的有效性，我们取出一个样本，并将其路径间自注意力所得的权重分配可视化为如图 4.6 所示的结果，其中颜色越深表示获得的关注更高，重要程度及融合增益越大。如图 4.6 为路径间自组合时某个样本的自注意力权重 $\boldsymbol{\alpha}$ 的分配情况，从图中可见，注意力权重主对角线及下方权值全为 0，这满足了在 Ground Truth 设定中的先序遍历顺序约束。根据权重分配的不同，每个叶子节点所在路径将根据自身编码自行选择组合度更高的其他叶子节点所在的路径。例如，有 $\{\alpha_{3,5}, \alpha_{3,9}\} > \{\alpha_{3,j}\}, \forall j \in [0, 11] \wedge j \neq 5 \wedge j \neq 9$. 这表明，节点 3 所在路径与节点 5 所在路径、节点 9 所在路径融合的增益更高。根据自注意力组合权重可知，对于任意一条路径，其会与其他路径生成一个信息融合权重 $\alpha_{i,j}$，并通过加权的方式将所有的信息融合到路径 i 中，表示 $l_i \rightarrow l_j, j \in (i, n]$ 的组合特征，l_i 表示第 i 个叶子节点。图中各线条用于区分不同的路径对组合，粗细表示组合权重，越粗表示该组合置信度越高。

问题 (C) 存在的简单证明　根据图 4.6 可以看出，权重分配值 $\alpha \geq 0.35$ 的数量大致为 13 个，因此可以认为对代码理解影响较大的路径为 13 条：(1) 0-2; (2) 1-2;

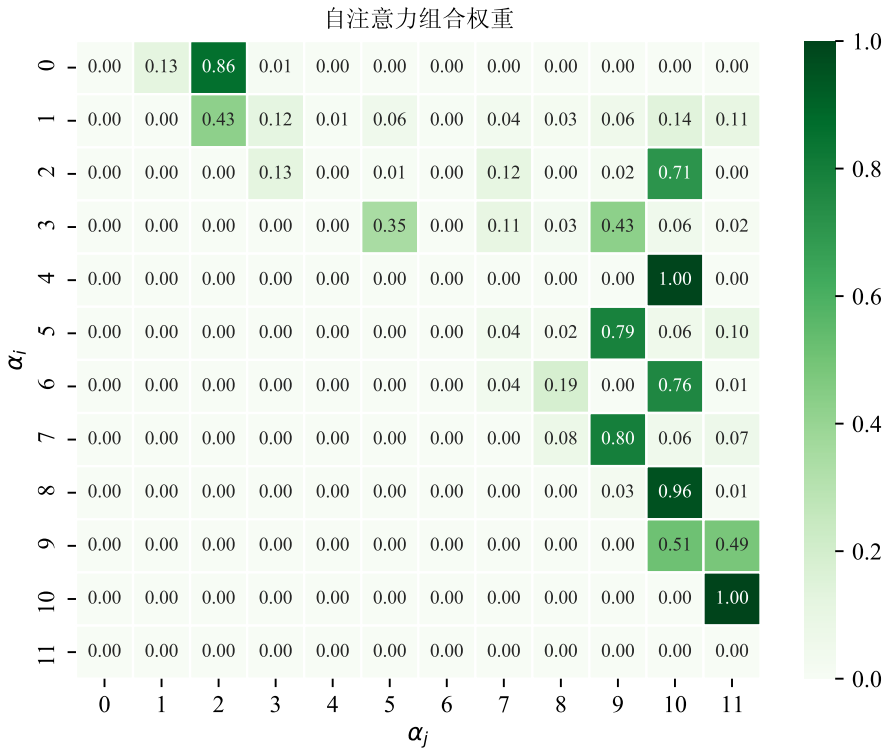


图 4.6: 自注意力权重分配图

(3) 2-10; (4) 3-5; (5) 3-9; (6) 4-10; (7) 5-9; (8) 6-10; (9) 7-9; (10) 8-10; (11) 9-10; (12) 9-11; (13) 10-11. 而以两两组合的方式所组成的路径，则有 $\frac{1}{2}n(n-1)=55, n=11$ 条，显然 $55>4\times13$，即引入的噪声为有效数据的 4 倍左右，且随着叶子节点的增多，该噪声比例会更大 (指数增长比线性增长的快)。综上，可以得出两两组合的路径带来的噪声，会随着叶子节点数量（AST 的宽度）的增加而快速增加。

为了更直观地分析自注意力机制的合理性，这里根据图 4.6 所示权重，可将该样本对应的 AST 路径可视化为图 4.7 所示的路径。从图 4.7 中可看出，自注意力机制能够有选择地着重于某一些特殊的模式，如 “／(④)” 的组合，则是一条由输入到输出的组合路径，而 “／(⑨)” 则是由条件判断到方法返回值的路径。从 AST 路径组合图中可以看出，通常贡献度较高的路径叶子节点的跨度较小，其中④跨度最大仅为 8 个节点。相反，在 “SPF o/SelfAtten” 模型中，数据被先验地任意两两组合，对于较先序顺序较前的叶子节点，产生了大量跨度较大的路径对，从而引入了过多噪声数据，对模型最终表现产生干扰。将图 4.7 与图 3.1 对照来看，经过自注意力机制得到的路径对具有一定的物理意义，同时也验证了定义 3.2 的正确性，说明本文依据的假设以及基本事实符合业务场景。综上可知，

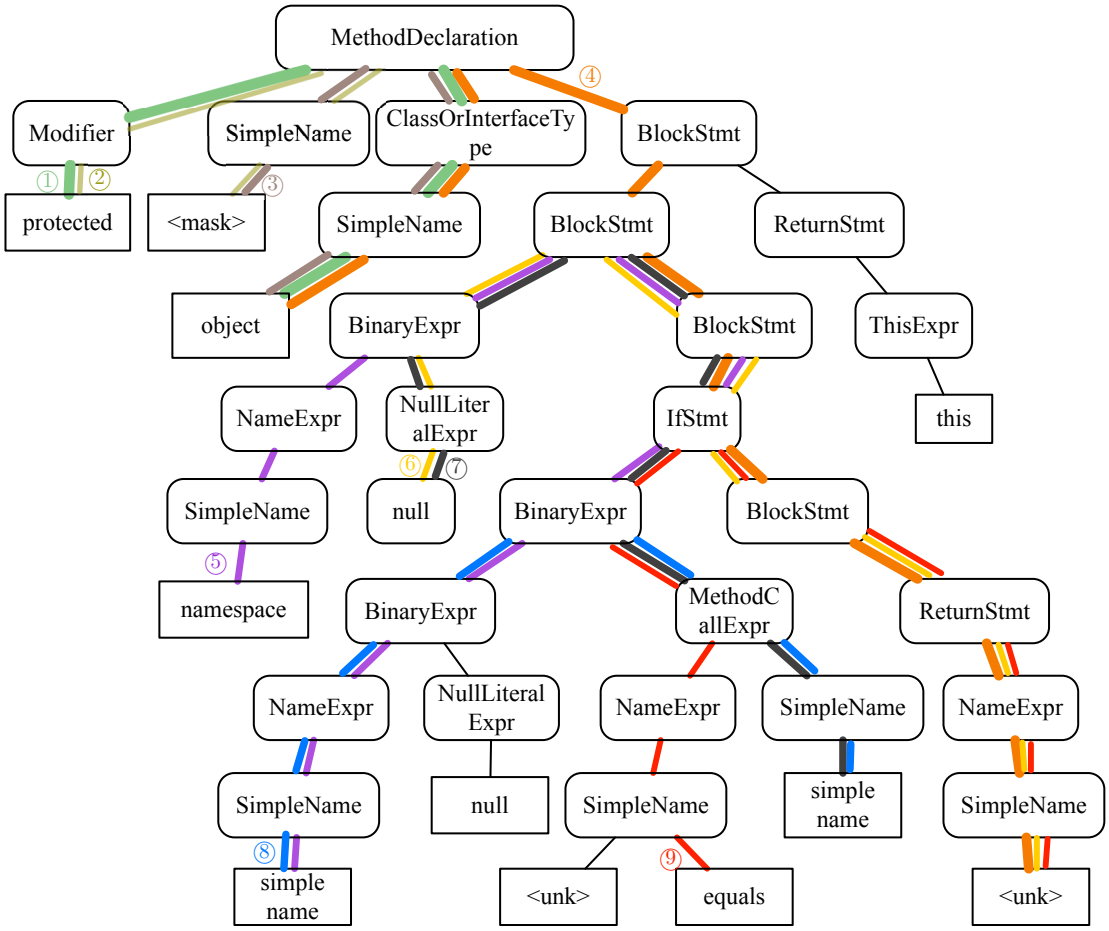


图 4.7: AST 路径组合示意图（部分）

静态路径对构造的数据集存在大量不可控的噪声，且随着方法体的复杂度升高，噪声越多。基于自注意力机制的路径对特征融合方案，解决了已有研究中存在的问题，获得的特征更加高效准确。

从方法论的角度来看，自注意力机制的路径对构造策略是合理并且有效的。同时，也说明自注意力机制不仅能够用于处理序列间的依赖关系（Transformer 结构），还可以用于特征融合过程，该方案为特征融合提出了一个全新的视角。由模型自动化地融合所需要的特征，能避免引入过多的先验知识，即减少特征工程的要求，更加端到端。

4.5　实验结论

通过对实验数据的分析，本文认为，本文提出的方法能够解决现有研究中暂未解决的问题。现有的工作存在三个主要问题：1）任务建模简单，数据标签

存在较为明显的长尾问题；2）未能采用时间效率高、准确性高的特征抽取方法；3）数据集定义存在较多噪声，速度慢且干扰大。本文详细分析了上述问题的存在，通过对数据集分析，本文提出的子序列拆分方法能够有效地缓解长尾效应，降低目标的词典大小。在实验过程中可以发现，基于 RNN 的网络相较于 FCN 具有更强的结构特征抽取能力，结合子序列方法，实现了从语法和语义两个角度对源代码建模目标，成功地将代码文本转化为对应的向量。额外地，本文针对数据集中的噪声，提出了一种更为简单的代码特征融合方法，基于自注意力机制的路径对融合方法，不仅在大大提升了时间效率，而且 “端到端” 地降低了数据噪声，显著提升了代码理解效果。将自注意力机制可视化，可得到 AST 及各条路径对模型的贡献，结果印证了假设：代码的语句对应了 AST 的若干路径。

从任务上看，本文构建的方法名生成和文本-代码语义匹配任务，验证了本文的目标：将代码的含义映射到文本空间。本文的实验所基于的数据定义、假设和基本事实，存在合理性，能够适应具体业务的需要。方法名生成任务的成功，可为注释生成、摘要生成等任务提供依据，而文本-代码语义匹配方法，可以应用到代码推荐、代码搜索、克隆检测等场景中，具有一定的实际应用价值。

从成果上看，本文利用大量的 API 文档重新构建了 Java 代码领域的词典，并基于该语料得到了预训练词向量，丰富了领域专业词汇，更具针对性地适配代码领域的相关任务。考虑代码特性和 AST 结构，本文提出了基于路径对的数据处理方法，并构建了用于方法名生成与语义匹配的数据集。此外，本文还将自注意力机制作为编码器的下游结构之一，成功地将 Transformer 应用到 RNN 结构的网络中，并取得了显著成效。

从数据上看，本文提出的网络 SPF w/SelfAtten 效果优于 SPF o/SelfAtten，SPF o/SelfAtten 优于基线模型。SPF o/SelfAtten 在方法名生成任务上 BLEU 分数达到了 31.74%，而 SPF w/SelfAtten 更是达到了 34.7%。类似地，在语义匹配任务中，SPF w/SelfAtten 的 Top5 召回率达到 27.05%，效果提升显著。同时也验证了全局注意力池化在表现上，要优于均值池化的观点。

不要停止预训练 [16] 说明了预训练在 NLP 任务中的重要性，本文在实验过程中也得出了类似的结论。本文的实验首先预训练了代码领域的词向量，相较于随机初始化，使用预训练词向量使得模型训练更加稳定。在使用随机词向量时，模型收敛更慢，且经常不收敛。在语义匹配任务上，也使用迁移学习思想，使用方法名生成任务中的编码器用于微调。实验发现，使用预训练好的编码器，更能帮助模型更快地收敛，且收敛情况更加稳定。由此也可推断，对于源代码相关的其他任务，使用本文提出的编码器结构，改变具体下游结构微调同样可以实现下游任务快速迁移，并取得相对较好的效果。

由于语义匹配任务使用了余弦相似度此类简单的 VSM 方法，因此在具体应用上，可以离线地处理所有源代码得到其语义表示向量。再获得到文本形式的 Query 后，可以仅对文本快速编码，得到文本向量后快速召回一批相关的源代码，实现文本至代码片段的查询。此时，源代码编码器便可视为一个“代码嵌入”组件。这表明，本文的实验找到了一个相对较好的映射函数 $f : AST \rightarrow \boldsymbol{V}$，使得代码能够用固定维度的向量表示。

4.6　本章小结

本章主要讨论了第三章中模型的具体实验细节和取得的实验结果。首先，本文利用开源项目的 API 文档，得到了各个场景下的领域词汇以及文档语料，训练获得初始词向量。类似地，基于开源项目提供地源代码，获得了每个方法的 AST 以及文档注释，根据第三章的定义，构建了用于方法名生成和语义匹配两个任务的数据集。通过分析本文提出的三个研究问题，结合实验数据证明了这三个问题的存在，根据实验取得的成果，可以发现本文提出的方案能够有效地解决存在的问题。

第五章　总结与展望

5.1　总结

本文主要聚焦于使用深度学习方法，实现对源代码内容理解，并使用一段向量表示。文章详细分析了当前领域的相关背景和工作现状，明确了研究目标。总的来说，本研究需要从语义、语法和模式三个层次，理解源代码中的含义，即将源代码表示成为既含有语义又含有语法特征的低维向量。结合应用场景，本文使用方法名生成和代码同文本的语义匹配两个任务共同评估代码理解的效果。

本文根据现有的研究方法分析，确立其中尚存在的问题，作为本文的研究方向。本文从现有的研究上总结出了三个主要问题：1) 代码潜在语义提取不准确问题；2) 代码语法结构抽取不完善问题；3) 代码文本到 AST 数据构造的噪声问题。围绕这三个问题，设计了对应的模型结构以及处理方法。由于需要从语法、语义和模式等层面理解代码，本文采用抽象语法树来结构化源代码，根据 AST 同代码的对应关系提出了路径对同代码语句对应的基本事实。此外，为避免人工数据标注，本文还提出代码同方法名称和文档注释含义相同的基本事实，用于构建两个代码理解下游任务。

现有研究由于未对 AST 叶子节点额外处理，导致语义信息缺失，本文进一步使用软件领域词典、API 预训练词向量和代码子序列编码器，将叶子节点中的文本拆分为词序列，获得更准确的代码语义信息。在基准 Code2Vec 方法中，其使用 FCN 结构处理各个节点，导致语法信息缺失，本文提出定义 3.1 和定义 3.2 ，结合使用 RNN 网络实现对 AST 序列中的语法依赖特征提取，即 SPF o/SelfAtten(Static Path Fusion without Self Attention) 网络。在本文实验过程中，发现定义 3.1 对应的静态路径对规模庞大，有明显的噪声，故而改进提出更简单的定义 3.3 和 SPF w/SelfAtten 网络，用于动态构造路径对抑制噪声。

代码方法名生成和代码文本匹配两个任务，均能证明 SPF o/SelfAtten 和 SPF w/SelfAtten 模型能够相对 Code2Vec 方法实现更准确的代码理解，并在下游任务上取得了显著提升。根据具体样本分析，可以认为 SPF 很好地改善了现有方法中的三大问题，使代码理解可以更准确地应用于代码检索、相似代码推荐等自动化软工任务。

综上，本文提出的 SPF 方法克服了现有问题，实现了准确地代码理解，获得了更全面的代码特征向量，从而支持更多的代码相关的任务。

5.2 未来工作展望

虽然本文的研究实现了既定目标，但是仍然存在较多不足。目前，本文的工作仍然只集中在理论实验阶段，未从工程角度对本文的任务和模型深入实践，**后续可将模型落地至实际的应用，进一步从工程视角验证本文的方案**。同时，本研究虽然解决了一部分现存的问题，但该方案也并非最优，并且仍有许多剩余的问题暂未解决。在数据的角度，本文将 AST 树结构处理为序列结构（即定义 3.1 的路径集合），这个过程将丢失一部分结构化特征。使用 RNN 的方案是为权衡模型的时间效率和复杂度而得出，最优的方案应是采用能够高效处理树或图结构数据的网络。

在方法的角度，当前图网络和无监督学习方法在诸多任务上都取得了成功。利用图网络处理 AST 具有巨大潜力，进一步提高代码理解效果。借助无监督学习方法，可以使模型更加关注代码数据，而无需将其他模态、领域的数据来指导代码理解，也可获得更广泛的训练数据。因此，**图网络结合无监督学习方法，可在垂直领域上获得更好的表现，这也可作为该工作后续的主要方向之一**。同样是序列模型，相较于 RNN 结构，得益于 Transformer 结构的强大特征抽取能力，使其在 NLP、CV 领域上均取得了重要成果。结合代码数据，将 **RNN 编码器替换为 Transformer 编码器**是一个必要尝试，本研究的后续工作也正在基于 Transformer 编码器改造本文提出的两个任务。

本文使用了迁移学习的思想，将较易收敛的生成任务作为难收敛的语义匹配任务的预训练任务，能够快速地迁移编码器结构与参数。现在的研究发现，多任务学习框架在某些场景下（如槽位填充与意图识别任务）能够提升任务整体上的表现，即要求模型能够学习多个任务通用的特征表示，更能增强模型的泛化能力。多任务学习经典的模型结构之一便是 Shared Bottom 结构，意在共享多个任务间的下游，独立上游任务特有结构，指导下游结构更加鲁棒和通用。本文的两个任务后续也可**使用共享编码器结构，将多个任务联合训练**，增强编码器的泛化能力。借助多任务的思想，使用门控机制**增强特征交互**，也能进一步提升语义匹配的效果。

语义匹配任务中，负样本的质量极大地影响了最终模型的效果。本文的语义匹配研究中，采用了一种较为简单的采样策略，使用在线采样，让模型尽可能地接触到更多的负样本，通过增强样本多样性来增大困难样本在训练过程的覆盖率。[17] 但在语义匹配领域，已经有越来越多的挖掘策略，如 TriHard 损失 [18]，MSML 损失 [58] 等。**借助机器学习中的 Boosting 策略，从损失或采样手段上改进负样本获取方式**，可以产生更高质量的负样本，提升效果。

参考文献

[1] Allamanis, M., Sutton, C., 2013. Mining source code repositories at massive scale using language modeling, in: Zimmermann, T., Penta, M.D., Kim, S. (Eds.), Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013, IEEE Computer Society. pp. 207–216.

[2] Alon, U., Zilberstein, M., Levy, O., Yahav, E., 2019. code2vec: learning distributed representations of code. Proc. ACM Program. Lang. 3, 40:1–40:29.

[3] Arora, S., Liang, Y., Ma, T., 2017. A simple but tough-to-beat baseline for sentence embeddings, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net. pp. 1–16.

[4] Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate, in: Bengio, Y., LeCun, Y. (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, pp. 1–15.

[5] Bavishi, R., Pradel, M., Sen, K., 2018. Context2name: A deep learning-based approach to infer natural variable names from usage contexts. CoRR abs/1809.05193. 1809.05193.

[6] Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022.

[7] Chaparro, O., Florez, J.M., Marcus, A., 2019. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empir. Softw. Eng. 24, 2947–3007.

[8] Chen, P., Sun, Z., Bing, L., Yang, W., 2017. Recurrent attention network on memory for aspect sentiment analysis, in: Palmer, M., Hwa, R., Riedel, S. (Eds.), Proceedings of the 2017 Conference on Empirical Methods in Natural Language

Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, Association for Computational Linguistics. pp. 452–461.

[9] Chen, X., He, K., 2020. Exploring simple siamese representation learning. CoRR abs/2011.10566. 2011.10566.

[10] Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., Sun, J., 2019. Detnas: Backbone search for object detection, in: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 6638–6648.

[11] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Moschitti, A., Pang, B., Daelemans, W. (Eds.), Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL. pp. 1724–1734.

[12] Chopra, S., Hadsell, R., LeCun, Y., 2005. Learning a similarity metric discriminatively, with application to face verification, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA, IEEE Computer Society. pp. 539–546.

[13] Elman, J.L., 1990. Finding structure in time. Cogn. Sci. 14, 179–211.

[14] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., 2020. Codebert: A pre-trained model for programming and natural languages, in: Cohn, T., He, Y., Liu, Y. (Eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020, Association for Computational Linguistics. pp. 1536–1547.

[15] Guo, S., Zhang, X., Yang, X., Chen, R., Guo, C., Li, H., Li, T., 2020. Developer activity motivated bug triaging: Via convolutional neural network. Neural Process. Lett. 51, 2589–2606.

[16] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., Smith, N.A., 2020. Don't stop pretraining: Adapt language models to domains and tasks. 2004.10964.

[17] He, T., Li, Y., Zou, Z., Wu, Q., 2019. L2R-QA: an open-domain question answering framework, in: Cui, Z., Pan, J., Zhang, S., Xiao, L., Yang, J. (Eds.), Intelligence Science and Big Data Engineering. Big Data and Machine Learning - 9th International Conference, IScIDE 2019, Nanjing, China, October 17-20, 2019, Proceedings, Part II, Springer. pp. 151–162.

[18] Hermans, A., Beyer, L., Leibe, B., 2017. In defense of the triplet loss for person re-identification. CoRR abs/1703.07737. 1703.07737.

[19] Hofmann, T., 1999. Probabilistic latent semantic indexing, in: Gey, F.C., Hearst, M.A., Tong, R.M. (Eds.), SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA, ACM. pp. 50–57.

[20] Huang, P., He, X., Gao, J., Deng, L., Acero, A., Heck, L.P., 2013. Learning deep structured semantic models for web search using clickthrough data, in: He, Q., Iyengar, A., Nejdl, W., Pei, J., Rastogi, R. (Eds.), 22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013, ACM. pp. 2333–2338.

[21] Jiang, L., Liu, H., Jiang, H., 2019. Machine learning based recommendation of method names: How far are we, in: 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019, IEEE. pp. 602–614.

[22] Kamiya, T., Kusumoto, S., Inoue, K., 2002. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Software Eng. 28, 654–670.

[23] Kim, Y., 2014. Convolutional neural networks for sentence classification, in: Moschitti, A., Pang, B., Daelemans, W. (Eds.), Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL. pp. 1746–1751.

[24] Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net. pp. 1–14.

[25] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R., 2020. ALBERT: A lite BERT for self-supervised learning of language representations, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net. pp. 1–17.

[26] Li, X., Sun, X., Meng, Y., Liang, J., Wu, F., Li, J., 2020. Dice loss for data-imbalanced NLP tasks, in: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J.R. (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, Association for Computational Linguistics. pp. 465–476.

[27] Liang, H., Sun, L., Wang, M., Yang, Y., 2019. Deep learning with customized abstract syntax tree for bug localization. IEEE Access 7, 116309–116320.

[28] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. Roberta: A robustly optimized BERT pretraining approach. CoRR abs/1907.11692. 1907.11692.

[29] Luong, T., Pham, H., Manning, C.D., 2015. Effective approaches to attention-based neural machine translation, in: Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., Marton, Y. (Eds.), Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015, The Association for Computational Linguistics. pp. 1412–1421.

[30] Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space, in: Bengio, Y., LeCun, Y. (Eds.), 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, pp. 1–12.

[31] Mou, L., Li, G., Zhang, L., Wang, T., Jin, Z., 2016. Convolutional neural networks over tree structures for programming language processing, in: Schuurmans, D.,

Wellman, M.P. (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, AAAI Press. pp. 1287–1293.

[32] Neamtiu, I., Foster, J.S., Hicks, M., 2005. Understanding source code evolution using abstract syntax tree matching. ACM SIGSOFT Softw. Eng. Notes 30, 1–5.

[33] Neculoiu, P., Versteegh, M., Rotaru, M., 2016. Learning text similarity with siamese recurrent networks, in: Blunsom, P., Cho, K., Cohen, S.B., Grefenstette, E., Hermann, K.M., Rimell, L., Weston, J., Yih, S.W. (Eds.), Proceedings of the 1st Workshop on Representation Learning for NLP, Rep4NLP@ACL 2016, Berlin, Germany, August 11, 2016, Association for Computational Linguistics. pp. 148–157.

[34] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., Ward, R.K., 2014. Semantic modelling with long-short-term memory for information retrieval. CoRR abs/1412.6629. 1412.6629.

[35] Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation, in: Moschitti, A., Pang, B., Daelemans, W. (Eds.), Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL. pp. 1532–1543.

[36] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L., 2018. Deep contextualized word representations, in: Walker, M.A., Ji, H., Stent, A. (Eds.), 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), Association for Computational Linguistics. pp. 2227–2237.

[37] Pu, W., Liu, N., Yan, S., Yan, J., Xie, K., Chen, Z., 2007. Local word bag model for text categorization, in: Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA, IEEE Computer Society. pp. 625–630.

[38] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., 2018. Improving language understanding by generative pre-training. OpenAI .

[39] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., 2019. Language models are unsupervised multitask learners. OpenAI blog 1, 9.

[40] Ramay, W.Y., Umer, Q., Yin, X., Zhu, C., Illahi, I., 2019. Deep neural network-based severity prediction of bug reports. IEEE Access 7, 46846–46857.

[41] Robertson, S., 2004. Understanding inverse document frequency: on theoretical arguments for IDF. J. Documentation 60, 503–520.

[42] Sajnani, H., Saini, V., Svajlenko, J., Roy, C.K., Lopes, C.V., 2016. Sourcerercc: scaling code clone detection to big-code, in: Dillon, L.K., Visser, W., Williams, L.A. (Eds.), Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, ACM. pp. 1157–1168.

[43] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Transactions on Neural Networks 20, 61–80.

[44] Schuster, M., Paliwal, K.K., 1997. Bidirectional recurrent neural networks. IEEE Trans. Signal Process. 45, 2673–2681.

[45] Serrano, L., Nguyen, V., Thung, F., Jiang, L., Lo, D., Lawall, J., Muller, G., 2020. SPINFER: inferring semantic patches for the linux kernel, in: Gavrilovska, A., Zadok, E. (Eds.), 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020, USENIX Association. pp. 235–248.

[46] Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G., 2014. A latent semantic model with convolutional-pooling structure for information retrieval, in: Li, J., Wang, X.S., Garofalakis, M.N., Soboroff, I., Suel, T., Wang, M. (Eds.), Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014, ACM. pp. 101–110.

[47] Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition, in: Bengio, Y., LeCun, Y. (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, pp. 1–14.

[48] Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S., 2010. A discriminative model approach for accurate duplicate bug report retrieval, in: Kramer, J., Bishop, J., Devanbu, P.T., Uchitel, S. (Eds.), Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010, ACM. pp. 45–54.

[49] Sun, Z., Zhu, Q., Mou, L., Xiong, Y., Li, G., Zhang, L., 2019. A grammar-based structural CNN decoder for code generation, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press. pp. 7055–7062.

[50] Tai, K.S., Socher, R., Manning, C.D., 2015a. Improved semantic representations from tree-structured long short-term memory networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers, The Association for Computer Linguistics. pp. 1556–1566.

[51] Tai, K.S., Socher, R., Manning, C.D., 2015b. Improved semantic representations from tree-structured long short-term memory networks. CoRR abs/1503.00075. 1503.00075.

[52] Tian, Y., Sun, C., Lo, D., 2012. Improved duplicate bug report identification, in: Mens, T., Cleve, A., Ferenc, R. (Eds.), 16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012, IEEE Computer Society. pp. 385–390.

[53] Tufano, M., Watson, C., Bavota, G., Penta, M.D., White, M., Poshyvanyk, D., 2018. Deep learning similarities from different representations of source code, in: Zaidman, A., Kamei, Y., Hill, E. (Eds.), Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, ACM. pp. 542–553.

[54] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need, in: Guyon, I., von

Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pp. 5998–6008.

[55] Vinyals, O., Toshev, A., Bengio, S., Erhan, D., 2015. Show and tell: A neural image caption generator, in: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, IEEE Computer Society. pp. 3156–3164.

[56] Wang, Y., Gu, Z., Wang, H., 2011. A survey of data mining softwares used for real projects, in: 2011 IEEE International Workshop on Open-source Software for Scientific Computation(OSSC' 2011), CAJEPH. pp. 94–97.

[57] Woo, S., Park, J., Lee, J., Kweon, I.S., 2018. CBAM: convolutional block attention module, in: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII, Springer. pp. 3–19.

[58] Xiao, Q., Luo, H., Zhang, C., 2017. Margin sample mining loss: A deep learning based method for person re-identification. CoRR abs/1710.00478. 1710.00478.

[59] Xu, C., Huang, W., Wang, H., Wang, G., Liu, T., 2019. Modeling local dependence in natural language with multi-channel recurrent neural networks, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press. pp. 5525–5532.

[60] Yu, A.W., Dohan, D., Luong, M., Zhao, R., Chen, K., Norouzi, M., Le, Q.V., 2018. Qanet: Combining local convolution with global self-attention for reading comprehension, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net. pp. 1–16.

[61] Zaremba, W., Sutskever, I., Vinyals, O., 2014. Recurrent neural network regularization. CoRR abs/1409.2329. 1409.2329.

[62] Zhang, J., Wang, X., Hao, D., Xie, B., Zhang, L., Mei, H., 2015. A survey on bug-report analysis. Sci. China Inf. Sci. 58, 1–24.

[63] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., Liu, X., 2019. A novel neural source code representation based on abstract syntax tree, in: Atlee, J.M., Bultan, T., Whittle, J. (Eds.), Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, IEEE / ACM. pp. 783–794.

[64] Zhang, W., Yoshida, T., Tang, X., 2011. A comparative study of tf*idf, LSI and multi-words for text classification. Expert Syst. Appl. 38, 2758–2765.

[65] Zhu, Y., Wang, G., 2019. CAN-NER: convolutional attention network for chinese named entity recognition, in: Burstein, J., Doran, C., Solorio, T. (Eds.), 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), Association for Computational Linguistics. pp. 3384–3393.

[66] 于世英, 袁雪梅, 卢海涛, 任家东, 李硕, 2013. 基于序列聚类的相似代码检测算法. 智能系统学报 8, 52–57.

[67] 尹刚, 王涛, 刘冰F, 周明辉, 余跃, 李志星, 欧阳建权, 王怀民, 2018. 面向开源生态的软件数据挖掘技术研究综述. 软件学报 29, 2258–2271.

[68] 李政亮, 陈翔, 蒋智威, 顾庆, 2021. 基于信息检索的软件缺陷定位方法综述. 软件学报 32, 247–276.

[69] 杨芙清, 2005. 软件工程技术发展思索. 软件学报 16, 1–7.

[70] 梅锋, 蔡子仪, 陆璐, 2020. 面向软件缺陷预测的树状结构编码方式. 计算机应用研究 37, 205–209.

[71] 郎大鹏, 丁巍, 姜昊辰, 陈志远, 2019. 基于多特征融合的恶意代码分类算法. 计算机应用 39, 2333–2338.

简历与科研成果

基本情况 邹智鹏，男，汉族，1996 年 3 月出生，江西省南昌市人。

教育背景

2018.9-2021.6 南京大学 软件学院 硕士

2014.9-2018.7 江西财经大学 软件与通信工程学院 本科

攻读工学硕士学位期间完成的学术成果

1. He T, Li Y, **Zou Z**, et al. L2R-QA: An Open-Domain Question Answering Framework[C]//International Conference on Intelligent Science and Big Data Engineering. Springer, Cham, 2019: 151-162.
2. Wang H, He T, **Zou Z**, et al. Using case facts to predict accusation based on deep learning[C]//2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2019: 133-137.

攻读工学硕士学位期间参与的专利

1. 何铁科，黎宇，**邹智鹏**，顾宇，陈振宇，史洋洋，“一种使用神经网络和机器学习排序算法的问答系统实现方法”，申请号:201811298287.5，已受理。

致　　谢

值至论文完成之际，我要对所有帮助和支持本文工作的老师、同学和家人致以由衷的感谢。

首先，应感谢我的研究生导师陈振宇老师，在论文的全过程，乃至整个研究生的学习和工作中，他始终以科学、严谨、客观和耐心的态度指导我开展工作和研究。陈老师以独到的学术眼光，指导我研究生的研究方向，时刻督促着我的研究进展，不断地发现和指出我各项研究存在的问题。整个论文的实验和写作，也极大地受益于陈老师的批评、建议以及修改。我在研究生期间取得的成果，离不开陈老师的悉心指导。此外，实验室的其他老师也给予了我莫大的帮助，包括何铁科老师、房春荣老师等，他们也都随时同我保持密切的沟通，关注我的研究方向，指出我研究上的不足，并提出了许多珍贵建议，更是为本文的研究方案、应用场景和论文结构的确立贡献了大量的建设性意见。

其次，我还应感谢学校、学院，尤其是智能软件工程实验室 (Intelligent Software Engineering Lab, iSE Lab)，他们为我提供了广阔的平台，才能有我巨大的成长。同时，实验室的同学也对我给予了莫大的关心和帮助，有入学时热情友好、答疑解惑的学长学姐，还有合作研究，一起工作的实验室好友，更有积极提供支持、建议和帮助的学弟学妹们，他们都给我研究生生涯留下了深刻印象，也为本文的工作做出了巨大牺牲和帮助，再次感谢他们的指导、关怀和帮助。

感谢在腾讯公司以及实习期间的各位同事，他们让我体会到了真实工业界的研究和工作方法，极大地提高了我的工程实践和理论水平。

除此之外，我还要感谢我家人的理解和帮助，他们为我的工作和学习创造了良好的条件，他们的支持令我能够更加全身心地投入到毕业论文的研究和写作中来。也正是他们的理解，使我能够直面论文中面临的各种困难和挑战，给了我充足的信心和毅力。

最后，我还要感谢本次参与论文评审和答辩的各位专家和老师们，你们的批评指正，使我的工作更加圆满和丰富，使论文观点更加全面，条理更加清晰。