2024年度红山开源创新大赛  
操作系统竞赛

项目功能说明书

|  |  |
| --- | --- |
| **学 校：** | **国防科技大学** |
| **参 赛 队 伍：** | **啥都能做队** |
| **队 伍 成 员：** | **杨运珊 喻琦芮** |
| **指 导 教 师：** | **董攀** |
| **完 成 日 期：** | **2024.10.19** |

目 录

[第 1 章 绪 论 1](#_Toc180247943)

[1.1 目的与意义 1](#_Toc180247944)

[1.2 项目背景 2](#_Toc180247945)

[1.2.1 相关工作概况 2](#_Toc180247946)

[1.2.2 本项目优势 3](#_Toc180247947)

[第 2 章 技 术 理 论 5](#_Toc180247948)

[2.1 设计思想 5](#_Toc180247949)

[2.2 技术路线 5](#_Toc180247950)

[2.2.1 模型选择 5](#_Toc180247951)

[2.2.2 流程设计 6](#_Toc180247952)

[2.3 代码原创说明 10](#_Toc180247953)

[第 3 章 数 据 集 处 理 11](#_Toc180247954)

[3.1 数据来源 11](#_Toc180247955)

[3.2 数据预处理和数据集划分 11](#_Toc180247956)

[第 4 章 项 目 测 试 13](#_Toc180247957)

[4.1 实验设置 13](#_Toc180247958)

[4.2 评价指标 13](#_Toc180247959)

[4.3 实验结果 14](#_Toc180247960)

[4.4 结果分析 15](#_Toc180247961)

[第 5 章 实 现 难 点 说 明 16](#_Toc180247962)

[5.1 报告爬取与基于补丁的分类 16](#_Toc180247963)

[5.2 有效信息提取 16](#_Toc180247964)

[5.3 选择合适的模型 16](#_Toc180247965)

[第 6 章 总 结 17](#_Toc180247966)

[6.1 未来展望 17](#_Toc180247967)

[6.2 参赛选手感言 17](#_Toc180247968)

[6.3 鸣谢 17](#_Toc180247969)

[参 考 文 献 18](#_Toc180247970)

# 绪 论

## 目的与意义

在数字化时代，操作系统内核的安全性和稳定性对整个计算环境的安全至关重要。内核的安全性是保障系统可靠性和用户数据安全的基础，任何安全漏洞都可能导致攻击者获取系统最高权限，带来严重后果。绝大多数情况下，内核错误比普通应用程序错误更为严重，因为它们可能引发数据泄露或导致整个系统的崩溃，而不仅仅是单个应用程序的故障。这就要求内核漏洞需要被尽快修复，以防止潜在的安全威胁。

Linux内核，作为全球广泛使用的开源操作系统核心，为从嵌入式设备到超级计算机的多样化计算环境提供支持。然而，随着Linux内核的不断更新和扩展，错误报告的数量也在不断增加。这些报告中存在大量的重复问题，给内核维护者带来了巨大的工作负担。例如，尽管多个bug报告看似各不相同，它们实际上可能指向同一个根本问题。然而，维护人员在初期可能并未意识到这一点，这可能导致不同的维护人员重复劳动处理同一个问题，或者某些bug未能及时分配给合适的工程师。这不仅浪费了维护者的宝贵时间，也降低了内核修复的效率。因此，对bug进行去重和分类，可以显著提升开发人员修复效率，加快内核的更新和维护。

如意内核作为一个新兴的安全内核，同样面临着大量的bug修复任务。如何高效地处理这些bug报告，对于如意内核的发展至关重要。如意内核至今并未有开放的bug维护平台，这导致非维护人员想要针对如意内核做bug去重和分类的研究显得尤为困难。幸运的是，内核错误报告的表现形式往往具有相似性，例如，如意内核和Linux内核的错误报告中都存在call trace信息。而Linux社区已经建立了成熟的漏洞发布平台，能够收集大量的错误报告数据，为模型训练提供了丰富的资源。通过使用Linux内核错误报告数据对RoBERTa模型进行训练和微调，我们可以获得一个高效的bug去重和分流模型。这个模型不仅可以应用于Linux内核的bug处理，还可以迁移应用到如意内核的bug报告信息分类中，为如意内核的维护提供强有力的支持。

本项目的目的在于通过先进的机器学习技术，特别是RoBERTa模型，提升Linux内核及如意内核的bug去重和分流的准确性，从而提高内核维护的效率和质量。通过减少重复和错误分类的bug报告，我们能够帮助内核维护者更快地识别和修复关键的安全漏洞，保障系统的稳定性和安全性。同时，本项目也将为开源社区提供一个强大的工具，促进开源社区的协作和发展。我们期待通过本项目，不仅能够为Linux内核的维护工作带来实质性的改进，还能够为如意内核的发展提供强有力的支持，共同推动开源社区的进步。

## 项目背景

### 相关工作概况

在针对错误去重和分流问题的现有研究中，研究人员主要分析两种信息：错误报告和bug触发时的运行时环境。一部分研究人员在基于错误报告的研究中使用了信息检索、文本分类，基于简化的方法和基于tossing-graph的方法等方案，并取得了较好的bug分类效果，但是这些方案未覆盖到运行时信息。而堆栈路径追踪是运行时信息中最重要的组成部分，Qian[1]等研究人员的综述指出，针对堆栈路径追踪的分析能在错误分流中取得更好的结果，以下是一些相关工作。

Wang[2]等研究人员定义了崩溃相关组，用于描述一组相同或相关错误的错误报告。同时他们提出了自动识别相关错误类型的五条规则，其中三条基于堆栈跟踪签名，一条基于时间位置性，还有一条基于崩溃注释的文本相似性。通过对 Firefox 和 Eclipse 的实证研究，他们发现前三条规则可以利用堆栈跟踪信息识别崩溃相关组，Firefox 的精确度为 91%，召回率为 87%，Eclipse 的精确度为 76%，召回率为 61%。这表明该工作在使用堆栈跟踪信息识别崩溃相关组的情况下能取得较好的效果。

Pushpalatha[3]等研究人员使用分层聚类技术根据调用堆栈中的相似信息将重复的崩溃报告聚类到唯一的桶中，在桶聚类纯度的评估中纯度高达 80%，取得了较好的效果。

Neda[4]等研究人员提出了一种利用堆栈跟踪和HMM自动检测重复错误报告的新方法。他们将堆栈跟踪分为几组，每组包含一个主堆栈跟踪和多个标记为重复的堆栈跟踪。他们训练了一个HMM模型来比较和确定新的堆栈跟踪是否是重复的。实验结果表明，在Firefox和GNOME数据集上，平均准确率分别达到76.5%和73%。当k > 10时，平均准确率超过90%。这表明HMM 和堆栈跟踪是在大型错误库中检测和分类重复错误的强大组合。

Irving[5]等研究人员提出了一种名为TraceSim的方法，它以一种新颖的方式将 TF-IDF、最优全局对齐和机器学习（ML）结合在一起来删除错误报告重复数据。TraceSim首先使用Needleman-Wunsch算法通过考虑每个堆栈跟踪帧的位置和频率来分配权重来确定全局最优对齐。用于权重计算的参数是使用称为TPE的贝叶斯超参数优化器学习和调整的。最终，TraceSim 计算两个堆栈跟踪之间的相似性得分以此来判别重复数据。该方法在所有数据集上考虑所有指标的情况下都能取得有竞争力的结果。虽然该方法简单且易于使用，但其具有低鲁棒性。

在众多研究中，我们更加关注Shi[6]等研究人员取得的成果，他们提出了abaci-finder，一个专门针对Linux内核崩溃的基于深度学习的分类框架。他们首先使用预设的正则表达式来提取和修剪堆栈跟踪。然后，使用kstack2vec方法对堆栈轨迹进行矢量化，提取语义和内核相关的偏置信息、关键帧及其上下文。最后，使用基于注意力的BiLSTM神经网络对多个轨迹进行分类。实验结果表明，Abaci-finder的F1得分为0.83，优于BiLSTM和TF-IDF等模型。

Qian等研究人员的综述还预测了bug去重和分类领域未来的几个潜在研究方向，其中之一就是堆栈跟踪和新的NLP模型的组合。这正和我们的思路一致。

### 本项目优势

在提升bug去重准确度方面，为了达到更好的分类效果，我们选择了RoBERTa模型，理由如下：

1. RoBERTa模型是在更大的数据集上进行预训练的，这意味着它能够学习到更丰富的语言特征和模式。在学习bug信息特征时，RoBERTa模型能够更好地捕捉到bug报告中的复杂语言结构和专业术语，从而提高分类的准确性。
2. RoBERTa模型在BERT模型的基础上进行了优化，通过动态 masking 和更大的模型尺寸，提高了模型的性能和稳定性。这种优化使得RoBERTa在处理长距离依赖和复杂语义时表现更优，有助于理解bug报告中的长文本段落和技术细节。此外，RoBERTa模型通过其预训练阶段的掩码语言模型任务，能够更有效地利用上下文信息。这意味着在bug信息分类中，RoBERTa能够更准确地理解bug的上下文环境，从而提高分类的准确率
3. RoBERTa模型虽然参数量较大，但其训练和推理的效率通常高于传统神经网络。由于RoBERTa模型已经做过大量的预训练，它可以更快地适应特定的bug信息分类任务，而不需要像传统神经网络那样从零开始学习所有的特征表示。
4. 由于bug信息可能包含代码片段、日志文件等多模态信息，而RoBERTa模型在预训练中会对多模态信息进行学习，能够更好地整合和理解这些信息，在bug信息分类任务中能关注到文本序列之外的特征。

综上所述，我们使用RoBERTa模型进行bug文本信息分类，而实验结果也证明本项目方法优于传统神经网络分类方法，如Shi[6]等人使用的BiLSTM分类，也优于何林浩[7]等人提出的二部图网络模型的分类效果。

在模型可迁移性方面，我们的方法也有很好的效果。微调后的 RoBERTa 模型（基于标注的 Linux 内核错误报告数据）之所以可以迁移应用于如意内核错误报告的核心就在于其迁移学习和自然语言表示的泛化能力，理由如下：

1. RoBERTa 模型具有很强的语义迁移能力：RoBERTa 是一种基于 Transformer 的预训练语言模型，在大规模通用语料上（如新闻、百科、技术文档）训练，使其能够捕捉文本的深层语义和上下文依赖关系。因此，即使不同操作系统（如 Linux 和 RUXOS）中的错误报告使用了不同的术语或格式，RoBERTa 仍能识别类似的语言模式。因为基础语言模式具有共性，内核错误报告通常包含日志信息、堆栈跟踪、模块名称和错误描述。这些信息可以被认为是结构化的自然语言文本，不论来源是 Linux 还是 RUXOS，都遵循相似的语言逻辑。而且因为RoBERTa具有的语义嵌入方法，即使在报告格式或细节不同的情况下，模型仍然能够将不同报告映射到相近的向量空间。如果错误报告描述的潜在问题（如内存泄漏、竞争条件）相同，那么它们在嵌入空间中的距离也会较近。
2. 模型微调过程会引入领域适应性：在预训练的 RoBERTa 基础上，使用 Linux 内核的错误报告数据进行微调，使模型具备对内核错误领域的特定理解，如常见错误类型、内核模块相关词汇和堆栈跟踪特征。通过微调，模型可以学会识别特定领域中的语言特征，比如如何描述死锁、驱动问题或内存溢出。这些问题在不同系统中以不同细节呈现，但它们的底层逻辑和现象类似。RUXOS等其他操作系统的错误报告也可能会描述类似的问题（如文件系统错误、线程竞争）。微调后的 RoBERTa 模型能捕捉这些跨领域的共性，因此可以在迁移过程中保持较高的准确率。
3. 模型可以进行迁移学习与小样本适应：微调后的 RoBERTa 模型不仅能应用于 Linux，还能通过少量 RUXOS 数据的进一步微调实现快速适应。由于 RoBERTa 在Linux内核错误领域已经有一定的语义理解，在迁移到 RUXOS 时，只需要少量样本进行少量微调，即可适应新系统的错误报告格式和术语。除此之外，直接训练一个适用于 RUXOS 的模型需要大量标注数据和算力。而微调后的 RoBERTa 已经掌握了错误报告的语言模式，减少了迁移的标注成本。

对于RUXOS这样的新兴操作系统来说，能够用于标注的bug信息较少，而本方法中的模型具有的强可迁移性就有了很大优势，从而辅助如意内核开发人员提升bug修复效率。

# 技 术 理 论

## 设计思想

本研究旨在利用微调后的 RoBERTa 模型对内核错误报告数据进行高效分类与预测，通过捕捉报告中的语义信息，实现模型在测试集上的准确预测。我们的设计遵循以下核心目标：

1. 高效分类：确保 RoBERTa 模型能准确分类内核错误报告，检测报告是否属于同一类型，达到bug去重的目的。
2. 模型迁移性：微调后的模型不仅适用于 Linux，也能迁移到其他类似系统（如 RUXOS）。
3. 优化预测效果：通过合理的模型结构和数据预处理和清洗方法，提升模型在测试集上的性能表现。

## 技术路线

我们设计思想的核心在于利用预训练语言模型的语义泛化能力，结合微调过程中的领域特定知识，确保模型在跨系统迁移时的高效性和鲁棒性。

### 模型选择

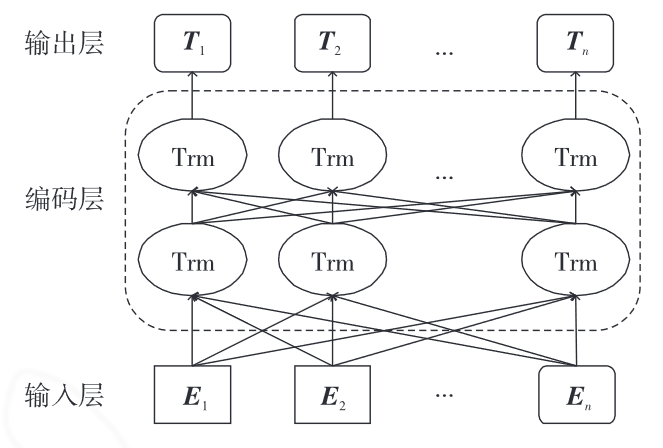
RoBERTa预训练模型是BERT模型的一个优化版本，采用Transformer架构来执行特征提取，通过自注意力机制深入分析词语之间的语义联系，从而生成能够反映上下文关系的动态词向量，这些词向量对于进一步的特征提取至关重要。如图1所示，其中，Trm 表 示 Transformer编 码 器，E1，E2，...， En 为词嵌入层的输出，T1，T2，...，Tn为 RoBERTa得到的符合上下文语境的动态词向量表示，可用于后续的二次特征抽取。

图 2‑1 RoBERTa模型结构图

### 流程设计

本方法基于RoBERTa模型，构建了一个 Linux 内核错误报告重复判定系统，工作流程如图2-2所示，由信息提取模块，特征编码模块，模型微调模块，预测分类模块组成。对于已标注的bug reports对，我们首先从其中提取堆栈跟踪，作为特征编码层的输入，再将生成的嵌入向量作为模型微调的训练集，使RoBERTa模型能够学习领域特征；对于未标注的报告对，我们将其处理后输入到微调后的模型中，在分类层输出模型的预测标签。

下面将分别介绍系统中的各个功能模块。



图 2‑2 工作框架概述图

#### 信息提取模块

在本模块中，我们需要从崩溃报告中提取内核堆栈跟踪信息。

1. BUG: KASAN: use-after-free in \_\_lock\_acquire+0x405b/0x5d40 kernel/locking/lockdep.c:4926
2. Read of size 8 at addr ffff888072212418 by task syz-executor.1/5256
3. CPU: 0 PID: 5256 Comm: syz-executor.1 Not tainted 6.2.0-syzkaller-06742-g307e14c03906 #0
4. Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 1.14.0-2 04/01/2014
5. Call Trace:
6. <IRQ>
7. \_\_dump\_stack lib/dump\_stack.c:88 [inline]
8. dump\_stack\_lvl+0xd9/0x150 lib/dump\_stack.c:106
9. print\_address\_description.constprop.0+0x28/0x370 mm/kasan/report.c:306
10. print\_report mm/kasan/report.c:417 [inline]
11. kasan\_report+0x11c/0x130 mm/kasan/report.c:517
12. \_\_lock\_acquire+0x405b/0x5d40 kernel/locking/lockdep.c:4926
13. lock\_acquire kernel/locking/lockdep.c:5669 [inline]
14. lock\_acquire+0x1e3/0x670 kernel/locking/lockdep.c:5634
15. \_\_raw\_spin\_lock\_irqsave include/linux/spinlock\_api\_smp.h:110 [inline]
16. \_raw\_spin\_lock\_irqsave+0x3d/0x60 kernel/locking/spinlock.c:162
17. p9\_tag\_remove net/9p/client.c:390 [inline]
18. p9\_req\_put net/9p/client.c:398 [inline]
19. p9\_req\_put+0xca/0x250 net/9p/client.c:395
20. req\_done+0x1e2/0x2e0 net/9p/trans\_virtio.c:147
21. vring\_interrupt drivers/virtio/virtio\_ring.c:2470 [inline]
22. vring\_interrupt+0x2a1/0x3d0 drivers/virtio/virtio\_ring.c:2445
23. \_\_handle\_irq\_event\_percpu+0x264/0x9f0 kernel/irq/handle.c:158
24. handle\_irq\_event\_percpu kernel/irq/handle.c:193 [inline]
25. handle\_irq\_event+0xab/0x1e0 kernel/irq/handle.c:210

表 2‑1 崩溃报告示例

首先介绍我们使用的崩溃报告，内容示例如表2-1中所示。

其中第1行是错误标题，报告中显示是因为KASAN (Kernel Address Sanitizer) 检测到了一个 use-after-free 错误而导致的崩溃，触发位置在发生在 kernel/locking/lockdep.c:4926 文件的 \_\_lock\_acquire 函数中，这个函数是 Linux 内核中的锁定机制，用于获取锁资源。根据第一行信息我们可以得出崩溃是因为在内核运行过程中，某个已释放的资源被重复使用而导致的错误。

第2行到第5行是相关的内存地址，线程号和硬件，操作系统版本等信息，这部分环境信息对崩溃分类判断并无作用，甚至反而会影响分类结果，所以在实际处理数据时，我们删除了这部分信息。

从第6行到结束的内容是崩溃发生时内核堆栈调用轨迹，展示了错误传播的路径，是我们进行崩溃分类的主要依据。此外，内核崩溃堆栈的顶部和底部函数大多是常见函数，对分类没有帮助，而指向错误根源的特定堆栈帧对分类至关重要，应给予更多关注，而RoBERTa模型中带有的自注意力机制就能有效地为不同重要程度的堆栈帧赋予不同权重。

综上所述，在信息提取模块中，我们根据开发人员在修复bug时实际关注的内容模块对原始的崩溃报告进行修剪，只保留对分类有用的信息，即崩溃标题和堆栈信息。

#### 特征编码模块

RoBERTa模型的核心在于其强大的字词理解能力，在特征编码模块中，我们需要将预处理剪枝后的内核错误报告对通过向量嵌入后再由RoBERTa模型分词器（Tokenizer）编码，生成特征向量，如图2-3所示。



图 2‑3 特征编码模块工作流程

在嵌入层编码阶段，每个报告调用栈对被分词，分别嵌入为标记向量（Token Embeddings）和段向量（Segment Embeddings），标记向量中每个词（如 bpf、link、release）都会被转换为高维向量表示 ，段向量对两个输入报告的调用栈部分进行区分，确保模型能够感知报告对之间的语义关系。这些输入会被嵌入到模型的底层，使得 RoBERTa 在捕捉语义时能保留词序和结构信息。输入内容信息如表2-1所示。

表 2‑1 不同嵌入形式及其含义

|  |  |
| --- | --- |
| 嵌入形式 | 含义 |
| Token Embeddings | 每个调用栈中的词被转化的向量表示 |
| Segment Embeddings | 区分调用栈的不同部分，标记调用栈对的前后关系 |
| Position Embeddings | 标记词语在序列中的位置，保留顺序信息 |

在本任务中，输入内容形式为报告对，那么向量格式同样需要拼接为向量对，格式为[CLS] CALL TRACE1 [SEP] CALL TRACE2 ，其中[CLS]是分类任务的特殊标记，它的输出代表报告对的语义，[SEP]是用来分隔两个调用栈部分的标记。

最终对于每一对调用栈信息，通过我们的特征编码模块后都能够转化为相应的向量表示，用于输入到RoBERTa 模型中进行训练微调。

#### 模型微调模块

通过对预训练后的 RoBERTa 模型进行微调，能使其更好地适应特定领域的数据。再本项目中，通过对Linux崩溃报告特征的学习，RoBERTa 模型能更准确地完成崩溃分类任务，具体的模型微调工作流程如图2-4所示：



图 2‑4 模型微调模块工作流程

在RoBERTa模型的微调阶段，我们将输入向量送入 RoBERTa Transformer 层，通过多个 Transformer Block 提取上下文信息，每个 Transformer Block 都包含多头自注意力机制和前馈神经网络机制，能够捕捉调用栈中各个词之间的依赖关系和全局上下文，处理后的调用栈帧经过多层 Transformer 生成特征向量。

在RoBERTa 的输入序列开头插入的 [CLS] 标记经过 Transformer 编码层的处理后，会作为整个序列的语义表示。微调时，[CLS] 标记所对应的向量作为输入文本对（X1 和 X2）的整体表示将被传递给下游分类层。

#### 预测分类模块

预测分类模块用于将 RoBERTa 编码后的特征映射到分类结果，模型训练和推理时的工作流程有所不同，如图2-5所示。



图 2‑5 预测分类模块工作流程

在模型微调阶段，输入序列经过 RoBERTa 的多层 Transformer 后，提取出第一个标记 [CLS] 的特征向量，作为整个输入序列的全局语义表示，输入到全连接层进行线性变换，再通过激活函数GELU引入非线性参数，增加模型表达能力；然后经过Dropout层丢弃20%的数据防止模型过拟合，提高模型泛化能力；接着将激活后的向量通过另一线性池化层，将维度压缩为二维，输出两个 logits，分别计算logits[0]和logits[1]的概率分布，根据标签值计算交叉熵损失；最后通过反向传播机制根据损失值计算梯度，并更新模型的参数，使其在训练过程中逐步优化。

在模型推理阶段，即在测试集上的预测过程中，微调后的模型无需再计算损失，而是直接通过 softmax 函数输出同类和不同类的概率，即logit[1] > logit[0]：预测为同类（Y=1）；logit[0] ≥ logit[1]：预测为不同类（Y=0），根据 logits 的输出，模型最终判定报告对是否属于相同类别，即X1,X2是否为重复错误报告。我们在推理阶段禁用 Dropout 层，来确保结果的稳定性。

## 代码原创说明

本团队郑重声明以下内容：

1. 本声明所涉及的代码是本团队独立开发完成的原创作品。
2. 代码中不包含任何侵犯他人知识产权的内容，包括但不限于版权、商标权、专利权等。
3. 代码中未使用任何未经授权的第三方代码或库，所有使用的第三方代码或库均已获得合法授权，并已按照相应的授权协议使用。
4. 代码中不包含任何恶意代码，如病毒、木马、间谍软件等，不会对任何计算机系统造成损害。
5. 代码的编写遵循了良好的编程实践和行业标准，确保了代码的质量和安全性。
6. 本团队对代码拥有完整的知识产权，包括但不限于著作权、使用权、修改权和分发权。

# 数 据 集 处 理

## 数据来源

谷歌在 2016 年初始化了 Syzkaller[8]和 Syzbot[9]项目，可自动、持续地模糊主要 Linux 内核分支，以发现漏洞。除了先进的模糊技术（Syzkaller）外，另一个关键优势是系统（Syzbot）会生成标准的崩溃报告和结构化信息字段，这使得安全分析人员更容易重现漏洞并分析根本原因。截至 2024 年 10 月，该系统已经发现了 2万多个内核漏洞，比 Syzkaller 之前 20 年发现的内核漏洞总数还要多。

Syzbot上提供了比CVE更完整且更新的漏洞报告，包含了重现漏洞所需的必要信息（例如内核版本、内核配置文件、Proof-of-Concept）。我们使用“Fixed bugs”中的数据，因为这些漏洞信息中包含了一个额外的“Fix commit”字段，显示了修复底层内核漏洞的补丁，我们能够通过补丁信息将漏洞报告进行分组，根据Linux 内核社区[10]向内核推送补丁的策略， 一个漏洞对应一个补丁，我们可以通过补丁是否相同对错误报告对进行标注。

我们爬取网页中2017年9月到2024年7月的“Fixed bugs”队列的崩溃报告，并随机两两配对生成了7381个bug组，其中包含6578非重复组，803重复组，其中bug是否重复是根据报告信息中补丁号是否相同判定的，最终数据集形式如表3-1所示，其中X1，X2分别代表bug组里的两篇不同错误报告，Y值代表这两篇报告对应是是否为同一个bug，若相同，则Y=1，反之Y=0。

表 3‑1 源数据集

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Y |
| KASAN: slab-out-of-bounds Write in rmd320\_final random: sshd: uninitialized urandom read (32 bytes read)  …… | KASAN: slab-out-of-bounds Write in tgr192\_final random: sshd: uninitialized urandom read (32 bytes read).  …… | ­1 |
| KMSAN: kernel-infoleak in vcs\_read random: sshd: uninitialized urandom read (32 bytes read)  …… | KASAN: slab-out-of-bounds Write in sha512\_final random: sshd: uninitialized urandom read (32 bytes read)  …… | 0 |

## 数据预处理和数据集划分

在这一步中，我们从崩溃报告中提取崩溃标题和内核堆栈跟踪，并将标题和报告内容进行拼接以崩溃报告序列。具体来说，我们通过预定义的正则表达式提取崩溃报告中的堆栈跟踪，在 "Call Trace "后保留日志，保留表2-1中的第 6-25 行。在实践中，提取是尽力而为的，因为当真正的 Linux 崩溃发生时，内核崩溃报告的部分内容可能会失序或遗漏，而在这些情况下提取准确的堆栈跟踪是相当复杂的。然后，我们将爬取到的“bug\_title”完整保留下来，和堆栈信息进行拼接后组成输入报告对。

二分类任务中，正负样本数据量的平衡至关重要。若样本不均衡，模型可能偏向于预测数量较多的类别，导致性能下降。这样，模型在少数类上的准确率和召回率可能显著降低，影响整体分类效果。通过平衡样本，可以提高模型对少数类的学习能力，从而提升其泛化能力和鲁棒性。

因此，我们在预处理后的数据集中抽取了正负例样本各600条，组成1200条数据集，将其作为训练集，用于微调模型。在构造测试集时，由于需要与其他的方法作比较，我们保持了同样大小的正例样本集，即104条bug组，再随机抽取预处理后的数据集中104条负例数据，整合作为模型的测试集，用于评估模型的分类准确度。

# 项 目 测 试

## 实验设置

在本实验中，我们利用标注后的 Linux 内核错误报告数据集，对预训练的 RoBERTa模型进行微调，以进行内核错误报告的重复检测与分类。实验主要分为两部分：模型的微调训练和测试集上的分类预测。

1. 模型微调

在微调阶段，我们将数据集中的一对报告（X1, X2）作为输入，分别通过 RoBERTa 的 Transformer 编码层进行嵌入提取，生成代表全局语义的 [CLS] 向量。将 [CLS] 向量输入到分类头，经过两层线性变换、GELU 激活函数和 Dropout 操作后，输出每对报告属于相同类或不同类的概率分布。

实验参数设置如表4-1所示。

表 4‑1 实验参数设置

|  |  |
| --- | --- |
| **参数** | **参数值** |
| AdamW 优化器的初始学习率 | 2e-5 |
| 权重衰减率 | 0.01 |
| 预热步数 | 500 |
| 训练批次大小 | 16 |
| 评估批次大小 | 64 |
| 训练的总轮数 | 5 |

2. 模型推理

在推理阶段，我们将训练好的微调模型应用到测试集上，输入一对报告的 [CLS] 向量，经过已训练好的分类模块进行预测。我们使用 Softmax 函数获取每对报告属于相同类的概率，并根据概率值进行分类预测（相同类或不同类）。

## 评价指标

为了评估模型的分类性能，我们二分类任务中的TPR指标作为模型性能评估指标，这是因为在实际应用场景中，我们关心的是模型对于重复bug的判定准确性，对应到实际指标上，TPR越高，表示模型在识别正样本方面的效果越好，即模型对于bug去重能力越强。此外，我们用于对比的其他方法均以TPR作为评价指标。

计算公式如下，其中，TP（True Positives）是正确预测的正样本数量，FN（False Negatives）是实际为正样本但被错误预测为负样本的数量。



## 实验结果

为了验证本文分类方法的有效性，选取了近几年提出的最先进的三种分类方法进行对比，均优于现有的方法。第一个方法为 clusterfuzz 使用的栈相似性对比算法[7]，第二个方法为Mu等人[11]使用的PoC分类算法，第三个方法为何林浩等人提出的二部图相似度算法，这三种方法均以成对的崩溃报告组作为输入，实验结果对比如下表所示。

|  |  |  |
| --- | --- | --- |
| **方法** | **TP率** |  |
| 栈相似性对比 | 49% |
| PoC程序对比 | 79.8% |
| 二部图相似度对比 | 84.6% |
| 本文方法 | 88.4% |

图表 4‑2 不同方法TP率比较

不同方法能够识别出的正例数量如图4-1所示。

图 4‑1 不同方法预测的正例数量

根据实验结果，我们发现与现有的方法相比，我们的模型具有更优越的性能。在TP率（True Positive Rate）对比中，本文方法的TP率为88.4%，显著高于二部图相似度算法的84.6%、PoC程序对比的79.8%和栈相似性算法的49%。从图4-1的正例数量对比中也可以看出，本文方法识别出的正例数量为92个，优于二部图相似度（88个）、PoC程序（83个）以及栈相似性算法（51个）。

## 结果分析

clusterfuzz 使用的栈相似性对比算法是通过对比堆栈之间的莱文斯坦距离来判定其相似程度的，这种方法容易因为堆栈帧的细微差异而导致错误分类，而RoBERTa模型能够通过语义理解对相似性进行更宽泛的捕捉。

PoC程序对比算法依赖于程序的重现能力和PoC的相似度来进行分类，但在很多情况下，程序的重现无法完全代表bug的根本原因，并且很大一部分bug并不具有单独的PoC程序，而我们的方法通过直接从报告中提取信息，避免了因PoC程序信息不全而难以捕捉语义细节的情况。

内核对象相似性算法侧重于使用二部图相似度算法对比内核中相关对象的相似性，适用于特定的bug类型，但其局限于内核对象的相似度范围和提取内核对象的巨大成本，难以泛化到其他不同场景下的崩溃报告去重。而RoBERTa能够跨越内核对象、栈结构、程序重现等多个维度，进行更为广泛的特征提取。

综上所述，RoBERTa的优势在于其能够在广泛的语义层面上捕捉复杂信息，而不仅限于特定的相似度量方法，这也是其在实验中表现最好的原因。

# 实 现 难 点 说 明

在本项目中，我们的目标是通过崩溃报告的自动化分类，提高对内核错误的处理效率。然而，实际执行过程中，存在多方面的挑战，主要包括报告的获取与分类、信息提取的有效性以及模型选择与优化。

## 报告爬取与基于补丁的分类

首先，我们需要从平台上自动化爬取大量的崩溃报告。为了确保分类的准确性和数据的一致性，我们依据“由同一个补丁修复的BUG是相同的”这一原则，对崩溃报告进行初步分类。由于崩溃报告的格式不统一，有些报告包含详细的堆栈信息、日志、内核数据，而另一些报告则可能非常简略。因此，在爬取数据后，还需要进行标准化处理，确保报告内容能够适应后续的分析和分类步骤。

## 有效信息提取

崩溃报告的结构通常包含多种不同的信息，如标题、描述、堆栈轨迹、日志文件等。为了有效进行分类，我们需要提取对分类最有帮助的信息。

崩溃报告的标题有的比较简略，甚至含糊不清，无法直接反映错误的核心原因，所以不能直接用其作为错误的区分，而syzbot平台是直接根据标题将bug作区分，这就导致了很多重复bug的出现。故而我们需要在bug标题的基础上增加信息量，选择能够体现崩溃发生时的函数调用信息的堆栈轨迹。但这些信息有时会非常长且冗余，如何从这些轨迹中提取最具代表性的函数调用序列，保留对分类最有价值的调用栈信息，过滤掉冗余数据是一个技术难点。我们使用正则表达式匹配来提取，在实践中，提取是尽力而为的，因为当真正的 Linux 崩溃发生时，内核崩溃报告的一部分可能会乱序或遗漏。

## 选择合适的模型

在完成信息提取后，我们需要选取一个能够理解语义的模型来完成报告的自动分类任务。经过对比，我们选用预训练的RoBERTa模型，增强分类系统的语义理解能力。在崩溃报告分类的具体场景中，RoBERTa需要从有限的训练数据中学习报告中的专业术语、函数调用以及堆栈信息等技术细节，因此，如何通过微调RoBERTa模型，使其能够更好地理解和分类崩溃报告，是一个关键的难点。

此外，崩溃报告包含的特征类型多样，例如堆栈轨迹和标题，这些特征的格式和内容差异较大。如何在RoBERTa的输入中有效融合这些特征，并保证模型能够处理好不同来源的信息，是模型优化的另一个难点。

# 总 结

## 未来展望

崩溃报告分类技术不仅限于操作系统内核，还可以扩展到各类软件应用和系统中，比如本次比赛的如意内核。未来的研究可以探索如何将这一技术应用于更广泛的软件工程领域，如Web应用、移动应用和大数据处理系统等，以帮助工程师更快速地识别和解决软件缺陷。

并且随着软件更新周期的加快和用户反馈的即时化，我们未来的方向是开发实时自动化处理系统，能够实时收集、分类和分析崩溃报告，并快速反馈给开发团队，从而提升软件质量管理的效率和效果。

## 参赛选手感言

在本次比赛过程中，我们团队成员共同合作开发，深入探索大模型新技术的潜力，同时希望通过我们的努力能够推动社区、行业的发展。感谢组织者提供这样的平台，让我们得以展示自己的创意与努力！

## 鸣谢

在此，衷心感谢所有支持和帮助我们的老师，同学。首先，感谢组委会为我们提供了这样一个宝贵的竞赛平台，让我们能够相互学习、分享经验。其次，感谢团队伙伴们的协作与智慧。最后，感谢导师的鼓励与指导。希望未来能够继续携手，共同探索更广阔的技术领域！

参 考 文 献

[1] C. Qian, M. Zhang, Y. Nie, S. Lu和H. Cao, 《A Survey on Bug Deduplication and Triage Methods from Multiple Points of View》, *Applied Sciences*, 卷 13, 期 15, 页 8788, 7月 2023, doi: 10.3390/app13158788.

[2] S. Wang, F. Khomh和Y. Zou, 《Improving bug management using correlations in crash reports》, *Empir Software Eng*, 卷 21, 期 2, 页 337–367, 4月 2016, doi: 10.1007/s10664-014-9333-9.

[3] P. M. N和R. M, 《Clustering The Duplicate Open Crash Reports Based on Call Stack Traces of Crash Reports》, *ijcse*, 卷 6, 期 9, 页 207–210, 9月 2018, doi: 10.26438/ijcse/v6i9.207210.

[4] N. Ebrahimi, A. Trabelsi, Md. S. Islam, A. Hamou-Lhadj和K. Khanmohammadi, 《An HMM-based approach for automatic detection and classification of duplicate bug reports》, *Information and Software Technology*, 卷 113, 页 98–109, 9月 2019, doi: 10.1016/j.infsof.2019.05.007.

[5] I. M. Rodrigues等, 《TraceSim: An Alignment Method for Computing Stack Trace Similarity》, *Empir Software Eng*, 卷 27, 期 2, 页 53, 3月 2022, doi: 10.1007/s10664-021-10070-w.

[6] H. Shi等, 《Abaci-finder: Linux kernel crash classification through stack trace similarity learning》, *Journal of Parallel and Distributed Computing*, 卷 168, 页 70–79, 10月 2022, doi: 10.1016/j.jpdc.2022.06.003.

[7] 何林浩, 魏强, 王允超和郭志民, 《对象驱动的Linux内核crash分类技术研究》, 小型微型计算机系统, 卷 45, 期 4, 页 926–932, 2024, doi: 10.20009/j.cnki.21-1106/TP.2022-0612.

[8] D. Vyukov, *Syzkaller*. (2016年). [在线]. 载于: https://github.com/google/syzkaller.

[9] *Syzbot Dashboard*. (2017年). [在线]. 载于: https://github.com/google/syzkaller/ blob/master/docs/syzbot.md

[10] L. Kernel, *Submitting patches: the essential guide to getting your code into the kernel*. (2021年). [在线]. 载于: https://www.kernel.org/doc/html/v4.10/process/ submitting-patches.html

[11] D. Mu等, 《An In-depth Analysis of Duplicated Linux Kernel Bug Reports》, 收入 *Proceedings 2022 Network and Distributed System Security Symposium*, San Diego, CA, USA: Internet Society, 2022. doi: 10.14722/ndss.2022.24159.