iResolveX：通过静态推理与学习增强精化实现多层间接调用解析

Monika Santra*、Bokai Zhang*、Mark Lim†、Vishnu Asutosh Dasu*、Dongrui Zeng†、Gang Tan*
*宾夕法尼亚州立大学{*mvs7165*、*bzz5205*、*vdasu*、*gtan*}*@psu.edu* †帕洛阿尔托网络公司{*malim*、*dzeng*}*@paloaltonetworks.com*

arXiv:2601.17888v1 [cs.SE] 25 Jan 2026

摘要—间接调用解析仍然是逆向工程和控制流图恢复中的一项关键挑战，尤其是对于被剥离符号或优化后的二进制文件。静态分析能够保证可靠性，但会进行过度近似，从而导致大量假阳性；而机器学习方法虽然能够提高精确率，却会牺牲完备性和泛化能力。现有的 Ghidra、IDA 和 Angr 等工具在效果方面落后于学术界的方法。我们提出了 iResolveX，这是一种将保守静态分析与基于学习的细化相结合的混合多层框架。第一层采用保守的值集分析（BPA）来确保高召回率。第二层引入基于学习的软签名评分器（iScoreGen），以及结合内存检查的选择性过程间反向分析（iScoreRefine），以减少假阳性。最终输出 p-IndirectCFG 会为间接边标注置信度分数，使其能够适应不同的下游分析，这些分析往往需要不同的精确率—召回率权衡。在 SPEC2006 和真实世界二进制文件上，iResolveX 的 iScoreGen 在保持 BPA 级别召回率（98.2%）的同时，使预测目标平均减少了 19.2%。与 iScoreRefine 结合后，相较于 BPA 的总体减少幅度达到 44.3%，召回率为 97.8%，仅损失 0.4%。此外，iResolveX 还能适应多样化的分析需求，提供保守、保持召回率或 F1 分数优化的配置，并且在这两类设置下均优于最先进的系统。

1. 引言

二进制分析，也称为二进制逆向工程（RE），对于当代软件安全实践至关重要。它支持漏洞检测、恶意软件分类、漏洞利用分诊以及对程序的整体理解等重要功能。这些操作的核心是控制流图（CFG），它能够有效地对二进制文件中的执行路径进行建模。构建准确且全面的 CFG [52]对于各种下游应用至关重要，包括控制流完整性（CFI） [1], [2], [8], [11], 符号执行 [34], [50], 反编译 [55],污点跟踪 [9],和模糊测试 [22], [26]。CFG 中的不准确之处，例如缺失的边或不精确的执行路径，可能导致漏洞被忽略、产生误导性发现，以及测试覆盖率不足。此外，随着图神经网络（GNN） [54], [57]和大语言模型（LLM） [31], [32] 越来越依赖源自 CFG 的表示，这些图的准确性对于监督信号变得至关重要。CFG 恢复中的错误可能以偏差训练数据的形式传播，从而损害正确性和可靠性。

恢复控制流图面临的一项重大挑战是解析间接调用 [20], ，其目标由寄存器、内存或计算地址在运行时确定。这带来了很大的不确定性，使静态分析更加复杂。此外，动态分析或混合分析 [27], [30], [37]通常受到覆盖率的限制；处理缺少关键符号和类型信息的剥离或优化二进制文件时，这一问题会更加突出。为应对这一挑战，研究工作主要沿着两条并行路径展开——静态分析 [14], [20], [29],[36], [37], [44], [45] 和基于机器学习的方法 [39], [58]——两者各有优势与局限，且相互补充。

静态分析（SA） [6], [20], [21], [45], [53], （如类型分析或值集分析（VSA） [5], ）通过保守地过度近似调用目标来实现较高的召回率。这确保了可靠性，但往往以牺牲精确率 [20]为代价。虽然这适用于控制流完整性（CFI），但会产生过多的假阳性，从而妨碍可达性分析和符号执行等对精确率敏感的任务。相比之下，基于机器学习（ML）的方法着重于通过过程内切片和模式匹配，将调用点与可能的被调用函数关联起来，以提高精确率 [58]。然而，这些方法通常假设直接调用和间接调用共享相同的输入特征空间——涵盖函数签名、代码模式和内存交互——却忽略了二者的特征分布可能存在根本差异。例如，直接调用通常指向固定目标，因此其反向特征或上下文特征较浅且结构稳定；而间接调用依赖运行时计算的指针，因此会产生更深且变化更多的上下文依赖。因此，主要基于直接调用训练的模型 [7], [48], [56] 在应用于语义更复杂的间接调用时往往表现不佳。若缺乏针对间接调用真实标注的显式监督和校准，这些模型的召回率往往较低。尽管它们

提供了一种颇有前景的方向，但无法彻底消除假阳性，也无法保证可靠性，因此限制了它们在精度与安全至关重要的应用中的可信度。

在实践中，两种方法都面临挑战：静态分析会产生大量噪声，令分析人员应接不暇；机器学习则可能遗漏关键行为。与此同时，主流逆向工程工具（如 IDA [16],、Ghidra [35], 和 Angr [38] ）仍依赖浅层启发式方法、简单的常量传播或不切实际的动态模拟，最终导致控制流图不准确。这种局面迫使分析人员在充满噪声的静态分析结果与不确定的机器学习预测之间做出选择，而能够灵活调整精确率-召回率配置以满足多样化安全需求的选项却十分有限。

局限性。我们指出了现有方法的三项主要局限。首先，静态分析与机器学习技术之间仍存在脱节；现有研究未能将静态推理的可靠性与基于学习的模型的适应性结合起来。其次，机器学习方法往往依赖过度简化的特征，限制了从直接调用到间接调用的泛化能力，并且必须依赖成本高昂的监督，或以牺牲可靠性为代价。例如，这些方法缺乏可扩展的跨过程支持。第三，逆向工程工具在技术上仍落后于学术解决方案，也缺乏精确率与召回率之间的灵活性，因此难以满足多样化的现实分析需求。**解决方案：**我们提出 iResolveX，这是一种用于间接调用解析的混合框架，可在最大限度减少召回率损失的同时提升精确率，并保留静态分析在可靠性方面的优势。该框架首先通过值集分析（BPA）保守地近似潜在的被调用函数。随后，采用轻量级、基于机器学习的软签名匹配器（iScoreGen），并使用直接调用对进行训练，以学习参数结构和内存访问模式等抽象兼容性特征。这些特征具有良好的泛化能力，即使面对剥离符号或高度优化的二进制文件，也适用于间接调用。接着，我们应用选择性过程间反向分析（iScoreRefine），提取关键代码上下文和内存—代码依赖。基于这些信息，我们重新调整初始预测分数，根据支持性的静态证据增强或抑制机器学习的输出。最终输出 p-IndirectCFG 为间接边标注置信度分数，使下游工具能够针对不同的安全关键型和精确率关键型任务调整精确率—召回率权衡。

贡献：我们提出 iResolveX，这是一种多层框架，旨在弥合间接调用解析中静态分析的可靠性与机器学习带来的精确率之间的差距。我们的主要贡献包括：

- **混合架构：**一种将可靠的静态分析与基于学习的细化相结合的统一设计。
- **零监督学习：**一种新颖的特征解耦策略，使模型能够在没有动态训练监督的情况下实现从直接调用到间接调用的泛化。
- **过程间推理：**一种可扩展的反向分析，用于恢复远程的过程间指针依赖关系，以验证机器学习模型的预测。
- **置信度标注与上下文自适应控制流图：**灵活的输出（p-IndirectCFG），支持根据应用需求动态定制控制流图。
- **实证验证：**与 BPA 相比，iResolveX 将平均间接调用目标（AICT）减少了 44.3%，召回率仅损失 0.4%。在召回率保持设置下，它以更高的召回率实现了比以往工具多 18–48.5% 的 AICT 降低。在 F1 优化模式下，它实现了高出 82–97% 的降低幅度，同时维持 80% 的 F1 分数和 82–89% 的召回率，优于现有系统。我们的制品见：Link。

2. 背景与相关工作

尽管经过数十年的研究，间接调用解析仍然需要在可靠性与精确率之间进行权衡。静态分析会产生过度近似，机器学习模型存在泛化不足且不完整的问题，而逆向工程工具则依赖脆弱的启发式方法。没有任何一种方法能够在各种应用场景中灵活平衡召回率与精确率。我们回顾了几类关键方法——程序分析（BPA [20], SchedExec [37], BinDSA [14]）、基于学习的方法（CALLEE [58], AttnCall [39]）以及逆向工程工具——并重点介绍它们的技术与局限性。

基于分析的方法：由分析驱动的技术仍然是间接调用解析的基础，因其可靠性和可解释性而受到重视，这些特性对于控制流完整性等安全任务至关重要。经过精心设计后，它们能够生成可靠但较为粗略的调用图，却牺牲了精确率和适应性。我们考察了三个代表性系统：SchedExec [37], BPA [20], 和 BinDSA [14],，每个系统都展现了不同的设计权衡，并促使人们探索更加灵活的解决方案。

SchedExec [37] 将动态执行与静态 IR 级重放相结合，提供了一种实用的混合方法。然而，它存在过早剪枝和过于简单的假设，例如假设函数指针位于连续的内存条目中。这种方法在包含稀疏数组、嵌套结构或混淆布局的真实二进制文件中会失效。SchedExec 忽略别名关系、栈参数和过程间流，也无法处理递归内存间接寻址。最关键的是，SchedExec 在完整的值集分析（VSA）之前就执行剪枝，在缺乏保守控制流目标近似的情况下移除程序路径。这会损害可靠性：一旦潜在目标被过早移除，任何事后分析都无法将其恢复。它无法处理 gcc 等复杂基准测试，进一步限制了其适用性。

BinDSA [24] 将数据结构分析（DSA） [25]应用于提升后的 IR，以推断结构和指向集。在指向分析期间，它利用恢复出的结构信息，在调用点剪除间接调用候选项。即使不考虑 IR 提升噪声，其过早剪枝也会导致 gcc 等大型基准测试中的召回率大幅下降。此外，数据结构恢复中的错误会导致许多有效调用点仍无法解析。

BPA [20] 采用完全静态的、基于 Datalog 的流水线 [19]，以构建可扩展且可靠的控制流图。其核心抽象将内存划分为互不相交的块，并假设指针算术不会超出块的边界。这避免了对块内偏移进行推理，并支持高效、保守的指向分析，适用于大型剥离符号的二进制文件和

控制流完整性。然而，块级建模会导致严重的过度近似，并且依赖脆弱的内存分区启发式方法。WhileDISA [51]通过机器学习驱动的基本块恢复改进了这一点，但仍然存在的精确率差距依旧会导致间接调用解析产生不可忽视的过度近似。

机器学习驱动的方法： 近期的基于机器学习的方法，如 CALLEE [58] 和 AttnCall [39]，旨在通过过程内寄存器级切片，学习调用点与被调用函数之间的兼容性，从而克服静态分析的过度近似。虽然这些方法实现了自动化并能快速推理，但它们依赖切片，这带来了重大局限，尤其是在真实世界二进制文件中。较小的被调用函数通常缺少足够的指令，无法生成有意义的切片；CALLEE 会将这些函数完全舍弃，从而漏掉有效目标。这两种方法在处理指向短包装函数或优化代码的间接调用时都表现不佳，而这类情况在性能关键型二进制文件中很常见。它们的过程内视角忽略了跨过程流程、别名关系 [4],以及代码与内存之间的交互，因此难以捕获长距离依赖。缺乏可解释性也进一步降低了它们在安全敏感任务中的适用性 [1], [2]。

这两种方法均以 x86-64 [18], [43]为目标，并遵循相似的流水线：从直接调用对中提取上下文切片，并预测间接调用关系。CALLEE 使用孪生网络，并进行有限的间接调用微调；AttnCall 则采用仅使用直接调用训练的 Transformer [48]。两者都依赖一个有缺陷的假设，即直接调用和间接调用的过程内寄存器级程序切片具有相似的上下文模式。实际上，与直接调用不同，间接调用的反向上下文通常涉及驻留在内存中的函数指针、存在别名的栈帧，或跨过程的控制／数据依赖关系，而这些模式往往难以通过局部切片捕获。在处理未经微调的间接调用时，AttnCall 的召回率会大幅下降（最低可至 70%）。CALLEE 虽然引入了微调，但需要成本高昂的动态真实标注；当切片缺失或指针结构复杂时，它仍会失效。AttnCall 沿袭了这些弱点，并因省略微调阶段而进一步损害了泛化能力。更深层的模型无法弥补结构推理的缺失。因此，这些模型不太适用于要求高精确率或涉及安全敏感的应用。

逆向工程工具的方法： 主流逆向工程工具（如 IDA [16], Ghidra [35], 和 Angr [38] ）通过启发式方法、有限的常量传播或代价高昂的仿真来解析直接调用，导致目标集欠近似、控制流图不完整。其中，Angr 凭借其仿真和值集分析后端表现最佳。然而，Angr 的分析缺乏过程间推理，且仿真开销很大，即使处理小型二进制文件也会变慢。这些局限使它不适用于大型或剥离符号的二进制文件；这类文件需要具备可扩展性、感知分析结果且可控可靠性与精确性权衡的控制流图构建方法。

3. 动机与概述

由于现有方法无法同时实现可靠性、精确率，间接调用解析仍然具有挑战性，

通过考察这些方法的局限性，我们提炼出一系列设计见解，为 iResolveX 奠定了基础。

静态分析和机器学习单独使用都不足够——必须采用统一方法： 只有将静态分析与机器学习紧密集成的框架，才能在间接调用解析中兼顾召回率、精确率和可扩展性。单独采用任何一种方法，都无法达到真实世界分析所需的均衡准确率。为弥合这一差距，iResolveX 采用多层架构，让静态分析发挥其所长——确保可靠的过度近似；同时让机器学习发挥其优势——学习可指导剪枝与细化的模式。

为保证可靠性的 VSA 后剪枝： 必须首先保留保守近似，仅在构建可靠的过度近似之后才能应用剪枝。为弥合这一差距，iResolveX 采用 BPA 的基于基本块的值集分析作为第 1 层（BPA），通过收集间接调用目标的超集来确保可靠性。然而，传统 VSA 的开销可能很高且精确率不足，尤其是在大型或剥离符号的二进制文件中。BPA 的基本块抽象兼具实用性和高召回率，但由于建模较为粗略，会产生超过 40% 的假阳性。为在保留覆盖率的同时实现可用的精确率，第 2 层（iScoreTune）引入了基于学习的剪枝策略，同时使第 1 层保持足够的模块化，以便替换为任意可靠的控制流图生成器。

泛化所需的不仅仅是切片： 要在本质不同的调用结构之间实现泛化，仅靠浅层切片远远不够。泛化需要将稳定的签名特征与间接调用特有的、上下文丰富的行为区分开来。如果不整合内存推理、上下文验证和过程间流，基于机器学习的解决方案仍会十分脆弱。为弥合这一差距，iResolveX 采用特征分离策略，区分对于准确间接调用解析至关重要的可泛化模式和上下文敏感模式。它利用函数签名特征——例如参数数量、寄存器使用情况等——这些特征在直接调用和间接调用之间保持一致，从而实现稳健的泛化。同时，它从后向切片中提取上下文敏感特征，例如指针解引用和内存加载，以捕捉间接调用的细微行为。这种分离构成了第 2 层（iScoreTune）的基础，其中包括（L2a）iScoreGen（一种用于软签名评分的基于学习的模块）以及（L2b）iScoreRefine（一种有针对性的过程间分析，用于根据代码和内存依赖调整分数）。

第 2a 层：iScoreGen – 软签名的作用： 稳健的泛化始于软签名匹配 *[29], [41],* 它抽象掉了静态分析无法处理的结构噪声和编译器引起的变化，使机器学习成为唯一可行的方法。在调用点之间匹配函数签名颇具挑战，尤其是在剥离符号的二进制文件中，因为优化会破坏既有约定。iScoreGen 通过软签名匹配解决这一问题。它采用一种轻量级机器学习模型，在直接调用对上进行训练，以学习在直接调用和间接调用中都保持一致的兼容性模式。这种设置无需耗时且详尽的间接调用真实标注，即可生成具有概率性的

表1：基于机器学习的工具与 iResolveX 的设计比较

方面	**CALLEE** [58]	**AttnCall** [39]	**iResolveX**
间接调用相关真实标注训练期间所需	Yes	No	No
特征设计	混合参数/上下文切片	混合参数/上下文切片	分离：签名 (L2a) + 上下文 (L2b)
模型类型	孪生网络 +微调	Transformer（大型、不透明）	轻量级 4 层隐藏层深度神经网络；无需微调；快速推理
直接到间接泛化	较弱；直接 ≠ 间接	声称具备泛化能力；实证依据薄弱	较强：基于直接调用进行训练，并通过静态分析加以优化
过程间支持	有限（仅限被调用者–调用者）	None	选择性后向分析 + 内存扫描
小型函数支持	No	No	Yes

分数，并且能够适应结构变化。然而，仅依靠基于签名的推理仍存在歧义，因为真实的被调用函数和错误的被调用函数可能具有相同的原型。因此，iScoreRefine 使用静态后向内存和控制流上下文，对机器学习预测进行验证或调整。

第 2b 层：iScoreRefine——可扩展过程间推理： 实用的过程间解决方案必须兼顾深度与可扩展性，在最关键之处有选择地进行过程间细化。 完全的过程间反向分析成本过高，而现有机器学习工具缺乏上下文感知能力。iScoreRefine 仅在调用点参数与调用方相符时，才执行选择性过程间分析。它会检查这些路径上的代码和内存，以捕捉内存—控制流耦合，并重新调整机器学习分数：依赖关系匹配时提高置信度，否则予以惩罚。这种选择性策略构成了紧密耦合的机器学习—静态分析框架，能够实现现实应用所需的精确率与可扩展性。

下游应用的不同需求： 在实践中，几乎无法实现完全准确的间接调用解析。在能够实现精确解析之前，控制流图构建期间的间接调用解析需要具备灵活性，以便下游分析根据精确率或召回率需求调整置信度。 学术界和逆向工程工具都将控制流图边视为二元关系，无法满足不同的精确率—召回率需求。例如，符号执行受益于剪枝后的控制流图，而控制流完整性或恶意软件分析则要求更高的召回率，即使这会带来误报边。p-IndirectCFG 通过为间接边分配置信度分数（由 iScoreGen 和 iScoreRefine 得出）来解决这一问题。需要精确率时，下游任务可以关注高置信度边；需要全面覆盖时，则可以纳入置信度较低的边。

在介绍 iResolveX 的完整工作流程之前，我们先概述其设计与以往基于机器学习的方法（表1）的不同之处。与 CALLEE 和 AttnCall 不同，后两者假设调用点切片与被调用函数切片共享相似的特征空间，而 iResolveX 将该空间划分为：（1）函数签名空间和（2）反向上下文空间。函数签名特征在直接调用和间接调用中保持稳定，因此，使用直接调用点—被调用函数对训练的软签名评分器也适用于间接调用。相比之下，直接调用与间接调用的反向上下文特征往往不同；从直接调用中学到的模式可能并不适用。因此，我们选择对反向上下文进行静态分析，以减少评分器导致的假阳性。

由于关键上下文通常位于过程间路径上，iResolveX 会针对间接调用执行过程间反向遍历，同时采用选择性遍历以提升可扩展性。这种结合式特征设计提升了泛化能力，避免了对直接调用进行不必要的分析，并使轻量级深度神经网络成为可能，而以往的方法无法做到这一点。

工作流程： 图 1 展示了 iResolveX 的工作流程。给定一个剥离符号的二进制文件，iResolveX 分三个阶段运行：**第 1 层（BPA）：** 受 BPA 启发的基本块级值集分析会计算出一个可靠的被调用函数超集。**第 2 层（iScoreTune）：** *L2a iScoreGen*：针对每个调用点—被调用函数对，提取特征并由轻量级机器学习模型进行评分。*L2b iScoreRefine*：针对所有这些配对，选择性后向切片会验证机器学习预测，同时避免代价高昂的完整分析。最终控制流图称为**p-IndirectCFG**，其中的间接边标注了置信度分数，从而支持精确率—召回率权衡。阈值$\in [0,1]$ 控制剪枝：阈值较低 时，控制流图的可靠性更高；阈值较高 时，控制流图的精确率更高。此外，可选的动态分析模块会提供动态跟踪，用于量化给定二进制文件在不同阈值下的召回率保持情况。

4. 方法论

4.1. Layer1：BPA 的设计

我们的流水线以 BPA [20],（一种基本块级指向分析框架）开始，通过保守地扩大间接调用目标的范围来确保可靠性。该过程包括从直接控制流图中创建粗粒度内存块模型，并迭代执行值跟踪和可达性传播，直至收敛。最终的不动点为每个间接调用点得出一组保守的候选目标。此外，通过基于可达性的过滤，该方法能够有效消除死代码中的间接调用，这是现有机器学习框架未涵盖的一项功能。有关实现细节，请参阅原始 BPA 框架。

4.2. 第2层：iScoreTune 的设计

为进一步筛选 BPA 候选项，iScoreTune 通过基于机器学习的软签名匹配（iScoreGen）为调用点—被调用者配对打分，并通过过程间分析（iScoreRefine）验证这些配对，从而有效区分可复用的签名模式与上下文。

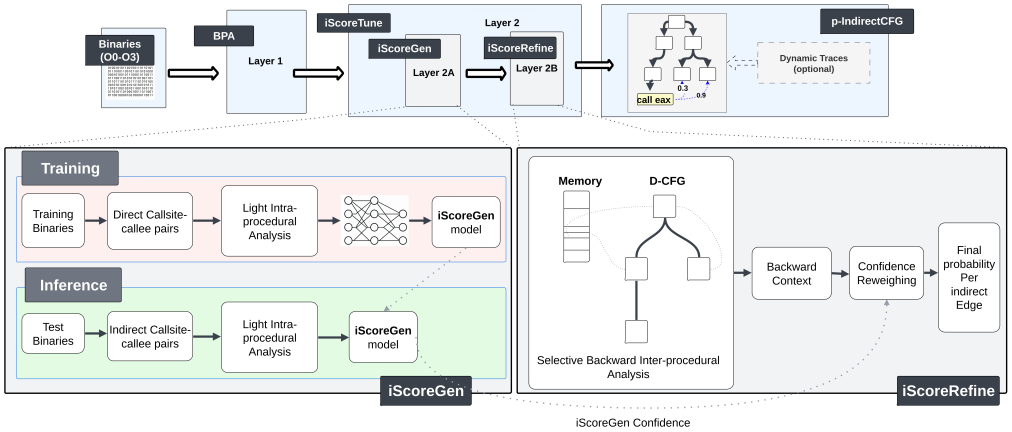


图 1：iResolveX 系统概览

4.2.1. 第2层A：iScoreGen的设计。iScoreGen 使用过程内数据流特征评估调用点—被调用者兼容性，这些特征能够捕捉关键行为模式。调用点特征提取算法见算法1。首先，TRACKINDIRECTCALL 在二进制文件中定位间接调用点，并保存附近的指令，以便后续理解参数使用模式。LIVEVARIABLE- ANALYSIS 会识别在调用点之后变为无效（即其值不再被使用）的变量（如寄存器、全局变量或栈槽），并将其视为可能的隐式参数。EXTRACTARGUMENTSFROMLIVEVARS 会分析活跃变量集，识别其中哪些变量用作函数参数，并提取其特征集。对于每个已识别的参数，CLASSIFYSOURCE 会确定其来源（例如栈、寄存器或全局变量），由此提供有关该值如何生成和传递的上下文。ANALYZE-MEMORYOPERAND 会检查参数是否通过取地址或解引用操作来使用，以判断其是否为指针。DETECTPRE-POSTCHECKS 会检查周围的指令，确认该参数是否在调用点之前或之后经过比较、边界检查或空值检查的验证。INFERPROBABLE- TYPE 会根据使用模式和表示形式分配粗略类型提示（例如整数、char * 等），以语义信息补充结构特征。最后，CHECKRETURNUSAGE 会确认调用是否存在返回值。

例如，清单1展示了一个最简程序，其中包含汇编代码，以及注释中对应的源代码。在 main 中，函数指针 fp 被设为 target_func（第 2 行）。局部变量包括 `= 42`（参数）、`b = 100`，以及栈缓冲区 buf`[]`（参数）。第 12 行的间接调用 fp(a, buf)（仅当 a $>$ 0 时才会调用）将返回值存入 r，随后将其打印出来（第 15 行），之后还会打印 b（第 16 行）。间接调用点之后已失效的变量以黄色突出显示，而调用点之后仍在使用的变量则以绿色标示；为简洁起见，仅标出了最相关的变量。这里，栈槽 -0x20(%ebp)（a）和缓冲区 -0x48(%ebp)（buf`[]`）已失效，并作为参数传递通过 push -0x20(%ebp) 和 push %eax（buf 的地址）。

随后为每个参数提取特征（算法1第 5–8 行）。对于示例中的变量 a，CLASSI- FYSOURCE 将其标记为栈局部变量。它被写入一个立即数常量，在比较中使用（cmpl $0x0, -0x20(%ebp)），并按值传递，表明它是非指针标量（ANALYZEMEMORYOPERAND）；其粗略类型提示被推断为整数（INFERPROBABLETYPE）。对于 buf，其来源同样是栈，但会获取其地址（lea -0x48(%ebp), %eax; push %eax），因此其指针性为真。DETECTPREPOSTCHECKS 中未观察到任何验证检查，这与另一个参数不同；作为一个在栈上分配并以字符串字面量初始化的字符数组，其粗略类型提示为字符指针。返回值使用情况（CHECKRETURNUSAGE，算法1第 10 行）表明，%eax 中的结果被保存到 -0x28(%ebp) 并立即打印出来，图中以紫色突出显示。最后，分析恢复出两个参数，并分别标注其来源、指针性、验证情况和类型提示。它们与返回值使用情况共同构成调用点的特征向量（AGGREGATECALLSITEFEATURES，算法1第 11 行）。

在被调用方侧（算法 2）中，我们通过使用 GETALLSTACKANDREGISTERVARS 检测变量来识别函数可能的参数，并使用 ISUSEDBEFOREWRITTEN 标记任何写入之前被读取的变量，这些变量表示预期由调用方提供的值。在剥离符号或高度优化的二进制文件中，由于内联和寄存器复用等因素，这项任务颇具挑战；我们发现，传统工具（如 Angr）往往无法可靠地识别此类参数。为解决这一问题，并涵盖被调用方未使用某些调用点参数的情况（例如可选参数），我们根据正栈偏移、早期使用、从未写入、跨基本块使用、高访问频率以及正偏移处的直接内存访问等线索，使用 COMPUTEARGUMENTHEURISTICSCORE 为每个候选项累加分数（更多详情见附录表7）；保留累积分数达到 $\geq \tau$ 的候选项作为可能的参数（在实验中我们设定 $\tau = 6.0$ ）。

算法1： 调用点特征提取（iScore-Gen）

输入： 二进制文件 B，间接调用点 c
输出： 表示 c的特征向量 F_c
$R \leftarrow$ 跟踪间接调用（B, c） ▷ 跟踪间接调用的相关信息
call
$L \leftarrow$ 实时变量分析（B, c, R） ▷ 查找调用点之后失效的变量
调用点 R
$A \leftarrow$ 从实时变量中提取参数(L, R) ▷ 推断参数
从死变量中提取，位于 R_c。
对于每个 $a \in A$ **执行**
  a.来源 $\leftarrow$ 分类源码(a) ▷ 确定来源：栈、
  全局变量、寄存器及其他
  a.指针相似性 $\leftarrow$ ANALYZEMEMORYOPERAND(a) ▷
  推断指针行为
  a.验证信息 $\leftarrow$ DETECTPREPOSTCHECKS(B, a, c) ▷
  检查参数是否在调用附近经过验证
  a.类型提示 $\leftarrow$ 推断可能类型(a) ▷ 推断粗略
  基于用法和表示形式的类型
$arg\_count \leftarrow |A|$ ▷ 推断参数的数量
$ret\_used \leftarrow$ 检查返回值使用(B, c) ▷ 检测返回值是否为
used
$F_c \leftarrow$ AGGREGATECALLSITEFEATURES(A, $arg\ count$, R,
ret_used)
返回值 F_c

清单1：调用点特征提取示例

```
1  080491be <main>:
2  080491d6: movl $0x8049196,  -0x1c(%ebp);   int (*函数指针)(int
       , char *) = target_func;
3  080491dd: movl $0x2a,  -0x20(%ebp);   int a = 42;
4  080491e4:移动 $0x64,  -0x24(%ebp)  ; int b = 100;
5  080491eb: lea  -0x48(%ebp) ,   %edi; 缓冲区目标
6  080491ee: mov  $0x804a024, %esi; 字符串字面量
7  080491f3: rep movsl; 字符缓冲区[] ="...";
8  08049200:比较 $0x0,  -0x20(%ebp)  ; 如果(a > 0)，则为函数指针
9  08049209: 取有效地址  -0x48(%ebp) ,   %eax ; 准备缓冲区地址
10 0804920c: 压入  %eax  ; 参数2 缓冲区
11 0804920d: 压入  -0x20(%ebp)  ; 参数1 a
12 08049213: 调用 *-0x1c(\%ebp) ; r = 函数指针(a, 缓冲区);
13 08049218: 移动 %eax,  -0x28(%ebp) ; 保存返回值 r
14 0804921e:压入  -0x28(%ebp)
15 08049226: 调用 printf@plt ; printf("%d",   r)
16 08049231: 压入  -0x24(%ebp)  ;打印("%d\n", b)的开始
```

在运行示例 `target_func` （清单 2）中，我们突出显示了写入之前就被使用的变量。在示例中，位于 `0x8(%ebp)` 处的 `a` 得分为2.0 （栈正偏移访问） + 1.5 （从未写入） + 2.0 （在函数中较早使用） + 0.0 （仅在单个基本块中使用） + 0.0 （使用次数少于三次） + 2.0 （在偏移量为 +处直接访问内存；此项用于指针间接寻址） = 7.5。位于`0xC(%ebp)`处的`buf`得分为2.0 （正偏移） + 1.5 （从未写入） + 2.0 （早期使用） + 0.0 （仅在单个基本块中使用） + 0.0 （访问频率不高）+ 2.0 （在偏移量为 +处直接访问内存） = 7.5。返回值检测对 `%eax` 采用类似规则（总分达到 ≥ 2.0 时标记为返回值）：1.0 （如清单 2 第 8 行写入`%eax` ） + 1.5 （在最后一个基本块中写入） + 0.0 （函数末尾附近没有常量移入`%eax` ）= 2.5，因此存在返回值。选定的参数随后会像调用点处的参数一样，提取相同的特征——CLASSIFYSOURCE、ANALYZEMEMORYOPERAND 和 INFERPROBABLETYPE

算法 2: 被调用函数特征提取 (iScore-Gen)

输入： 二进制文件 B，函数 f
输出： 表示 f 的特征向量 F_f
$V \leftarrow$ 获取所有栈和寄存器变量(f) ▷ 收集栈/寄存器
变量以供检查
$A \leftarrow [\,]$ ▷ 初始化推断参数列表
对每个 $v \in V$ **执行**
  如果 ISUSEDBEFOREWRITTEN(v)**则**
    $score \leftarrow$ COMPUTEARGUMENTHEURISTICSCORE(v) ▷
    为每个可能的候选项评分，以选出概率较高的候选项
    ones
    如果 $score > \tau$ **则**
      a.来源 $\leftarrow$ 分类来源(v) ▷ 确定
      来源：栈、全局变量、寄存器、其他
      a.指针相似性 $\leftarrow$
      分析内存操作数操作数OPERAND（v） ▷ 推断指针
      行为
      a.类型提示 $\leftarrow$ 推断可能类型类型类型类型（v） ▷ 推断一个
      基于用法和表示形式的粗略类型
      A.append(a)
$arg\_count \leftarrow |A|$ ▷ 推断参数的数量
$ret\_present \leftarrow$ CHECKRETURNUSAGE(B, f) ▷ 检测返回值是否
被使用
$F_f \leftarrow$ 聚合被调用者特征(A, $arg\ count$, $ret\_present$) _
返回值 F_f

清单 2：被调用函数特征提取示例

```
1  08049196 <目标_函数>: 2 80491a3: 将 0xc(%ebp) 压
入栈中 ; 缓冲区（使用                  将 0x8(%ebp) 压入栈
中 ; a（使用） 4 8049                  4a008 压入栈中 ; 格式
字符串 5 80491ae: 调用  8049070 <printf@plt>6
80491b3: add $0x10,%esp 7 80491b6: mov 0x8(%ebp),%
eax ; 使用 a ->eax 8 80491b9: add $0xa,%eax ; a  + 109
80491bd: ret ; 返回值 a+10
```

— 从而得到用于基于学习的兼容性评分的被调用者特征向量。

iScoreGen 的训练阶段使用从训练二进制文件中的直接调用获得的带标签调用点–被调用者对。对于每一对，我们使用调用点和被调用函数特征提取算法提取特征，并将其合并为一个特征向量。随后将该向量输入轻量级深度神经网络，以对调用点与被调用函数之间的软兼容性建模，并输出基于 sigmoid 的概率分数。在推理期间，第 1 层的 BPA 会生成候选间接调用点–被调用者对。iScoreGen 计算这些对的特征，并将其输入训练好的深度神经网络以生成兼容性分数，之后由 iScoreRefine 使用过程间静态分析进一步优化这些分数。

4.2.2. 第 2B 层：iScoreRefine 的设计。为了进一步提升预测能力，超越 iScoreGen，我们采用了一种选择性的过程间反向分析。此步骤可恢复长距离指针算术和递归内存链，这些因素会影响间接调用操作数——而过程内分析或基于机器学习的方法往往会遗漏这类依赖关系。恢复出的这些轨迹可提供静态证据，用于确认或排除潜在的调用点–被调用者配对。

算法 3 概述了计算每个间接调用点操作数可

能目标的过程。首先定位包含该调用点的函数和基本块。以此位置为种子，我们在直接控制流图（DCFG）上执行有界的过程间反向遍历（EXTRACTBACKWARDPATHS，第 6 行），其边界由向后探索的基本块数量上限表示。沿着每条反向路径（第 7–16 行），我们检查指令操作数：如果某个操作数直接对应已知的函数起始地址，就将其归类为控制流解析的函数指针。如果操作数指向全局内存（例如 .data、.rodata 或 .bss），则调用 RECURSIVEMEMORYSWEEP 遍历该内存区域。此解析会沿着指针链遍历，直至达到可配置的递归深度，从而恢复仅被间接引用的函数入口点。我们不依赖 BPA 的启发式内存块，而是利用 DISA 的 [51] 机器学习驱动的灵活内存布局，该布局在不同二进制文件间具有更好的泛化能力。

为扩大搜索上下文，我们通过 CROSSREFRESOLVE 纳入递归交叉引用推理。具体而言，我们识别出调用当前函数的调用点，该函数中包含间接调用，并检查这些调用点是否将有意义的操作数（例如指针或地址）作为参数压入。当存在此类证据时，我们会在交叉引用的调用方执行一次向后分析（范围为一个基本块），并按照原始调用点遍历中的步骤（第 7–16 行）重复检查。随后，我们使用相同的反向遍历和内存扫描策略，递归分析上游的交叉引用调用点，并将递归深度限制在最大值以内。这样一来，分析便能捕获源自当前函数作用域之外的符号指针流。

图2通过一个示例进行了说明：间接调用点从 BB1 开始。然而，该调用点压入的参数源自一条较长的跨过程链：它最初在 CallerAddr3 中引入，经 CallerAddr2 传播，再由 CallerAddr1 转发，最终在间接调用点被使用。首先，我们提取从 BB1 开始的反向路径（边 1），并分析其中的指令。在此过程中，我们遇到一条将函数指针 (fp1) 加载到 %eax 中的移动指令；我们将其标记为已解析控制流的指针，并存入该调用点的候选映射中。通过交叉引用解析（边 2- 3），我们发现 CallerAddr1 本身由 CallerAddr2（BB4）调用，后者传递了指针操作数；沿调用链继续向上追溯，CallerAddr3（BB6）则将 GlbAddr1 作为参数压入。此时，我们应用受 DISA 启发的全局块模型，递归扫描从 GlbAddr1 开始的全局内存槽，并发现 fp7 是有效的函数指针（边 4）。扫描还发现了一条指向 GlbAddr2 的指针链，随后我们递归探索该链，直至找不到更多链或达到递归深度上限。交叉引用解析也采用相同的递归策略。最后，为了确保分析大型二进制文件时的实用性，如果任何调用点的分析时间超过预设上限，我们会保留部分结果并继续处理下一个调用点。

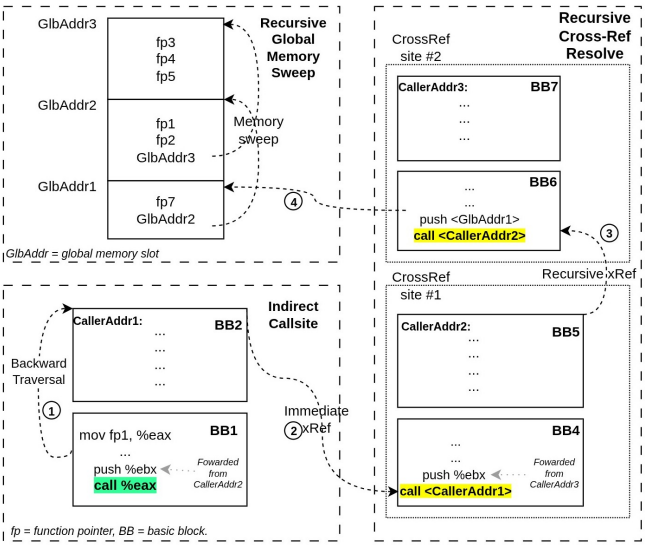


图2：使用反向遍历、交叉引用解析和全局内存扫描进行 L2b 分析的示例。

我们的分析兼具适应性与时间约束。控制流遍历深度、递归内存扫描层级以及交叉引用探索等参数均可调整，以在精确率、覆盖率和运行时之间取得平衡。更深入的设置能够扩大分析范围并提高覆盖率，但需要更多时间，因此在效率与完备性之间形成权衡。每个调用点都设有硬性超时限制，以避免分析大型二进制文件时出现瓶颈。

识别出与调用点相关的函数指针后，我们会细化 iScoreGen 的预测分数：如果 CALLEE 出现在恢复集合中，就将其分数提高一定幅度（例如 +0.1）；如果不在集合中，则将其分数降低相同幅度，并将结果截断至 [0,1]。这种小幅调整使静态证据能够发挥柔性修正作用，补充而非取代机器学习模型的置信度分布。若要更严格地区分正确目标与错误目标，可以采用更大的调整幅度（例如 0.5），以适度牺牲召回率为代价进行更激进的剪枝。这些机制让用户能够根据自身的分析需求，灵活权衡精确率与性能。

4.3. p-IndirectCFG

不同二进制文件中的间接调用用法差异很大，因此单一分数阈值无法在各个基准测试中提供统一的召回率特征。这种差异源于结构和语义因素——从稀疏且彼此隔离良好的分派点，到跨越过程边界的密集、存在别名或多态模式——并且还受到被调用函数密度、目标歧义、编译器优化（例如内联和合并），以及静态行为与运行时行为之间不匹配的影响。因此，尽管一般规律依然成立——较低的阈值会生成更健全的控制流图（召回率更高），而较高的阈值会生成更精确的控制流图（伪

算法 3： 选择性向后跨过程分析（iScoreRefine）

输入： 间接调用点列表 $\mathcal{C}$；DCFG $\mathcal{G}_{CFG}$；受 DISA 启发的全局内存块 $\mathcal{M}$；函数起始集合 $\mathcal{S}$；函数范围 $\mathcal{F}$；交叉引用映射 $\mathcal{X}$；最大控制流图反向遍历高度 H；最大内存扫描深度 R；最大交叉引用深度 D；超时 T

输出： 从调用点 → 到已解析目标集合的映射 $\mathcal{T}$

$\mathcal{T} \leftarrow \emptyset$ ▷ 为所有调用点初始化结果映射
对每个调用点 $C \in \mathcal{C}$ **执行**
  $T_C \leftarrow \emptyset$ ▷ 调用点 C的已解析目标
  开始时间 ← 获取当前时间() ▷ 启动计时器以检查超时
  f_C, b_C ← 包含该内容的函数和基本块 C ▷ 定位包含函数和基本块
  $\mathcal{P}_C$ ← 提取反向基本路径(b_C, H) ▷ 提取反向高度为 H 的基本块的控制流路径
  对于每条路径 $\pi \in \mathcal{P}_C$ **执行**
    对于每条 指令 $i \in \pi$ **执行**
      如果 已失效($start\_time$) $> T$ **则**
        全部中断 ▷ 此调用点已超时
      如果操作数 i 是立即数 $\in \mathcal{S}$ **则**
        T_C.add(函数_指针) ▷ 记录直接函数指针
      如果操作数 i 引用了 $\mathcal{M}$ 中的内存**则**
        F_{mem} ← RECURSIVEMEMORYSWEEP(i, $\mathcal{M}$, R) ▷ 解引用指针链
        遍历 $f \in F_{mem}$ **执行**
          T_C.add(函数_指针) ▷ 记录已解析的内存函数指针
    CROSSREFRESOLVE(f_C, T_C, $start\_time$, D, 0)
    $\mathcal{T}[C] \leftarrow T_C$
  返回值 $\mathcal{T}$
  函数 CrossRefResolve(f, T_C, $start\ time$, D, d)**:**
    如果 $d \geq D$ **那么**
      返回
    $\mathcal{X}_f \leftarrow \mathcal{X}[f]$ ▷ 获取函数 f 的交叉引用调用点
    对每个 $x \in \mathcal{X}_f$ **执行**
      如果 *arg at* x 是指针或基于内存的 **那么**
        如果 ELAPSED($start\_time$) $> T$ **那么**
          中断
        $\mathcal{P}_x \leftarrow$ EXTRACTBACKWARDPATHS(x,1) ▷ 交叉引用站点的 1-基本块后向切片
        对每个 $\pi \in \mathcal{P}_x$ ▷ 重复第 7-16 行的分析循环分析 xRef 路径并更新 T_C
        $f_x \leftarrow \mathcal{X}[x]$
        对每个 $f' \in f_x$ **执行**
          如果 x 处的参数是指针或基于内存的**则**
            CrossRefResolve(f', T_C, $start\_time$, D, $d+1$)

边更少）——但最佳截断阈值仍因基准测试而异。

为了针对每个二进制文件校准阈值，我们加入了一个可选的动态分析模块，用于运行程序并将间接调用边记录为调用点–被调用者地址对。由于动态跟踪始终依赖输入且并不完整，在这种情况下，即使跟踪数据严重不完整，也足以为阈值设定提供依据。此外，动态跟踪会突出显示实际执行过的边，引导阈值校准在保留关键行为的同时，安全地剪除不太可能在运行时出现的低置信度边，从而确保最终的控制流图对下游分析而言仍然准确且相关。即使没有动态跟踪，我们的工具仍然有用：将阈值设为零可保留所有第 1 层候选项，以确保可靠性；而提高阈值会剪除低置信度边，从而提高精确率，但会牺牲召回率。

5. 实现

BPA (L1)：我们直接复用了基于 Souffl´e 的 [19] BPA 实现，该实现为每个间接调用点生成一个可靠的被调用函数候选集。这个保守的超集构成了后续细化的输入。

iScoreGen (L2a) 利用 angr 的过程内数据流分析，包括活跃变量分析、快速变量恢复和调用约定分析，识别调用点和被调用函数两侧的栈变量与寄存器变量，并进一步将其识别为潜在参数。为缓解 Angr 在被调用方侧的精度不足问题，我们采用一种启发式过滤方案（详见附录 A.2）。提取的特征被编码为固定长度的调用点–被调用函数向量，并由一个包含四个隐藏层、批归一化、LeakyReLU 激活函数和 Dropout 的深度神经网络（TensorFlow2 [3]/ Keras [10]）进行分类。模型通过 Sigmoid 神经元输出兼容性分数，并使用二元交叉熵、Adam、提前停止和学习率衰减进行训练（详见附录 A.3）。

iScoreRefine (L2b) 在 DCFG 的 angr 生成的 CFGFast 上执行有界的过程间反向遍历（高度为 = 40 个基本块），并利用 angr 的交叉引用功能进行递归内存检查（深度为 = 1）。我们将目标集与 iScoreGen 的预测合并，并对匹配项保守地调整分数 +0.1，对不匹配项调整 -0.1，以保持召回率。我们还测试了更高的阈值，例如 0.2 到 0.5，以进行更激进的剪枝。

动态跟踪模块： 我们开发了一个基于 PIN 的 [33] 动态跟踪器，用于记录间接调用点及其已执行的被调用函数。对于 SPEC2006 基准测试，我们遵循与 BPA [20], AttnCall [39], 和 BinDSA [14], 相同的方法论，使用 SPEC 提供的输入种子以确保一致性。由于真实世界二进制文件（例如 exim、lighttpd、thttpd、nginx、memcached）缺少标准化的输入/输出配置，现有工具（包括 CALLEE [58]）无法提供动态跟踪。为此，我们手动设计了尽力而为且全面的输入场景，以生成具有代表性的跟踪。在评估中，为平衡 AICT 降低和召回率保持，我们通过分析随机抽取的 30% 动态跟踪中 L2a 和 L2b 的输出分布，为每个基准测试选定一个阈值。对于没有可用输入的二进制文件，我们的跟踪器可与 AFL-Pin [47], 集成，以进行跟踪采集。这种灵活性确保了能够针对不同的二进制文件和执行环境进行阈值校准。

超参数：我们的细化流程公开了用于平衡准确率与运行时的参数：控制流图反向遍历高度（40 个基本块）、内存扫描深度（1）、交叉引用递归深度（3），以及每个调用点 250 秒的超时时间。iScoreRefine 通过 ±0.1 保守地调整分数，以保持召回率，不过也可以采用更高的阈值进行更激进的剪枝。剪枝

p-IndirectCFG 的阈值可选用动态追踪驱动，从而实现精确率—召回率权衡——较高的阈值只保留高置信度边，较低的阈值则倾向于扩大召回率。

数据集与实验设置：我们使用 GCC 7.5 编译了来自 binutils、coreutils 和 MiBench 的 2,664 个 x86-32 [42]二进制文件。Angr 用于提取直接调用点–被调用者对的随机子集，得到 333,063 个有效对和 469,872 个无效对，并按 80% 的训练集和 20% 的验证集进行划分。遵循先前的研究 [14], [20], [39], [58], ，我们的测试数据集由 SPEC2006基准测试和五个真实世界二进制文件（thttpd、memcached、lighttpd、exim 和 nginx）组成，这些文件使用 GCC 9.2.0 编译。在极少数情况下，由于 Angr 的实现问题，其过程内分析未产生结果，我们会在 iScoreGen 中跳过该项，并在后续流程中重新引入，同时将其 L2a 分数设为 0。实验在 Ubuntu 22.04（内核 6.8.0）上运行，配备 352 GB 内存和一颗 44 核 Intel Xeon 处理器。

6. 评估

我们通过考察各层——BPA（L1）、iScoreGen（L2a）和 iScoreRefine（L2b）——对间接调用目标解析的贡献，评估 iResolveX 的有效性。我们还通过案例研究和消融实验，进一步探讨该框架相较于学术工具和逆向工程工具的性能，以及其对真实场景的适应能力。为系统地开展分析，我们提出了五个关键研究问题：

- **RQ1：** iScoreGen（L2a）和 iScoreRefine（L2b）如何与 BPA（L1）互补，以改进间接调用目标解析?
- **RQ2：** iScoreRefine（L2b）的意义何在？在 L2b 中，分数调整阈值的选择对于平衡保守剪枝和激进剪枝有多重要?
- **RQ3：** 与最先进的学术框架和广泛使用的逆向工程工具相比，iResolveX 表现如何?
- **RQ4：** 正如我们的消融分析所研究的那样，移除单个层或多个层的组合会产生什么影响?
- **RQ5：** p-IndirectCFG 中的阈值敏感性如何影响其性能权衡？进而，这会如何影响下游分析?

6.1. RQ1：分层贡献分析

为评估 iResolveX，我们使用两项指标分析其三个层的贡献：BPA（L1）、iScoreGen（L2a）和 iScor- eRefine（L2b）。这两项指标分别是平均间接调用目标数（AICT）和召回率。遵循先前工作 [14], [20],[37], [39], [58],，AICT 作为我们的精确率指标：它表示每个间接调用点预测目标数的平均值（越低越好）。召回率是相对于根据动态执行轨迹得出的伪真实值计算的，应被视为覆盖率的估计，而非绝对召回率，因为可能存在未观测到但有效的目标。我们的目标是在保留覆盖率的同时逐步降低 AICT；因此，我们会结合两项指标进行解读：高召回率与低 AICT 表示预测集较为均衡，而高 AICT 则意味着过度近似。

正如先前工作所确立的那样， [39], [58], 召回率通过两种方式进行评估：全局召回率，即汇总所有调用点上的正确预测；以及 AICT 召回率，即计算每个调用点的召回率并取平均值。这两项指标的正式定义见附录 A.1。尽管全局召回率能够反映整体正确性，但当数据集包含许多简单调用点和少数复杂调用点时，它可能会产生误导。在这种情况下，较高的全局召回率可能掩盖关键复杂案例上的失误，造成系统鲁棒性很强的假象。相反，AICT 召回率对每个调用点一视同仁，能够更准确地体现平均性能。因此，我们将 AICT 召回率作为主要指标。

我们的分层评估（表 2）突显了 iResolveX 的有效性。我们展示了对 BPA 的重新评估结果：仅使用 L1 会生成保守的、过度近似的目标集合，但能确保最高的召回率。加入 iScoreGen（L2a）后，通过机器学习驱动的软签名匹配，AICT 精确率提升了 19.2%；而 iScoreRefine（L2b）进一步将剪枝效果提升了 31%，使整体表现较 BPA 提升 44.3%，且召回率几乎没有损失（0.4%）。由于可达性分析会过滤掉死代码，部分 AICT 值低于 1；这是以往机器学习研究未提供的功能 [39], [58]。perlbench、h264ref、gobmk、exim 和 nginx 等基准测试相较于纯静态分析，表现提升了 30–75%。与 [20]，中采用僵化参数数量过滤器的 BPA 混合设计相比，iResolveX 额外减少了 24.2%，验证了其基于学习的分层设计。

6.2. RQ2：iScoreRefine — 静态过程间验证的重要性

本节探讨（i）iScoreRefine（L2b）在间接调用解析中的作用，以及（ii）其保守模式与激进模式之间的平衡。L2a 通过机器学习驱动的软签名匹配增强剪枝，而 L2b 则采用双重策略：保守模式优先考虑召回率，激进模式则接受召回率方面的权衡，以获得更高的精确率。当放宽召回率限制时，L2b 的优势便会显现，从而实现更有力的剪枝和显著的 AICT 降低——这是现有工具所不具备的能力。我们通过使用不同的调整分数对 L2a 的输出重新评分，评估 L2b 的有效性。较低的分数调整可确保召回率损失较小，而较高的分数调整则支持激进剪枝，显著降低 AICT，但会牺牲部分召回率。这种方法可根据具体应用需求调整精确率与召回率之间的平衡。我们以 gobmk 基准测试为例进行说明（图 3）。我们选择 gobmk 是因为其 AICT 较高，改进效果更明显；为简洁起见，展示 O0 和 O1。相同趋势也适用于

表 2：间接调用预测的逐层比较。

二进制文件	Size	L1	L2a	L2b
bzip2-O0	0.2	2 / 100	1.6 / 100	**1 / 100**
bzip2-O1	0.2	2 / 100	**1.6 / 100**	**1.6 / 100**
bzip2-O2	0.2	2 / 100	**1.5 / 100**	**1.5 / 100**
bzip2-O3	0.3	2 / 100	**1.5 / 100**	**1.5 / 100**
sjeng-O0	0.3	7 / 100	**6 / 100**	**6 / 100**
sjeng-O1	0.4	7 / 100	**6 / 100**	**6 / 100**
sjeng-O2	0.5	7 / 100	**6 / 100**	**6 / 100**
sjeng-O3	0.6	7 / 100	7 / 100	7 / 100
milc-O0	0.4	2 / 100	2 / 100	2 / 100
milc-O1	0.5	2 / 100	2 / 100	2 / 100
milc-O2	0.5	2 / 100	2 / 100	2 / 100
milc-O3	0.6	1 / 100	1 / 100	1 / 100
sphinx3-O0	0.5	1.3 / 100	**0.8 / 100**	**0.8 / 100**
sphinx3-O1	0.7	1.3 / 100	**1 / 100**	**1 / 100**
sphinx3-O2	0.7	0.7 / 100	**0.6 / 100**	**0.6 / 100**
sphinx3-O3	0.8	0.7 / 100	**0.6 / 100**	**0.6 / 100**
hmmer-O0	0.7	2.9 / 100	**2.4 / 100**	**2.4 / 100**
hmmer-O1	1	4.3 / 100	**2.8 / 100**	**2.8 / 100**
hmmer-O2	1.1	2.8 / 100	**2.3 / 100**	**2.3 / 100**
hmmer-O3	1.3	1 / 100	1 / 100	1 / 100
h264ref-O0	1.8	5.7 / 100	3.6 / 100	**3.4 / 99.8**
h264ref-O1	2.1	5.2 / 99.7	4.2 / 99.7	**3.6 / 99.2**
h264ref-O2	2.1	26.7 / 100	6.3 / 100	**5.9 / 99.7**
h264ref-O3	3	18.5 / 100	14.7 / 100	**6.9 / 99**
gobmk-O0	5.2	884.6 / 100	715 / 99.1	**532.9 / 98**
gobmk-O1	5.9	1336.3 / 100	1033 / 99.1	**625.8 / 99**
gobmk-O2	6	1337.7 / 100	1196 / 99.1	**815.1 / 99**
gobmk-O3	6.4	1416.2 / 100	1184 / 99.1	**1001.5 / 98**
perlbench-O0	2.8	400.3 / 99.1	202.7 / 99.1	**182.1/ 99**
perlbench-O1	3.9	379.7 / 100	356.9 / 100	**170.1 / 96**
perlbench-O2	4.1	377.6 / 100	343.1 / 100	**192 / 95**
perlbench-O3	4.9	453.4 / 100	440.8 / 100	**291.8 / 98**
gcc-O0	8.7	534.8 / 99.3	486.3 / 99.3	**241.9 / 99**
gcc-O1	11.9	491 / 99.5	478.5 / 99.5	**357.8 / 98**
gcc-O2	12.1	427.7 / 99.5	397.7 / 99.5	**388.8 / 99**
gcc-O3	14.3	544.3 / 99.6	503.4 / 99.6	**474.9 / 99**
thttpd-O0	0.2	8 / 100	5 / 100	**3 / 100**
thttpd-O1	0.2	8 / 100	**5 / 100**	**5 / 100**
thttpd-O2	0.2	14 / 100	4 / 100	**3 / 100**
thttpd-O3	0.3	14 / 100	**1 / 100**	**1 / 100**
memcached-O0	0.4	1 / 92	**0.9 / 92**	**0.9 / 92**
memcached-O1	0.5	0.9 / 71	**0.7 / 71**	**0.7 / 71**
memcached-O2	0.6	1.4 / 70	0.9 / 70	**0.8 / 70**
memcached-O3	0.6	1.4 / 71	0.9 / 71	**0.8 / 71**
lighttpd-O0	0.8	35.5 / 100	9.7 / 100	**7.7 / 100**
lighttpd-O1	1.1	35.5 / 100	11.7 / 100	**11 / 100**
lighttpd-O2	1.2	34.8 / 100	15.4 / 100	**14.6 / 100**
lighttpd-O3	1.3	35.1 / 100	**15.3 / 100**	**15.3 / 100**
exim-O0	3.4	31.1 / 100	11.3 / 100	**7.8 / 99**
exim-O1	4.1	29.6 / 100	12.5 / 100	**10.6 / 100**
exim-O2	4.3	30.6 / 100	12.6 / 100	**10.5 / 100**
exim-O3	4.5	40.6 / 100	18.2 / 100	**12.7 / 100**
nginx-O0	4.4	444 / 100	306.1 / 100	**143.6 / 100**
nginx-O1	5.1	463.4 / 100	305.8 / 100	**160.4 / 99**
nginx-O2	5.2	527.3 / 100	359.1 / 99.9	**183.6 / 99**
nginx-O3	5.4	515.1 / 100	351.2 / 99.9	**176.9 / 99**
平均值	2.7	195.9 / 98.2	158.3 / 98.2	**109.2 / 97.8**

每个单元格报告 AICT / AICT_召回率。二进制大小以 MB 为单位。召回率相近时，较低的 AICT 表示精确率有所提高。

各项基准测试，尤其是在代码复杂度增加、假阳性增多的大型二进制文件中。图 3 展示了 L2b 的两种模式：保守模式（以 ±0.1 重新调整）和激进模式（±0.5）。随着召回率下降，两种模式之间的差距不断扩大。例如，在 gobmk-O0 中，召回率为 93.6% 时，激进剪枝比保守模式使 AICT 降低 6.6%；召回率为 90.8% 时，降低 59%；召回率为 85% 时，降低 76%，依此类推。这种效果在较低优化级别下最为显著，但在各种配置中都保持一致。总体而言，iScoreRefine 通过优化预测并对精确率-召回率权衡提供精细控制，增强了 iScoreGen。

6.3. RQ3：与现有解决方案的比较

学术工具：为评估我们的框架 iResolveX 的有效性，我们将其与三个具有代表性的学术基线进行比较：CALLEE [58]，AttnCall [39],和 BinDSA [14]。我们不纳入 BPA [20]，因为它构成了 iResolveX 自身的基础静态引擎（Layer1）。此外，我们也未纳入 SchedExec [37]，因为它排除了关键的基

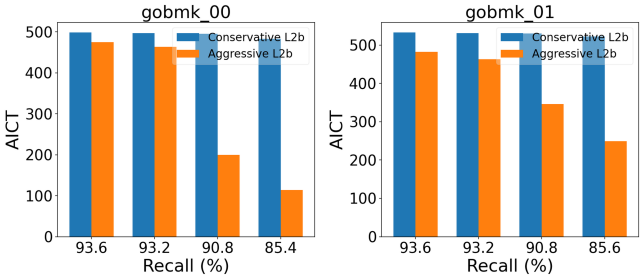


图3：不同召回率下 L2b剪枝强度对 AICT 的影响：激进方案与保守方案。

表3：iResolveX 在不同剪枝方案和评估目标下的配置。

Mode	方案	L2b Δ	p-IndirectCFG τ	关注点
iResolveX@C-R	保守	±0.1	轨迹引导（召回率调优）	面向完备性
iResolveX@C-F	保守	±0.1	轨迹引导（F1 调优）	平衡剪枝
iResolveX@A-R	激进	±0.5	轨迹引导（召回率调优）	更强剪枝
iResolveX@A-F	激进	±0.5	轨迹引导（F1 调优）	高置信度剪枝

Δ：L2b 分数调整，用于控制基于学习的细化程度（Δ 越大，剪枝越强）。 τ：pIndirectCFG 边纳入阈值；有动态跟踪时采用轨迹引导方式，否则采用静态方式——召回率保持模式和 F1 保持模式使用较低阈值。保守方案侧重完备性；激进方案则侧重保留高置信度边。

准测试项，例如 perlbench、gcc 和其他真实世界二进制文件，因而限制了其完备性和泛化能力。

评估参数与配置。 在 RQ1 中，我们主要使用 AICT 召回率来评估剪枝效果，同时保持完备性。为适应不同的分析目标，iResolveX 支持四种评估模式，它们在两个互补维度上有所不同：剪枝强度和保持目标。如表3所示，保守模式采用较温和的 L2b 细化以保持完备性，而激进模式则更大幅度地调整 L2a 的分数，以抑制过度近似的目标。此外，每种模式都可以优先考虑召回率保持——倾向于全面覆盖潜在目标；或优先考虑 F1 平衡——倾向于减少假阳性。请注意，我们省略了追求最佳精确率的情况，因为我们认为，在不保持合理召回率的前提下追求精确率毫无意义。通过相应选择 pIndirectCFG 的边纳入阈值，即可平衡这两个目标。这些配置共同展示了 iResolveX 如何适用于不同的分析场景：召回率保持模式适用于完备性至关重要的安全关键任务，而 F1 保持模式则适用于旨在获得精简、高置信度控制流边的分析。F1 按调用点计算（附录 A.1）。

公平的指标比较与可复现性障碍。评估指标——召回率和 F1 分数——依赖于动态轨迹覆盖率。为确保比较公平，我们采用与 AttnCall 和 BinDSA 相同的、由 SPEC 提供的基于输入的跟踪策略，使报告的指标具有可比性。然而，正如先前研究所指出的，由于动态目标仅覆盖真实标注的一部分，召回率往往会被高估，而精确率

则会被低估。

然而，CALLEE 的设置与其他工具不同，导致召回率比较更加复杂。尽管其发布了动态真实标注，但未提供二进制文件，且工件中缺少嵌入模型，因此无法复现。即使我们根据其论文尽最大努力重新实现 CALLEE，并使用自己的动态数据重新评估，测得的 AICT 召回率也只有 50%，全局召回率为 70%，远低于其宣称的 90.9% 全局召回率。此外，CALLEE 依赖 IDA 的 DataXRef，导致对取地址函数的估计不足——在 gcc 中仅检测到 414 个，而 iResolveX 平均恢复了 1,207 个[1]——因而漏掉了 417K 个函数对。重新运行其模糊测试设置 [17] 后，我们确认有 33.4% 的有效被调用函数被漏掉。perlbench 和 gobmk 也呈现出类似但程度较轻的情况（见附录 A.4 中的表9）。总体而言，这些问题凸显了 CALLEE 在可扩展性和完备性方面的局限。

此外，鉴于 CALLEE 使用了相同的 gcc-9 编译后的二进制文件，并与 BPA 进行了比较，其基于模糊测试的动态数据原则上应产生数量相当或更多的调用点–被调用者对。然而，它在每个基准测试中平均仅记录了 187 个调用点–被调用者对，而我们记录了 353 个。perlbench、milc、sphinx3 等基准测试以及真实世界二进制文件的动态真实值并未出现在其公开数据集中。这一 47% 的差距源于：（1）未处理命令行参数；（2）朴素模糊测试设置中的输入探索深度不足。假设召回率随跟踪规模线性增长，其真实召回率可能低至：

$$0.53\times R_{\text{obs}} \le R_{\text{true}} \le R_{\text{obs}} \quad \Rightarrow \quad 42.3\% \le R_{\text{true}} \le 90.9\%.$$

然而，我们承认，在所有工作中实现较高的动态覆盖率仍然是一项挑战，这凸显了程序感知型、基于语法的模糊测试的必要性。尽管如此，为确保公平性，我们遵循了与 AttnCall 和 BinDSA 相同的实验设置。此外，虽然我们将 CALLEE 纳入比较，但由于动态数据设置不同，其召回率和 F1 分数无法直接比较。由于 AICT 独立于动态跟踪，我们仅依据该指标评估 CALLEE。我们还忽略了其因 IDA DataXRef 导致的缺失调用点–被调用者对——将这些对纳入比较会进一步有利于我们的结果。AttnCall 和 BinDSA 同样存在可复现性问题：AttnCall 的代码缺少训练好的模型，而 BinDSA 尚未公开发布。因此，我们完全依赖 CALLEE、AttnCall 和 BinDSA 报告的数字。由于这些工作均未说明其评估二进制文件的优化级别，我们将 AICT 的基线与 BPA 对齐：CALLEE 和 BinDSA 使用 O2 二进制文件，AttnCall 使用 O0。

与最先进基线的比较。为清晰起见，我们仅使用 O0 和 O2 二进制文件将 iResolveX 与最先进基线进行比较，不过结果也适用于其他优化级别。AICT 比较（不依赖动态跟踪）见表4：比较 iResolveX 在不同配置下的 AICT 与最先进基线。

基准测试	C-R	C-F	A-R	A-F	CALLEE	AttnCall	BinDSA
bzip2-O0	1	0.25	1	1	–	1	–
bzip2-O2	1.55	0.25	2	1.20	1.4	–	1
sjeng-O0	6	6	6	6	–	6	–
sjeng-O2	6	6	7	7	7	–	7
milc-O0	2	2	2	2	–	3	–
milc-O2	2	2	2	2	2	–	1
sphinx3-O0	0.75	0.75	1.25	1.25	–	5.7	–
sphinx3-O2	0.57	0.57	0.57	0.57	5.6	–	–
hmmer-O0	2.40	1.44	2.78	1.44	–	6.6	–
hmmer-O2	2.30	2.30	1.70	1.70	7.2	–	3.3
h264ref-O0	3.41	1.10	2.40	0.03	–	21.5	–
h264ref-O2	5.90	0.02	1	0.06	20.9	–	2.04
gobmk-O0	532.90	170.18	485.48	130.52	–	422	–
gobmk-O2	815.07	21.91	564.43	151.89	672.4	–	86.1
perlbench-O0	182.10	1.06	166.53	2.78	–	313.2	–
perlbench-O2	192	16.36	148.50	39.15	354	–	20.8
gcc-O0	241.90	5.45	209.40	6.92	–	274	–
gcc-O2	388.81	0.73	113.98	5.90	338	–	111.2
thttpd-O0	3	3	3	3	–	7	–
thttpd-O2	3	3	3	3	–	–	–
memcached-O0	0.89	0.89	0.93	0.93	–	2.1	–
memcached-O2	0.79	0.60	0.71	0.71	11.3	–	1.2
lighttpd-O0	7.68	0.02	7.66	0.22	–	17.4	–
lighttpd-O2	14.61	0.01	7.20	0.20	31.7	–	17.6
exim-O0	7.81	0.19	6.17	0.45	–	15.1	–
exim-O2	10.51	0.08	8.78	0.28	22.4	–	5
nginx-O0	143.59	0.18	131.86	2.42	–	296.5	–
nginx-O2	183.60	0.62	170	4.09	383	–	8.7
平均-O0	81.10	13.80	73.30	11.40	–	99.36	–
平均-O2	116.20	3.90	73.60	15.60	142.84	–	22.08

C-R = iResolveX@C-R（保守召回率保持），C-F = iResolveX@C-F（保守 F1 值保持），A-R = iResolveX@A-R（激进召回率保持），A-F = iResolveX@A-F（激进 F1 值保持）。每个单元格报告 AICT。

表4，召回率/F1分数比较见表5。由于先前的工作仅针对 SPEC 收集动态跟踪数据，表5涵盖 SPEC。

如表4所示，iResolveX 在所有评估模式下均实现了稳定的 AICT 降低。在保守模式下，iResolveX@C-R 的降低幅度比 CALLEE 和 AttnCall 均高**18%**。在 F1 保持设置（iResolveX@C-F）下，其表现优于 CALLEE、BinDSA 和 AttnCall，幅度分别为 **97%、82% 和 86.2%**。在激进模式下，iResolveX@A-R 的表现优于 CALLEE 和 AttnCall，幅度分别为 **48.5% 和 26.2%**；而 iResolveX@A-F 的表现优于 CALLEE、BinDSA 和 AttnCall，幅度分别为**89.2%、30% 和 88.6%**。值得注意的是，在召回率保持模式下，召回率仍然很高：iResolveX@C-R 为 98.1%，iResolveX@A-R 为 96.4%。即使在 F1 保持模式下，iResolveX 也保持了较高的召回率——iResolveX@C-F 为 82.6%，iResolveX@A-F 为 88.8%——同时平均维持 80% 的 F1 分数和精确率。[2]

尽管 BinDSA 在召回率保持配置下报告了更低的 AICT，但由于数据结构恢复不够精确，其完整性存在严重不足。即使对于 sphinx3 这样的小型二进制文件，BinDSA 也未能解析出任何目标，尽管存在动态证据；而对于较大的

1. 尽管 CALLEE 报告的取地址计数略高，但其结果仍然是欠近似的；我们在 x86/x64 构建版本上的测试证实，IDA DataXRef 并不可靠。由于 CALLEE 的二进制文件并未发布，我们无法验证其说法。

2. 召回率/F1分数/精确率是根据完整评估日志计算的平均值；表4未列出这些指标，表5则展示了部分 SPEC 的召回率。

表5：iResolveX 在不同配置下与 SPEC2006基准测试上的最先进基线相比的召回率 / F1分数。

基准测试	C-R（召回率）	C-F (F1)	A-R（召回率）	A-F (F1)	CALLEE	AttnCall（召回率）	BinDSA (F1)
bzip2-O0	100	79	100	75	–	100	–
bzip2-O2	100	79	100	72	–	–	100
sjeng-O0	100	100	100	100	–	100	–
sjeng-O2	100	93	100	93	–	–	92.5
milc-O0	100	100	100	100	–	100	–
milc-O2	100	100	100	100	–	–	100
sphinx3-O0	100	91	100	88.9	–	100	–
sphinx3-O2	100	94.3	100	94.3	–	–	NA
hmmer-O0	100	89.8	100	89.8	–	100	–
hmmer-O2	100	95.1	100	95.4	–	–	95.3
h264ref-O0	99.8	89.8	99.3	87.8	–	92.3	–
h264ref-O2	99.7	98.9	99	97.1	–	–	78.8
gobmk-O0	98	55.7	96	64.7	–	96.9	–
gobmk-O2	99	42	86	53.4	–	–	70.5
perlbench-O0	99	55.6	93	65.3	–	78.4	–
perlbench-O2	95	56.3	85	57	–	–	65.6
gcc-O0	99	70.8	98.5	70.2	–	93.2	–
gcc-O2	99	70.7	90	68.6	–	–	54.5
平均-O0	99.5	81.3	98.5	82.4	–	95.6	–
平均-O2	99.2	81	95.6	81.2	–	–	82.1

C-R = iResolveX@C-R（保守召回率保持），C-F = iResolveX@C- F（保守 F1 值保持），A-R = iResolveX@A-R（激进召回率保持），A-F = iResolveX@A-F（激进 F1 值保持）。对于保持召回率的配置，每个单元格报告 AICT_召回率；对于保持 F1 的配置，每个单元格报告 AICT_F1分数。CALLEE 未报告基准测试特定数值；对于 AttnCall，每个单元格报告 AICT_召回率；对于 BinDSA，每个单元格报告 AICT_F1分数。

表6：逆向工程工具比较

二进制文件	IDA	GHIDRA	Angr	
			CFGEmulated	CFGFast
bzip2	0 / 0	0.9 / 25	0.1 / 5	0 / 0
sjeng	7 / 100	0 / 0	1 / 0	0 / 0
milc	0 / 0	0 / 0	1 / 100	0 / 0
sphinx3	0 / 0	0 / 0	0 / 0	0 / 0
HMMER	0 / 0	0 / 0	0.9 / 87.4	0 / 0
h264ref	0 / 0	2.1 / 8.3	0.9 / 1.2	0 / 0
gobmk	0 / 0	1 / 26.6	0.5 / 27.8	0 / 0
perlbench	7.4 / 32.1	0.9 / 30.1	14.1 / 29.3	0.3 / 26
gcc	1.7 / 11.9	2.5 / 16.4	超时	0.2 / 11.8
平均值	1.8 / 16	0.8 / 11.8	**1.9 / 31.4**	0.1 / 4.2

每个单元格报告 AICT / AICT_召回率

程序（perlbench、gcc、nginx、lighttpd），它遗漏了 18–37 % 的有效调用点。相比之下，iResolveX 提供了可配置的精度–覆盖率权衡，既支持高覆盖率场景，也支持高精度场景。如表5所示，在 SPEC2006 上，iResolveX 在召回率保持配置下实现了更高的召回率和可比的 F1分数，同时仍大幅降低了 AICT。

逆向工程工具： 我们观察到，主流逆向工程工具与学术方法在解析间接调用方面存在显著差距。平均而言，IDA 和 GHIDRA 的 AICT（1.8 和 0.8）及召回率（16% 和 11.8%）都较低，且 GHIDRA 还会引入假阳性。Angr 的 CFGEmulated 召回率较高（31.4%），但速度仍慢到难以实际应用，在大型二进制文件上会超时（表6）。这些结果表明，我们需要一种灵活的解决方案来连接学术方法与逆向工程工具。p-IndirectCFG 弥补了这一差距。

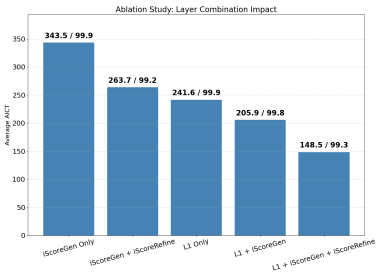


图4：各柱标注的平均 AICT 和 AICT 召回率的消融研究。_

6.4. RQ4：层组合消融研究

为评估 iResolveX 各组件的贡献，我们针对其三个层（L1（BPA）、L2a（iScoreGen）和 L2b（iScoreRefine））的所有有意义排列进行了消融研究。能够分别考察静态分析和机器学习影响的五种有效组合如下（图4）：(1) 仅 L1（BPA），(2) L1+iScoreGen（L1+L2a），(3) 完整流水线 L1+iScoreGen+iScoreRefine（L1+L2a+L2b），(4) 仅 iScoreGen（L2a），以及 (5) iScoreGen+iScoreRefine（L2a+L2b）。请注意，L1+L2b 无效，因为 L2b 依赖 L2a 的输出。在这项研究中，我们报告保守、保持召回率的配置（iResolveX@C-R）下的结果；其他配置也呈现出类似结果。结果表明，L1 的召回率较高，但存在过度近似的问题。加入 iScoreGen（L2a）可显著提高精确率，而 iScoreRefine（L2b）则通过静态过程间验证进一步减少假阳性。相比之下，仅使用机器学习的配置（L2a 和 L2a+L2b）由于缺少保守的基础，会扩大目标集。总体而言，结合所有层的效果最佳：L1 奠定可靠性基础，L2a 提高精确率，L2b 则实施上下文剪枝。

6.5. RQ5：p-IndirectCFG 阈值及其影响

我们通过对随机选取的基准测试进行敏感性分析，研究 p-IndirectCFG 中阈值的选择如何影响评估指标。这些基准测试在优化程度和二进制文件大小方面各不相同，涵盖了最小的二进制文件（bzip2）、最大的二进制文件（gcc）以及真实场景中的 O3 二进制文件（图5）。本研究报告的是保守且保持召回率的配置（iResolveX@C-R）下的结果；其他配置也呈现出类似结果。总体趋势十分明显：提高阈值会持续降低 AICT 和召回率，同时提升 F1分数。尽管各基准测试中的变化速率有所不同，这一趋势在评估集中的其他基准测试上也始终可见。值得注意的是，所有基准测试的 AICT 都会先急剧下降；对于 bzip2 这样较小的二进制文件，F1分数会立即提升，而对于 gcc 这样较大的二进制文件，则要等到剪除更多假阳性后才有所改善。可调节的阈值使 p-IndirectCFG 能够适应不同的分析需求：较低的阈值可以保留对召回率至关重要的边，而较高的阈值则倾向于通过剪枝来提高精确率。

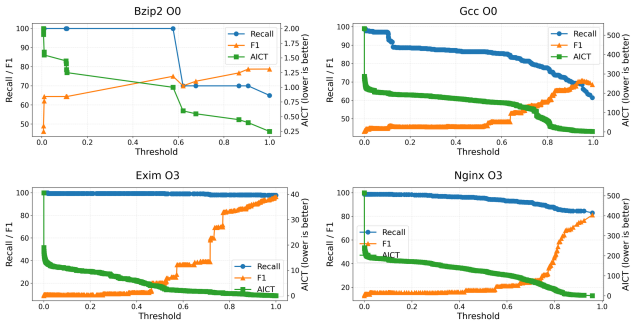


图5：阈值敏感性分析，展示召回率、F1分数和 AICT 随阈值的变化。左轴：AICT_召回率或 AICT_F1分数；右轴：AICT。

如案例研究（附录 A.5）所示，这种适应性有助于保留对恶意载荷可达性至关重要的别名驱动的边，同时较高的阈值会消除 BPA 引发的自环和结构上无效的路径。

7. 讨论与局限性

L2a 微调与设计选择： L2a 学习一种函数签名引导的特征空间，可将从直接调用中学到的知识泛化到间接调用，从而减少对大规模动态跟踪的依赖——我们的评估验证了这一点。为评估微调的益处，我们使用额外的 1 万个动态提取的间接调用点–被调用者对对 L2a 模型进行了微调。结果显示，AICT 降低了 1.5%，而召回率没有损失，表明微调有助于捕捉罕见模式。然而，在微调后的 L2a 之上应用 L2b 时，收益降至 0.8%，因为反向推理已涵盖了这项益处的大部分。尽管额外的动态数据可能带来增量收益，但我们预计，加入 L2b 后收益会逐渐递减。此外，用表达能力更强的模型（如 Transformer）替换 L2a 的深度神经网络，可能会改善剪枝，这将留待未来工作。L2a 的算法 2 还纳入了根据我们的观察总结出的编译器特定的 x86 启发式规则，以改善 Angr 不可靠的调用约定分析。由于 Angr 本就依赖启发式规则，我们对其进行了增强以提高稳定性；不过，更强大的静态分析前端或许可以免去这一需求。

实用性与可扩展性： 我们评估了各层的运行时间——L1（静态分析）、L2a（特征提取与 +推理）、L2b（反向推理）和动态跟踪。开销最大的组件是 L1 和动态跟踪，但它们可并行执行，因此端到端开销有限。与 Angr 高召回率的 CFGEmulated 相比，iResolveX 在小型二进制文件上的速度最高可提升 74%（召回率为 100% 时，平均耗时 <100s）。对于 gcc 这类大型二进制文件，Angr 会超时（>24 h），而 iResolveX 可在 10 小时内完成，召回率达到 98.8%——这是我们测试套件中耗时最长的一项。尽管速度不如纯机器学习方法，iResolveX 仍提供了它们所不具备的能力，包括静态过程间推理，以及可通过阈值灵活控制、适用于对召回率与精确率敏感的剪枝策略。灵活控制召回率与精确率，适用于对二者敏感的剪枝策略。

指令集架构和语言支持： 受 BPA 引擎的限制，iResolveX 目前面向基于 C 的 x86-32 二进制文件，而 BPA 引擎仅支持这种配置。虽然 L1 依赖 BPA 并继承了这一限制，但从理论上说，它与架构和语言无关。相比之下，层 2a 和 2b 与 x86-32 和 C 均解耦，可在任何由静态分析驱动的控制流图上运行。如我们的消融实验（图4）所示，iResolveX 无需 L1 也能运行：将取地址函数视为可能的目标并应用 L2 剪枝，可使 AICT 降低 23.2%，接近使用 L1+L2 时的 38.5% 降低。要支持 x86-64，可使用 TypeArmor [46]，这类可靠的替代方案来调整 L1，它能提供类似的过度近似。值得注意的是，iResolveX 的 L2 不要求 L1 可靠；也就是说，L2 可独立处理来自任何工具的控制流图，包括 CALLEE 和 AttnCall，并对间接边进行事后排序或剪枝。因此，iResolveX 可广泛应用于各种工具链。相比之下，现有机器学习工具与 x86-64 紧密耦合，因为它们依赖特定于调用约定寄存器的切片逻辑，因此要支持 x86-32 并非易事。未来一个重要方向是通过可插拔前端扩展对 x86-64 和 ARM 的支持。

iScoreRefine 中的分数调整（L2b）： L2b 目前使用静态分数来调整 L2a 的输出，从而适应不同的分析目标。虽然这种方法有效，但未考虑程序特有的因素，例如间接调用点密度、被调用者歧义或控制流复杂度。为提高适应性，我们计划用异构图神经网络（GNN）[23], [49] 取代阈值处理，利用过程间结构和语义上下文学习如何调整分数。该 GNN 基于由调用点、候选被调用者、内存位置和指令构成的多关系图运行，图中的边用于捕获控制流和数据流。节点则通过子图进行增强，以捕获实参流、解引用和跨函数控制模式。我们使用 iScoreGen 的兼容性分数作为弱监督，以实现无监督细化。在部分基准测试上的初步结果表明，这种基于学习的细化能够进一步降低 AICT，且不会损失召回率，验证了其潜力。我们将完整集成和评估留待后续工作。

逐基准测试阈值选择中的动态跟踪稀疏性： 不同程序的间接调用行为差异很大，因此难以构建通用的动态剖析。p-IndirectCFG 则使用可选的部分跟踪记录（占总量的 30%），避免依赖穷尽式性能剖析。作为未来工作，我们计划探索轻量级强化学习 [28], [40]，以实现自动剪枝——仅在训练期间使用强制执行跟踪记录，并在测试时采用静态的策略驱动阈值；强化学习能够自然地将精确率—召回率权衡建模为奖励驱动的优化问题。

8. 结论

我们提出了 iResolveX，这是一种通过结合值集分析来重新解析间接调用目标的混合框架，

基于机器学习的签名评分以及选择性过程间推理。该设计在可靠性与精确率之间取得平衡，无需动态监督即可区分可泛化的签名与特定于上下文的行为。我们的 p-IndirectCFG 进一步弥合了学术研究与逆向工程工具之间的鸿沟。实证结果表明，iResolveX 在保持高召回率的同时降低了 AICT，性能优于当前最先进的方法。作为一种灵活且模块化的框架，iResolveX 能够很好地适用于各种需要可控精确率-召回率权衡的下游任务，填补了当前文献中的关键空白。

9. 伦理考量

本研究仅提升自行编译的 SPEC 和开源二进制文件的间接调用解析能力，不涉及个人数据或人类受试者；更准确的控制流图以及更便于实施的加固措施，将使开源维护者、安全分析师和终端用户受益。尽管逆向工程具有双重用途，我们仅发布附有指导说明、供授权用途使用的制品，并且只记录控制流元数据，因此我们预计防御方面的收益将超过潜在的滥用风险。

参考文献

[1] Mart´ın Abadi、Mihai Budiu、 ´Ulfar Erlingsson 和 Jay Ligatti。控制流完整性。载于第*12*届*ACM*计算机与通信安全会议论文集，CCS '05，第340–353页，美国纽约州纽约市，2005年。美国计算机协会。[2] Mart´ın Abadi、Mihai Budiu、 ´Ulfar Erlingsson 和 Jay Ligatti。控制流完整性原则、实现与应用。*ACM*信息与系统安全汇刊，13(1)，2009年11月。[3] Mart´ın Abadi 等。Tensorflow。https://www.tensorflow.org，2015年。[4] Lars Ole Andersen 和 Peter Lee。C编程语言的程序分析与特化。2005年。[5] Gogul Balakrishnan 和 Thomas Reps。分析 x86可执行文件中的内存访问。载于 Evelyn Duesterwald 编，编译器构造，第5–23页，柏林、海德堡，2004年。Springer Berlin Heidelberg。[6] Markus Bauer、Ilya Grishchenko 和 Christian Rossow。Typro：通过类型传播为 C 风格的间接函数调用实现前向控制流完整性。载于第*38*届年度计算机安全应用会议论文集，第346–360页，2022年。[7] Jane Bro mley、Isabelle Guyon、Yann LeCun、Eduard S¨ackinger 和 Roopak Shah。使用"孪生"时延神经网络进行签名验证。载于 J. Cowan、G. Tesauro 和 J. Alspector 编，神经信息处理系统进展，第6卷。Morgan-Kaufmann，1993年。[8] Nicholas Carlini、Antonio Barresi、Mathias Payer、David Wagner 和 Thomas R. Gross。控制流弯曲：控制流完整性的有效性。载于第*24*届*USENIX*安全研讨会（*USENIX Security 15*），第161–176页，华盛顿特区，2015年8月。USENIX协会。[9] Sanchuan Chen、Zhiqiang Lin 和 Yinqian Zhang。SelectiveTaint：通过静态二进制重写实现高效的数据流跟踪。载于第*30*届*USENIX*安全研讨会（*USENIX Security 21*），第1665–1682页。USENIX协会，2021年8月。[10] Franc.ois Chollet 等。Keras。https://keras.io，2015年。[11] Tommaso Frassetto、Patrick Jauernig、David Koisser 和 Ahmad-Reza Sadeghi。CFInsight：衡量控制流完整性策略的综合指标。载于 *NDSS*，2022年。

[12] 自由软件基金会。GNU binutils。https://www.gnu.org/software/binutils/，2023年。访问日期：2025-10-03。[13] 自由软件基金会。GNU coreutils：GNU 核心工具。https: // www.gnu.org/software/coreutils/，2023年。访问日期：2025-10-03。[14] Lian Gao 和 Heng Yin。BinDSA：采用上下文敏感的堆重建实现高效、精确的二进制级指针分析。*ACM* 软件工程论文集，2(ISSTA)，2025年6月。[15] M.R. Guthaus、J.S. Ringenberg、D. Ernst、T.M. Austin、T. Mudge 和 R.B. Brown。MiBench：一个免费的、具有商业代表性的嵌入式基准测试套件。载于第四届 *IEEE* 国际工作负载特征分析研讨会论文集，*WWC-4*（目录号 *01EX538*），第 3–14 页，2001年。[16] Hex-Rays。IDA Pro 反汇编器和调试器。https://hex-rays. com/ida-pro/。7.5 版，访问日期：2025年7月29日。[17] Immunant, Inc. ibresolver：用于解析间接分支的 QEMU TCG 插件。https://github.com/immunant/ibresolver，2025年。访问日期：2025年7月29日。[18] 英特尔公司。*Intel 64* 和 *IA-32* 架构软件开发人员手册，第 *1–3* 卷，2021年。https:// www.intel.com/content/ www/us/en/developer/articles/technical/intel-sdm.html。[19] Herbert Jordan、Bernhard Scholz 和 Pavle Suboti´c。Souffl´e：论程序分析器的合成。载于国际计算机辅助验证会议，2016年。[20] Sun Hyoung Kim、Cong Sun、Dongrui Zeng 和 Gang Tan。在二进制层细化间接调用目标。载于网络与分布式系统安全研讨会（*NDSS*），2021年。[21] Sun Hyoung Kim、Dongrui Zeng、Cong Sun 和 Gang Tan。Bin-pointer：迈向精确、可靠且可扩展的二进制级指针分析。载于第 *31* 届 *ACM SIGPLAN* 国际编译器构造会议论文集，第 169–180 页，2022年。[22] James C. King。符号执行与程序测试。*ACM* 通讯，19(7)：385–394，1976年7月。[23] Thomas N. Kipf 和 Max Welling。基于图卷积网络的半监督分类。载于第 *5* 届国际学习表征会议（*ICLR 2017*），法国土伦，*2017*年*4*月*24–26*日，会议论文集。OpenReview.net，2017年。[24] Chris Lattner、Andrew Lenharth 和 Vikram Adve。让上下文敏感的指向分析通过堆克隆在真实世界中切实可行。载于*2007*年 *ACM SIGPLAN* 编程语言设计与实现会议论文集（*PLDI'07*），美国加利福尼亚州圣迭戈，2007年6月。[25] Chris Lattner、Andrew Lenharth 和 Vikram Adve。让上下文敏感的指向分析通过堆克隆在真实世界中切实可行。*SIGPLAN* 通知，42(6)：278–289，2007年6月。[26] Gwangmu Lee、Woochul Shim 和 Byoungyoung Lee。约束引导的定向灰盒模糊测试。载于第 *30* 届 *USENIX* 安全研讨会（*USENIX Security 21*），第 3559–3576 页。USENIX协会，2021年8月。[27] Yuan Li、Mingzhe Wang、Chao Zhang、Xingman Chen、Songtao Yang 和 Ying Liu。寻找盾牌上的裂缝：控制流完整性机制的安全性。载于 *2020* 年 *ACM SIGSAC* 计算机与通信安全会议论文集，CCS '20，第 1821–1835 页，美国纽约州纽约，2020年。美国计算机协会。[28] Yuxi Li。深度强化学习：综述，2018年。[29] Yan Lin 和 Debin Gao。当函数签名恢复遇上编译器优化。载于*2021* 年 *IEEE* 安全与隐私研讨会（*SP*），第 36–52 页，2021年。[30] Ziyi Lin、Jinku Li、Bowen Li、Haoyu Ma、Debin Gao 和 Jianfeng Ma。TypeSqueezer：当二进制可执行文件的函数签名静态恢复遇上动态分析。载于*2023* 年 *ACM SIGSAC* 计算机与通信安全会议论文集，第 2725–2739 页，2023年。

[31] Peipei Liu、孙健、陈力、严兆腾、张培正、孙大鹏、王大为和李丹。基于大语言模型的控制流增强型反编译器，2025年3月。[32] Peipei Liu、孙健、孙荣康、陈力、严兆腾、张培正、孙大鹏、王大为、张晓玲和李丹。Codeinverter 套件：借助大语言模型增强控制流与数据映射的二进制反编译，2025年。[33] 陆志强、Robert Cohn、Robert Muth、Harish Patil、Artur Klauser、Geoff Lowney、Steven Wallace、Vijay Janapa Reddi 和 Kim Hazelwood。PIN：通过动态插桩构建定制化程序分析工具。载于*2005*年 *ACM SIGPLAN* 编程语言设计与实现会议论文集，PLDI '05，第 190–200 页，美国纽约州纽约，2005年。美国计算机协会。[34] 马建强、Khoo Yit Phang、Jeffrey S. Foster 和 Michael Hicks。定向符号执行。载于 E. Yahav（编），静态分析，第 95–111 页，柏林、海德堡，2011年。Springer Berlin Heidelberg。

[35] 美国国家安全局。Ghidra 软件逆向工程框架。https://ghidra-sre.org/。访问日期：2025年7月29日。

[36] Weizhong Qiang、Yingda Huang、邹德清、金海、Shizhen Wang 和孙国忠。面向商用现成软件二进制文件的完全上下文敏感 CFI。第 435–442 页，2017年5月。[37] 石洋洋、田丽娜、陈立伟、杨燕琪和石刚。基于调度执行的二进制间接调用目标细化。载于 Joaquin Garcia-Alfaro、Rafał Kozik、Michał Chora´s 和 Sokratis Katsikas 编，《style id='4'>计算机安全——ESORICS 2024》，第 3–23 页，尚，2024年。Springer Nature Switzerland。[38] Yan Shoshitaishvili、王若愚、Christopher Salls、Nick Stephens、Mario Grosen、Sophia Dutcher、John Feng、Christophe Hauser、Christopher Kruegel 和 Giovanni Vigna。SoK：angr 框架中符号执行的现状与前沿。载于《style id='7'>IEEE 安全与隐私研讨会 (S&P)》，第 89–104 页。IEEE，2016年。[39] 孙睿、郭英刚、王子成和曾庆凯。At-tncall：利用注意力机制细化二进制文件中的间接调用目标。载于 Gene Tsudik、Mauro Conti、Kaitai Liang 和 Georgios Smaragdakis 编，《style id='10'>计算机安全——ESORICS 2023》，第 391–409 页，尚，2024年。Springer Nature Switzerland。[40] Richard S. Sutton 和 Andrew G. Barto。《style id='13'>强化学习：导论》。麻省理工学院出版社，第 2 版，2018年。[41] Sami Ullah、金文辉和 Heekuck Oh。用于多架构二进制可执行文件函数匹配的高效特征。《style id='16'>IEEE Access》，9:104950–104968，2021年。[42] UNIX 系统实验室。System V 应用程序二进制接口：Intel386 架构处理器补充规范，1997年。https://refspecs. linuxfoundation.org/elf/abi386-4.pdf。[43] UNIX 系统实验室。System V 应用程序二进制接口：AMD64 架构处理器补充规范，2013年。https://refspecs. linuxfoundation.org/elf/x86 64-abi-0.99.pdf。[44] Victor van der Veen、Dennis Andriesse、Enes G¨oktas¸、Ben Gras、Lionel Sambuc、Asia Slowinska、Herbert Bos 和 Cristiano Giuffrida。实用的上下文敏感 CFI。载于《style id='23'>第 22 届 ACM SIGSAC 计算机与通信安全会议论文集》，CCS '15，第 927–940 页，美国纽约州纽约，2015年。美国计算机协会。[45] Victor van der Veen、Enes Goktas、Moritz Contag、Andre Pawlowski、陈曦、Sanjay Rawat、Herbert Bos、Thorsten Holz、Elias Athana-sopoulos 和 Cristiano Giuffrida。棘手的调用：在二进制层缓解高级代码复用攻击。载于《style id='26'>S&P》，2016年5月。[46] Victor van der Veen、Enes G¨oktas¸、Moritz Contag、Andre Pawloski、陈曦、Sanjay Rawat、Herbert Bos、Thorsten Holz、Elias Athana-sopoulos 和 Cristiano Giuffrida。棘手的调用：在二进制层缓解高级代码复用攻击。2016年5月。

[47] THC van Hauser。AFL-Pin：通过 PIN 插桩进行覆盖率引导模糊测试。https://github.com/vanhauser-thc/afl-pin。访问日期：2025-08-18。[48] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Łukasz Kaiser 和 Illia Polosukhin。注意力机制就是你所需的一切。载于《style id='3'>第 31 届神经信息处理系统国际会议论文集》，NIPS'17，第 6000–6010 页，美国纽约州雷德胡克，2017年。Curran Associates Inc。[49] Petar Veli˘ckovi´c、Guillem Cucurull、Arantxa Casanova、Adriana Romero、Pietro Li`o 和 Yoshua Bengio。图注意力网络。载于《style id='8'>国际学习表征会议》，2018年。[50] Ya¨elle Vinc¸ont、S´ebastien Bardin 和 Micha¨el Marcozzi。符号执行与模糊测试的紧密集成（短文）。载于 Esma A¨ımeur、Maryline Laurent、Reda Yaich、Beno ˆıt Dupont 和 Joaquin Garcia-Alfaro 编，《style id='16'>安全基础与实践》，第 303–310 页，尚，2022年。Springer International Publishing。[51] Peicheng Wang、Monika Santra、Mingyu Liu、Cong Sun、Dongrui Zeng 和 Gang Tan。DISA：利用注意力机制实现准确的基于学习的静态反汇编。载于《style id='19'>第 31 届 ACM 计算机与通信安全会议（CCS）》，2025年。[52] Qianjin Wang、Xiangdong Li、Chong Yue 和 Yuchen He。二进制代码控制流图恢复综述。载于 Min Zhang、Bin Xu、Fuyuan Hu、Junyu Lin、Xianhua Song 和 Zeguang Lu 编，《style id='22'>计算机应用》，第 225–244 页，新加坡，2024年。Springer Nature Singapore。[53] Tianrou Xia、Hong Hu 和 Dinghao Wu。Deeptype：利用强大的多层类型分析细化间接调用目标。载于《style id='25'>USENIX安全研讨会》，2024年。[54] Xiaojun Xu、Chang Liu、Qian Feng、Heng Yin、Le Song 和 Dawn Song。用于跨平台二进制代码相似性检测的基于神经网络的图嵌入。载于《style id='28'>2017年 ACM SIGSAC 计算机与通信安全会议论文集》，CCS '17，第 363–376 页。ACM，2017年10月。[55] Khaled Yakdan、Sebastian Eschweiler、Elmar Gerhards-Padilla 和 Matthew Smith。不再使用 goto：采用模式无关的控制流结构化和保持语义的转换进行反编译。2015年2月。[56] Lei Yu、Karl Moritz Hermann、Phil Blunsom 和 Stephen Pulman。用于答案句选择的深度学习，2014年。[57] Zeping Yu、Rui Cao、Qiyi Tang、Sen Nie、Junzhou Huang 和 Shi Wu。顺序很重要：用于二进制代码相似性检测的语义感知神经网络。《style id='33'>AAAI人工智能会议论文集》，34(01):1145–1152，2020年4月。[58] Wenyu Zhu、Zhiyao Feng、Zihan Zhang、Jianjun Chen、Zhijian Ou、Min Yang 和 Chao Zhang。CALLEE：通过迁移学习和对比学习恢复二进制文件的调用图，2022年。

附录 A. iResolveX 组件的补充细节

A.1. 召回率和 F1分数的定义

为保持一致，全局变量和 AICT 指标均针对 n 间接调用点定义，其中 TP_i、 FP_i和 FN_i分别表示调用点 i处的真正例、假阳性和假阴性。

$$\text{Global\_Precision} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FP_i)} \qquad \text{AICT\_Precision} = \frac{1}{n}\sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i}$$

$$\text{Global\_Recall} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)} \qquad \text{AICT\_Recall} = \frac{1}{n}\sum_{i=1}^{n} \frac{TP_i}{TP_i + FN_i}$$

$$\text{Global\_F1} = \frac{2 \cdot \text{Global\_Precision} \cdot \text{Global\_Recall}}{\text{Global\_Precision} + \text{Global\_Recall}} \qquad \text{AICT\_F1} = \frac{2 \cdot \text{AICT\_Precision} \cdot \text{AICT\_Recall}}{\text{AICT\_Precision} + \text{AICT\_Recall}}$$

全局指标汇总所有调用点的计数，而 AICT 指标计算逐调用点平均值，以反映单个调用点层面的分析质量。

表7：参数和返回值检测的启发式权重。

启发式	权重
正栈偏移	2.0
从未写入	1.5
早期使用（前3条指令内）	2.0
在多个基本块中使用	1.0
高访问频率	1.0
偏移量为正的内存操作数	2.0
返回值 (EAX) 权重	
写入 EAX	1.0
在最后一个基本块中写入	1.5
赋予一个常量值	1.0

表8：神经网络架构和训练参数。

层	配置
输入	标准化调用点–被调用者向量
隐藏层 1	1024 个神经元，LeakyReLU，Dropout（0.4）
隐藏层 2	512 个神经元，LeakyReLU，Dropout（0.4）
隐藏层 3	256 个神经元，LeakyReLU，Dropout（0.3）
隐藏层 4	128 个神经元，LeakyReLU，Dropout（0.2）
输出层	1 个神经元，Sigmoid 激活函数
优化器	Adam（lr = 0.001）
Loss	二元交叉熵
提前停止	耐心值 = 8 个周期
学习率衰减	耐心值 = 4 个周期
批量大小	256

A.2. iScoreGen中的参数和返回值检测启发式方法

表7总结了 iS-coreGen 筛选变量所使用的启发式权重。当变量的累积分数超过 6.0 时，该变量会被保留为可能的参数。对于通过 EAX 进行的返回值检测，我们采用类似规则，阈值为 2.0。

A.3. iScoreGen模型详情

iScoreGen使用的深度神经网络是一个全连接前馈模型，其架构如表8所示。训练前，我们使用StandardScaler对输入向量进行标准化。

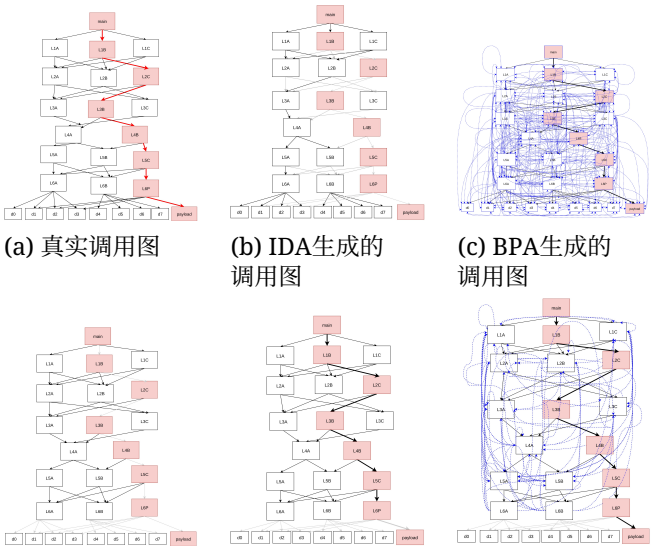
A.4. RQ3：额外的CALLEE评估——取地址函数检测比较

表9比较了 iResolveX 与 CALLEE 在检测取地址函数方面的表现。对于我们编译的二进制文件，iResolveX 在所有基准测试中始终能恢复更多函数，而 CALLEE 会漏掉相当一部分真实被调用函数，尤其是在 gcc 这类大型二进制文件中。

表9：取地址函数检测比较。

二进制文件	iResolveX	CALLEE	漏检的真实目标函数	遗漏率
gobmk	1787	1757	29	8
perlbench	720	689	9	3
gcc	1207	414	138	33.4

漏检的真实目标函数：CALLEE 漏掉的真实目标函数。遗漏率按全部真实目标函数计算。



(a) 真实调用图 (b) IDA生成的调用图 (c) BPA生成的调用图

(d) p-间接控制流图 at 阈值>=0.7 (e) p-间接控制流图 at 阈值>=0.6 (f) p-IndirectCFG 位于 阈值>=0.57

图6：不同工具生成的概念验证二进制调用图。（黑色：工具检测到的边；蓝色虚线：假阳性；灰色：真实标注的边）。除 TrueCall 图外，所有图均使用此颜色标注。

A.5. 案例研究——为什么 p-IndirectCFG 对真实世界逆向工程至关重要

间接调用解析的理想结果是获得完整且精确的控制流图，但无论是学术界还是商业领域，现有工具都无法兼顾这两点。逆向工程师只能在多个工具（IDA、Ghidra 和学术界的原型工具）生成的不完整控制流图之间周旋，或转而采用动态跟踪。像 BPA 这样可靠的工具可以确保召回率，却会产生大量假阳性，使分析人员不堪重负，从而影响人工检查和自动推理。p-IndirectCFG 通过提供灵活的置信度标注控制流图，弥补了这一不足，支持对精确率和召回率要求各异的任务。调整剪枝阈值即可在假阳性与覆盖率之间取得平衡。我们通过两个截然不同的场景展示了这一点：

召回率敏感场景：我们的概念验证二进制文件模拟了真实恶意软件：它们将有效载荷隐藏在多层指针别名、内存—代码依赖和过程间间接调用之后。232条路径中，只有一条能够到达有效载荷。该概念验证同时采用基于索引的分派和由别名驱动的函数指针解引用，以模拟真实的混淆手法（见代码清单3）。只有路径 main → L1B → L2C→ L3B → L4B → L5C → L6P → 有效载荷能够到达隐藏函数（图6a）。Ghidra 完全无法解析；IDA 虽能部分解析调用，却遗漏了有效载荷（图6b）。BPA 达到了 100% 的召回率，但 AICT 为 16.9——乍看之下，每个调用点平均有 16.9 个目标，绝对数量似乎不大；但对于大型程序，这很快就会累积成数百甚至数千条额外的边（图6c）。这些伪边会导致结构失真，例如有效载荷

代码清单3：召回敏感型案例研究的概念验证

```
1 /*2     * 只有 main->L1B->L2C->L3B->L4B->L5C->L6P->有效载荷使用
指
数
t)(
P4
相
L1
荷
d3
v(
(i
21
pi
谓
模
谓
择
L3
L5
主
(a
T4
进
达
```

看起来可通过虚假路径到达，而符号执行等静态分析会被不可实现路径压垮。

相比之下，p-IndirectCFG 能逐步优化召回率与精确率之间的权衡：在较高阈值下（图6d），仅保留高置信度边——没有误报，但路径不完整。降低阈值后（图6e），在没有误报的情况下恢复了恶意链的大部分内容，表现优于 IDA/Ghidra。进一步放宽阈值（图6f），即可捕获完整的载荷路径，同时相比 BPA 减少噪声。这种灵活性支持基于边置信度的自动遍历策略，适用于人工或基于智能体的恶意软件分析。这说明了安全关键型逆向工程中召回率敏感性的必要性。我们还测试了 CALLEE。如第6.3节所述，我们无法复现其结果。我们重新设计的 CALLEE 控制流图比 BPA 的更不精确，这可能是由于在存在参数相似性和深度内存别名的情况下，切片能力有限。由此产生的过度近似使其不适用于逆向工程工作流程。

精确率敏感场景：假阳性增加会严重降低下游分析的效果。单条虚假的过程间边就可能合并区域、产生不存在的循环，并增加不动点迭代次数。符号执行会遭遇路径爆炸和求解器超时；支配树和 SSA形式会出现人工合流点，支配关系和结构也会失真，变量/类型会退化为“top”或未知。随着每个额外的被调用方都会增加调用字符串和摘要并集，过程间分析将面临上下文爆炸。切片和污点分析会过度近似，将无关的调用者和汇点也纳入分析。噪声较多的控制流图也会损害机器学习，导致模型适应错误边，并在干净的图上表现不佳。对于通用分析，通常最好控制假阳性，即使因此牺牲一些召回率也在所不惜，只要控制流图仍忠实反映程序的预期语义。

我们通过两个 SPEC 基准测试来说明这一点：在这些基准测试中，所有逆向工程工具都会漏掉大多数间接调用，而静态工具则会使控制流图充斥假阳性；随着程序规模增大（从 h264ref 到 gobmk），这一问题尤为突出。

在 h264ref-O0 中，两个全局结构控制着间接分派。DataPartition 的字段 writeSyntaxElement 决定使用 UVLC 还是 CABAC 编码器（第一次间接调用）；而 SyntaxElement 字段（UVLC 对应 mapping，CABAC 对应 writing）则驱动每个编码器内部的第二次间接调用（见 Listing 4）。UVLC 路径不会调用 CABAC 例程，反之亦然，因此，任何跨越这两个例程族的控制流图边在语义上都无效。逆向工程工具无法解析此处的许多目标，而 BPA 会引入大量假阳性。具体而言，BPA 会构造一个有向环，其中 writeSyntaxElement_CABAC 似乎调用了 writeSyntaxElement_UVLC，随后又返回自身；此外，这两个例程上还有自环；此时 AICT 为 $= 5.7$，召回率为 100%。iResolveX 的细化首先将 AICT 降至 3.4（仍存在虚假路径），随后在召回率为 95.4% 时将其降至 1.68，并消除了该有向环和一个自环。仅保留高置信度边后，在召回率为 80% 时，所有这些虚假路径均被移除，从而得到足以用于精度关键型分析的忠实控制流图。相比之下，CALLEE 和 AttnCall 在此案例中的 AICT 值为 ≥ 20，不仅超过了 BPA 的假阳性率，还引入了更多虚假路径。

对于 gobmk-O0 这类大型二进制文件，问题会更加严重。这里同时存在两类互不相关的间接调用：棋形评估中，check_pattern_hard 只通过 pattern->autohelper/pattern->helper 进行分派；GTP 命令处理中，gtp_main_loop 只通过 commands[i].function 进行分派（见列表 5）。这两类调用之间存在严格的语义边界；任何从棋形回调指向 GTP

处理程序（或反向）的边都是伪边。尽管如此，BPA 粗粒度的全局内存模型仍将它们混为一谈，产生了 3,554 个虚假环路（最大环路长度为 19），而合法环路仅有 115 个——在 AICT = 884.6 下增加了 **2,990%**，同时召回率为 100%。这会扭曲控制流图，并导致下游分析发生路径爆炸。借助 p-IndirectCFG，iResolveX 提高剪枝阈值，以移除所有虚假环路，并将 AICT 降低 50%，同时仍保持 95.5% 的召回率，从而恢复语义完整性，避免污染下游工作流程。

列表 5：gobmk：最简的两层间接调用说明。首先，Patternevaluation 仅调用 pattern->autohelper/pattern->helper。

其次，GTP 循环仅通过 commands[i].function 进行分派。

```
1 /* -- 与第一层间接调用相关的数据结构 -- */ 2 struct pattern {int
autohelper_flag; 4 double constraint_cost; 5 int (*autohelper)(int ll, int
m
in
调
cc
17
in
{ 2
令
static int 29 check_pattern_hard(int move, int color, struct pattern
*p
((p
cc
35
*/
*
Bl
46
*/
st
```

清单 4：h264ref 两步间接调用：首先，通过 DataPartition->writeSyntaxElement 选择 UVLC 或 CABAC 编码；其次，通过 SyntaxElement.mapping/SyntaxElement.writing 进行间接调用。

```
1 /* -- 与第一层间接调用相关的数据结构 -- */ 2 typedef struct
datapartition { 3 /* ... 其他字段 ... */ 4 int
(*
接
的
va
层
El
SyntaxElement; 13 14 /* -- 第一层间接调用的设置（0=UVLC，1=CABAC） --
*/
da
18
U
21
=
26
lin
w
32
34
CA
*/
点
D
se
inf); // 第二层间接调用 45 /* -- UVLC 编码步骤 -- */ 46 }
```

```
/* 通过命令进行间接调用 [i] 函数 */ for (int i = 0; 命令[i]名
```

称

数