华北计算技术研究所

研究生毕业论文开题报告

**基于深度学习的程序生成模型**

**研究与设计**

**Research and Design of Program Generation Model Based on Deep Learning**

学生姓名： 赵东杰

指导教师： 仇建伟

部 门： 系统一部

专 业： 计算机应用技术

2019年10月

目录

[1 选题背景和意义 1](#_Toc287665574)

[1.1 选题背景 1](#_Toc166072798)

[1.2 选题意义 1](#_Toc2118785581)

[2 国内外发展动态 2](#_Toc2107163804)

[3 研究目标 4](#_Toc1398592525)

[4 研究内容 4](#_Toc478795689)

[4.1 基础自然语言处理的研究 5](#_Toc755906434)

[4.2预训练语言模型的研究 5](#_Toc810571059)

[4.3程序表示方法的研究 5](#_Toc1152970342)

[4.4 数据集的研究 6](#_Toc411054993)

[4.5 模型评价指标的研究 6](#_Toc2003935442)

[5 关键技术 6](#_Toc177707251)

[5.1 通用预训练语言模型 6](#_Toc503151176)

[5.2 选择合适的程序表示方式 7](#_Toc1057223034)

[6 论文实施计划 8](#_Toc723936405)

[7 参考文献 8](#_Toc474490995)

# 选题背景和意义

## 选题背景

随着深度学习在2006年正式提出，标志着人工智能进入了一个新的阶段，其中深度神经网络在各个领域的实际应用，也极大的促进了人们生活的智能化。特别是数据量和计算机计算能力快速发展的背景下，神经网络在诸多学术领域变成最先进的技术，并且已经成功地在生产中得到部署，在自然语言处理[1]、图像处理[2]和语音处理[3]等方面都得到了广泛的应用。

然而作为诸多应用背后的开发人员，编写代码是基本功，但是编写冗长的代码也极大的消耗了开发者的耐心，深度学习却没有为程序员的基础开发工作带来实质上的改进。尽管常见的开发平台通常整合了代码补全工具，但是往往都是基于静态词频统计，然后将候选结果按照字典序排列，顺序靠前的候选项往往并不是开发人员所需要的，这样的代码补全在实际开发场景中反而可能增加开发人员的负担。

部门项目组为了满足部门综合业务的开发需求，设计并实现了综合业务可视化开发平台，满足了开发人员快速构建系统架构的需求。但是对于编码过程，综合开发平台相对于通用的开发环境并没有很大提升。

因此，提高开发过程中代码编写的自动化、智能化程度[4]就成为项目组目前需要解决的问题，其中最主要的问题在于提高代码补全中的准确性和完整性，以及根据自然语言生成相应代码两方面[5]。

## 选题意义

现代集成开发环境(Integrated Development Environment,简称IDE)为程序员提供了基本的拼写错误检查，预测函数名、关键字、方法等基本功能，通常给出的预测信息都是以提示的方式给出，并且按照字母排序的方式排列，这往往不符合程序员本身的意图，更多的时候，开发人员希望给出的预测是根据与代码功能的匹配度排序，从而能够更快的进行选择。

开发过程中的文档注释，往往包含了开发人员在设计中的诸多信息，如果能够利用开发人员给出的自然语言的注释信息，自动生成能够实现开发人员意图的代码，对于提高开发人员的开发效率，以及促进整个业务的开发进程，都会有很大的作用。

因此，本文希望基于深度学习模型，在现有的技术基础上，分析其特点及不足，从而研究设计相应的程序生成模型，为开发人员提供更符合自身开发习惯、更加规范、以及更加智能化的的代码补全等程序生成服务，从而提高开发人员的开发效率以及开发规范性，加快项目开发进度，促进部门整体业务的快速发展。

# 国内外发展动态

代码补全是现代IDE的重要组成部分，也是开发人员判断一个IDE是否好用的重要标准，同时代码补全也是程序生成最常见的技术。通过帮助开发人员预测方法名、关键字、属性等，代码补全对下一个Token给出预测，并以字母序进行排序，增加了开发人员选择的时间。传统代码补全方法主要基于两方面：一是根据利用静态类型信息，加上设计的各种启发式的规则来决定预测的Token，如Eclipse通过静态的类别信息给开发人员推荐方法，其中候补信息通过字母进行排序；二则是利用代码样例以及语义信息来进行Token的补全，如2010年Huo等人[6]通过称为BBC的技术，该技术通过对API进行排序和筛选，从而提高Eclipse本身基于类型的代码补全效果。但这两种传统的方法一般都要求人为的设计启发式的规则，因此限制了方法本身的发展。

深度学习的出现，改变了传统的代码补全的方式，通过从大量的代码中进行学习，深度学习可以了解到代码Token之间的概率分布，从而基于学习到知识，提高Token预测的准确率。目前通过深度学习进行代码补全的主要流程如下：

1. 训练阶段：通过从开源社区或者Github等开源代码库获取大量语料库，并通过代码解析器对源代码进行处理，如将代码转化为语法树或者Token序列，之后选择一个合适的深度神经网络模型，如语言模型，对语料库中的数据进行训练。而目前效果最好的语言模型包括BERT[7]、GPT以及GPT-2[8]等。
2. 代码补全阶段：在当前需要补全的的位置调用训练好的模型，该模型根据当前已经输入的代码片段来预测需要补全的Token。

其中主要工作包括：基于程序序列化特征的补全，如Hindle在2012提出的利用语言模型学习程序的顺序概率特征[9]，并据此对下一个Token进行预测；基于程序局部性特征的补全，如Tu等人[10]提出通过在已有的语言模型的基础上，添加缓存机制，从而来维护程序代码的局部信息；基于程序结构化特征的补全：如Liu[11]等人通过遍历语法树，对得到的语法树序列进行建模，或者Raychev等人[12]直接利用树结构对程序的语法树进行建模。

基于功能描述的程序生成，体现了从自然语言到程序语言的翻译过程。而自然语言的文本化以及文本对应的二义性，程序代码的复杂结构以及多样性，使得自然语言到程序语言的翻译过程成为一个难题。而深度学习正好可以用来解决这个问题。

Betagy等人[13]利用深度学习从自然语言生成IFTTT代码。IFTTT是If-This-Then-That的简写，相对于普通程序来说，IFTTT程序结构更加简单也更加容易学习该程序的结构规则。Gu等人[14]利用深度学习提出了DEEPAPI，一个根据自然语言生成API序列的工具，从而能够生成通用程序设计语言比如Java、Python等代码片段。Cai等人[15]通过将传统查询解析方法与深度学习方法相结合的方式提高了SQL生成的准确率。

目前程序生成的深度学习模型主要包括两种，一种是基于语言模型，另一种是基于机构化代码生成网络。语言模型通常直接将解析过的代码当做文本作为输入，对文本的概率分布来进行建模。常见的语言模型有N-gram模型和循环神经网络(Recurrent Neural Network,简称RNN)。其中N-gram可以有效的学习代码的上下文，但是对代码的语义信息却无能为力。而RNN可以捕捉句子中词与词之间的规律，因此许多N-gram与RNN结合的模型，以及RNN的变体都相继被提出并得到了广泛的应用，包括Raychev等人[16]通过将N-gram与RNN结合的方法，在Java中进行API级别的补全，以及长短记忆模型(Long Short-Term Memory,简称LSTM)等。结构化代码生成网络则是通过对代码的强结构性进行结构化的建模，具体又分为基于语法树进行建模以及基于图网络进行建模。如Dong[17]通过提出一种树解码器SEQ2TREE，将编码器的结果通过RNN生成相应的Token。而Allamanis等人通过图的网络来表示代码结构和语义特征。

高质量的代码数据集是进行深度学习的前提，在图像和自然语言处理处理方面都有各自公开的数据集，如图像处理的ImageNet，而在程序语言中，这方面的数据集非常少。目前已有的一些公开的数据集包括：Xing等人[18]提供的从GitHub中抽取的Java方法以及相应的JavaDoc描述，Bhoopchand等人[19]提供的GitHub上fork数前949以及星级大于100的Python项目。目前这些数据集大多来自于GitHub，同时为了保证代码质量，都会对star数以及fork数目进行一定的限制，从而抽取质量较高的代码片段加入数据集之中。

目前国内外已经实现商用的基于深度学习的智能代码补全工具包括Kite、TabNine，以及aiXcoder。其中Kite是由来自MIT和斯坦福的开发人员组成的团队，主要是针对Python语言的智能开发插件，它提供了整行代码补全，代码片段补全以及直接在编辑器中查询官方文档等功能，来加速Python代码的开发速度。而TabNine主要是由来自加拿大的大四学生Jackson开发完成，并在主流开发工具中都已上线，作者是基于OpenAI提供的GPT-2模型，并通过GitHub上上百万的代码文件对模型进行优化，从而实现了对多达23种编程语言的支持。aiXcoder的开发人员来自于北京大学高可信软件技术教育部重点实验室，aiXcoder融合了基于序列的程序代码语言模型、基于抽象语法树和程序逻辑关系的图神经网络等方法，共同打造一个完整的系统，从而能够更准确地进行代码生成。同时aiXcoder也主动与其他两款应用进行了对比，结果显示aiXcoder可以用更少的键盘输入次数来完成相同的代码输入。而在使用方面，因为Kite只针对Python语言，所以只有单一的版本。aiXcoder和TabNine都是基于多种语言的，其中aiXcoder根据不同的语言实现了不同的模型，从而提供不同的版本，而TabNine因此是基于通用语言模型进行开发，所以对于不同的语言，都提供了统一的版本。

# 研究目标

基于所内创新基金项目——综合业务信息系统可视化开发平台，通过预训练通用语言模型与基于抽象语法树的结构化网络相结合的方法，研究并设计程序生成模型，并对模型的实际效果进行评测，从而能够为部门开发人员提供更加准确、智能化的程序生成服务，提供开发效率。

# 研究内容

为了能够顺利的对程序生成模型进行研究与设计，主要对以下五个方面进行研究：基础自然语言处理(Natural Language Processing,简称NLP)的研究、主流预训练语言模型的研究、程序生成模型的研究、数据集处理的研究以及模型评价指标的研究。

## 4.1 基础自然语言处理的研究

基于深度学习的程序生成本质上依旧是从大规模代码中学习知识并最终进行应用，而程序代码除了它本身的特点之外，本质上依旧是文本，因此在NLP流程中的基本技术，如词向量、词性标注、命名实体识别、LSTM、Attention、Transformer架构等，都会在程序生成的研究中起到关键作用，对NLP基础知识的掌握程度，直接影响到后续研究能不能进行顺利展开。

## 4.2预训练语言模型的研究

在NLP中，各种预训练语言模型层出不穷，如ELMO[20]、GPT、BERT、Transformer-XL[21]、GPT-2等，这些通用的预训练语言模型在各种NLP任务上一次次的刷新了上限。从ELMO长短期记忆模型到GPT的单向Transformer架构的成功应用，再到BERT模型中将双向Transformer架构[7]应用到模型当中，用全Attention[22]的结构代替了LSTM，抛弃了传统编码器——解码器模型中必须结合RNN等模型的固有模式，成功减少了计算量以及提高了并行效率。而近期GPT-2的发布更是又刷新了一系列自然语言任务的记录，可以完成阅读理解、、问答、机器翻译等多项不同的语言建模任务。

程序语言与自然语言在很多方面都有着类似的地方，因此如何将这些强大的模型应用到程序生成技术当中，也就成了至关重要的问题。

## 4.3程序表示方法的研究

不同的程序生成任务需要根据其需求采用不同的程序表示方式，然后根据不同的程序表示方式构建相应的模型，从而完成相应的任务。基本的基于深度学习的程序生成模型包括三类：基于序列的程序生成模型、基于结构的程序生成模型以及基于执行的程序生成模型。

在基于序列的生成模型中又包括基于字符和基于Token的程序生成模型，其中基于字符的方法，在构建词汇表的时候不会产生词表之外(Out of Vocabulary,简称OoV）的问题，如Cummins等人[23]根据基于字符序列模型构建的代码生成工具CLgen。但是基于字符会大大的增加序列的长度，同时无法捕捉到程序的更多语义信息。因此基于Token序列的程序模型被应用到代码补全的工作中，如White等人[24]提出的基于RNN的对程序语言的Token序列进行建模，并成功的应用到了Java语言的代码补全任务中。但基于Token序列的程序模型会丢失程序中的结构信息，因此不能对程序结构进行建模。

在基于结构的程序模型中，抽象语法树(Abstract Syntax Tree,简称AST)可以有效的表示程序的语法结构及其内容，被广泛地应用到了程序生成的相关任务中。在基于AST的程序生成模型中，程序被解析为AST并对它进行遍历从而得到节点序列，之后再根据节点序列进行建模，从而更加有效的利用程序本身的结构性。而通过在AST上增加相应的边从而构建程序图，再通过构建图神经网络对程序的结构和数据流进行建模，也可以完成相应的程序生成任务。因此基于图的程序 生成模型可以看做基于AST模型的扩展，通过增加更多的信息，使得相应的图神经网络具有更强的建模能力。但是同时复杂的图结构也存在着难以训练的问题。

## 4.4 数据集的研究

所有的深度学习应用，最基础的资源就是大量高质量的数据，高质量的、规范的代码数据集更有利于深度学习网络学习，在程序生成领域，公开的数据集与图像处理或者语音处理等领域相比，不管是数量还是质量都有很大差距。如前文提到的公开的数据集，都存在着一定噪声，如代码编写不规范，以及由于重载或者重写机制导致的模型很容易出现过拟合现象。如何设计规则对数据进行清洗，从而将它们应用到自己的模型构建当中去，是需要仔细思考的问题。

同时，如何将部门内部开发人员的代码数据进行处理并用于模型的训练和调优，从而使模型最终达到的效果能够更加符合实际开发人员的需求。

## 4.5 模型评价指标的研究

目前在程序生成方面，现有文献中没有一个统一的来评判程序生成模型能力的指标，通常采用的指标有机器翻译方面的BLEU指标、信息检索方面的MRR指标和预测下一个token的准确率三个方面，这三个指标没法进行直接的比较，并且这些指标的高低与生成的代码质量高低之间的关系也并不能清晰地表达。所以如何选择一个合适的指标来对模型的实际效果进行评测，对研究后期验证阶段至关重要。

# 关键技术

## 通用预训练语言模型

Attention机制由深度学习三巨头之一Bengio团队在2014提出并在深度学习的各个领域得到广泛应用，例如在计算机视觉中Attention机制被应用于捕捉图像的感受野，在NLP中用于定位关键Token及特征。而完全采用Attention机制组成的Transformer架构在最新的如BERT以及GPT语言模型中，在NLP的11项任务中都取得了效果的大幅度提升。

传统RNN的计算限制为顺序的，即RNN相关算法只能从左到右或者反过来依次计算，这样就会带来两个问题：一是时间t的计算依赖于时间t - 1时刻的计算，从而严重限制了模型的并行化；而是顺序计算中信息会有一定程度的丢失，尽管有相应的LSTM机制来缓解远距离依赖问题，但是对于长期的依赖，仍然没有好的方法可以解决。而Transformer架构中采用的Attention机制解决了这两个方面的问题，通过Attention机制，将序列中任意两个位置任意的距离缩小为了一个常量，其次采用了非顺序结构，从而提高了并行性，更适合现代GPU框架。

目前最先进的语言模型包括GPT、BERT与GPT-2，这些模型通过大规模无监督训练从大量高质量的语料集中学习了大量的语义知识，如何将这些模型应用到程序生成模型，是解决程序生成问题的关键所在。

## 选择合适的程序表示方式

不同的程序表示方式在构建程序生成模型的过程中有着极其重要的作用，对于不同的程序生成任务，不同的表示方式有着不同的应用。在代码补全方面，主要有基于Token级别和基于AST级别的模型，并通过RNN，CNN等应用到实际的代码补全任务中去。而在开发人员的自然语言描述，生成程序代码的任务中，主要包括基于AST和基于图的表示方法，其中基于图的表示方法是在程序AST的基础上增加相应的边来构建程序图，并据此构建图神经网络来对程序以及数据流进行建模。

如TabNine采用的基于通用预训练语言模型GPT-2构建的应用，即是基于Token序列进行的，因此可以在多达23种语言上实现更加智能化的效果，而北京大学开发团队开发的aiXcoder则是采用通用语言模型和图神经网络相结合的方法，能够在单一语言的程序生成方面取得更好的效果。因此选择正确合适的程序表示方式，对程序模型的构建也至关重要。

# 论文实施计划

毕业设计任务时间划分为五个阶段：

|  |  |
| --- | --- |
| 第一阶段  （2019.09.01-  2019.10.31） | 收集整理资料，对毕设题目进行确定，对整个课题研究进行初步的了解和掌握。对收集的资料进行分类、阅读。归纳相关文献。 |
| 第二阶段  （2019.11.01-  2020.05.31） | 通过收集整理材料，对相关数据进行处理，对所收集的材料进行系统性的学习。充分认识相关研究课题的关键难点问题。在学习的过程，完成小论文的发表。 |
| 第三阶段  （2020.06.01-  2020.10.31） | 对相关模型进行设计与实现。 |
| 第四阶段  （2020.11.01-  2021.01.31） | 对模型进行测试与验证。 |
| 第五阶段  （2021.02.01-  2021.03.31） | 完成毕业设计和毕业论文。 |

# 参考文献

[1] Bordes A, Glorot X, Weston J, et al. Joint learning of words and meaning representations for open-text semantic parsing[C]//Artificial Intelligence and Statistics. 2012: 127-135.

[2] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[3] Bordes A, Glorot X, Weston J, et al. Joint learning of words and meaning representations for open-text semantic parsing[C]//Artificial Intelligence and Statistics. 2012: 127-135.

[4] 胡星, 李戈, 刘芳, 等. 基于深度学习的程序生成与补全技术研究进展[J]. 软件学报, 2019, 30(5): 1206-1223.

[5] 刘芳, 李戈, 胡星, 等. 基于深度学习的程序理解研究进展[J]. 计算机研究与发展, 2019, 56(8): 1605-1620.

[6] Hou D, Pletcher D M. Towards a better code completion system by API grouping, filtering, and popularity-based ranking[C]//Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering. ACM, 2010: 26-30.

[7] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[8] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners[J]. OpenAI Blog, 2019, 1(8).

[9] Hindle A, Barr E T, Su Z, et al. On the naturalness of software[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 837-847.

[10] Tu Z, Su Z, Devanbu P. On the localness of software[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 269-280.

[11] Liu C, Wang X, Shin R, et al. Neural code completion[J]. 2016.

[12] Raychev V, Bielik P, Vechev M. Probabilistic model for code with decision trees[C]//ACM SIGPLAN Notices. ACM, 2016, 51(10): 731-747.

[13] Beltagy I, Quirk C. Improved semantic parsers for if-then statements[C]//Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2016: 726-736.

[14] Gu X, Zhang H, Zhang D, et al. Deep API learning[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016: 631-642.

[15] Cai R, Xu B, Yang X, et al. An encoder-decoder framework translating natural language to database queries[J]. arXiv preprint arXiv:1711.06061, 2017.

[16] Raychev V, Vechev M, Yahav E. Code completion with statistical language models[C]//Acm Sigplan Notices. ACM, 2014, 49(6): 419-428.

[17] Dong L, Lapata M. Language to logical form with neural attention[J]. arXiv preprint arXiv:1601.01280, 2016.

[18] Hu X, Li G, Xia X, et al. Deep code comment generation[C]//Proceedings of the 26th Conference on Program Comprehension. ACM, 2018: 200-210.

[19] Bhoopchand A, Rocktäschel T, Barr E, et al. Learning python code suggestion with a sparse pointer network[J]. arXiv preprint arXiv:1611.08307, 2016.

[20] Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations[J]. arXiv preprint arXiv:1802.05365, 2018.

[21] Dai Z, Yang Z, Yang Y, et al. Transformer-xl: Attentive language models beyond a fixed-length context[J]. arXiv preprint arXiv:1901.02860, 2019.

[22] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.

[23] Cummins C, Petoumenos P, Wang Z, et al. Synthesizing benchmarks for predictive modeling[C]//2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2017: 86-99.

[24] White M, Vendome C, Linares-Vásquez M, et al. Toward deep learning software repositories[C]//Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 2015: 334-345.