

基于Transformer与BiLSTM的Docker构建预测

蔡美玲¹, 杨 佶¹, 张 锦²

(1. 湖南师范大学 信息科学与工程学院, 湖南 长沙 410081;

2. 长沙理工大学 计算机与通信工程学院, 湖南 长沙 410076)

摘要: 在Docker构建过程中,通过建立提前预警机制能够精准预测构建失败情况,从而有效节省因等待构建完成而耗费的时间成本。现有基于配置语义特征与传统机器学习算法的Docker构建预测模型存在难以有效提取语义特征、捕捉复杂特征模式和深层语义关系等问题,对此提出一种基于改进源代码表征方式和多头注意力机制的深度神经网络模型DeepPDBR。首先,使用AST分阶段解析对源代码进行表征,提取深层次语义特征,并将AST节点序列化后通过Word2Vec进行特征向量化;其次,使用Transformer编码器捕捉全局联系并进一步提取输入数据特征;再次,利用BiLSTM网络进行长距离依赖特征提取,以保留数据的序列化特征;最后,通过全连接层和Softmax层获得分类结果。实验结果表明,相较于现有模型,DeepPDBR在精确率、F1分数和曲线下面积方面分别提升了5.63%~12.28%、1.47%~50.49%和61.20%~93.03%,证实了其在Docker构建预测任务中的有效性。

关键词: Docker; 构建预测; 抽象语法树; Transformer; BiLSTM

DOI:10.11907/rjdk.251065

扫描二维码阅读全文:



中图分类号: TP183

文献标识码: A

文章编号: 1672-7800(2026)003-0063-06

Prediction of Docker Construction Based on Transformer and BiLSTM

CAI Meiling¹, YANG Ji¹, ZHANG Jin²

(1. College of Information Science and Engineering, Hunan Normal University, Changsha 410081, China;

2. School of Computer and Communication Engineering, Changsha University of Science & Technology, Changsha 410076, China)

Abstract: In the process of building Docker, establishing an early warning mechanism can accurately predict the occurrence of build failures, thereby effectively saving time costs incurred due to waiting for the completion of the build. A deep neural network model DeepPDBR based on improved source code representation and multi head attention mechanism is proposed to address the problems of difficulty in effectively extracting semantic features, capturing complex feature patterns, and deep semantic relationships in existing Docker prediction models based on configuration semantic features and traditional machine learning algorithms. Firstly, the source code is characterized using AST staged parsing to extract deep semantic features, and the AST nodes are serialized and vectorized using Word2Vec for feature vectorization; Secondly, using Transformer encoder to capture global connections and further extract input data features; Again, using BiLSTM network for long-distance dependency feature extraction to preserve the serialized features of the data; Finally, the classification results are obtained through the fully connected layer and Softmax layer. The experimental results show that compared to existing models, DeepPDBR has improved accuracy, F1 score, and area under the curve by 5.63%~12.28%, 1.47%~50.49%, and 61.20%~93.03%, respectively, confirming its effectiveness in Docker based prediction tasks.

Key Words: Docker; building prediction; abstract syntax tree; Transformer; BiLSTM

收稿日期: 2025-01-25

基金项目: 全军共用信息系统装备预研项目(31511010402);湖南省重点研发计划项目(2023GK2042)

作者简介: 蔡美玲(1982-),女,博士,湖南师范大学信息科学与工程学院副教授、硕士生导师,研究方向为软件工程、人工智能等;杨佶(1999-),男,湖南师范大学信息科学与工程学院硕士研究生,研究方向为软件工程;张锦(1979-),男,博士,长沙理工大学计算机与通信工程学院教授、博士生导师,研究方向为软件工程、人工智能等。本文通讯作者:张锦。

0 引言

容器化技术是一种将应用程序及其依赖打包在一起,确保在不同环境中一致运行的技术。2013年,随着 Docker 的推出,容器化技术实现了重大飞跃^[1-3]。目前,有大约 79% 的 IT 组织在使用 Docker,该技术对开发人员的日常工作产生了深远影响^[4]。开发人员通过编写 Dockerfile 配置代码来构建一个包含应用程序、依赖项和运行环境的镜像,然后基于镜像创建和运行容器^[5-8]。在构建过程中,需要通过互联网检索大量输入文件,这可能会导致构建过程异常缓慢^[9-11]。Wu 等^[12]通过实证研究发现 Docker 构建总体失败率为 17.8%,且多数项目至少经历过一次构建失败。同时,他们在另一项研究中发现 Docker 构建普遍耗时较长,其中 48.3% 的构建过程超过 10 min^[13]。这可能会对开发人员的工作效率带来负面影响,因为他们可能会等待很长时间才能看到最终构建结果,进而确定构建是否失败。因此,研究构建预测技术来辅助开发人员预判潜在的构建失败情况对于减少时间成本、提高开发效率具有重要意义。

1 相关研究

为了应对 Docker 构建失败风险并提高构建效率,许多研究集中在 Dockerfile 代码异味的检测与修复上。例如,Henkel 等^[14]提出一种 Dockerfile 人机协同修复系统 Shipwright,在对失败构建进行聚类后,通过人工参与确定可能的故障原因来制定 Dockerfile 代码异味修复规则,据此对有问题的 Dockerfile 提供修复建议。詹威霖等^[15]提出一种新的 Dockerfile 代码异味挖掘和检测方法 DRIVE,通过算法挖掘频繁模式自动生成规则,依据规则类型使用自定义领域特定语言描述规则并检测违规情况。Hao 等^[16]提出一种 Dockerfile 安全扫描系统 DFScan,通过指令类别设计

涵盖 8 种指令的检查方案,从多种渠道提取分析规则以挖掘潜在漏洞。Bui 等^[17]提出一种 Dockerfile 自动修复工具 Dockercleaner,基于行业最佳实践制定修复措施。尽管这些研究通过静态检测和修复代码异味的方式提高了 Dockerfile 质量,但仍需要进一步的构建预测研究,以动态识别和预防潜在的构建失败风险。在构建预测研究中,Wu 等^[18]提出的自动化方法是目前主要的解决方案。其通过将 Dockerfile 解析为包含指令和参数两类节点的抽象语法树 (Abstract Syntax Tree, AST) 的源代码表征方式来提取语义特征,将其映射为整数向量后,再通过训练 K-最近邻、支持向量机等 6 种机器学习算法来构建预测模型。然而,该方法在提取 Dockerfile 源代码语义特征以及捕捉特征的复杂模式和深层语义关系方面存在局限性。

针对以上问题,本文提出一种基于改进源代码表征方式与多头注意力机制的深度神经网络模型 DeepPDBR (Deep Predict Docker Build Result)。该模型通过改进源代码表征方式对 Dockerfile 进行表征,从而有效提取语义特征;同时通过词嵌入得到特征向量。考虑到 Dockerfile 的上下文关联和长距离依赖,该模型还引入多头注意力机制和 BiLSTM 网络,然后采用全连接层和 Softmax 层对输出进行分类。

2 DeepPDBR 模型构建

DeepPDBR 模型整体框架如图 1 所示。首先,使用 AST 分阶段解析对源代码进行表征,提取深层次语义特征,并将 AST 节点序列化后通过 Word2Vec 进行特征向量化;其次,使用 Transformer 编码器捕捉全局联系并进一步提取输入数据特征;再次,利用 BiLSTM 网络进行长距离依赖特征提取,以保留数据的序列化特征;最后,通过全连接层和 Softmax 层获得分类结果。

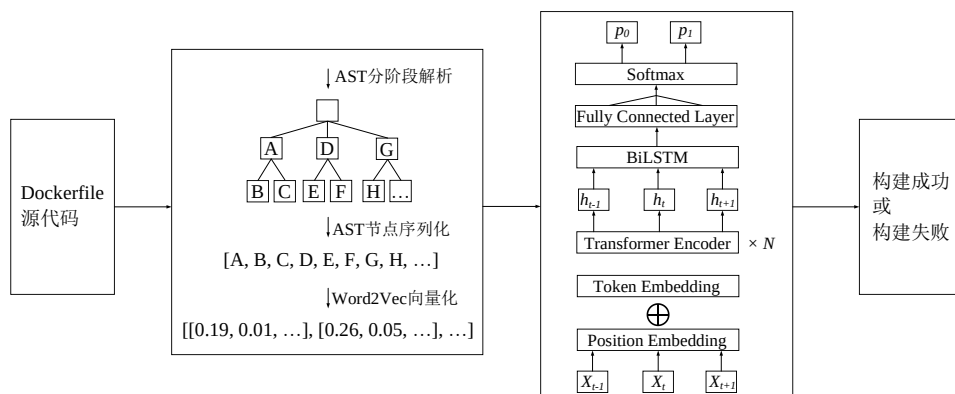


Fig. 1 Overall framework of DeepPDBR model

图1 DeepPDBR 模型整体框架

2.1 AST 分阶段解析

传统的 Dockerfile 源代码表征方式只能将 Dockerfile 解析为包含指令节点和参数节点两类节点的浅层结构,难以有效提取语义特征。针对该问题,本文提出基于 AST 分阶

段解析的源代码表征方式,以便提取更丰富的语义特征。解析流程如图 2 所示。首先,使用 PyPI 的 Dockerfile 包对 Dockerfile 源代码进行顶层解析。第一阶段只能对 Dockerfile 指令和参数进行初步解析。为进一步提取语义特征,

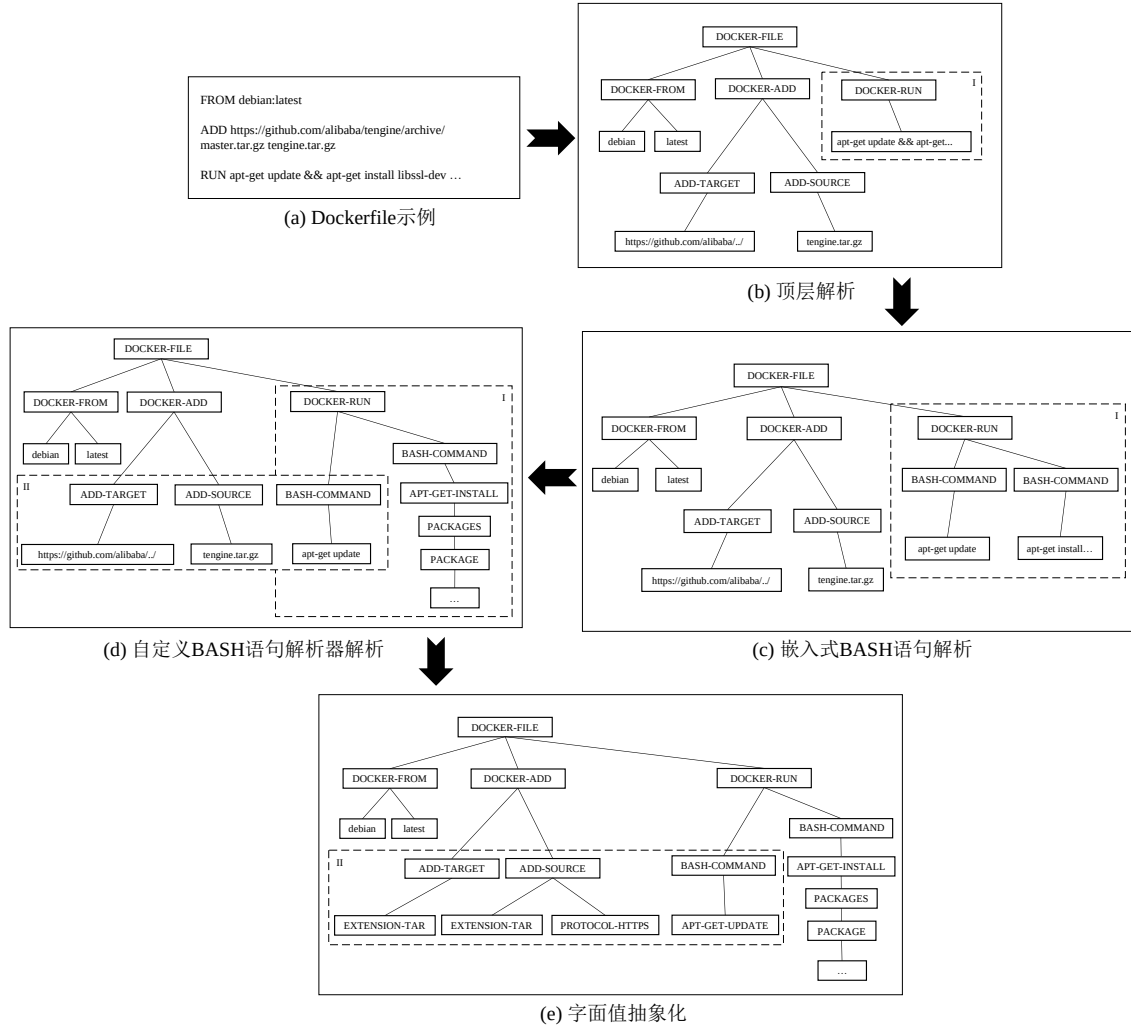


Fig. 2 Phased parsing flow of abstract syntax tree

图2 AST分阶段解析流程

第二阶段通过ShellCheck工具解析第一阶段无法解析的嵌入式BASH。第三阶段通过为BASH命令行工具设计自定义解析器来进一步解析BASH指令、选项和参数。其次, 为避免由于不同命名习惯带来的差异干扰, 对具体参数节点的字面值进行抽象化。相较于传统的Dockerfile源代码表征方式, 本文方法能够构建包含语法层级、语义关系以及上下文依赖的细粒度表征, 显著提升了代码特征的表征深度和语义完整性, 从而更有效地提取语义特征, 提高模型的预测性能。

2.2 AST节点序列化与Word2Vec向量化

为了将分阶段解析得到的AST转换为可以输入到神经网络中的向量, 首先使用先序遍历算法将每个AST节点添加到序列中, 得到AST节点序列 $W = \{w_1, w_2, \dots, w_n\}$, 其中 w_i 表示第 i 个AST节点; 其次, 使用词嵌入模型Word2Vec将AST节点序列映射到其向量表示 $H = \{h_1, h_2, \dots, h_n\}$, 其中 h_i 表示第 i 个AST节点的词嵌入向量^[19]。具体公式为:

$$H = \text{Word2Vec}(W) \quad (1)$$

2.3 Transformer编码器

传统机器学习算法由于具有线性或低复杂度的模型架构, 难以有效捕捉高维非结构化数据(如代码)中隐含的复杂模式与深层语义关联。尤其是在处理长距离依赖、多层次抽象语义时, 传统模型易受局部特征主导, 导致全局信息整合能力不足。相比之下, 深度学习模型通过多层非线性网络结构, 能够逐步从简单特征中提取复杂模式, 每一层都能对输入进行更加精细的抽象, 从而捕捉到数据中的深层语义关系。然而, 深度学习模型的早期架构存在一定局限性, 例如循环神经网络(Recurrent Neural Network, RNN)在处理序列依赖时效率低下, 卷积神经网络(Convolutional Neural Network, CNN)的局部感受野限制了全局关系捕捉等。因此, 本文选择Transformer编码器作为核心组件, 其优势在于通过注意力机制突破序列建模的时空约束, 可直接建立输入元素间的全局关联。相较于传统模型, Transformer能够动态分配注意力权重, 自适应聚焦关键语义节点, 从而实现对复杂语义关系的多粒度建模。

Transformer编码器包括多头注意力机制、前馈神经网络、残差连接和归一化等模块。结构如图3所示。多头注

注意力机制为编码器的核心,其使用多个注意力头对输入序列进行线性变换,可独立计算注意力分数,再根据注意力权重将所有头的加权输出拼接,通过线性变换得到最终输出。多头注意力机制可进一步扩展模型的表征空间,允许并行学习不同语义子空间的交互特征,显著提升模型对深层语义的解析能力。该机制使用点积注意力进行并行运算。注意力的计算公式为:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V \quad (2)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (3)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (4)$$

式中: Q 、 K 、 V 分别表示查询、键、值; d_k 表示键的维度; W_i^Q 、 W_i^K 、 W_i^V 、 W_i^O 表示权重矩阵。

前馈神经网络由两个全连接层组成,中间通过激活函数 ReLU 连接,可以缓解传统激活函数的梯度消失问题。通过对每个位置的特征进行非线性变换可学习输入数据的复杂模式与关系。前馈神经网络的计算公式为:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (5)$$

式中: x 表示输入向量; W_1 、 W_2 表示权重矩阵; b_1 、 b_2 表示偏置向量。

残差连接通过构建一种跨越机制,将输入信息直接绕过一层或多层网络结构进行传递,从而有效缓解了随着网络深度不断增加而引发的网络退化难题,确保深层网络能够稳定且高效地学习特征。归一化则通过线性变换提高模型训练的稳定性。残差连接和归一化的计算公式为:

$$\text{output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \quad (6)$$

式中: x 表示输入张量; $\text{Sublayer}(x)$ 表示当前层的处理结果; $\text{LayerNorm}(x)$ 表示层归一化操作。

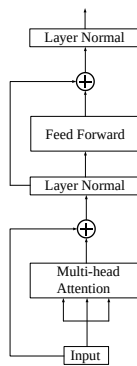


Fig. 3 Structure of Transformer encoder

图3 Transformer 编码器结构

2.4 BiLSTM

在序列建模任务中,长距离依赖特征的有效捕捉是提升模型语义理解能力的关键。尽管 Transformer 通过自注意力机制实现了全局交互建模,但其对序列位置局部动态演化规律的捕捉仍存在局限。因此,本文引入 BiLSTM 作为补充组件,其双向循环架构能够同时融合序列的历史与

未来上下文信息。BiLSTM 由一个前向 LSTM 和一个反向 LSTM 组成,结构如图 4 所示。前向 LSTM 通过时间顺序传递序列信息,获得前向隐状态 $\vec{h}_t$;反向 LSTM 通过时间逆序传递序列信息,获得反向隐状态 $\overleftarrow{h}_t$;随后将前向输出与反向输出拼接得到最终输出。通过双向隐状态的动态耦合, BiLSTM 可显式捕获代码片段中跨越多层的语法约束,从而增强模型对长程语义依赖的敏感性与鲁棒性。特征的计算公式为:

$$\vec{h}_t = \text{LSTM}(\text{output}, \vec{h}_{t+1}) \quad (7)$$

$$\overleftarrow{h}_t = \text{LSTM}(\text{output}, \overleftarrow{h}_{t-1}) \quad (8)$$

$$h_t = w \vec{h}_t + v \overleftarrow{h}_t + b \quad (9)$$

式中: output 表示 Transformer 编码器层的输出; $\vec{h}_{t+1}$ 表示下一时刻隐藏层的短期记忆; $\overleftarrow{h}_{t-1}$ 表示上一时刻隐藏层的短期记忆; w 、 v 分别表示前向和后向输出的权重; b 表示偏置向量。

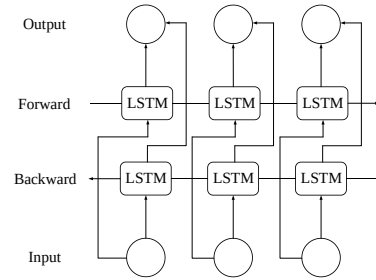


Fig. 4 Structure of BiLSTM

图4 BiLSTM 结构

2.5 全连接层和 Softmax 层

全连接层将 BiLSTM 输出的高维特征映射到与类别数量相对应的维度,Softmax 层将向量转换为每个类别的概率分布。为避免出现过拟合,提升模型的鲁棒性,本文采用正则化方法 Dropout 在训练过程中随机关闭一些神经元,以降低模型对训练数据的依赖。同时使用在二分类任务上表现优异的二元交叉熵损失函数作为模型的优化方向。该函数通过概率输出与对数损失的结合,能够灵敏地反映模型预测概率与真实标签之间的差异,在处理不平衡数据集方面相较其他损失函数具有显著优势。概率分布和损失函数的计算公式为:

$$p = \text{Softmax}(Wh + b) \quad (10)$$

$$L = -[y \log(p) + (1 - y) \log(1 - p)] \quad (11)$$

式中: W_y 表示权重矩阵; b_y 表示偏置向量; p 表示类别概率; y 表示实际类别。

3 实验方法与结果分析

3.1 实验环境与参数设置

本文实验环境设置如表 1 所示。

Table 1 Experimental environment setting**表 1 实验环境设置**

名称	值
CPU	Intel Xeon(R) Platinum 8481C
GPU	NVIDIA RTX 4090 24 GB
内存	200 G
编程语言	Python3.8
深度学习框架	Pytorch1.10
操作系统	Ubuntu 20.04 LTS

模型参数设置如表2所示。

Table 2 Setting of model parameters**表 2 模型参数设置**

参数名称	参数值
Epochs	50
Batch_size	64
Learning rate	0.000 3
Dropout	0.5
Input_dim	100
Attention_head	4
FeedForward_hidden_size	64
BiLSTM_hidden_size	64

3.2 数据集

本文使用 Wu 等^[12-13]研究中的实验数据集。该数据集基于托管在 GitHub 和 Docker Hub 公共项目中的 Docker 日志构建,具体步骤为:①使用 Google BigQuery 查询 GitHub 中包含 Dockerfile 的仓库;②使用 DockerHub API 根据仓库名称过滤构建数据未托管在 DockerHub 上或非公共的项目;③在构建日志中提取历史构建数据,标注 Dockerfile 对应的构建结果;④过滤掉 Dockerfile 为空或无法解析为 AST 的构建数据。对于具有相同 Dockerfile 的构建,只要有一次构建成功,便将构建结果标注为成功。最终数据集中包含来自 3 646 个项目的 37 792 个 Docker 构建,其中有 32 359 次成功构建和 5 433 次失败构建。

3.3 评价指标

采用精确率(Precision)、召回率(Recall)、F1 值(F1 Score)和曲线下面积(Area Under Curve, AUC)对模型性能进行量化。其中,精确率表示预测为成功的 Docker 构建占实际成功构建的比率;召回率表示正确预测的 Docker 构建占实际成功 Docker 构建的比率;F1 值为基于精确率与召回率的调和平均数;AUC 值介于 0 和 1 之间,值越高表示模型的分类能力越强,可以避免因类别分布不均衡而产生的误导性结果,从而综合反映模型在正负样本上的表现。以上指标的计算公式分别为:

$$Precision = \frac{TP}{TP + FP} \quad (12)$$

$$Recall = \frac{TP}{TP + FN} \quad (13)$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (14)$$

$$AUC = \frac{\sum_{I \in positiveClass} R_i^{rank} - M(1 + M)/2}{M \times N} \quad (15)$$

式中: TP 表示真正例,即正确预测为正类的样本数; FP 表示假正例,即错误预测为正类的样本数; FN 表示假负例,即错误预测为负类的样本数; R_i^{rank} 表示样本序号;

M 、 N 分别表示正样本和负样本的数量; $\sum_{I \in positiveClass}$ 表示只

将正样本相加。

3.4 实验结果与分析

3.4.1 比较实验

将本文模型与 PDBR^[18]、GRU^[20]、LSTM^[21]、RoBERTa^[22]、CodeBERT^[23]和 CodeT5^[24]进行比较,结果如表3所示。可以看出,本文模型的精确率、F1 分数和 AUC 值分别达到 92.73%、93.43%、86.71%,与基准模型相比分别提升了 5.63%~12.28%、1.47%~50.49% 和 61.20%~93.03%。基于配置语义特征与传统机器学习算法的 PDBR 的优点为训练成本低,但存在难以有效提取语义特征、捕捉复杂特征模式和深层语义关系等缺点。LSTM 和 GRU 在处理长距离依赖特征上具有局限性,且在训练时间上表现一般。CodeBERT、RoBERTa 和 CodeT5 在召回率上表现较好,但在其他指标上表现较差,且参数量太大导致模型训练时间较长。综合来看,本文模型通过创新源代码表征技术创新和优化算法,在性能上取得了显著提升,证实了其在 Docker 构建预测任务中的有效性。

Table 3 Result comparison of different models**表 3 不同模型结果比较**

模型	Precision/%	Recall/%	F1/%	AUC/%	训练时间
PDBR-KNN	85.43	83.94	84.68	53.79	56 s
PDBR-SVM	83.85	49.00	61.85	45.27	8 min 49 s
PDBR-LR	86.34	90.96	88.59	52.67	13 s
PDBR-DT	82.59	68.87	75.11	49.42	9 s
PDBR-RF	84.68	85.59	85.13	44.92	14 s
PDBR-NB	87.79	69.02	77.28	57.49	5 s
GRU	86.92	92.12	89.44	52.73	42 min
LSTM	87.02	92.05	89.46	52.78	47 min
RoBERTa	86.82	97.12	91.68	50.12	18 h 20 min
CodeBERT	86.72	97.28	91.70	50.59	19 h 10 min
CodeT5	86.76	97.30	91.73	50.52	37 h 15 min
DeepPDBR	92.73	93.43	93.08	86.71	19 min 25 s

3.4.2 消融实验

为研究各模块对所提模型的贡献程度,进行以下消融实验,通过移除或替换部分模型结构来观察其对实验结果的影响:①将原模型的源代码表征方式替换为 PDBR 的源代码表征方式;②将原模型的 Word2Vec 向量化替换为独热(one-hot)编码;③移除源代码表征;④移除模型的 Transformer 编码器;⑤移除模型的 BiLSTM 网络。消融实验结果如表4所示。可以看出,在移除或替换部分结构后,模型性能均有所下降,这表明模型的所有结构均对性能提升有积极影响。所提改进源代码表征方式能更有效地提取源代码语义特征,且使用 Word2Vec 进行词嵌入向量化能够得到准确的向量表示,同时 Transformer 编码器和 BiLSTM 能更好地捕捉特征的全局关系和学习长距离依赖

特征。

Table 4 Ablation experiment result

表 4 消融实验结果 (%)

消融模型	Precision	Recall	F1	AUC
替换源代码表征方式	89.45	90.96	90.20	74.25
替换 Word2Vec 向量化	79.21	80.33	79.77	58.16
移除源代码表征	45.74	57.66	51.01	39.70
移除 Transformer 编码器	85.03	89.72	87.31	50.23
移除 BiLSTM 网络	90.13	91.24	90.68	77.54
原模型	92.73	93.43	93.08	86.71

4 结语

本文针对现有基于配置语义特征与传统机器学习算法的 Docker 构建预测模型存在的难以有效提取语义特征、捕捉复杂特征模式和深层语义关系等问题,提出一种基于改进源代码表征方式与多头注意力机制的深度神经网络模型 DeepPDBR。该模型通过改进源代码表征方式对 Dockerfile 进行表征,能有效提取语义特征;同时,基于 Transformer 编码器和 BiLSTM 网络更好地捕捉特征的全局关系和学习长距离依赖特征,在相关数据集上取得了较好表现。

然而,本文仅考虑了 Dockerfile 配置代码数据,未涵盖其他模态信息,如构建日志、docker-compose.yml 等数据。同时,本文所采用的数据集基于 GitHub 上的开源项目构建且规模较小,所提模型可能无法适用于其他平台或闭源环境中的项目。未来将通过开展实证研究分析 Docker 构建失败的具体原因来扩展数据集,进而更好地定位 Docker 构建失败的故障点。

参考文献:

- BERNSTEIN D. Containers and cloud: from lxc to docker to kubernetes [J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
- ANDERSON C. Docker [software engineering] [J]. IEEE Software, 2015, 32(3): 102-c3.
- WU Y, ZHANG Y, WANG T, et al. Characterizing the occurrence of dockerfile smells in open-source software: an empirical study [J]. IEEE Access, 2020, 8: 34127-34139.
- KSONTINI E, KESSENTINI M, FERREIRA T N, et al. Refactorings and technical debt in docker projects: an empirical study [C]//2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021: 781-791.
- CITO J, SCHERMANN G, WITTERN J E, et al. An empirical analysis of the docker container ecosystem on github [C]//2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017: 323-333.
- ZHANG Y, VASILESCU B, WANG H, et al. One size does not fit all: an empirical study of containerized continuous deployment workflows [C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018: 295-306.
- MOUAT A. Using Docker: developing and deploying software with containers [M]. California: O'Reilly Media, Inc., 2015.
- MERKEL D. Docker: lightweight Linux containers for consistent development and deployment [J]. Linux Journal, 2014, 239(2): 2.
- HUANG Z, WU S, JIANG S, et al. Fastbuild: accelerating docker image building for efficient development and deployment of container [C]//2019 35th Symposium on Mass Storage Systems and Technologies (MSST), 2019: 28-37.
- SU H, ZHANG J H, ZENG J J, et al. A rule-based verification method for static detection of temporary files in Dockerfiles [J]. Computer Engineering, 2025, 51(3): 189-196.
苏珲, 张建辉, 曾俊杰, 等. 基于规则验证的 Dockerfile 临时文件静态检测方法 [J]. 计算机工程, 2025, 51(3): 189-196.
- DURIEUX T. Empirical study of the Docker smells impact on the image size [C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024: 1-12.
- WU Y, ZHANG Y, WANG T, et al. An empirical study of build failures in the Docker context [C]//Proceedings of the 17th International Conference on Mining Software Repositories, 2020: 76-80.
- WU Y, ZHANG Y, XU K, et al. Understanding and predicting Docker build duration: an empirical study of containerized workflow of OSS projects [C]//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022: 1-13.
- HENKEL J, SILVA D, TEIXEIRA L, et al. Shipwright: a human-in-the-loop system for dockerfile repair [C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021: 1148-1160.
- ZHAN W L, ZHOU Y. A Dockerfile rule automatic extraction tool based on sequence mining [J]. Computer Systems Application, 2023, 32(7): 293-298.
詹威廉, 周宇. 基于序列挖掘的 Dockerfile 规则自动提取工具 [J]. 计算机系统应用, 2023, 32(7): 293-298.
- HAO J, LU H, JIANG Y, et al. DFScan: security scanner of the Dockerfile based on instruction coverage and attack perspective [J]. Human-centric Computing and Information Sciences, 2024, 14: 10.
- BUI Q C, LAUKÖTTER M, SCANDARIATO R. DockerCleaner: automatic repair of security smells in Dockerfiles [C]//2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2023: 160-170.
- WU Y, ZHANG Y, CHANG J, et al. Using configuration semantic features and machine learning algorithms to predict build result in cloud-based container environment [C]//2020 IEEE 26th International Conference on Parallel and Distributed Systems, 2020: 248-255.
- MIKOLOV T, CORRADO G, CHEN K, et al. Efficient estimation of word representations in vector space [C]//Proceedings of the Workshop at ICLR, 2013: 1-12.
- ZAREMBA W, SUTSKEVER I. Empirical evaluation of gated recurrent neural networks on sequence modeling [C]//Montreal: Proceedings of the 28th Conference on Neural Information Processing Systems, 2014.
- HOCHREITER S, SCHMIDHUBER J. Long short-term memory [J]. Neural Computation, 1997, 9(8): 1735-1780.
- LIU Y. RoBERTa: a robustly optimized BERT pretraining approach [C]//Proceedings of the 36th International Conference on Machine Learning, 2019: 364-373.
- FENG Z, GUO D, TANG D, et al. CodeBERT: a pre-trained model for programming and natural languages [C]//Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020: 1949-1958.
- WANG Y, WANG W, JOTY S, et al. CodeT5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation [C]//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021: 1388-1399.

(责任编辑:尹晨茹)