从静态分析到有效警告识别：我们离真正的有效警告识别还有多远?

蔡之恒 林浩然 沈霁昀 熊丘桓

{201250127, 201250184, 201250048, 201250172} @smail.nju.edu.cn
南京大学软件学院

摘 要 静态分析技术产生的大量的假阳性警告极大降低了静态分析器的使用体验，有效警告识别技术志于从程序编译到静态分析的各种阶段寻找警告的特征，推理标记警告的有效性。本文聚焦于近 10 年来的有效警告识别技术工作，从“警告的特征”、“推理警告有效性的方法”、“评估标准”三方面进行整理，并最终对于“我们离真正的有效警告识别有多远”以及一些其他相关问题给出自己的思考与结论。

关键词 静态分析 (Static Analysis)；文献综述 (Literature Review)；有效警告识别 (Actionable Warning Identification)；机器学习 (Machine Learning)；用户反馈 (User's Feedback)

1 引言

1.1 背景

静态程序分析（Static Program Analysis）是指在不运行代码程序的情况下，进行程序分析的方法。今天，静态分析技术已经在工业界（Oracle, Google, Microsoft, IBM, Facebook……）得到广泛应用。但静态分析并不完美。理论研究*已经证明，不存在一个即具备完备性（Completeness）又具备准确性（Soundness）的静态分析，因此，静态分析永远都是在“漏报”和“误报”之间取得一个平衡。为了尽可能多地反应真实存在的 bug ，现有的静态分析通常秉持着“宁可误报，不要放过”的原则，会产生大量的假阳性警告（False Positives）。

在静态分析器的真实应用中，多篇调研 [17,21,24] 均提及一个观点：尽管静态分析器有效，但可能并不易用。“在真实项目开发中有一大部分开发者不倾向于使用静态分析器”[2]。这其中有工作流未充分整合等其他原因，但大量假阳性可能已经成为阻碍静态分析器进一步被应用的主要障碍。

如何改进静态分析器的使用体验？如何从大量警告中识别出有效（actionable）的部分？这两个问题重要性逐渐显现。小组成员从这两个问题出发，对于有效警告识别（Actionable Warning Identification, AWI）的相关技术进行调研，并尝试回答“我们离真正的有效警告识别还有多远”的问题。

*莱斯定理，Rice's Theorem

1.2 研究过程

1.2.1 问题提出

小组成员将编译技术、静态分析技术以及它们和有效警告识别技术的关系整理在图 1 中。如图所示，静态分析器从编译器前后端执行过程中获取输入，而 AWI 技术从静态分析器和编译器执行的不同阶段获取输入，应用相应的推理警告有效性的方法对于警告的有效性进行推理和标记。

我们据此提出如下三个研究问题：

- RQ1：如何理解警告（警告的特征）?
- RQ2：如何推理警告的有效性?
- RQ3：如何评估 RQ2 中方法的效果?

1.2.2 文献搜集

小组成员通过各种渠道共搜索到相关文章 37 篇，其中 2020 年以后共 17 篇，由于**综述末尾的引用与原文献列表有出入**，特在此附上小组搜索到的**完整文献列表**。†。

1.2.3 调研结果与讨论

“引文”后的三节（“警告的特征”、“推理警告有效性的方法”、“评价标准”）将分别回答 RQ1, 2, 3，并在第五节“讨论、反思与总结”中给出我们的一些思考。

†**完整文献列表**：https://yjiqhlihmt.feishu.cn/sheets/shtcn0hVdiygileWvCzQw02QgdL

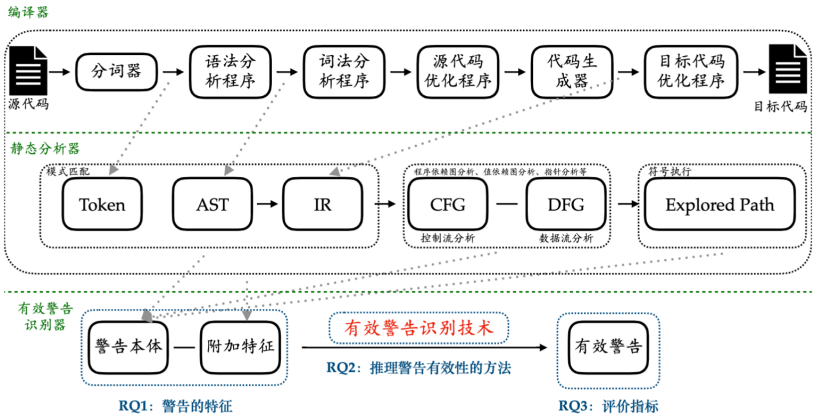

图 1 相关技术概览、RQ 的划分

2 RQ1：警告的特征

面对“有效警告识别”，一个初步的问题是：我们应该如何理解（建模）警告？也就是说，这些 AWI 的方法分别使用了哪些信息作为输入。

Sarah Heckman 等人 [8] 给出了他们的分类方法：

- 警告特征（AC, Alert characteristics）：警告自身的信息，如警告种类、对应代码位置、静态分析器生成的优先级或严重程度。
- 代码特征（CC, Code characteristics）：包含警告的代码片段的特征。
- 源码仓库特征（SCR, Source code repository metrics）：代码仓库中的信息，如提交记录。
- 外部 bug 数据库特征（BDB, Bug database metrics）：对于代码的修改链接到外部的 bug 数据库，可以一定程度上反应 bug 的修复情况。
- 动态分析提供的额外信息（DA, Dynamic analyses metrics）：动态分析提供的额外信息。

这样的划分粗看有些笼统。事实上，我们在阅读大部分文献后发现，这些 AWI 的方法大都融合了多种信息，而从源代码到静态分析的过程中，信息的流动贯通，划分阶段较难。

我们仍以 Sarah Heckman 等人 [8] 提出的分类方法为模板，并在此基础上加入一类用户标注（UA, User’s annotations），即用户对于警告提供的额外信息，对于搜集到的文章进行分类（表 1）。

近年来的工作很少采用单一的警告特征，尤其是 AC 和 CC 特征在 99% 的文章中都成对出现。再加上机器学习方法在 AWI 领域的广泛应用，导致 AC 和 CC 特征的出现率过高。

有关警告特征分类一块，我们也进行过一些其他的尝试，最后结果均不如表 1，尽管这样的分类还有诸多不足，但须认为这是当前较合适的做法。

3 RQ2：推理警告有效性的方法

这一部分着重分析相关文章解决问题的基本思路，根据有无用户参与“监督”，可将这些思路大致划分为“无监督”和“半监督”两类。

按照“监督”这个概念的原始出处（人工智能领域），除去“无监督”、“半监督”，还有“全监督”，但是在 AWI 方向中，“全监督”的工作如果存在，似乎也无太大意义，将其略去。

3.1 “无监督”方法

3.1.1 “无监督”方法综述

获取了警告的特征后，一种非常自然的方法是据此直接推理警告的有效性。“无监督”的意义是没有用户参与的、全自动化的方法，小组成员将这些方法大致分为两类，一类是数据驱动的，另一类是逻辑驱动的。

数据驱动

当提及数据驱动时，其实已经踏入了近年来兴起的人工智能领域，也只有机器学习等方法具备数据驱动的特征。事实上，警告的有效性只有 True 和 False 两类结果，这天然是一个二分类的预测问题。类比于图像，警告的内在维度不知，训练模型似乎是一个好的办法。

在 2011 年的文献综述 [8] 中已经提及了一些使用机器学习方法的尝试；在 2013 年，Ulas Yuksel 等人 [31] 对于一些常见的机器学习方法在一个具体项目上进行了评估和对比；在 2018 年，Junjie Wang 等人 [27] 声称为 AWI 方向的机器学习方法找到了“黄金”特征集（golden feature set），即机器学习中最重要的若干输入（警告的特征）。这篇文章首先通过文献综述的方法，调研整理了相关工作使用的一

表 1 部分工作使用的警告特征分类

警告特征	使用文章
AC	[1, 4, 5, 9–14, 19, 20, 22, 23, 25, 27–33]
CC	[1, 4, 5, 9–15, 19, 20, 22, 23, 25, 27–33]
SCR	[11, 22, 27–30]
BDB	[9, 12, 22]
DA	[10]
UA	[11, 15, 22, 23, 32, 33]

些参数，然后设计实验分别验证其效果，从中筛选出了最具效果的（“黄金”的）23 个特征（图 2）。这份工作给出了一个自信的结论：使用这些“黄金”特征，可以将预测的效果达到接近完美（≈ 100%）。

Table 4: Commonly-selected features (RQ1)

Cat.	Feature	#Rev. (Revision Information)
wCmb ***6/15=40%***	F107: warning context in method	30(*lu:5, tm:5, mv:5, dr:5, po:5, ph:5*)
	F108: warning context in file	26(*lu:2, tm:5, mv:5, dr:4, po:5, ph:5*)
	F112: defect likelihood for warning pattern	23(*lu:3, tm:5, mv:2, dr:3, po:5, ph:5*)
	F115: discretization of defect likelihood	23(*lu:5, tm:5, mv:1, dr:4, po:5, ph:3*)
	F116: average lifetime for warning type	22(*lu:5, tm:3, mv:1, dr:3, po:5, ph:5*)
	F106: warning context for warning type	20(*lu:5, tm:4, mv:2, dr:0, po:5, ph:4*)
cChr ***5/15=33%***	F24: method depth	24(*lu:5, tm:5, mv:1, dr:4, po:4, ph:5*)
	F31: classes in package	22(*lu:4, tm:5, mv:3, dr:2, po:5, ph:3*)
	F25: file depth	20(*lu:5, tm:2, mv:3, dr:2, po:5, ph:3*)
	F23: comment-code ratio	18(*lu:0, tm:3, mv:3, dr:5, po:4, ph:3*)
	F28: methods in file	18(*lu:0, tm:4, mv:4, dr:1, po:4, ph:5*)
wChr ***4/7=57%***	F91: warning priority	24(*lu:5, tm:4, mv:4, dr:3, po:5, ph:3*)
	F89: warning pattern	18(*lu:4, tm:4, mv:1, dr:2, po:5, ph:2*)
	F90: warning type	18(*lu:0, tm:5, mv:3, dr:1, po:5, ph:4*)
	F96: warnings in package	18(*lu:1, tm:5, mv:0, dr:2, po:5, ph:5*)
fHst ***3/8=37%***	F18: developers	26(*lu:5, tm:5, mv:1, dr:5, po:5, ph:5*)
	F16: file creation revision	24(*lu:5, tm:5, mv:2, dr:4, po:5, ph:3*)
	F15: file age	18(*lu:1, tm:3, mv:2, dr:3, po:5, ph:4*)
cAnl ***2/19=10%***	F72: parameter signature	24(*lu:5, tm:5, mv:2, dr:2, po:5, ph:5*)
	F84: method visibility	19(*lu:5, tm:5, mv:0, dr:2, po:4, ph:3*)
cHst ***2/24=8%***	F40: added lines of code in file during the past 25 revisions	22(*lu:5, tm:5, mv:1, dr:3, po:5, ph:3*)
	F46: added lines of code in package during the past 3 months	18(*lu:1, tm:4, mv:2, dr:2, po:4, ph:5*)
wHst ***1/4=25%***	F99: warning lifetime by rev	25(*lu:5, tm:4, mv:4, dr:3, po:5, ph:4*)
fChr(0)		

P:X denotes the feature is selected in X revisions of project P.
Projects are abbreviated as follows: **lu**:Lucene-solr, **tm**:Tomcat, **mv**:Maven, **dr**:Derby, **po**:Poi, **ph**:Phoenix.

图 2 警告的“黄金”特征集 [27]

该工作一经发表，反响不小，Xueqi Yang 等人 [28] 利用“黄金”特征集，使用常见的机器学习（深度学习）模型进行了实验，一方面实验验证了声称的“完美”效果，另一方面也对这些模型的能力进行了基于事实的对比。对比过程中发现，普通机器学习模型优于深度学习模型，并据此得出了“警告的内在数据特征比较低维”的结论。Xueqi Yang 等人 [29] ‡还为“黄金”特征集挖掘了一些支持证据（documenting evidence）。

工作进展到这里，问题似乎已经比较完美地解决了。但是尽管有可复现的实验数据作为支撑，“黄金”特征集在迁移后的效果显著下降。后来者们发现了一些端倪。在 ICSE 2022 上，Hong Jin Kang 等人 [11] 对此提出了质疑。他们的工作探究了“黄金”特征集可以起到“完美”的效果的原因，并评估了一种常见的标记警告有效性的启发式算法的可靠性。他们发现，在使用“黄金”特征集进行训练的过程中，发生了微妙的数据泄漏（图 3 ）：那些最具分类效果的特征（leaked features）在计算的过程中其实是巧妙地利用了未来版本中的一些信息；而另外一个原因则是数据重复（data duplication），即在划分训练集和测试集的时候由于难以察觉的疏忽，将一致的数据同时划分入了训练集和测试集中。这解释了为什么“黄金”特征集的应用效果并不如实验展示得那么完美。而对于用于标注警告有效性的启发式算法，Hong Jin Kang 等人 [11] 在人工标注后发现只有大约 47% 的被标记为有效的警告是真正有效的，因而这种方法也并不如人们想象得那么可靠。

Table 3: Effectiveness of an SVM using the Golden Features after removing the leaked features and removing the duplicate warnings between the training and testing dataset. The numbers in parentheses are the F1 obtained by the baseline classifier that predicts all warnings are actionable.

Project	All Golden Features		– leaked features		– data duplication		– leak, duplication	
	F1	AUC	F1	AUC	F1	AUC	F1	AUC
ant	0.94 (0.09)	1.00	0.11 (0.09)	0.67	-	-	-	-
cassandra	0.92 (0.24)	1.00	0.45 (0.24)	0.86	0.9 (0.41)	0.99	0.29 (0.41)	0.54
commons	0.65 (0.10)	0.99	0.16 (0.10)	0.65	0.75 (0.25)	0.97	0.11 (0.25)	0.49
derby	0.95 (0.09)	1.00	0.39 (0.09)	0.93	0.97 (0.28)	0.97	0.30 (0.28)	0.59
jmeter	0.94 (0.38)	0.99	0.53 (0.38)	0.76	1.00 (0.14)	1.00	0.25 (0.14)	1.00
lucene-solr	0.87 (0.51)	0.97	0.59 (0.51)	0.74	0.87 (0.53)	0.98	0.23 (0.53)	0.62
maven	0.86 (0.07)	1.00	0.27 (0.07)	0.9	0.95 (0.24)	0.99	0.27 (0.24)	0.58
tomcat	0.93 (0.37)	1.00	0.48 (0.37)	0.73	0.95 (0.70)	1.00	0.65 (0.70)	0.39
phoenix	0.89 (0.25)	1.00	0.42 (0.25)	0.78	0.83 (0.37)	0.99	0.40 (0.37)	0.63
Average	0.88 (0.23)	1.00	0.38 (0.23)	0.76	0.90 (0.37)	0.99	0.31 (0.37)	0.59

图 3 leaked features, data duplication 的对比实验 [11]

毫无疑问，Hong Jin Kang 等人 [11] 的工作给大热的机器学习方法泼了一盆冷水。在这之后，也仍然有不少工作在进行机器学习相关方法的尝试 [9, 12, 20, 30] ，但已经不再那么狂热§，也有一些工作（比如 [20] ）结合了用户反馈来改善传统的启发式算法直接标注的不足。

逻辑驱动

逻辑驱动的方法主要挖掘静态分析过程中（如数据流、控制流等）和代码片段（比如某种特定的模式）中的各种信息及它们的联系，并以此作为判断警告有效性的依据 [1, 10, 19, 25]。静态分析信息和传统代码片段信息的界限本就难以区分，而这些

‡是的，与 [28] 是同一个人的不同文章

§之前，Xueqi Yang 等人 [28] 在文章标题里面写 intrinsically easy）

工作所利用的信息获取来源广，难以直接分类，因此将在下文的代表工作中选择一些详细介绍。

3.1.2 “无监督”代表工作

- Peter Hegedus 等人 [9] 也试图采用机器学习的方法来预测警告的有效性。与之前的工作不同，他们没有试图去警告产生的过程中追溯特征信息，而是借助于自然语言处理中的 word2vec 的思想，利用挖掘到的庞大数据集训练了一个 code2vec 的模型，并根据人工标注结合多种常见模型进行了训练（图 4）。实验结果表明 RForest 算法表现最为优秀，能达到 0.910 的准确率和 0.813 的 F1-score。这种方法跳出束缚，迁移了相关领域的思想，较为新颖。

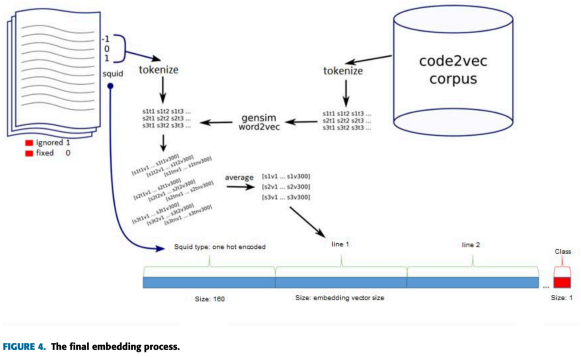

FIGURE 4. The final embedding process.

图 4 一种 code2vec 的 embedding 方法 [9]

- Akshay Utture 等人 [25] 的工作基于的假设是：在静态调用图（static call graph）中，比起产生有效的警告，大多数的边都会产生更多的假阳性，因此可以在调用图上删去一部分边来减少假阳性警告出现的频率，但如何在删边的过程中达到平衡非常有挑战，也就是说，既要删去尽可能多的 False Positive，但又不能删去过多的 True Positive。他们为此构建了一个调用图惩罚器（图 5），对于动态和静态的调用图构造器应用了超前学习（ahead of learning）的过程，评估每一条边生成 False Positive 的概率，最后删去评估值较高的那些边。
- Thu Trang Nguyen 等人 [19] 尝试利用 SEI CERT C Coding Standard 对于静态分析器产生的警告位置进行演绎验证（deductive verification）。他们首先使用静态分析器产生警告，然后利用 ACSL（ANSI/ISO C Specification Language）产生并结合 SEI CERT C Coding Standard 要求产生程序在警告位置的行为性质（behavior properties），最后利用演绎验证的方法验证警告的有效性（图 6）。这种方法在他们设计的实验中能够减少约 90% 的 False

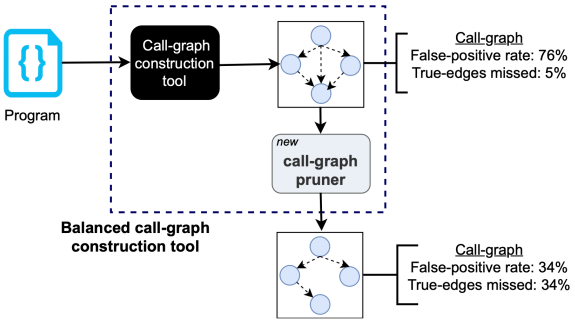

Figure 1: Overview of our technique

图 5 调用图惩罚器的工作流程 [25]

Positive 。这种方法利用了形式化验证的技术，但由于该方法通常开销较大，因此通常只在一些要求较高的领域（比如安全领域）使用。

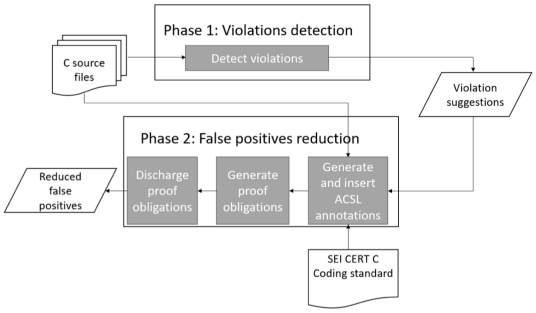

Figure 1: Overview of the approach to reduce false positives

图 6 一种基于演绎验证的工作流程 [19]

3.2 “半监督”方法

3.2.1 “半监督”方法综述

我们认为，“无监督”方法才是有效警告识别方向追求的终极目标，但在现阶段技术难以突破的困境下，引入用户这样一个真正的智能，不仅可以有效提高识别的精度，而且可以直接改善用户体验。“半监督”很显然是一种妥协，但可以扎实地改进开发者使用静态分析器的体验，具有实际意义。

在过去，一种常见的警告消减的语用的（pragmatic）方法包括应用启发式算法来抑制它们的根本原因（root cause）。例如，Bessey 等人 [3] 构造的启发式规则会忽略可能导致高误报警率的特定形式的代码。然而，尽管这种试探法可以减轻用户负担，但也可能使分析结果不可靠。为保证可靠性，静态分析内部机理可能是提高精度和效率的切入口。

在发展过程中，有部分工作开始关注静态分析

内部机理。只要是相对复杂的（静态）分析，都不可避免面对复杂的数据流和控制流信息，结论也从这些复杂的信息网络中得出。处在同一张信息网络之间，True Positive 和 False Positive 其实在性质上有很强的联系（图 7）。

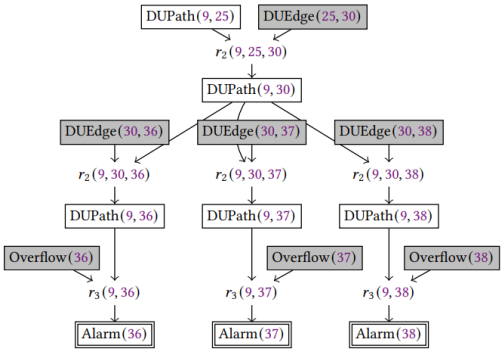

图 7　事实之间的依赖耦合关系 [4]

正如 Xin Zhang 等人 [33] 中指出的，“大多数警报通常是一组相对较小的共同根本原因的症状。例如，错误地认为可访问的方法可能会在该方法的主体中导致许多错误警报。”具体来说，由于静态分析所采取的包括但不限于对控制流的处理方式（如控制流汇聚时的过近似、流敏感与否）、为了在精度和速度间平衡的分析策略、程序分析得出的内部事实（fact）之间存在着很强的依赖关系。这些内部事实（包括警告）经过一个映射关系得出了程序分析的结果。我们可以说，警告是事实的体现，二者（在某种不严谨的意义下）同构。所以，这种事实之间的依赖关系导致了警告之间本身也存在依赖关系。利用这种依赖关系，当某个元素被判定为 True 或者 False 时，可以通过各种方式推断与它相关的其他元素的真假性（图 8）。这也是这一类半监督做法成立的理论基础。

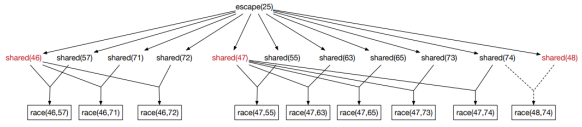

Fig. 3. Derivation of dataraces in example program. We only show parts related to false alarms for brevity.

图 8　根本原因与警告、警告与警告间的联系

值得一提的是，真假性的判断也不宜武断，否则会过度上海静态分析的可靠性。因而，许多工作将最后结果表现形式退化为对警告的排序：将 True Positive 可能性高的警告排在前列。这非常贴合开发者阅读警告的习惯，能够极大改善开发者的体验。

3.2.2 “半监督”代表工作

- Xin Zhang 等人 [33] 的工作的关键思想是，与其直接应用特定的启发式算法，不如组合方法，向用户提出关于启发式算法所针对的根本原因的有效性的问题。他们观察到，大多数警报通常是一组相对较小的共同根本原因的症状。在效率方面，由于不同根本原因导致的错误警报的数量可能不同，用户可能希望只回答收益相对较高的问题。他们的方法通过以迭代方式与用户交互而不是一次提出所有问题来实现这一目标。在每一轮迭代中，所问的问题经过评估挑选，使得期望的回报最大化。
- Ulas Yuksel 等人 [32] 在工作流中引入了外部知识库（knowledge base）。它们构建了一个 Prolog 知识库，用于捕获源代码中的数据流信息以及列出的警告、其属性和根本原因。当开发人员根据列出的警告逐一检查源代码，并确定它们是否指向实际故障时，知识库将动态更新，并根据不断更新的知识库重新刷新其余警告的优先级（图 9）。他们的目标是提供一种侵入性较小的方法，在这种方法中，对开发人员来说警告审查过程保持不变。唯一的额外工作为标记警告（可操作或不可操作），因为它们已经被审查。开发人员不会受到任何形式主义的影响。这种低侵入性对于让程序员乐于接纳静态分析并融入日常工作流中至关重要。

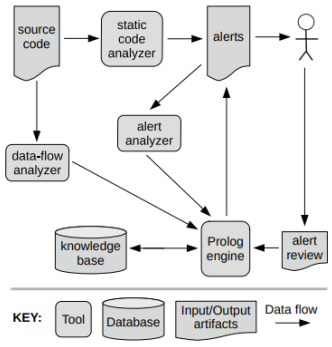

图 9　一种基于外部知识库警告动态刷新方法 [32]

- Tianyi Chen [4] 的工作基于 Xin Zhang [33] 的工作，在其基础上引入动态分析，通过贝叶斯公式来计算相互的条件概率，以此挖掘警告以及背后的事实之间的依赖关系。他们将动态分析中得到的实际执行中可能出现的中间数据流事实作为构建贝叶斯模型的算法的输入，根据这些可能的中间数据流，统计警告被同时触发的事件，构造贝叶斯网络，计算警告之间的概率关系。于是，当某个事实中的某些假阳性信

息被用户排除后，计算剩余警告的条件概率并排序，向开发者推荐更可能是缺陷的警告（图 10）。

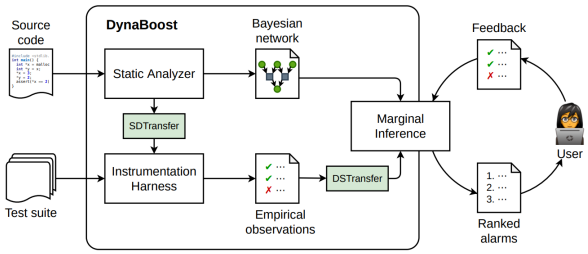

图 10 DynaBoost[4] 的工作流程

4 RQ3：评价标准

这一节中，我们整理了一些常见的评估指标，将少数几篇涉及到该方向评价标准的文章脉络进行初步的整理。

4.1 数据集选择

在调研中，小组成员很少看到 AWI 方向公开的、标准的数据集（测试集），我们认为这对该方向的发展起到不容忽视的阻碍作用。

Peter Hegedus 和 Rudolf Ferenc [9] 提出了类似的观点。他们认为：“大多数工作都集中在自己的（小规模的）实验项目中，因此难以评判这些工作在真实世界的项目中的有效情况。”

事实上在这一领域研究早期（2008 年），Sarah Heckman 等人提出了构建基准的方法 [7]，他们通过收集来自不同领域的一组小的、真实的和含有典型异常的程序来作为基准，并指出，一个成功的基准应包含如下几个特性：可访问性、可负担性、清晰度、相关性、可解决性、可移植性和可扩展性。

目前该领域大部分的方法，都是针对性的，即只处理少数 SCA [¶] 警告类型，并基于合成基准或小规模手动收集的数据集（典型的样本大小为数百）进行评估，甚至是基于一些公司的不对外公开的项目代码进行的评估工作。

Koc 等人 [14] 已经意识到，这样的小规模测试集（数据集）存在一定的问题。尽管应用各种技术对假阳性分析报告进行分类和过滤的结果在训练集测试集上均有不错的表现，但数据集获取代码特征往往依赖专家的人工操作，既耗时又会损失代码的一般性，并且由于缺乏详细的大规模实证评估，这一领域研究的长期潜力和最佳实践尚不清晰。他们还强调，由于这些工作大多是在自行合成的基准上进行的，因此，所提出方法的复制和验证工作很难进行，需要规模更大的真实世界项目数据集来验证先前的工作。

基于他们的研究，Peter 等人 [9] 做了进一步的经验性分析，指出收集大规模的真实世界数据的必要性。他们利用广泛流行、存储有数百万项目的 GitHub ，使用数据挖掘技术创建了一个训练数据集。该数据集比现有我们已知的真实世界 SCA 研究报告的任何其他现有数据集大一到三个数量级（通常比公开可用的开源数据集大 100 倍）。从 GitHub 的 9958 个不同的开源 Java 项目中收集样本，构建了一个包含 224484 个真实世界警告样本的数据集，样本中包含被开发人员修复的真阳性缺陷以及显式忽略的假阳性警告。

4.2 评估指标

这些评估指标并不独立，它们广泛地存在于其他领域。最简单直观的部分指标整理如下（图 11），2011 年的一篇早期文献综述 [8] 中已经提及了这些指标。

		Anomalies are observed (oracle)	
		Actionable	Unactionable
Model predicts alerts	Actionable	True positive (TP)	False positive (FP)
	Unactionable	False negative (FN)	True negative (TN)

Figure 1: Classification Table [8, 12] (adapted from Zimmerman, et al. [25])

图 11 部分评估指标定义

根据上图中列出的生成的预测与警报预言的关系的分类指标，将结果分为 TP，FP，FN，TN，基于此我们可以计算几个其他度量，这些度量用于评估静态分析警报分类工具的有效的程度，在该领域研究中被普遍使用。

精度 Accuracy

$Accuracy(A) = \frac{\#\ of\ TP+TN}{\#\ of\ all\ samples}$

Accuracy 是一个很好的指标，因为没有简单的方法，每个类别的样本分布均匀地拥有很高的准确率。

召回率 Recall

$Recall(R) = \frac{\#\ of\ TP}{\#\ of\ TP+FN}$

当无法接受缺陷漏报时（例如，分析安全关键系统时），召回率指标更有效，可以更好地找到更多缺陷。

精准度 Precision

$Precision(P) = \frac{\#\ of\ TP}{\#\ of\ TP+FP}$

当开发人员审核假阳性报告的成本过高、不可

[¶] SCA, Static Code Analysis，即静态分析

接受时，Precision 可能更有用，使得报告结果更加可信。

F1 评分 F1 score

$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$

为了衡量模型的稳健性，取精确度和召回率的调和平均数定义 F1 score。F1 score 是两者的综合。

小组成员还发现以下两个指标应用得特别广泛 [6]：

接收者操作特征曲线 ROC curve, receiver operating characteristic curve

ROC 曲线用于绘制不同分类阈值下正例率‖和假正例率**的比值，可以展示分类模型在所有分类阈值下的性能。

ROC 曲线下面积 AUC, Area under the curve

AUC 用来描述 ROC 曲线下的面积，对于所有可能的分类阈值的结果进行汇总衡量。曲线下面积的一种解读为，模型对于随机正类别样本的排名高于随机负类别样本的概率。

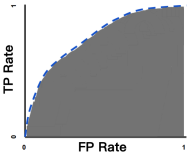

图 12 ROC 和 AUC 示意 [6]

5 讨论、反思与总结

5.1 “遗漏”的部分：为什么未整理和对比实验结果?

文献综述的过程中理应对现有技术的实验结果进行整理和比较，但这一部分在这份课程作业里面“遗漏”。

小组成员在阅读文献的过程中发现，AWI 方向是一个前沿领域，似乎还没有统一的标准对于现有的技术进行足够公平的评估和对比。而大多数工作所提出的方法，都能在自己的实验中取得较为不错的效果（比如，声称减少了 70% ∼ 90% 的警告），但大都具有两个问题：一是实验数据集规模小、较为单一，这是指一份工作通常只在少数几个项目上进行测试（但应当承认这是正常的，毕竟获取警告有效性的开销不小）；二是缺少公开的、标准的数据集，因此无法进行集中的比较和评估，也有一些学者尝试迁移一部分技术，但效果不好，通常显著劣于原实验结果。事实上，Hong Jin Kang 等人 [11] 在文章中已经呼吁建立标准化的数据集，而 Peter Hegedus 等人 [9] 已经率先做出了尝试。我们也认为，建立公开的、标准的、规模足够的实验数据集，应该是 AWI 方向继续发展的必经之路。

有了以上观点，应当认为整理一份完整的实验数据效果对比并无太大意义，这也是我们最终决定忽略这部分的最重要原因。

5.2 机器学习与否：如何看待机器学习方法在 AWI 方向的应用?

原则上这个问题过于宏大，我们不能够也没有资格回答。但小组在调研的过程中看到了如雨后春笋一般出现的机器学习方法，也确实有一些想法和思考，因而“斗胆直陈”。

上文提到，警告的有效性天然是一个二分类的预测问题，这个问题看上去非常适合用机器学习方法，但是以“黄金”特征集 [27] 为代表的一众工作过于狂热，得出的一些结论过于激进，令人难以接受。联想到机器学习近年来的大热，很难不让人觉得这是一种跟风的行为。而这样的结论确实也站不住脚，很快就被 [11] 有针对性地提出了质疑。我们非常喜欢 [11] 这篇文章，包括本文的标题也来源于这篇文章。原因并不是因为它带来的“打脸”的爽快感，而是它有理有据、令人信服，得出了符合认知的结论，许多观点都和我们的想法不谋而合。

事实上，机器学习也好，传统方法也罢，都是解决问题的工具而不是问题本身，我们不能脱离实际情况来直接强行地去拟合问题域的解，脱离目标数据也确实为机器学习领域的大忌。我们并不排斥这种方法，但我们希望看到更多对于数据内在的深入挖掘和思考，而不是唯经验论的直接应用。

5.3 元问题探索：AWI 技术应该立足于何处?

这一小节中，我们尝试给出有关“有效警告识别”这个问题本身的一些思考。

小组不仅调研了 AWI 方向的技术资料，也对这个问题的（姑且称）“上游”的静态分析器的真实应用情况做出了一些调研 [2,16,21,24,26]，应该承认，改善静态分析器的使用体验，是客观存在的需要，而最直观的方法或许就是对于警告的有效性进行推理。但这些工作的界限并不清晰，想要推理警告的有效性，原则上就必须结合警告在产生路径上的各种信息，如果完全舍弃编译阶段、静态分析阶段的各种信息，单纯把有效警告识别作为静态分析的下游任务，似乎丢失了太多有价值的信息，看上去不是一个明智的选择；但如果我们千方百计地去追溯各种控制流和数据流信息，又和直接改进静态分析有什么差别呢？我们甚至可以大胆地说，这两个问题是同质的。引言中已经提到，不存在一个完全精准的静态分析。因此这个问题并不平凡（trivial）。

‖即召回率 Recall

**与正例率对称定义，即 $\frac{FP}{FP+TN}$

这也是为什么我们看到一些唯经验论的机器学习方法就觉得不靠谱的重要原因之一。

至此，我们的思考似乎已经拐入了死胡同。静态分析发展几十年，至今都没有得出很好答案的问题，不可能被轻易解决。但我们惊喜地发现，仍有一部分工作 [4, 32, 33] 为这个问题提供了全新的思路，这就是在推理的过程中引入用户反馈（user's feedback），它们的思路也非常简单，那就是回归问题的初心，即“如何改进静态分析器的使用体验”，而在使用体验中，另一个一直被忽略的、但极为重要的角色就是使用者。我们不得不承认，这样的办法是在现阶段无法自动化推理警告有效性情景下的一种妥协，但这样的工作扎实、有效、富有人情味，它们确实可以切实改善静态分析器的使用体验。我们认为，从这个方向入手，可能是现阶段最有效的思路之一。在 AWI 这个方向，除去等待极具创新性的理论突破；在此之前，引入一些接地气的“人情味”，不失为一个很好的立足点。

5.4 我们离真正的有效警告识别还有多远?

如果要给一个结论，我们的答案是“很远”。这是客观因素和主观因素相统一的结果。

5.4.1 客观角度

AWI 是一个小众且前沿的领域，相关工作其实较少。从现有成果来看，现有的 AWI 技术大都只能在小规模的实验项目上取得不错的效果，但难以迁移和推广；从实际应用来看，我们几乎没有看到有技术被工业界认可并被真实地整合到了他们的工作流中；从评估标准来看，大规模的、标准的、被广泛认可的评估指标（实验数据集等）目前仍未建立。这也说明了这些技术虽经过多年的发展，却依然不够成熟，考虑到这个问题的复杂程度，这确实符合情理，但距离大规模的应用应该还有很长的路要走。

5.4.2 主观角度

在上一节中已经给出了我们的思考：我们认为这个问题并不平凡。为了在现阶段改善静态分析器的使用体验，科研工作者们做出了诸多妥协，比如，在工作中引入“用户”这个角色，或是退而求其次，在无法标注警告有效性的前提下，仅仅对警告进行排序或是聚类 [18]，以帮助开发者更好地发现真正的 bug 。上文中我们也分析过，AWI 这个问题和“如何改善静态分析器”这个终极问题似乎是同质化的，这就注定了这不是一个简单的问题。因此，针对这个问题的任何尝试都应该怀抱着极其严谨的态度，我们不应轻信任何冒进的、轻率的结论。

真正的有效警告识别，仍任重道远。

6 * 感想与后记

工作进行至此，确有不少感想。尽管这一部分原则上不应该在正文中出现[††]，但我还是想要留下一些书面的记录，以表达我对他人帮助的感谢之情。

在整个调研和文献综述的写作过程中，我最大的感受是“战战兢兢，如履薄冰”，因为我们所能接触到的真正的科研工作与通常在课本上接触到的简洁优美的方法和结论不同，这些工作繁琐且复杂，涉及不懂的知识和领域太多，而我们的知识与经验太少，再加上时间紧迫，颇有力不从心的感觉。

这份报告写完后，我最害怕的有两点：一是由于我们对于一些方法的理解不到位和文献搜索过程中的疏忽导致出现的明显的错漏之处；二是由于我们的观点太过激进而显得自高自大、自以为是，对他人有所冒犯。

如果真的有第一点的情况出现，我确实具有无可辩驳的责任；但关于第二点，我想要在此声明：我们仅仅只是本科三年级学生，在文献综述的写作过程中也很努力地在保持“探讨问题”的态度。我们绝对尊重所有科研工作者的辛勤劳动，如有冒犯，非常抱歉。

感谢小组中的每一个成员。四人小组中，只有我（组长）是方向课，而其他人都是非方向课，感谢他们百忙之中抽出时间陪我进行讨论和调研，共同完成了这份“热情洋溢”的课程作业，尽管结果未知，但这份作业也确是我们努力的结晶。

感谢葛修婷学姐的指导，感谢房春荣老师和诸位其他助教的付出。在做作业的过程中，我确实过了一把科研的瘾，能做这样的作业，我感觉到十分满足。

感谢在我心情极差的时候安慰我、帮助我、听我倾诉、帮我分担工作的同学们。感谢 22 级的学妹陈洁，她在我最崩溃的时候轻轻拉了我一把，蓦然回首处，云散月明。我才能在 DDL 堆积的日子里保持热情和专注，要不然我现在可能已经被退学了。

我在这里向他们表示深深的敬意。

蔡之恒

2022 年 11 月 17 日

参 考 文 献

[1] Steven Arzt, Siegfried Rasthofer, Robert Hahn, and Eric Bodden. Using targeted symbolic execution for reducing false-positives in dataflow analysis. In Proceedings of the 4th

[††]所以能不能不要计算到页数中去

ACM SIGPLAN International Workshop on State Of the Art in Program Analysis, SOAP 2015, page 1 – 6, New York, NY, USA, 2015. Association for Computing Machinery.

[2] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. Analyzing the state of static analysis: A large-scale evaluation in open source software. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 1, pages 470–481, 2016.

[3] Al Bessey, Ken Block, Benjamin Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later using static analysis to find bugs in the real world. Commun. ACM, 53:66–75, 02 2010.

[4] Tianyi Chen, Kihong Heo, and Mukund Raghothaman. Boosting static analysis accuracy with instrumented test executions. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, page 1154 – 1165, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Katarzyna Filus, Paweł Boryszko, Joanna Domańska, Miltiadis Siavvas, and Erol Gelenbe. Efficient feature selection for static analysis vulnerability prediction. Sensors, 21(4), 2021.

[6] Google. Classification: Roc curve and auc. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=en. Accessed: 2022-11-16.

[7] Sarah Heckman and Laurie Williams. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, page 41 – 50, New York, NY, USA, 2008. Association for Computing Machinery.

[8] Sarah Heckman and Laurie Williams. A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 53(4):363–387, 2011. Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[9] Péter Hegedűs and Rudolf Ferenc. Static code analysis alarms filtering reloaded: A new real-world dataset and its ml-based utilization. IEEE Access, 10:55090–55101, 2022.

[10] Ashwin Kallingal Joshy, Xueyuan Chen, Benjamin Steenhoek, and Wei Le. Validating static warnings via testing code fragments. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2021, page 540 – 552, New York, NY, USA, 2021. Association for Computing Machinery.

[11] Hong Jin Kang, Khai Loong Aw, and David Lo. Detecting false alarms from automatic static analysis tools. In Proceedings of the 44th International Conference on Software Engineering. ACM, may 2022.

[12] Hyunsu Kim, Mukund Raghothaman, and Kihong Heo. Learning probabilistic models for static analysis alarms. In 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pages 1282–1293, 2022.

[13] Ugur Koc, Parsa Saadatpanah, Jeffrey S. Foster, and Adam A. Porter. Learning a classifier for false positive error reports emitted by static code analysis tools. In Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL 2017, page 35 – 42, New York, NY, USA, 2017. Association for Computing Machinery.

[14] Ugur Koc, Shiyi Wei, Jeffrey S. Foster, Marine Carpuat, and Adam A. Porter. An empirical assessment of machine learning approaches for triaging reports of a java static analysis tool. In 2019 12th IEEE Conference on Software

Testing, Validation and Verification (ICST), pages 288–299, 2019.

[15] Seongmin Lee, Shin Hong, Jungbae Yi, Taeksu Kim, Chul-Joo Kim, and Shin Yoo. Classifying false positive static checker alarms in continuous integration using convolutional neural networks. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), pages 391–401, 2019.

[16] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandè. Avatar: Fixing semantic bugs with fix patterns of static analysis violations. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 1–12, 2019.

[17] Diego Marcilio, Rodrigo Bonifácio, Eduardo Monteiro, Edna Canedo, Welder Luz, and Gustavo Pinto. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pages 209–219, 2019.

[18] Tukaram Muske and Alexander Serebrenik. Survey of approaches for handling static analysis alarms. In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 157–166, 2016.

[19] Thu Trang Nguyen, Pattaravut Maleehuan, Toshiaki Aoki, Takashi Tomita, and Iori Yamada. Reducing false positives of static analysis for sei cert c coding standard. In 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SERIP), pages 41–48, 2019.

[20] José D'Abruzzo Pereira, João R. Campos, and Marco Vieira. Machine learning to combine static analysis alerts with software metrics to detect security vulnerabilities: An empirical study. In 2021 17th European Dependable Computing Conference (EDCC), pages 1–8, 2021.

[21] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at google. Commun. ACM, 61(4):58 – 66, mar 2018.

[22] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. Tricorder: Building a program analysis ecosystem. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, volume 1, pages 598–608, 2015.

[23] Miltiadis Siavvas, Ilias Kalouptsoglou, Dimitrios Tsoukalas, and Dionysios Kehagias. A self-adaptive approach for assessing the criticality of security-related static analysis alerts. In Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Chiara Garau, Ivan Blečić, David Taniar, Bernady O. Apduhan, Ana Maria A. C. Rocha, Eufemia Tarantino, and Carmelo Maria Torre, editors, Computational Science and Its Applications – ICCSA 2021, pages 289–305, Cham, 2021. Springer International Publishing.

[24] Mohammad Tahaei, Kami Vaniea, Konstantin (Kosta) Beznosov, and Maria K Wolters. Security notifications in static analysis tools: Developers'attitudes, comprehension, and ability to act on them. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[25] Akshay Utture, Shuyang Liu, Christian Gram Kalhauge, and Jens Palsberg. Striking a balance: Pruning false-positives from static call graphs. In 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pages 2043–2055, 2022.

[26] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Harald C. Gall, and Andy Zaidman. How developers engage with static analysis tools in different contexts. Empirical Software Engineering, 25(2):1419–1457, Mar 2020.

[27] Junjie Wang, Song Wang, and Qing Wang. Is there a ”golden” feature set for static warning identification? an experimental evaluation. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM ’18, New York, NY, USA, 2018. Association for Computing Machinery.

[28] Xueqi Yang, Jianfeng Chen, Rahul Yedida, Zhe Yu, and Tim Menzies. Learning to recognize actionable static code warnings (is intrinsically easy). Empirical Software Engineering, 26(3):56, Apr 2021.

[29] Xueqi Yang and Tim Menzies. Documenting evidence of a reproduction of ‘is there a “golden”feature set for static warning identification? —an experimental evaluation’. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, page 1603, New York, NY, USA, 2021. Association for Computing Machinery.

[30] Xueqi Yang, Zhe Yu, Junjie Wang, and Tim Menzies. Understanding static code warnings: An incremental ai approach. Expert Systems with Applications, 167:114134, 2021.

[31] Ulas Yüksel and Hasan Sözer. Automated classification of static code analysis alerts: A case study. In 2013 IEEE International Conference on Software Maintenance, pages 532–535, 2013.

[32] Ulaş Yüksel and Hasan Sözer. Dynamic filtering and prioritization of static code analysis alerts. In 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pages 294–295, 2021.

[33] Xin Zhang, Radu Grigore, Xujie Si, and Mayur Naik. Effective interactive resolution of static analysis alarms. Proc. ACM Program. Lang., 1(OOPSLA), oct 2017.