基于静态分析的警报识别的研究与优化

孙骁 董志昂 唐瑞 王海鹏

南京大学软件学院 南京 211100

摘 要 本文是一篇关于静态分析与警报识别的综述。我们研读了 30 篇相关的论文（引用在文末），从引言、问题与挑战、方法与实现、未来研究方向、结论等内容出发，细致详实地讲解了这些论文的主要方向以及研究成果，给出了我们自己的一些主观看法与评价。基于静态分析的警报识别从 21 世纪以来就被提及，其热度从未衰减，并随着代码量级的提升而显著升高。在实践中，人们发现了诸多有待解决的问题，包括但不限于警报的高误报率、用户无视警报、错误遗漏等等。越来越多的专家学者开始研究相关领域，研发出了诸如 URSA、DynaBoost 的优秀工具来消除误报，或是通过机器学习的手段来鉴别警报的可操作性。我们发现现今大约有 7 种处理静态分析器生成警报的方法，分别是聚类、排序、修剪、误报消除、静态和动态分析组合、简化检查、轻量级静态分析工具，我们在方法陈述部分挑选了其中比较有代表性的优秀方法进行了讲述。总体而言，到 2022 年为止，基于静态分析的警报识别的研究与优化已经取了相当喜人的成果，但依旧有很多复杂的问题有待我们未来去解决。

关键词 静态分析；警报；检查；消除

Research and Optimization of Alarm Recognition Based on Static Analysis

Sun Xiao Dong Zhiang Tang Rui Wang Haipeng

NanJing University Software Engineering NanJing 211100

Abstract (**500 英文单词，内容包含中文摘要的内容**). **字体为** Times new Roman, **字号** 5 **号**

This article is an overview of static analysis and alert recognition. We have studied 30 related papers (cited at the end of the text), starting from the introduction, problems and challenges, methods and implementation, future research directions, conclusions, etc., and explained the main directions and research results of these papers in detail. Some of our own subjective views and evaluations. Alert recognition based on static analysis has been mentioned since the 21st century, and its popularity has never waned, and it has increased significantly with the increase of code size. In practice, people have found many problems to be solved, including but not limited to high false positive rate of alerts, users ignoring alerts, errors and omissions, and so on. More and more experts and scholars have begun to study related fields, and have developed excellent tools such as URSA and DynaBoost to eliminate false alarms, or identify the operability of alarms by means of machine learning. We found that there are about 7 methods for dealing with alerts generated by static analyzers today, namely clustering, sorting, pruning, false positive elimination, combination of static and dynamic analysis, simplified inspection, lightweight static analysis tools, which we describe in the method statement section Selected and representative excellent methods are described. In general, by 2022, the research and optimization of alert recognition based on static analysis has achieved quite gratifying results, but there are still many complex problems to be solved in the future.

Keywords static analysis; alarm; inspection; elimination

孙骁，男，2002 年生，本科. 董志昂，男，2002 年生，本科. 唐瑞，男，2002 年生，本科. 王海鹏，男，2002 年生，本科.（排名不分先后）

1 引言

现如今，随着软件规模的不断扩大，越来越多的源码等待测试，而这些代码中极可能隐藏着潜在的问题。如果不进行警报识别，潜在的问题可能会导致更严重的后果，甚至造成难以估量的损失。进入 21 世纪以来，人们逐渐认识到了代码分析和警报识别的重要性，越来越多的人开始关注和研究相关的领域，并取得了卓越的成就。然而，在有所成就的背后，问题与挑战依旧存在；问题就像一个圆，被中间的成果撑大后，接触到的挑战就变得更多更复杂。

作为一篇综述，本文在 Google Scholars 上收集并仔细阅读了 30 篇相关领域的论文，搜索关键词如表 1 所示。其中我们着重搜索了 elimination, static analysis, alarm。

TABLE I: Keywords used for database search

I	1) elimination, 2) reduction, 3) simplification, 4) ranking, 5) classification, 6) reviewing, 7) inspection
II	1) static analysis, 2) automated code analysis, 3) source code analysis, 4) automated defects detection
III	1) alarm, 2) warning, 3) alert

本文分析了现今存在的问题与挑战，回顾已经取得的技术成果与实现细节，并以此为基础对未来的工作展开预想与规划。通过图 1 我们可以看到如今静态分析工具作用于不同项目的性能：

Project	# open warnings	# filtered	% filtered
jmeter	710	6	1%
tomcat	1624	9	1%
commons-lang	106	19	18%
flink	4934	4754	96%
hadoop	3053	269	9%
jenkins	1212	178	15%
kudu	1873	464	25%
kafka	4668	2993	64%
morphia	65	0	0%
undertow	347	113	33%
xmlgraphics-fop	949	909	96%
Average (Mean)	1666	818	31%
Average (Median)	1212	178	18%

图 1 不同项目的警报识别过滤

对于不同的代码项目，静态分析的过滤率有着显著的不同。对于 flink 和 xmlgraphics-fop 这款静态分析器有着明显的高适应性，但是对于 jmeter 和 tomcat 而言，这款静态分析器就有着显著的不足。不同的分析技术对不同的项目过滤的差异性也提醒了我们在使用静态分析是时要做到 “因地制宜”，为源码项目找到合适的静态分析器。

除了上述提到的挑战之外，还存在着更普遍、更复杂的问题与挑战。本文的第 2 节给出了对于现今静态分析警报识别的详细的分析。

本文不仅会探讨存在的问题与挑战，还会详细的讲述在应对这些挑战时，人们所研发出来的不同的算法和展现出来的令人叹服的开创性。相关的内容会在本文的第 3 节被详细探讨。从每年相关论文的发表数量我们就不难发现，越来越多的学者和研究人员对相关领域展开了研究, 如图 2 所示：

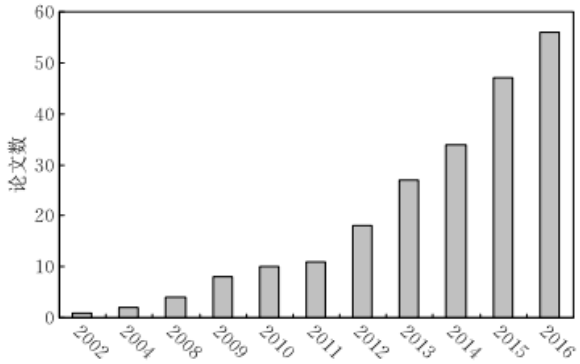

图 2 截止 2016 年逐年累计发表的相关论文数量

本文所引用论文的相关分析讨论的类型众多，包括但不限于：

（1）静态分析产生误报，需要用户手动检查的问题；

（2）基于 web 应用的静态分析与警报标记；

（3）静态分析对比于动态分析存在优势；

（4）启发式方法的优缺点。

本文的第 4 节我们会讨论未来研究与展望。现如今静态分析依旧存在各种各样的问题与潜力，包括但不限于未来代码量的几何级数暴增带来的分析难度的陡增，新的 project 的出现导致静态分析器过滤效果的失控和错误警报问题，人工成本的上升和人工精准度的不可控等因素导致的人工检测越发困难等等，这些问题在未来都亟待解决。

最后一节，我们会对本文讲述的内容做出总结，对于现今的警报识别技术，尤其是静态分析，给出一个主客观相结合的评价，并对未来给出一个大胆新奇的展望。文章末尾是我们所引用到的论文引用出处。

2 问题与挑战

2.1 静态分析产生误报

在我们整理的文章中，相当一部分[2][3][4][6][7][8][9][10][12][13][14][16][17][18][19][25][29]都提及甚至特意强调了该问题。静态分析工具在实践中存在很高的误报率，经常导致开发人员花费大量的时间和精力去检查分析报告的真伪，为生产实践带来了极大的阻碍。

经过研究，导致静态分析工具在进行代码分析的过程中会产生大量的误报的可能原因包括但不限于以下这些：

（1）无论任何该工具多么精确，在实际大小的工业软件上首次运行它通常会产生至少一些错误警报[6]，这是理论与实践的 gap。

（2）由于所分析的软件没有被执行，静态分析工具必须推测实际的程序行为将是什么。他们经常高估可能的程序行为，导致与真正缺陷不对应的虚假警告[12]。

（3）可能存在过度分析，即对于微不足道、没有影响的缺陷发出警报，这一行为会导致大量的警报产生[12]。

2.2 用户无视警报

在早些年的研究中，这一问题并未被发觉。但是随着警报识别与静态分析技术的快速进步，用户检查发现越来越多的警报属于误报，需要付出大量努力来修复而几乎没有感知到的好处的真正缺陷[12]。随着相关文章的发表与问题的提出，用户中间普遍形成了警报大概率是误报的认知，这导致一些真正的警报被当做误报，从而不被处理。

2.3 用户对于警报给出负面反馈

因为人工检查在现今的静态分析中必不可少，因此就难免会存在用户的负面反馈[8]。所谓的负面反馈，指的是用户在检查警报时对于警报内容产生了错误的判断，例如误报当真，从而对整个代码的后续迭代产生了负面的影响。

2.4 启发式标签过于乐观

启发式产生的标签与人类预言不符，如果在实践中采用，会对其真实性能过于乐观[10]。启发式方法有其优越性，例如速度快，精度高等，但它的代码结构经常存在缺陷，导致分析结果不健全。这些不健全的结果容易给用户带来“虚假的繁荣”，导致对其真实性能过于乐观。

2.5 采用动态分析的弊端

出于分析精度亦或是项目特性的考虑，部分代码在警报识别时选择了动态分析而非静态分析。动态分析相较于静态分析拥有对代码的实时分析与反馈，这会带来更优的精度，但是其弊端也是不容忽视的：

（1）动态分析的测试套件会导致低覆盖率，会遗漏错误[8]。

（2）动态分析测试昂贵，成本比较高[18]。

（3）动态分析由于很难运行到所有的代码路径，因此无法消除缓冲区溢出及其影响，静态测试是解决问题的唯一方法[18]。

3 方法与实现

考虑到我们依然面对着大量的问题与挑战，研究人员倾注了大量的时间和精力来研发缓解或解决上述问题的方法，目前已经卓有成效。下面会主要介绍几种典型且重要的实现方法，并给出相对详细的介绍与说明，最后会对各个方法的解决实际问题的性能给出一个主观评价。

3.1 URSA 消除错误警报[3]

在介绍 URSA 之前，先讲解一下我们提出的一种新的方法——通过与用户互动，向用户提出问题。如果用户确认了这些问题，启发式方法才会在用户知情的情况下被用于消除错误警报。

为了有效、综合的方法我们必须完成两个目标：概括化和优先化。

（1）概括性。与被消除的错误警报的数量相比，我们的方法向用户提出的问题的数量应该小得多。

（2）确定优先次序。按照回报率递减的顺序提出问题，允许用户在回报率递减时停止回答问题。迭代过程允许将用户对过去问题的回答纳入到对未来问题的更好选择中，从而进一步放大了用户努力的减少。

将问题简化为一连串的整数线性编程（ILP）实例。每个 ILP 实例对分析事实之间的依赖关系进行编码，并权衡询问不同的根本原因集的收益和成本。然后解决了一连串的 ILP 实例，在收益成本比的上限和下限之间进行二进制搜索。我们在一个叫 URSA 的工具中实现了我们的方法。

URSA 在每个基准中消除了 74% 的错误警报，每个问题的平均回报率为 12 倍。此外，URSA 有效地对具有高回报的问题进行优先排序。此外，根据从 40 名 Java 程序员那里收集的数据，我们观察到，

解决根本原因的平均时间只有解决警报的一半。这些结果共同表明，我们的方法实现了对用户工作的显著减少。

（1）迭代 1。URSA 选择 shared(47) 作为最佳根集，因为它的期望报酬率为 5，是所有根集中最高的。其他根集最终的期望报酬率较低。

（2）迭代 2。在这个迭代中，我们有两个最佳根集的候选者。虽然解决（shared(46) 有望消除 race(46, 57), race(46, 71), race(46,72)，但解决 shared(48) 只能消除 race(48, 74)。

（3）迭代 3。在这个迭代中，只有一个根集 shared(48)，它的预期报酬为 1。

图 3 是显示每个迭代中所问问题与解决的错误警告的图表。从图表中我们不难看出大多数的实际数值与 ideal 数值已经十分接近。

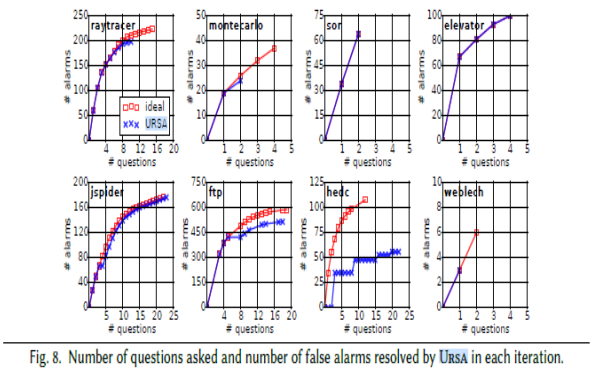

Fig. 8. Number of questions asked and number of false alarms resolved by URSA in each iteration.

图 3 URSA 在每次迭代中提出的问题数量和解决的误报数量

URSA 通过只问两个（或三个）问题，以合理的方式解决了 9 个错误警报中的 8 个（或 9 个）。此外，它在早期迭代中优先考虑具有高回报率的问题。综上，URSA 在消除了错误警报方面起到了显著的效果。

3.2 逻辑回归预测警报的可操作性 [7]

尽管静态分析工具很常见，但它们识别的反模式经常被开发人员忽略，因为静态分析工具可能会产生很高的误报率，这可能非常耗时。考虑到生产力损失，研究人员一直在尝试区分可操作（必须修复）和不可操作（可能未修复，没有足够严重的问题来修复）警报。研究人员经过实验，得出了可操作警报规则的图表如下：

研究结果表明，已采取措施的错误中有 80% 的错误优先级为 3。其他分别为 5、2 和 1。在已解决的所有错误中，最常见的错误是 DataflowAnomalyAnalysis，它属于 Error Prone 规则集。

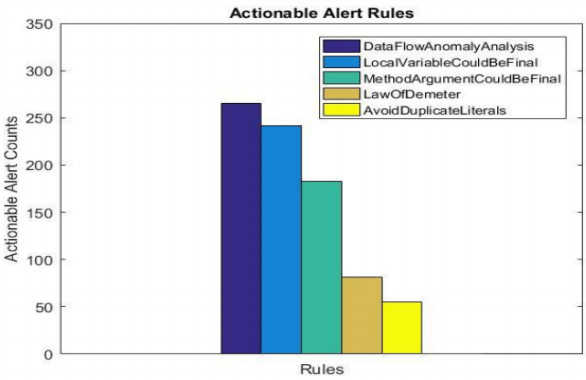

图 4 Actionable Alert Rules

得到警报错误的类型常见类型后，我们使用 PMD 作为静态软件分析工具来分析软件，并假设后期版本中修复的反模式是可操作的。

Ruthruff et. al. 从谷歌收集了样本警告数据，并提出了一种机器学习方法，该方法使用逻辑回归来预测 FindBugs 警告是可操作的警告还是琐碎（不可操作）的警告。

对于两个连续的版本，我们比较了每个函数中的所有错误。如果在字母版本中没有以相同的方法检测到相同优先级的错误和规则，则认为错误已经解决，这表明它是可操作的。其余错误被假定为不可操作。

最后给出一个基于 Fiware 版本的一个可操作警告编号图表。据观察，大多数可操作的警报已在版本 8 中引入系统，所有这些都在版本 9 中得到解决。这可能是由于软件不断发展的性质。当我们分析 GE 的版本历史时，我们注意到在版本 8 中向系统添加了新组件，并且增加了可操作的警报数量。

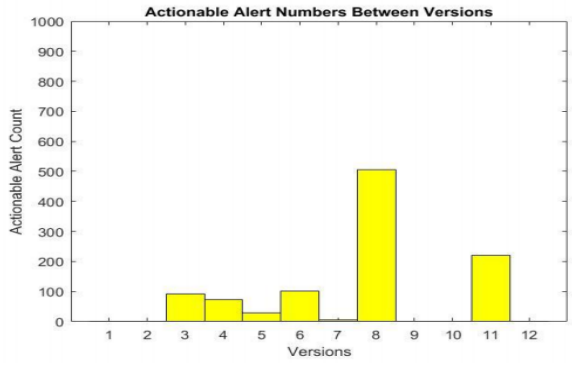

图 5 Actionable Alert Numbers between versions

最后，我们给出了关于警报可操作性的简单易懂的结论：

（1）如果警报在软件的修订历史记录中关闭，则警报是可操作的。

（2）如果警报因文件删除而关闭，则警报既不可操作也不不可操作，并从警报集中删除。

（3）其他警报可以通过检查进行分类，也可以标记为不可操作。

3.3 “动静结合”提高警报的有效精度 [8]

首先，我们使用传统的 Sparrow 程序分析器。它使用稀疏分析框架计算的 def-use 图来确定导致敏感内存访问的所有数据依赖关系，然后执行区间分析以证明内存安全。正如人们所预料的那样，它无法证明在第 38 行调用 memmove 的安全性，并在该行发出警报。

因此我们重构了推导图。第一步是重建导致 Sparrow 报告每个警报的推理轨迹。这个推理过程——应用于 def-use 图的区间分析——可以用图 6 所示的推导规则来近似描述。从程序中的变量定义和一步数据流边开始，由 VarDefn(a) 和 DUEdge(a,b)，分析器计算 DUPath(a,b) 形式的数据流路径。如果每个派生的 DUPath(a,b) 中的最后一个节点 b 对应于一个数组访问，则分析器执行额外的推理来证明程序点 b 处操作的安全性。请注意，我们没有对这个子分析的细节进行建模——它采用了区间抽象——而是提供了溢出（b）形式的输入元组作为推理过程中未建模部分的存根。当分析器发现导致潜在不安全内存访问的此类数据流路径时，它会在适当的点报告警报，如推导规则 r3 所示。

Input relations

VarDefn(a): Variable definition at program point a

Overflow(b): Possible buffer overflow at point b

DUEdge(a, b): Direct dataflow edge between program points a and b

Output relations

DUPath(a, b): (Transitive) Dataflow path from program point a to b

Alarm(b): Alarm indicating possible buffer overflow at b

Derivation rules

r_1 : DUPath(a, b) :- VarDefn(a), DUEdge(a, b)

r_2 : DUPath(a, c) :- DUPath(a, b), DUEdge(b, c)

r_3 : Alarm(b) :- DUPath(a, b), Overflow(b)

图 6 Modeling the program analyzer using derivation rules, represented here as a Datalog program

我们在警报排名系统的概率框架中结合了静态和动态分析，例如通过插入动态检查来验证无法静态证明的属性，使用从测试执行中收集的信息来为后续分析运行优化设置旋钮，通过代码片段来指导测试的 concolic 执行难以探索的，使用静态分析来最小化动态监控的数量，或有限的穷举测试。

不了解分析设计细节的静态分析器的人类用户只能提供有限形式的反馈，例如警报标签。DynaBoost 通过结合来自测试用例的动态分析结果克服了这一限制，从而提高了警报排名系统的性能。

最后总结一下，该方法通过了一种概率技术来利用动态分析的结果，从而提高静态分析器的有效精度。通过程序的针对性检测，我们能够通过实验确认静态分析器得出的中间结论的存在，并使用此反馈来确定生成警报的优先级。在实验中，不难看出人工警报检查负担的显着减少，以及其他相关指标的改进，例如初始排名的质量和错误的泛化事件。总而言之，“动静结合”不失为一种有效且高明的方法。

3.4 使用黄金特征的机器学习技术 [10]

因为 Findbugs 等自动静态分析工具 (ASAT) 的误报率很高，所以研究人员建议使用机器学习来消除误报，并仅向开发人员提供可操作的警告。最先进的研究已经根据对文件、代码和警告的特征和历史计算的指标确定了一组“黄金特征”。最近的研究表明，使用这些特征的机器学习非常有效，并且它们达到了几乎完美的性能。

许多研究人员提出了修剪误报和识别可操作警告的技术，这些警告是开发人员将修复的警告。这些方法考虑了项目中 Findbugs 报告的警告的不同方面，包括有关源代码的因素、存储库历史记录、文件特征以及有关项目中 Findbugs 警告修复的历史数据。Wang 等人完成了对文献中提出的特征的系统评估，并确定了 23 个“黄金特征”，这是检测可操作 Findbugs 警告的最重要特征。一个完美的预测器具有 100% 的召回率、精度和 AUC，这表明使用黄金特征的机器学习技术几乎是完美的。Wang 等人对 Golden Features 的进一步研究表明，使用黄金特征，只需标记一小部分数据集即可训练有效的分类器。

接下来详细介绍一下用于生成标签的封闭警告启发式方法。图 7 显示了构建和标记真实数据集的过程。为了评估检测误报的方法，收集了 Findbugs 警告的数据集。虽然一些研究人员通过手动标记单个修订中的警告来构建标记数据集，但其他研究人员通过自动地面实况数据收集过程收集数据集。收集测试修订和至少一个培训修订的数据，按时间顺序设置在测试修订之前。这模拟了该工具的实际使用情况，其中对项目历史进行了培训，然后在测试修订时使用。

最后点评一下该方法。Golden Features 虽然存在与数据泄漏和数据重复相关的细微错误，但其确实在消除误报、提供可操作警报方面卓有成效，并几乎达到了完美的性能。

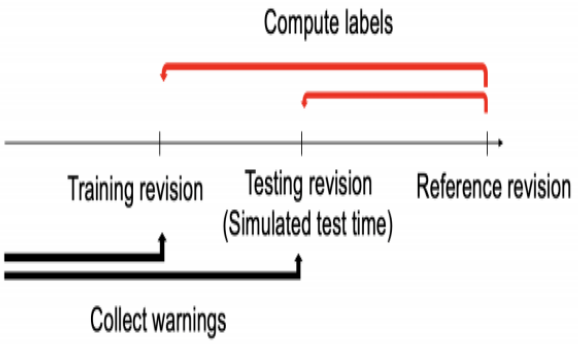

图 7 数据集包含在训练和测试修订之前创建的警告。每个警告的标签由封闭警告启发式确定；如果在参考修订版处关闭警告并且文件尚未删除，则它是可操作的。

3.5 增量主动学习框架识别静态警告 [19]

由于 SA 工具错误地建议大量警告为给开发人员的错误，压倒少数可操作的（真正的错误）。由于无法采取行动的警告率很高，这种静态代码分析工具的实用性值得怀疑。以前的研究工作表明，大约 35%SA 工具报告为错误的 91% 警告实际上是无法采取行动。

本方法应用增量主动学习框架来识别静态警告。这源于主动学习，已被证明在解决多个领域的全面召回问题，例如电子发现、循证医学、初级研究选择、测试用例优先级等。如图 8 所示，我们的目标是通过以下方式实现更高的召回率减少检查 SA 工具生成的警告的工作量。

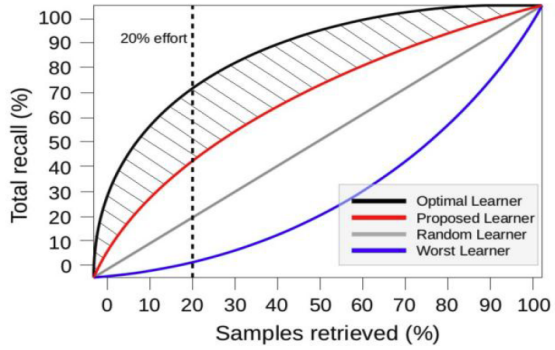

图 8 不同学习者的学习曲线

每个算子的具体细节如下所示：分类器：我们采用三个机器学习分类器作为嵌入式主动学习模型、具有加权方案的线性 SVM、随机森林和具有默认参数的决策树，因为这些分类器在软件工程领域得到了广泛的探索，并且在 Wang 等人的文章中也有报道。所有分类器都是来自 Sckit-learn(Pedregosa et al., 2011) 的模块，这是一个用于机器学习的 Python 包。

Cormack 等人提出的推定不相关示例 (Cormack and Grossman, 2015)，是一种减轻不平衡数据集中负样本的样本偏差的技术。具体来说，在每个训练过程之前，模型从未标记的池中随机采样，并假设采样的实例在训练中被标记为负样本，因为负样本的普遍存在。

积极欠采样（Wallace et al., 2009）是一种处理不平衡数据集的采样方法，通过丢弃靠近 SVM 决策平面的多数负训练点并积极访问少数正点直到这两个类别的比率达到平衡。这是消除数据集中不平衡偏差的有效方法。

主动学习的程序分为以下四步：

（1）初始抽样

（2）人工或数据库标记

（3）模型训练和更新

（4）查询策略

通过使用人类在环主动学习器，我们进行了实证学习 9 个软件项目和 3 个机器学习分类器来验证当前 SA 的性能如何可以通过有效的增量主动学习方法改进工具。我们发现，如果仅检查大约 20% 到 30% 的警告报告而不使用其历史版本信息，则可以识别大约 90% 的可操作静态警告。我们的研究试图弥合通过深入分析监督学习和努力感知主动学习模型之间的研究差距降低静态警告识别问题的成本。我们的方法显着降低了检查静态代码生成的错误报告警告的成本软件工程师的分析工具（尤其是在软件项目生命周期的早期阶段）并提供提高当前 SA 工具性能的有意义的指南。接受和采用未来的静态通过结合 SA 特征提取和自适应增量主动学习，可以增强分析工具。

3.6 处理静态分析器生成警报的方法总结 [17]

(A) 聚类：根据警报之间的相似性或相关性，将警报聚类或划分为若干组。

(B) 排序：警报和源代码的各种特征、错误/警报修复和代码更改的历史等用于对警报进行排序或优先级。

(C) 修剪：将警报分为可操作或不可操作，对不可操作的警报进行修剪。

(D) 误报消除 (FPE)：使用更精确的技术处理警报，例如模型检查和符号执行，以自动识别警报中的误报。

(E) 静态和动态分析组合（SDC）：静态和动态分析相结合来处理警报。

(F) 简化检查：通过半自动诊断、新颖的用户界面和代码导航工具、信息辅助审查等方式，简化了人工检查报警。

(G) 轻量级静态分析工具（LSAT）的设计：LSAT

旨在解决大量分析警报的问题。

以上就是常见的 7 种处理方法，涵盖了绝大多数的处理方法。此前的 5 个方法探讨也包含在这 7 种之中，只不过有的采取了更新的处理手段，比如机器学习的处理手段。

4 未来研究方向

4.1 分析警报的邻域影响 [7]

在将来，我们可以去分析警报之间邻域的影响，并调查解决出现在同一文件中的不可操作警报的可能趋势，或与高优先级可操作警报一起工作。通过观察由于其可操作的邻居而由开发人员解决的不可操作的警报，可以处理许多值得信赖的数据集并获得更稳定的结果。

4.2 机器学习模型的未来优化

Wang 等人对 Golden Features 的进一步研究表明，机器学习模型的选择对有效性缺乏影响。他们认为线性 SVM 是最优的，因为它需要较低的训练成本 [10]。相比之下，虽然深度学习方法达到了相似的有效性水平，但它的训练时间更长。在未来，我们可以着重研发线性 SVM 相关的模型，以此来实现更高效的警报识别。

除了效率，该技术也存在与数据泄漏和数据重复相关的细微错误。这强调了对实验结果进行更深入分析的重要性，定量和定性分析都是必不可少的，这同时也强调了将现有技术和新提出的技术与简单基线进行比较的必要性。我们呼吁需要更多的复制研究，因为这样的工作可以突出未来工作的机遇和挑战。

4.3 Golden Features 的未来优化

现有的 23 个特征被称为黄金特征，其已经在一款静态分析工具（FindBugs）中发挥了近乎完美的性能。但是，目前尚且没有证据说明 Golden Features 的 23 个特征同样可以匹配其他的静态分析工具。因此，未来我们计划通过工业和开源案例研究，对其他常用的静态分析工具进行 Golden Features 的性能评估并对 23 个特征做出相应的调整改善。

4.4 提前计算数据维度

正如 Parsimony 原理所建议的那样，在本质上是低维的数据上使用复杂的模型（如深度学习）是有害的 [13]。尽管我们并不推荐轻易地降维来实现数据简化，但实际上依然存在数据降维所导致的错误遗漏与误报。对于软件分析的未来工作，我们建议分析师将其分析工具的复杂性与其研究问题的潜在复杂性相匹配。具体来说，我们强烈敦促 SE 社区计算其数据的维度，然后使用这些结果来选择合适的分析算法。

5 结论

本文提及了 5 种常见的问题与挑战，其核心就是静态分析工具在进行警报识别时的高误报率。基于这一核心问题，我们又可以将问题细分为警报的检查标准、预测标准等等。基于这些细分问题，我们探讨了 5 种优秀的解决方法，从各自的维度与方向给出了自己的解决思路，开发出了原创或优化的技术工具。例如 URSA，它就是为了消除错误警报而研发的，其成功地将迭代数值优化成几乎贴近 ideal 的曲线，大大降低了警报的误报率。

基于机器学习技术的 Golden Features 技术是值得特意拿出来探讨的。目前该技术趋近成熟，但依旧存在错误遗漏等细微问题。出于效率考虑，未来会进一步与线性 SVM 结合。但是这些现存的瑕疵都不足以掩盖 23 个 Golden Feature 的光辉——近乎 100% 的召回率、精度与 AUC。这是目前我们发现的最成功的一项技术，而且其研究成果是 2022 年刚刚发标出来的，站在了相关领域的前沿。当然，与实践的结合是否能如同理论一般近乎完美还需要更多工业与开源代码的验证。

最后，我们展望了一下未来的研究方向。从分析警报的邻域影响、机器学习模型的未来优化、Golden Features 的未来优化和计算数据维度四个方向讲述了我们对未来有待研究进步的方向的看法。此外，在我们没有单独提及的方面，即用户层面，我们觉得在未来技术供应方需要为用户提供更易懂易操作的工具指南，同时在主观层面上更好的消除警报误报的影响。

致 谢 感谢葛修婷助教为查找文献提供了宝贵的意见，感谢房春荣老师为本课程所做出的贡献！

参 考 文 献

[1] Zhongyang, Yibing, et al. ”DroidAlarm: an all-sided static analysis tool for Android privilege-escalation malware.” *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 2013.

[2] Mansoor, Niloofar, et al. ”An Empirical Assessment on Merging and Repositioning of Static Analysis Alarms.

[3] Zhang, Xin, et al. ”Effective interactive resolution of static analysis alarms.” *Proceedings of the ACM on

Programming Languages* 1.OOPSLA (2017): 1-30.
[4] Bavishi, Rohan, Hiroaki Yoshida, and Mukul R. Prasad. ”Phoenix: Automated data-driven synthesis of repairs for static analysis violations.” *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019.
[5] Liu, Kui, et al. ”Avatar: Fixing semantic bugs with fix patterns of static analysis violations.” *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019.
[6] David Delmas and Jean Souyris. ”Astrée: from research to industry.” *International Static Analysis Symposium*. Springer, Berlin, Heidelberg, 2007.
[7] MİNTEMUR, Ömer, et al. ”Automated Identification of Actionable Static Code Analysis Alerts on Open Source Systems: Fiware as an Example.”
[8] Chen, Tianyi, Kihong Heo, and Mukund Raghothaman. ”Boosting static analysis accuracy with instrumented test executions.” *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021.
[9] Medeiros, Ibéria, Nuno Neves, and Miguel Correia. ”Detecting and removing web application vulnerabilities with static analysis and data mining.” *IEEE Transactions on Reliability* 65.1 (2015): 54-69.
[10] Kang, Hong Jin, Khai Loong Aw, and David Lo. ”Detecting False Alarms from Automatic Static Analysis Tools: How Far are We?.” *arXiv preprint arXiv:2202.05982* (2022).
[11] Wagner, David, and R. Dean. ”Intrusion detection via static analysis.” *Proceedings 2001 IEEE Symposium on Security and Privacy. SP 2001*. IEEE, 2000.
[12] Wang, Junjie, Song Wang, and Qing Wang. ”Is there a” golden” feature set for static warning identification? an experimental evaluation.” *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*. 2018.
[13] Yang, Xueqi, et al. ”Learning to recognize actionable static code warnings (is intrinsically easy).” *Empirical Software Engineering* 26.3 (2021): 1-24.
[14] Goseva-Popstojanova, Katerina, and Andrei Perhinschi. ”On the capability of static code analysis to detect security vulnerabilities.” *Information and Software Technology* 68 (2015): 18-33.
[15] Amer, Eslam, and Shaker El-Sappagh. ”Robust deep learning early alarm prediction model based on the behavioural smell for android malware.” *Computers Security* 116 (2022): 102670.
[16] Yungbum Jung, Jaehwang Kim, Jaeho Shin, Kwangkeun Yi. ”Soundness by Static Analysis and False-alarm Removal by Statistical Analysis: Our Airac Experience” ROPAS MEMO 2005-24 April 26, 2005
[17] Tukaram Muske and Alexander Serebrenik. ”Survey of Approaches for Handling Static Analysis Alarms” 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM) 02-03 October 2016
[18] Misha Zitser, Richard Lippmann and Tim Leek. ”Testing Static Analysis Tools using Exploitable Buffer Overflows from Open Source Code” [SIGSOFT ’04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering](https://dl.acm.org/doi/proceedings/10.1145/1029894)
[19] Xueqi Yang, Zhe Yu, Junjie Wang, Tim Menzies. ”Understanding Static Code Warnings: an Incremental AI Approach” Expert Systems with Applications: An International Journal Volume 167 Issue C, Apr 2021
[20] Arjun Guha, Shriram Krishnamurthi and Trevor Jim ”Using Static Analysis for Ajax Intrusion Detection” [WWW ’09: Proceedings of the 18th international conference on World wide web](https://dl.acm.org/doi/proceedings/10.1145/1526709)
[21] Soleimani-Babakamali, Mohammad Hesam, and Mohsen Zaker Esteghamati. ”Estimating seismic demand models of a building inventory from nonlinear static analysis using deep learning methods.” Engineering Structures 266 (2022): 114576.
[22] Nachtigall, Marcus, Michael Schlichtig, and Eric Bodden. ”A large-scale study of usability criteria addressed by static analysis tools.” Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. 2022.
[22] Rodrigues, A. David. ”Reimagining the Framework Supporting the Static Analysis of Transporter Drug Interaction Risk; Integrated Use of Biomarkers to Generate Pan-Transporter Inhibition Signatures.” Clinical Pharmacology Therapeutics (2022).
[23] Samhi, Jordan, et al. ”JuCify: a step towards Android code unification for enhanced static analysis.” Proceedings of the 44th International Conference on Software Engineering. 2022.
[24] Dewangan H C, Sharma N, Panda S K. Numerical nonlinear static analysis of cutout-borne multilayered structures and experimental validation[J]. AIAA Journal, 2022, 60(2): 985-997.
[25] Vo D, Suttakul P, Rungamornrat J, et al. Static analysis of planar arbitrarily curved microbeams with the modified couple stress theory and Euler-Bernoulli beam model[J]. Applied Mathematical Modelling, 2022, 112: 358-390.
[26] Draiche, Kada, et al. ”Static analysis of laminated reinforced composite plates using a simple first-order shear deformation theory.” Computers and Concrete, An International Journal 24.4 (2019): 369-378.
[27] Bushong, Vincent, Dipta Das, and Tomás Cerný. ”Reconstructing the Holistic Architecture of Microservice Systems using Static Analysis.” CLOSER. 2022.
[28] Feist, Josselin, Gustavo Grieco, and Alex Groce. ”Slither: a static analysis framework for smart contracts.” 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019.

[29] Tikhomirov, Sergei, et al. ”Smartcheck: Static analysis of ethereum smart contracts.” Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. 2018.
[30] Lezgy-Nazargah, M. ”A finite element model for static analysis of curved thin-walled beams based on the concept of equivalent layered composite cross section.” Mechanics of Advanced Materials and Structures 29.7 (2022): 1020-1033.