



改进的神经语言模型及在代码提示中的应用

A Modified Neural Language Model and Its Application in
Code Suggestion

张献 贵可荣

(海军工程大学 武汉 430033)

主要内容

- 研究背景
- 预备知识
- 本文方法
- 实验与结果分析
- 近期工作

研究背景

近年来，伴随软件的更新演化、开源思想的深入，软件仓库，如 GitHub、Stack Overflow 等，累积了海量的代码和其他元数据。这些日益扩增的软件制品促使着新的、数据驱动的方法分析软件。目前已有一批重要成果引入 **自然语言处理** (natural language processing, NLP)、**机器学习** 等方法解决不同软件分析任务。可以看出，这些技术的应用正成为近年来软件分析的热点内容。

- [1] 王千祥, 张健, 谢涛, 等. 软件分析: 技术、应用与趋势/ CCF 2015-2016 中国计算机科学技术发展报告[M]. 北京: 机械工业出版社, 2016: 55-113.
- [2] 江贺, 彭鑫, 魏俊, 等. 软件智能化开发技术的进展与趋势/ CCF 2016-2017 中国计算机科学技术发展报告[M]. 北京: 机械工业出版社, 2017: 88-115.
- [3] 陈海波, 刘杰, 顾荣, 等. AI 与系统软件的深度融合研究进展与趋势/ CCF 2016-2017 中国计算机科学技术发展报告[M]. 北京: 机械工业出版社, 2018: 100-146.
- [4] Allamanis M, Barr E T, Devanbu P, et al. A Survey of Machine Learning for **Big Code and Naturalness**[J]. ACM Computing Surveys, 2018, 51(4): Article 81. (arXiv:1709.06182)
- [5] 刘斌斌, 董威, 王戟. 智能化的程序搜索与构造方法综述[J]. 软件学报, 2018, 29(08): 2180-2197.
- [6] 张峰逸, 彭鑫, 陈驰, 等. 基于深度学习的代码分析研究综述[J]. 计算机应用与软件, 2018, 35(06): 9-17+22.
- [7] 张猷, 賁可荣. 深度学习方法在软件分析中的应用[J]. 计算机工程与科学, 2017, 39(12): 2260-2268. (截止 2017 年 5 月)

语言模型 (language model) 是 NLP 领域中的一类重要模型，其通过语料库学习对文本段的发生概率进行估计，进而刻画出语言规律。目前，该模型已在自动分词、句法分析和机器翻译等问题中得到了广泛应用。

近些年，不少研究者开始尝试采用语言模型来分析程序语言/软件代码，应用包括代码提示 (code suggestion)、代码补全 (code completion)、API 推荐 (application programming interface recommendation)、编程规范 (coding convention) 提示、缺陷检测、程序合成 (Program Synthesis)、伪代码自动生成等。

近年来，迅速兴起的深度学习方法为克服传统语言模型固有的维数灾难问题，目前已在 NLP 领域取得了显著进展。

本文利用深度学习方法对程序语言进行建模，依据任务特点提出了一种改进的循环神经网络 (recurrent neural network, RNN) 语言模型 *CodeNLM*，并以代码提示任务为例对模型应用进行了验证。考虑到现有模型仅学习代码数据，信息利用不充分，本文提出了**附加信息引导策略**，通过非代码信息的辅助提高了代码规律的刻画能力。

预备知识

语言模型旨在刻画语言规律，其通过语料库学习对序列的联合概率分布进行估计，是处理语言任务的一类基础模型。

形式化地，给定一个语言序列 $S = w_1 w_2 \cdots w_l$ ，那么序列 S 的发生概率可用其联合概率表示：

$$\begin{aligned} P(S) &= P(w_1)P(w_2 | w_1) \cdots P(w_l | w_1, w_2, \cdots, w_{l-1}) \\ &= \prod_{i=1}^l P(w_i | w_1, \cdots, w_{i-1}) \end{aligned} \tag{1}$$

估计 $P(S)$ 的方法有多种，目前最为常用的是 n -gram 模型和神经语言模型 (neural language model)

(1) n -gram 语言模型

n -gram 模型是此类模型的一个典型代表，为简化计算，其给定了一个马尔科夫假设：当前词的出现概率仅仅与前面 $n-1$ 个词相关，因此式(1)被近似表示为：

$$P(S) = \prod_{i=1}^l P(w_i | w_1, \dots, w_{i-1}) \approx \prod_{i=1}^l P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) \quad (2)$$

对于式(2)，常采用频数计数法估算其中的条件概率：

$$P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \dots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \dots, w_{i-1})} \quad (3)$$

虽然 n -gram 模型对概率计算进行了简化, 但其依然难以学习长距离信息。这主要是由于模型将语言单元视为孤立的符号, 无法有效地利用词间的语义关系, 造成维数灾难问题。例如, 考虑一个大小为 2000 的词汇表, 令序列最大长度为 10, 那么不同的语言序列就约有 10^{33} 种可能, 因此 10-gram 模型在式(3)的频数统计中, 将造成数据稀疏和内存爆炸问题。为此, 实际应用中常选择 5-gram 规模以下模型。

(2) 神经语言模型

神经语言模型借助神经网络估计序列的联合概率分布，通过利用词间语义相似性克服传统模型带来的维数灾难问题。

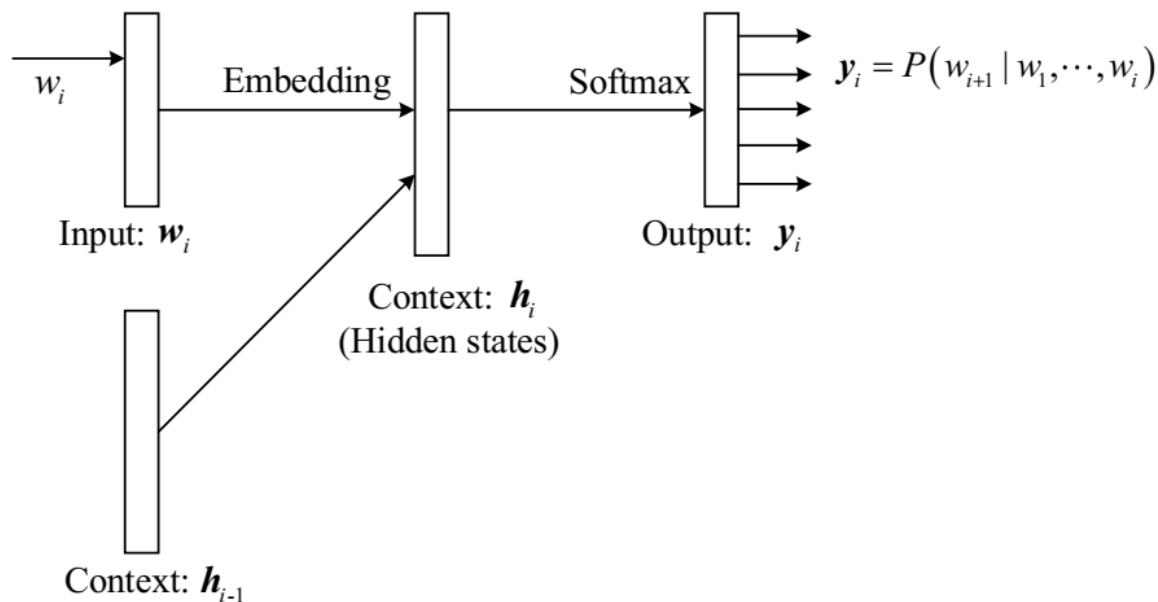



Figure 1. The scheme of basic RNN language model



神经语言模型一方面利用词嵌入（word embeddings）技术将离散的语言符号映射为连续的一组矢量，语义相似的词将拥有相似的矢量，增强了模型对词间语义关系的学习能力；另一方面借助高度非线性的神经网络拟合序列的发生概率，可以更灵活地刻画语言规律；此外 RNN 等网络的兴起，还使得模型在长程依赖（long-term dependency）学习方面具有显著优势。

代码提示 (code suggestion) 作为一项重要技术常集成于软件开发环境，其通过即时推荐候选代码辅助编程任务。

```
public class JDBCTest
{.....
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.out.println(e. ?

        .....

    try {
        Category log = Category.getInstance("main");
        log.debug("Debug 1");
        Thread.sleep(500);
        log.info("info 1");
        Thread.sleep(500);
        log.warn("warn 1");
        Thread.sleep(?
```

提示候选项:

1. toString ✓
2. getMessage
3. getClass
4. getName
5. printStackTrace

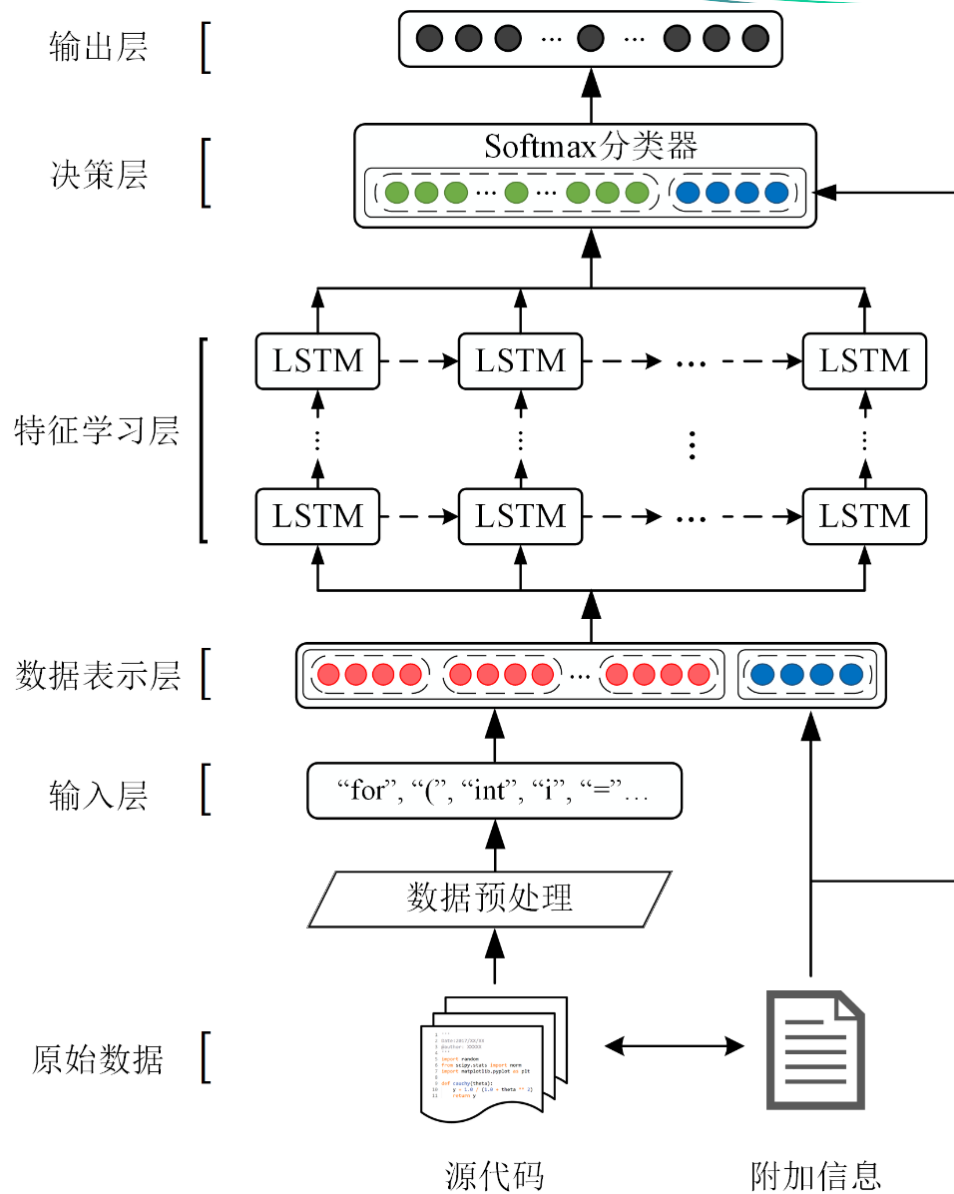
提示候选项:

1. 500 ✓
2. 200
3. 100
4. 10
5. 50

改进的神经语言模型（本文方法）

为增强代码特征的学习能力,本文提出了一种改进的 RNN 语言模型 CodeNLM。为改善现有语言模型在数据学习和网络结构设计方面的盲目性,本文提出两种改进策略:

- 1) 考虑到现有模型仅学习代码数据,信息利用不充分,本文提出了附加信息引导策略,通过非代码信息的辅助提高了代码规律的刻画能力。
- 2) 针对语言建模任务特点,提出了节点逐层递增策略,通过优化网络结构改善了信息传递的有效性。



考虑到现有的语言模型在数据学习和网络结构设计方面仍具有一定盲目性，本文提出两种改进策略，以增强模型对软件任务的理解、改善程序语言建模能力。

附加信息引导策略：

现有语言模型大都从源代码数据中提取知识，而有助于解决软件任务的部分样本信息(如与项目的对应关系、软件类型、软件重要性等级)难以在代码中体现，导致模型未能对软件进行充分分析。为克服这一问题，本文提出将任务相关的非代码信息融入网络，引导模型更准确地刻画语言特征，提高原有网络对数据学习的针对性。需要指出，应依据具体应用任务，抽取不同附加信息。下文将以代码提示任务为例进行阐述。

具体地，该策略首先将抽取的附加信息以实值向量形式表示，然后分别同词向量输入和 softmax 层输入相融合，用于辅助网络学习与决策。

$$\begin{array}{l} \mathbf{x}_t^1 = \mathbf{w}_t \\ \mathbf{x}_t^{softmax} = \mathbf{h}_t^{N_{layer}} \end{array} \longrightarrow \begin{array}{l} \tilde{\mathbf{x}}_t^1 = [\mathbf{w}_t \quad \mathbf{p}_t] \\ \tilde{\mathbf{x}}_t^{softmax} = \left[\mathbf{h}_t^{N_{layer}} \quad \mathbf{p}_t \right] \end{array}$$

代码提示旨在为编程人员提供准确的代码候选项，以提高开发效率。由于不同的软件项目拥有不同的应用背景及开发过程，致使反映在代码产品上，存在统计差异。因此，为使模型更精准地捕获代码规律、更准确地完成代码提示，本文将**样本与项目的关系信息**融入网络。针对代码提示应用，附加信息即为量化后的项目名称/ID，其量化使用词嵌入技术。



节点逐层递增策略：

在神经网络结构设计方面，目前缺乏经验性总结，导致实验盲目、低效。本文通过分析神经语言模型特点，提出一种经验性的网络规模设置方法：节点逐层递增策略，即由输入层到特征学习层再到输出层的网络节点数按递增模式设置：

$$\dim_{\text{in}} < \dim_{\text{h}}^1 < \dim_{\text{h}}^2 < \cdots < \dim_{\text{h}}^{N_{\text{layer}}} < \dim_{\text{out}}$$

这一经验策略有悖于常见的 DNN 结构设计方法，即整体上网络节点逐渐递减或节点先增后减的设计模式，例如经典的 LeNet5、AlexNet 和 GoogLeNet 等 DNN。

LSTM 单元由 Schmidhuber 等提出，通过使用内存门控机制防止梯度消失问题，克服了早期 RNN 难以记忆长程依赖的问题。LSTM 单元的数学模型为：

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

其中 $\sigma(x) = 1 / (1 + e^{-x})$ 和 $\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$ 为激活函数； \mathbf{W} 和 \mathbf{b} 分别表示权重矩阵和偏置向量； \odot 表示按元素相乘，即 Hadamard 积； $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ 分别表示输入门、遗忘门和输出门操作； $\mathbf{x}_t, \mathbf{c}_t, \mathbf{h}_t$ 分别表示网络输入、记忆单元状态和隐含层状态； t 表示展开时间。

实验与结果分析

在验证实验中，针对 9 个 Java 项目共 203 万行代码，CodeNLM 得到的**困惑度指标**明显优于对比的 n-gram 类模型和神经语言模型，在代码提示应用中得到的**平均准确度**（MRR 指标）较对比方法提高 3% ~ 24%。实验结果表明，CodeNLM 能较好地进行程序语言建模，具有较强的**长距离信息学习能力**，适用于代码提示等软件分析任务。

表1 实验数据集

Table1 Information of the evaluated dataset

项目名	版本	文件数	代码行	有效代码行
Ant	20110123	1,193	254,457	128,266
Batik	20110118	1,657	367,293	195,467
Cassandra	20110122	545	135,992	97,029
Log4J	20101119	353	68,528	34,479
Lucene	20100319	2,279	429,957	266,573
Maven2	20101118	386	61,622	38,593
Maven3	20110122	847	114,527	70,462
Xalan-J	20091212	973	349,837	171,619
Xerces	20110111	827	257,572	138,841
总计	—	9,060	2,039,785	1,141,329

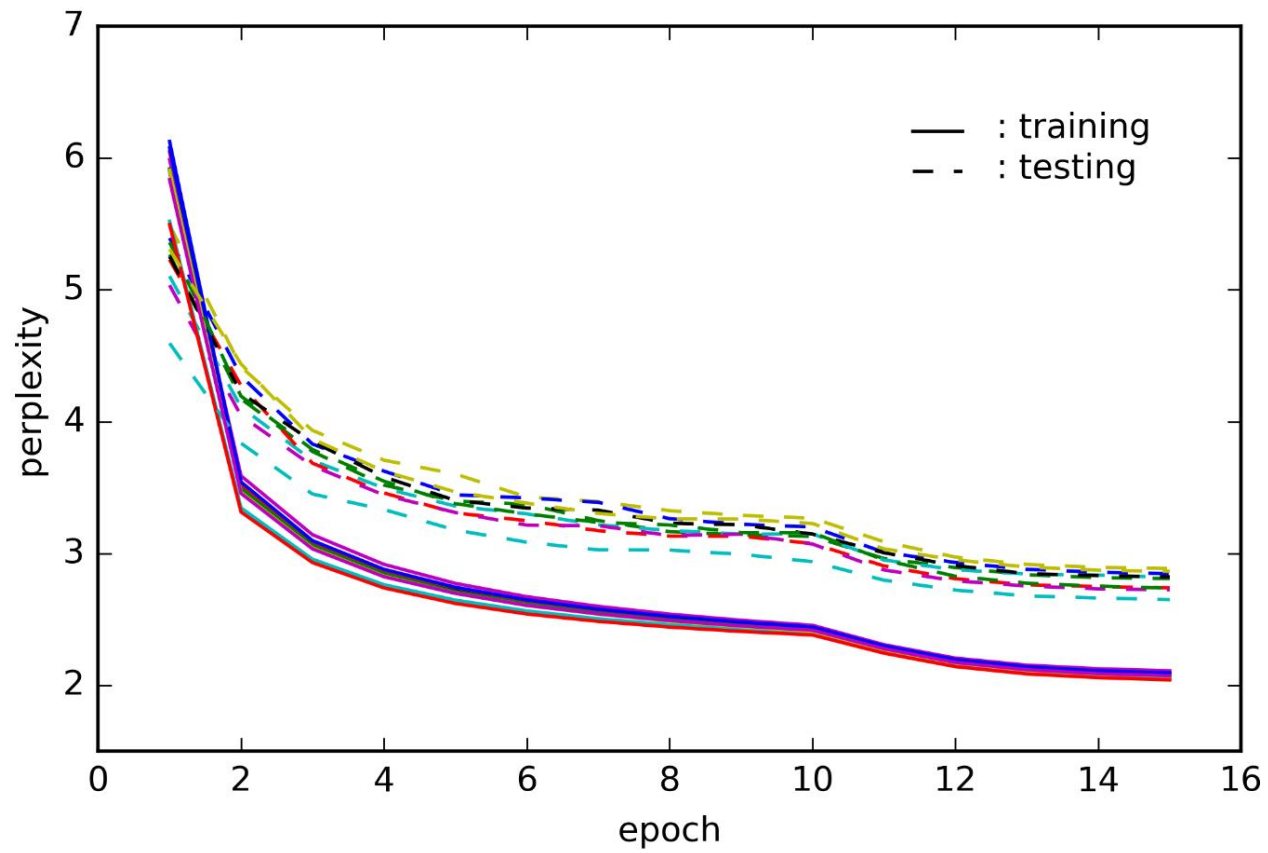




表2 不同模型的困惑度指标值
Table2 Perplexity values of the investigated models

<i>n</i> -gram 类模型			Regularized LSTMs [22]			本文方法		
<i>n</i> -gram [8]	Cache LM [9]	DeepSoft [24][25]	Small	Medium	Large	CodeNLM**	CodeNLM*	CodeNLM
12~14	8~9	4.720	3.739	3.044	3.152	2.946	2.881	2.802

注：*表示未使用策略 1；**表示未使用策略 1 和策略 2。

本文选用两类常用指标衡量模型的代码提示性能^{[8]-[10]}：平均排序倒数（mean reciprocal rank, MRR）和 Top- n 准确率。指标的计算方法如下：

$$\text{MRR} = \frac{1}{|T|} \sum_{i=1}^{|T|} \frac{1}{\text{rank}_i}$$

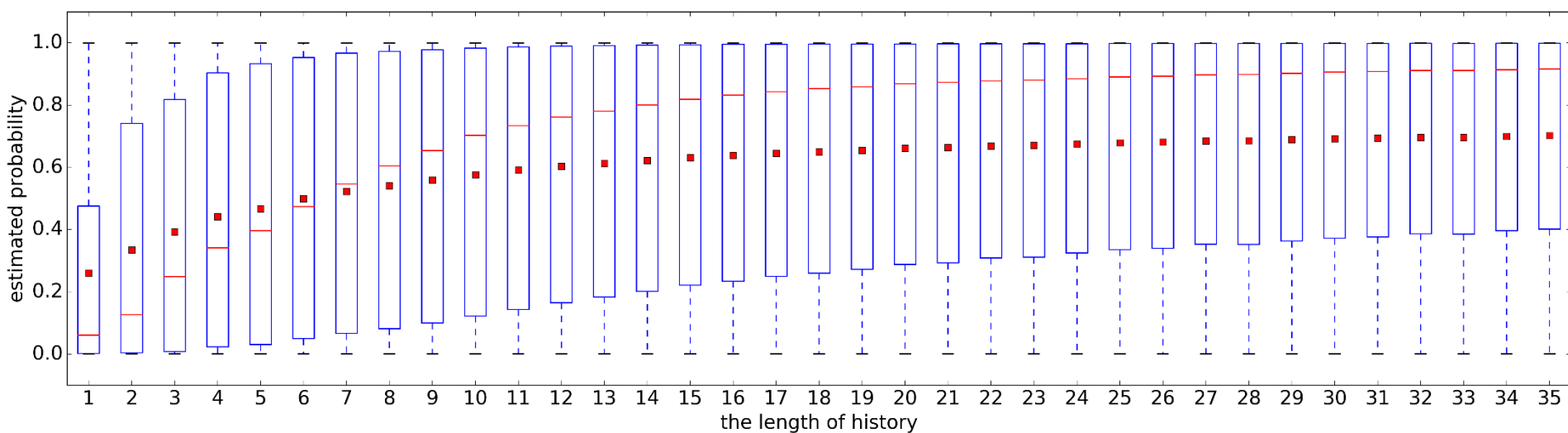
$$\text{Top-}n = \frac{1}{|T|} \sum_{i=1}^{|T|} r_i$$


其中

$$r_i = \begin{cases} 1, & \text{rank}_i \leq n \\ 0, & \text{rank}_i > n \end{cases}$$

表3 不同模型在代码提示任务中的性能对比
Table3 Performance of the investigated models for code suggestion

性能指标	<i>n</i> -gram类模型			Regularized LSTMs [22]			本文方法		
	<i>n</i> -gram [8]	Cache LM [9]	SLAMC [10]	Small	Medium	Large	CodeNLM**	CodeNLM*	CodeNLM
MRR	52.8%	67.6%	—	68.5%	74.0%	72.9%	75.4%	76.2%	77.2%
Top-1	46.3%	—	65.6%	58.9%	65.1%	64.0%	66.9%	67.7%	68.5%
Top-5	56.8%	—	78.2%	80.3%	85.0%	83.8%	86.0%	86.8%	88.0%





实验结果表明，CodeNLM 能较好地进行程序语言建模，具有较强的长距离信息学习能力，适用于代码提示等软件分析任务。下一步工作将考虑不同方面附加信息的提取与融合，并应用语言模型于缺陷检测等其他软件分析问题。

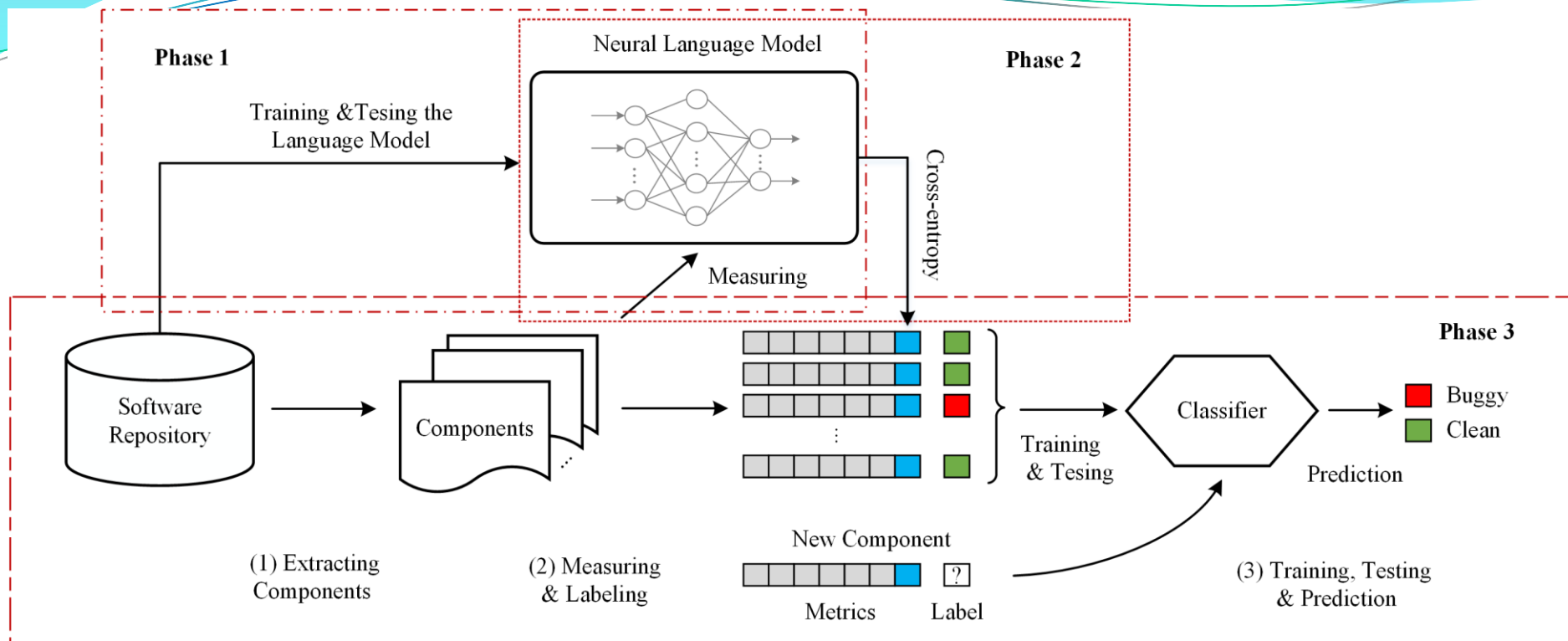
近期工作

代码自然性 (code naturalness)

- Hindle, et al. “On the naturalness of software,” (ICSE’12)
- Z. Tu, et al. “On the localness of software,” (FSE’14)
- Ray, et al. “**On the naturalness of buggy code**,” (ICSE’16)
- S. Wang, et al. “Bugram: Bug detection with n-gram language models,” (ASE’16).
- M. Allamanis, et al. “A survey of machine learning for big code and naturalness,” (ACM Computing Surveys, 2018. arXiv:1709.06182)

基于神经语言模型的代码熵值度量及在缺陷预测中的应用：

- [1] **Zhang Xian, Ben Kerong, Zeng Jie.** *Cross-entropy: a new metric for software defect prediction*[C]. //Proc. of the 18th IEEE International Conference on Software Quality, Reliability and Security (QRS), Lisbon, Portugal, 2018: 111-122.
- [2] **Zhang Xian, Ben Kerong, Zeng Jie.** *Using cross-entropy value of code for better defect prediction*[J]. International Journal of Performability Engineering, 2018, 14(9): 2105-2115. (CE-AST, the cross-entropy value of the sequence of AST nodes)



The overall workflow of *DefectLearner*

相对于缺陷预测，语言模型的训练是无监督的。

思路:

考虑到现有模型仅学习代码数据 (狭义的), 信息利用不充分, 使用**附加信息引导策略**, 通过非代码信息的辅助提高代码规律的刻画能力。

The language model can be regarded as a learner. It should pay more attention to the codes with **high quality**. Therefore, we try to feed the model the **quality information** of codes, so that different samples will have different weights. After training, the cross entropy measured by the weighted language model can get better performance.

切片粒度缺陷预测:

Z. Li, D. Zou, S. Xu, et al., “VulDeePecker: A deep learning-based system for vulnerability detection,” in *NDSS*’18.

谢 谢

