COVERT：通过微体系结构事件的统计激活进行商用现货硬件木马检测

Mahmudul Hasan*、Sudipta Paria†、Swarup Bhunia† 和Tamzidul Hoque*
*堪萨斯大学电气工程与计算机科学系，美国堪萨斯州劳伦斯市，邮编66045
†佛罗里达大学电气与计算机工程系，美国佛罗里达州盖恩斯维尔市，邮编32611 电子邮箱：
{m.hasan@ku.edu, sudiptaparia@ufl.edu, swarup@ece.ufl.edu, hoque@ku.edu}

arXiv:2601.11939v1 [cs.ET] 17 Jan 2026

摘要——**商用现货（COTS）硬件（如微处理器）能够相比定制解决方案缩短开发时间并降低成本，因此被广泛应用于系统设计。然而，从消费者的角度来看，参与商用现成组件设计和制造的供应链实体均不受信任，因为其中可能被植入隐藏的恶意逻辑或硬件木马（HT）。由于商用现成组件具有黑盒特性且无法获取黄金模型，现有的木马检测解决方案大多不适用。一些能够应用于此类场景的研究需要昂贵的设备，缺乏可扩展性，并且仅适用于有限类别的木马。在本研究中，我们提出了一种新颖的、无需黄金模型的信任验证框架COVERT，面向商用现货微处理器，通过识别微体系结构罕见事件，并将现有处理器设计中的激活知识迁移至极易触发的内部节点，从而高效测试硬件木马植入物是否存在。COVERT利用大语言模型自动生成能够触发罕见微体系结构事件的测试程序，而这些事件可能被利用来构建木马触发条件。通过从公开可用的寄存器传输级实现中推导这些事件，COVERT能够验证广泛采用相同指令集架构的商用现成微处理器。我们在开源RISC-V商用现成微处理器上评估了所提出的框架，结果表明，它能够以较高覆盖率有效激活组合型和时序型木马触发器，凸显了该信任验证方法的高效性。通过从mor1kx Cappuccino OpenRISC处理器设计中筛选罕见微体系结构事件，COVERT在or1k Marocchino和PicoRV32这两种商用现成处理器中最罕见的5%事件上，实现了超过80%的触发覆盖率。**

索引词—**木马检测、测试生成、微架构事件、商用现成产品、组合型和时序型木马。**

I. 引言

与定制设计方法相比，商用现成组件（COTS）凭借缩短开发时间、降低硬件成本以及市场可获得性，提供了一种极具吸引力的系统设计范式。因此，商用现货微处理器已广泛应用于军事领域、航空电子、金融领域和商业领域。根据2022年的一份报告，国防应用中使用的微电子组件约有98%属于商用现成产品（COTS） [1]。对商用现成组件日益增加的依赖也同时带来了重大的安全问题。商用现成组件的供应商将设计、制造和测试分散到国内外不同的不可信实体中。任何能够接触设计的不可信实体都可能植入隐蔽的恶意逻辑或硬件木马，从而导致功能故障或敏感信息（例如加密密钥）泄露 [2]。现实世界中的许多网络攻击表明，不可信组件中的硬件木马威胁正在日益加剧 [3]。

现有的大多数硬件木马研究主要关注来自不可信晶圆厂或不可信知识产权供应商 [4], [5] 的植入威胁。这些对策大多依赖以下一项或多项条件：（i）能够访问黄金（无木马）芯片；（ii）能够白盒访问设计；或（iii）在设计时进行修改以集成对策 [2]。对于商用现成组件而言，这些假设均不成立，因此现有方法无法奏效。例如，逻辑测试方法需要访问设计才能生成有效的测试向量 [6]，[7]，，而大多数侧信道分析（SCA）方法则需要黄金芯片来建立参考特征，并需要访问设计以生成测试，从而确保木马中的开关活动 [8]。已有少量研究提出了适用于商用现货硬件的运行时方法，用于在实际运行环境中检测、容忍或防止木马激活 [9]–[11]。尽管运行时解决方案增加了一层安全防护，但信任验证方法通常更适合作为主要防御机制。已有侧信道辅助的无黄金芯片信任验证方法可用于商用现货硬件，但这些技术同样假设存在能够确保木马网络激活的、基于设计信息的测试生成方法[12], [13]。因此，商用现货硬件的高置信度黑盒信任验证仍存在亟待解决的关键需求。

本文提出 COVERT（商用现成组件木马检测验证框架），这是一种通过微体系结构事件的统计激活来检测商用现成微处理器中硬件木马的新型测试生成方法。COVERT 利用大语言模型（LLM）的软件代码生成能力，生成定向测试程序，以激活可能构成潜在木马触发器的罕见内部事件。尽管商用现成处理器的黄金设计可能无法获取，但商用现成组件的采购方通常会收到关于指令集架构（ISA）和架构特性的完整文档，其中包括中断、事件列表以及有限的硬件组件细节。COVERT 利用公开可用的 ISA 以及该 ISA 的开源寄存器传输级实现，探索罕见的微体系结构事件。这些事件中的大多数与实现无关，也就是说，它们适用于采用相同 ISA 的商用现成处理器。接下来，COVERT 构建定向测试程序，以触发这些罕见的微体系结构事件；这些事件是构造难以激活的木马触发条件的常见目标

本文已获准在 DATE 2026 发表。这是作者的初稿版本。最终论文将收录于会议论文集。

。COVERT 利用跨设计抽象之间的关系，将低层木马网络与高级事件关联起来。

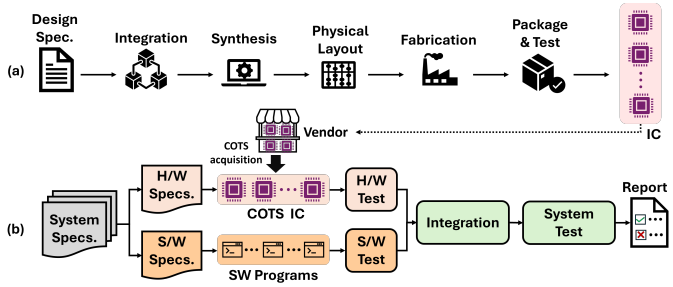


图 1. (a) 现代集成电路设计流程；(b) 商用现成集成电路集成流程 [14], [15]。

微体系结构事件，如流水线冒险、缓存行为或异常条件，直接源自 RTL 设计，并代表处理器中的精确功能场景。如果某个事件在微体系结构层面很少发生，那么在门级实现该事件的各个寄存器和逻辑锥也同样很少经历开关活动。COVERT 根据公开信息和开源实现，为给定的指令集架构精简罕见微体系结构事件空间，并基于这些事件开发测试，以全面探索潜在的木马设计空间。本文的主要贡献包括：

- 我们提出了一种名为 COVERT 的新型框架，无需获取商用现成微处理器的实现，即可通过从指令集架构和现有处理器设计中迁移激活知识来检测木马。
- 我们提出了一套系统化流程，用于从已知的微处理器 RTL 实现中提取罕见的微体系结构事件。为激活这些事件，我们开发了一种大语言模型辅助程序生成框架，其中包含一个基于 LangGraph（构建于 LangChain 之上）的智能体工作流，并通过自动反馈迭代修正程序错误。
- 我们通过将基于 OpenRISC mor1kx Cappuccino 参考设计生成的测试程序应用于两款黑盒商用现成处理器，对 COVERT 进行了广泛评估：一款是采用相同指令集架构的 OpenRISC Marocchino，另一款是遵循类似 RISC 设计理念的 PicoRV32。我们发现，两款处理器中最稀有事件组的触发覆盖率均超过 80%。我们还分析了通过多次激励这些事件来触发复杂木马时的覆盖率。我们已在此链接 [16] 开源了所有相关实验产物。

II. 背景

A. 商用现成微处理器的信任问题

定制集成电路的设计流程包含多个步骤，从规格说明开始，直至版图完成，如图 1(a) 所示。或者，如果市场上有商用现成集成电路，系统开发者便可跳过这些步骤，直接从市场上获取商用现货组件。例如，采购方可以根据所需的规格、性能和指令集架构，采购商用现成处理器，如图 1(b) 所示。然而，由于对设计和制造流程缺乏了解，使用商用现货硬件会给采购方带来信任问题，因为其中任何参与方都可能引入恶意篡改或硬件木马。

B. 现有验证技术

现有硬件木马检测方法适用于满足以下一项或多项假设的情况：i) 可获取作为参考的黄金（无木马）设计；ii) 具备芯片设计的白盒访问能力；iii) 通过设计修改来促进验证。然而，在商用现成组件 [2], [4], [9], [17]的场景下，这些条件均无法满足。例如，MERO [6] 等逻辑测试方法旨在生成测试向量，以确保激活设计中难以激活的稀有网络。然而，现有专注于激活稀有网络的测试生成方法依赖于对网表 [6]–[8]的访问。少数现有验证技术利用侧信道（例如功耗或延迟）。文献 [12] 中的作者采用激光辅助侧信道，观测不属于扫描链且难以激活的可疑触发器。文献 [13], 中的作者提出对功耗侧信道进行无监督聚类，以便在没有黄金特征的情况下识别可疑芯片。然而，这两项研究都要求掌握设计信息，以创建能够将稀有网络激活至其稀有取值的测试模式。

C. 大语言模型辅助的处理器验证

基于大语言模型的测试程序生成方法已用于功能验证和漏洞检测 [18]–[21]。这些技术展现出无需访问处理器设计即可实现高代码/功能覆盖率的潜力。然而，它们的目标是通过功能验证发现非故意设计漏洞，而非进行信任验证；它们不会生成能够触发复杂微体系结构事件或激活常作为木马触发器的稀有网络的程序。由于缺少面向安全的覆盖计划、假设存在用于正确性验证的黄金模型、不围绕安全目标重新定义覆盖区间，且在某些情况下需要访问内部 RTL覆盖率（这对商用现货硬件而言不可行），因此也无法轻易扩展用于木马检测。表I对所提出的COVERT框架与适用于处理器的现有验证技术进行了比较分析，并指出了这些技术在检测商用现货硬件中的硬件木马方面的局限性。

III. 方法论

图2展示了所提出的COVERT框架。该流程首先识别目标商用现货微处理器的指令集架构（ISA），并获取实现相同指令集架构或遵循相同架构理念的开源RTL设计。随后，我们从该RTL中提取详细的微体系结构信息，以定义高层事件。接着，利用信号概率分析按罕见程度对这些事件进行排序。最后，状态驱动代理生成测试程序以触发罕见事件，并报告其触发覆盖率。

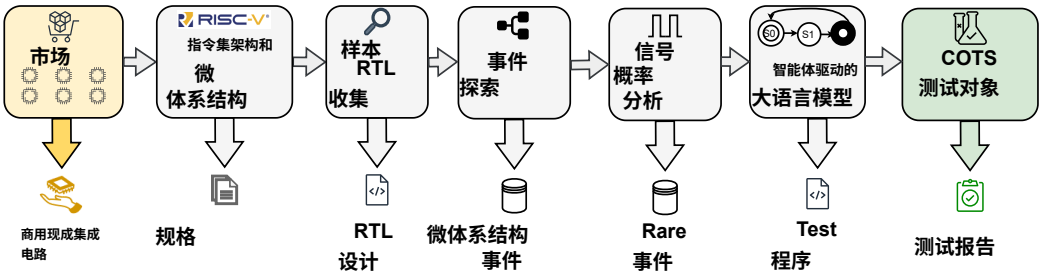


图2。C语言 OVERT框架：用于商用现成产品硬件木马检测的概览　　硬件。

表 I 现有验证技术与所提出的COVERT 框架的比较分析。

所提出的解决方案	Goal	Task	木马检测?	事件数 / 程序数。
LLM4DV [18]	功能验证	激励生成	No	不适用 / 不适用
Xiao 等人 [19]	处理器验证	测试程序生成	No	不适用 / 164
LLM-TG [21]	处理器验证	测试生成	No	不适用 / 132
Paria等人 [15]	COTS验证	测试程序生成	No	9–11 / ∼10
COVERT（本研究）	COTS验证 +木马检测	测试程序生成	Yes	368 / ∼500

A. 微体系结构事件探索

A微体系结构事件是指处理器微体系结构中可观测的内部条件或状态转换，它概括了处理器内部特定低级网络激活后产生的有意义的高层行为。例如，如果某个罕见网络受指令页故障标志、特权模式位和TLB未命中状态影响，则可将其抽象为微体系结构事件：“用户模式下发生指令页故障，且有待进行TLB重填。”这类事件抽象能够有效地将低级信号映射到架构不变状态，从而通过测试程序实现精准定向。这里，**架构不变意味着，该事件对于实现相同指令集架构的处理器始终有效且语义一致，不受内部设计差异的影响**。图3展示了事件探索的系统化流程。

该流程首先从1 **识别目标商用现货微处理器的指令集架构**并选择符合相同指令集架构和规格说明的代表性 RTL 实现开始。选择一个2 **Verilog RTL 设计**作为**参考**实现，以便识别罕见事件并随后生成定向测试程序。我们选择 RTL 设计来生成事件，因为它揭示了定位难以触达条件所需的内部控制逻辑和依赖关系，而公开的指令集架构手册和数据手册并未披露这些信息。随后，对选定的3 **RTL 进行解析并转换为抽象语法树**，从而能够对设计进行结构和语义分析。自定义脚本4 **在抽象语法树中，沿着扇入逻辑递归地向后追踪每个已识别的罕见信号，**同时追踪模块内和跨模块的信号依赖关系，直至预设深度。此类向后追踪通常以译码器阶段为界，这非常适合揭示触发罕见信号的具体指令。生成的分层追踪结果在底层网络活动与相应的高层架构行为之间建立了关键联系，并存储在 JSON 数据库中。随后，追踪结果会进一步抽象为简化的过程逻辑和控制逻辑块的文本表示，为大语言模型提供必要的5 **上下文，使其能够用自然语言解读结构依赖关系**并**推断相应的微体系结构事件**。为确保语义准确并避免幻觉，我们通过精心编写的系统提示词引导大语言模型，其中规定了其角色、上下文信息、分析规则以及预期的输出格式。基于这些信息，大语言模型会生成事件名称及其高层描述，并将其与追踪到的信号关联起来，同时提供架构摘要、测试生成指导和相关指令类别，为生成定向测试程序奠定基础。

B. 罕见事件识别

我们进行罕见性分析，以识别发生频率较低、可作为木马触发器的微体系结构事件。我们使用多种基准测试程序激励参考寄存器传输级设计，生成真实的开关活动。这些基准测试程序涵盖了广泛的指令类型、数据模式和控制流行为，既能真实反映实际应用，也有助于提取罕见事件。仿真期间会生成值变化转储（VCD）轨迹，以记录信号随时间的变化。从 VCD 文本文件中，信号被分为单比特网络和多比特总线，两者需要采用不同的活动度指标进行罕见性分析。对于**单比特网络**，我们根据仿真周期估算信号处于逻辑高电平（$\hat{}p_1$）和逻辑低电平（$\hat{}p_0$）的概率。罕见性指标定义为$\theta = \min(\hat{}p_0, \hat{}p_1)$。对于**多比特总线**，活动度通过归一化翻转率来衡量，其定义如下：

$$\theta = \frac{\text{Number of value changes}}{\text{Total number of consecutive cycle pairs}}$$

C. 利用大语言模型进行测试程序生成

COVERT 整合了大语言模型的软件代码自动生成能力，用于生成以 C语言/C++ 或内联汇编编写的定向测试程序。基于大语言模型的状态驱动型智能体

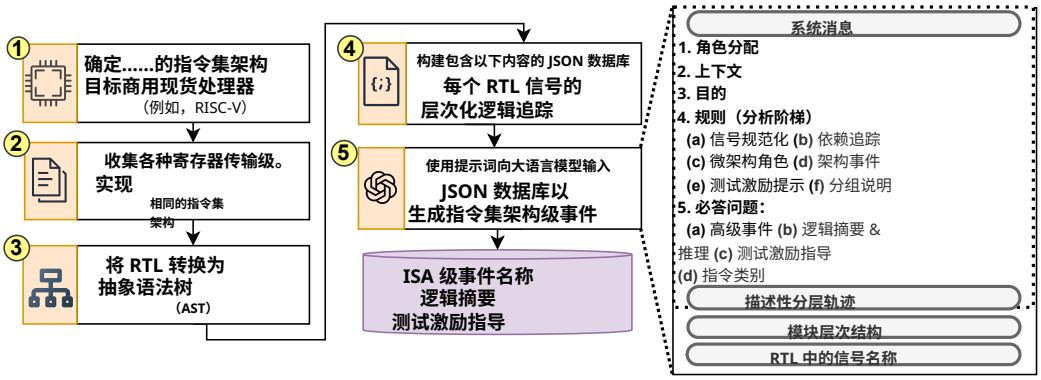


图3. 事件探索流程概览：该流程通过基于抽象语法树的分层逻辑展开，将寄存器传输级信号映射为微体系结构事件。

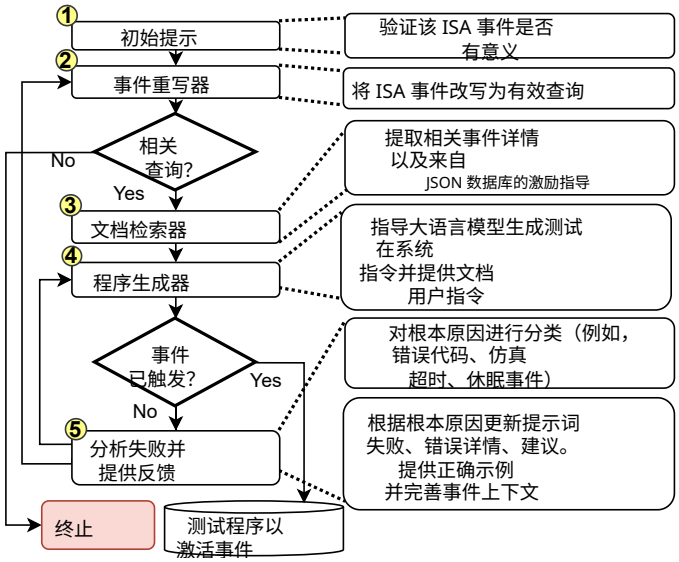


图4. COVERT 中由智能体驱动的工作流程，该流程利用大语言模型生成测试程序，以激活罕见的微体系结构事件。

程序生成工作流如图 4 所示，其中每个状态都会按顺序执行一项特定任务。以下小节将简要介绍各个状态。

1）事件重写器和主题分类器：COVERT 首先处理用户提供的事件查询，以1 验证其正确性，确保事件名称有效、一致且不存在语言错误。如果检测到错误，智能体会2重写查询，同时保留其预期语义。随后，修订后的查询将传递给主题分类器，由其验证该查询是否符合目标指令集架构的约束，并映射到有效的微体系结构事件。

2) 文档检索器：在验证事件查询后，智能体继续3 检索与该事件相关的微体系结构知识。通过提供底层逻辑推理，这一步对于引导大语言模型生成超越表层事件名称的有意义程序至关重要。用于此次检索的知识库在事件生成阶段（第 III-A 节）构建，并存储在结构化 JSON 文件中。对于每个事件，数据库除事件名称外还包含三个关键元数据字段：（**a**）描述导致该事件的推理链的逻辑摘要、（**b**）激励生成指南，以及（**c**）相关指令类别，指出最适用的指令集架构操作。

3）程序生成器：COVERT利用大语言模型4，通过系统化的提示模板生成测试程序。该过程从迭代检查开始，由于没有先前的输出，第一次迭代不应用任何反馈。随后，状态不断迭代，直至生成的程序满足触发条件。如果生成的程序未能通过测试，智能体会结合失败反馈并收紧约束来完善提示词，从而逐步提高成功的可能性。提示词通过现代大语言模型 API 组织为两层：系统指令和用户指令。这样既能动态更新用户层，又能保持系统层任务稳定。系统提示词定义模型的角色，强制执行全局规则（例如，只允许一个代码块、不得捏造操作码），并概述分步分析流程。此外，它还通过可运行示例实现 k-shot 学习：分别提供一个标准 C语言示例和一个内联汇编示例，以展示预期的输出格式。用户提示包含四个关键组成部分：(a) 来自前一状态的特定事件元数据（见第 III-C2 节）；(b) 用户定义的约束和目标；(c) 生成前的自检清单；以及 (d) 严格定义的输出结构。

另一个挑战是，大语言模型往往难以从大型、无结构的 PDF 手册中持续准确地提取正确的指令集架构语义。为缓解这一问题，我们将指令集架构文档重新组织为两部分：(a) 用于记录有效操作码、操作数和编码规则的 JSON 表示；以及 (b) 原始 PDF 内容，用于提供高层次的体系结构背景（例如内存模型、异常和专用寄存器）。这种双格式知识库结合自动化的检索增强机制，显著提高了代码的正确性和有效性，同时减少了成功实现事件触发所需的迭代次数。

4) 事件触发器检测：在此阶段，智能体会编译并仿真上一阶段生成的测试程序，并生成 VCD 跟踪。随后，将此跟踪与通过执行空主函数生成的黄金参考跟踪进行比较，以检测微体系结构

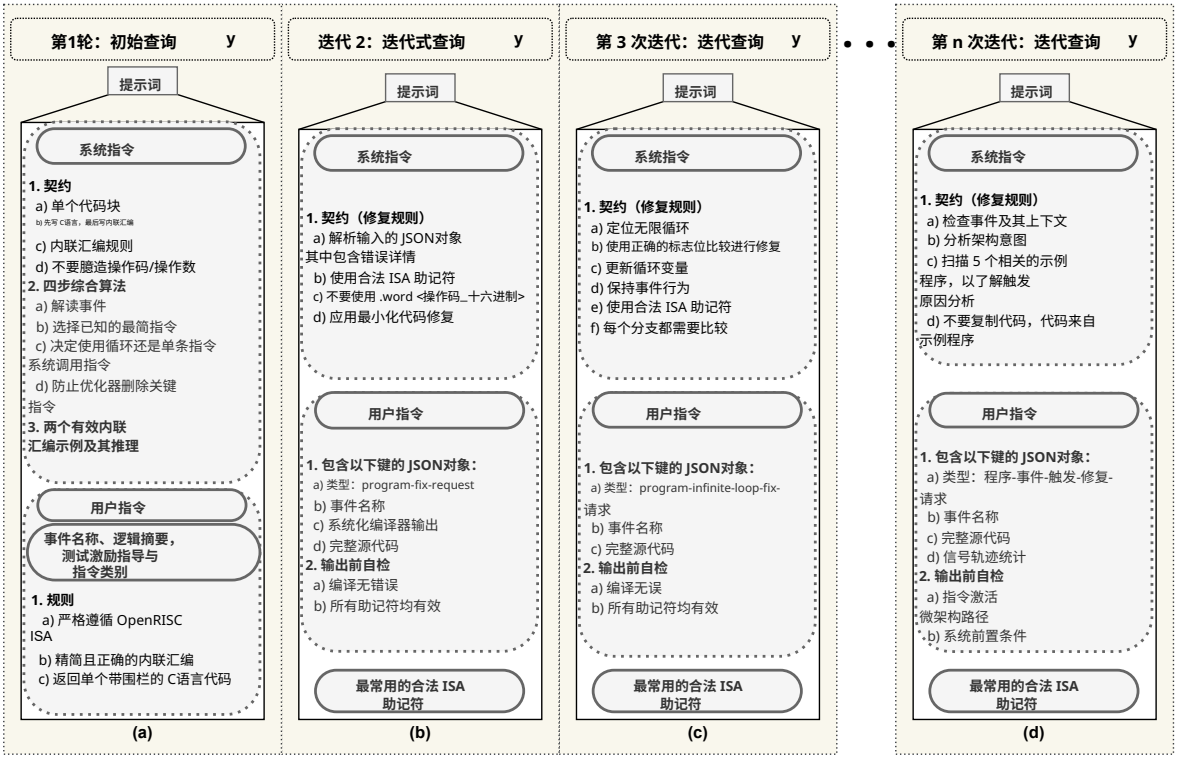


图 5. COVERT框架中迭代式提示词优化流程概览。该图展示了从 (a) 初始查询到 (b-c) 基于修复的迭代，再到 (d) 后续优化的过程。初始提示词指示大语言模型生成一个满足事件条件的最简 C/C++ 或内联汇编程序。如果编译或仿真失败，则将错误输出附加到下一条提示词中，以便模型修复代码。如果程序能够运行但未激活该事件，则将逻辑反馈和先前的成功示例添加到下一条提示词中。这个循环会持续进行，直到优化后的提示词生成一个合法程序，并成功触发这一罕见事件。

转换差异。转换次数增加表明测试程序已成功触发目标事件并到达终止状态。如果事件未被触发，框架会将失败归因于以下三种主要原因之一：(a) 编译错误，(b) 仿真超时，或 (c) 未发生事件激活。智能体会记录失败原因及相关输出，并将其转发至反馈阶段，以指导下一轮测试生成。

5) 失败分析器与反馈： 主要目标是建立一个结构化的 5 反馈循环，使大语言模型能够根据失败的根本原因迭代优化其输出。智能体会分析由三种主要原因之一导致的失败输出，并在后续提示词迭代中加入有针对性的反馈。对于编译器错误，智能体会将编译器日志解析为结构化的 JSON 格式，详细说明错误类型、位置和原因。对于仿真超时，系统提示词会更新，指示大语言模型检测可能存在的无限循环、添加终止条件，或修改有问题的内联汇编片段。当程序能够编译并执行，却未能触发目标事件时，智能体会在提示词中补充此前成功且与事件相关的示例，以改善上下文。在所有迭代过程中，系统提示词都会强制执行修复策略和约束规则，而用户提示则包含结构化错误报告以及先前代码尝试的历史记录。为了帮助生成有效的内联汇编，智能体会在用户提示中添加一份常用指令集架构指令列表，从而减少模型为查找目标指令而搜索 JSON 数据库的需要。这种引导式反馈机制确保大语言模型能够获得精确且结合上下文的信息，显著提升其生成有效测试程序的收敛速度。图 5 展示了 COVERT框架中大语言模型的迭代式提示词优化流程。

IV. 结果

我们采用来自 OpenRISC [23] 架构的 mor1kx Cappuccino 内核 [22] 作为样例设计，提取微体系结构事件。为进行验证，我们选择了 or1k Marocchino [24] 处理器，它实现了完全相同的指令集架构，但采用乱序流水线；此外还选择了 PicoRV32 [25] 处理器，其指令集架构和设计理念与之相似。我们使用 Verilator [28] 对 CHStone [26] 和 MiBench [27] 基准测试套件进行随机测试，以获得精确到周期的仿真结果，随后计算信号概率以识别稀有网络。我们使用 PyVerilog [29] 解析寄存器传输级代码并生成抽象语法树，从而将这些稀有网络映射到架构事件。我们借助 LangChain [30] 中的 LangGraph 构建状态驱动代理，以完成框架并实现自动化测试生成。该智能体通过文件助手连接 OpenAI Assistant API [31], , 以检索指令集架构文档信息。在 API 中，我们选择了 OpenAI 4.1 模型（ mini），其编码能力出色，且相较于更大型的模型具有更高的性价比

表 II MOR1KX Cappuccino 流水线模块中的信号分布，以及映射到相应高级事件的 RTL 信号示例。

阶段	总计信号	总计事件	Rare 事件	示例信号	对应的高级事件
译码	56	45	11	译码_操作码_-movhi o_	指令发生流水线停顿因依赖于高位立即数而进行译码。
取指	237	149	65	指令缓存_填充_-done_o	缓存填充完成，允许指令取指恢复。
ALU	93	76	16	溢出_集_o 检测	有符号溢出或除以零。
LSU	278	173	37	对齐_错误_短	未对齐内存访问。
Ctrl	194	90	25	控制_操作_-移至特殊用途寄存器_i	移至特殊用途寄存器指令传输来自通用寄存器寄存器（PR）至特殊用途寄存器。

。所有实验均在一台配备 24 核 Intel Core i7-13700F 处理器（基准频率 2.1 GHz、64 GB RAM）并运行 Ubuntu 22.04.5 的系统上完成。我们已在 [16]开源所有相关实验产物。

A. 将信号映射至架构事件

表 II 总结了 mor1kx Cappuccino 内核五个流水线模块中的可用信号。原则上，可以通过多个信号的有意义组合来推导微体系结构事件。然而，在我们的实验中，我们将每个信号都视为潜在的架构事件，同时排除在基准测试仿真期间从未发生切换或在 AST 抽象过程中失败的信号。我们还报告了稀有度阈值为 θ<=0.05 和$\theta \in [0,1]$ 时的罕见事件数量。表 II 还列出了五个代表性示例，每个流水线模块各一个，展示了如何将 RTL 信号映射为架构事件。

B. 商用现成产品的测试程序生成

要使用现代大语言模型生成有效的高级程序和裸机程序，并可靠地触发特定事件，需要准确了解寄存器地址和位级配置，而大语言模型无法从指令集架构文档中推断这些信息。为克服这些局限，我们逐步改进了提示策略。图 6 展示了每次改进如何逐步提高样例设计中被触发事件的比例，从基线提示到迭代反馈皆是如此。

T1. 基线提示：在初始阶段，我们只向大语言模型（GPT 4.1 mini）提供了 PDF 格式的指令集架构文档，并且仅在用户角色中设置了少量规则。输入内容包括事件名称和简要的架构摘要。在此设置下，测试程序生成的正确率最低。大多数输出要么无法编译，要么导致仿真超时，因此如图 6 所示，流水线模块中成功触发的事件数量较少。

T2. 角色引导提示：识别出常见问题后，我们通过两种方法加以解决：(i) 将 PDF 中基于表格的指令集架构描述转换为 JSON 模式 {id, syntax, encoding, example_hex, description}，并仅在查看内存和特殊用途寄存器（SPR）详情时保留 PDF；(ii) 强制采用严格的

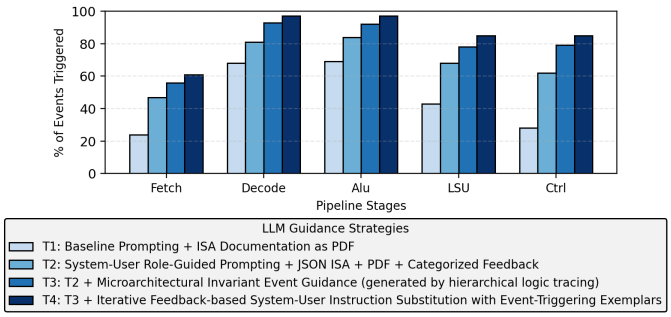


图 6. 使用四种逐步改进的提示策略（T1–T4）时，样例设计中各流水线模块触发事件的百分比。

编码规则，并建立修复循环，将编译器和仿真的错误反馈给大语言模型，以修正当前程序，而不是重新生成程序。我们还纳入了提供预定义宏和 SPR 访问函数的编译器头文件定义。因此，我们得以减少错误，这些错误主要源于无效的操作码/操作数组合以及 SPR 位配置。图 6 显示，触发的事件数量显著增加：取指从 24% 提升至 47%，加载存储单元从 43% 提升至 68%，控制从 28% 提升至 62%；而译码和算术逻辑单元仅略有改善，因为其中大多数事件本来就不需要太多底层编程工作。

T3. 纳入事件引导：对于仍未能触发的事件，我们加入了更高层次的上下文信息，即对信号行为的逻辑摘要、关于合适测试激励的指导，以及可能影响该事件的指令类别。这种抽象使大语言模型能够将底层活动与指令集架构层面的结果联系起来，同时保持架构无关性。在 T2 的基础上，这些补充提高了所有流水线模块中事件的触发百分比。

T4. 使用示例进行逐次运行角色定义：最后，T3 扩展为迭代反馈循环，根据程序生成的当前状态调整系统指令和用户指令。当某个事件未能触发时，框架会从现有池中检索少量此前成功的程序（例如 3–5 个），并将其作为示例与更新后的指令一同提供。这种方法进一步改善了事件触发效果，如图 6 所示。即使应用了 T4，取指模块的覆盖率仍然较低，因为大语言模型生成了规模较小的基于循环的测试，试图触发存储器事件，例如指令缓存未命中和重新填充、页错误、通过特殊寄存器实现的缓存失效，以及指令总线取指。这些测试没有创建必要的环境，包括规模大且多样化的代码、跨越相距较远代码位置的大量跳转，以及能够改变缓存和存储器设置的长时间运行过程。

C. 商用现成产品的事件覆盖率

为评估 COVERT 在未知商用现成处理器上的有效性，我们运行了根据样例实现生成的测试程序。对于商用现成产品，由于实现方式不同，要将由信号产生的事件与样例设计进行直接的一对一匹配颇具挑战。因此，我们首先在 Marocchino 和 PicoRV32 上重新运行相同的 CHStone 和 MiBench

表 III 使用样本 MOR1KXCAPPUCCINORTL 中的测试程序评估 COVERT，针对两个目标：OpenRISC Marocchino处理器的模块级稀有事件覆盖率，以及单模块 RISC-V PICORV32 的总体稀有事件覆盖率。

OpenRISC Marocchino																		RISC-V PICORV32IMC		
模块	取指			译码			ALU			LSU			Ctrl			合并		
θ	Rare 事件	Rare 事件触发	% of 触发	Rare 事件	Rare 事件触发	% of 触发	Rare 事件	Rare 事件触发	% of 触发	Rare 事件	Rare 事件触发	% of 触发	Rare 事件	Rare 事件触发	% of 触发	Rare 事件	Rare 事件触发	% of 触发
	模拟的测试程序 = 80			模拟的测试程序 = 41			模拟的测试程序 = 70			模拟的测试程序 = 91			模拟的测试程序 =77			已模拟的测试程序 =364		
0.05	78	62	79.49	21	21	100	12	11	91.67	107	85	79.44	86	74	86.05	101	86	85.15
0.15	110	94	85.45	34	34	100	26	25	96.15	195	172	88.21	95	83	87.37	134	119	88.81
0.25	120	102	85.00	41	41	100	42	41	97.62	220	197	89.55	99	87	87.88	147	132	89.80
0.35	125	106	84.80	50	50	100	56	55	98.21	247	224	90.69	103	91	88.35	154	139	90.26
0.45	130	111	85.38	71	71	100	63	62	98.41	253	230	90.91	108	96	88.89	167	152	91.02
0.55	165	143	86.67	90	90	100	65	64	98.46	258	235	91.09	116	104	89.66	170	155	91.18
0.65	176	153	86.93	94	94	100	67	66	98.51	260	237	91.15	120	108	90.00	176	161	91.48
0.75	180	157	87.22	99	99	100	70	69	98.57	260	237	91.15	120	108	90.00	179	164	91.62
0.85	183	160	87.43	101	101	100	73	72	98.63	261	238	91.19	125	113	90.40	179	164	91.62
1	199	175	87.94	104	104	100	75	74	98.67	266	243	91.35	135	123	91.11	182	167	91.76

基准测试，以识别罕见事件。对于 Marocchino，我们模拟了特定于模块的测试程序。对于 PicoRV32，由于它采用单一顶层模块架构，没有明确的流水线阶段，因此我们运行了整套测试程序。我们使用大语言模型将 OpenRISC 内联汇编自动转换为适用于 PicoRV32 的 RV32IMC 指令集。

表 III 总结了结果。当稀有度阈值（见第 III-B 节）为 5% 时，Marocchino 在取指和加载存储单元模块中实现了近 80% 的事件覆盖率，而译码和算术逻辑单元超过 90 %，控制模块则超过 86%。PicoRV32 的激活率与之相近。随着稀有度阈值放宽，所有模块的覆盖率都稳步提升。译码和算术逻辑单元对微架构特征的依赖较少，而取指和加载存储单元涉及缓存、内存管理单元和特殊用途寄存器的交互，这解释了它们在 Marocchino 中覆盖率略低的原因。PicoRV32 是一种尺寸优化内核，不具备缓存、内存管理单元和访问模式；即使运行转换后的汇编程序，覆盖率也只额外提升了约 2%。这些结果表明，源自样本处理器核设计的微架构程序在不同实现中仍然有效，即使缺乏内部设计知识，也能实现较高的覆盖率。遵循相似指令集架构的商用现成微处理器总会存在重叠的罕见事件，而独有事件则源于实现特有的特性。通过持续从多种开源 RTL 设计中生成微架构事件，采购方可以逐步覆盖共有和独有的罕见事件，提高稀有节点激活率，并建立信心。木马触发器可以是组合型或时序型。根据定义，组合逻辑木马由单个事件触发；因此，此分析可视为组合触发覆盖率。

D. 商用现货 *Marocchino* 的复杂触发覆盖率

一次或多个不同事件的发生可视为组合触发器，而时序触发器则可通过多个事件组成的序列来设计。我们通过联合激活各个事件构造时序木马实例，从而扩展了分析。为使激活概率保持较低但又符合实际情况， [32], 我们将事件组合限制在图 7 所示的实际稀有度阈值范围内。我们观察到，OpenRISC Marocchino 所有模块的时序触发覆盖率从取指模块的约 60% 到译码和算术逻辑单元模块的近 100%，合并后的联合 θ值约在 4.6×10^{-16} 至 0.24 之间。

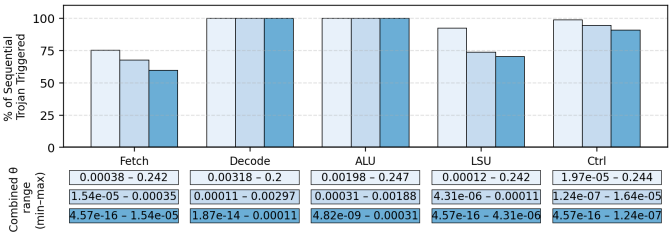


图 7. 在 OpenRISC Marocchino 微处理器中，联合稀有度阈值为 θ 时，各流水线模块的顺序触发覆盖率百分比。

E. 覆盖率数学分析

覆盖率计算为测试期间触发的木马数量占木马总数的比例。

◯i 对于单个测试程序，每个木马 j 由一个测试程序作为目标 t_j ，该程序以概率 p_j触发它。假设所有 T 测试程序彼此独立地执行，则触发的木马期望数量为：

$$\mathbb{E}[\text{Triggers}] = \sum_{j=1}^{T} p_j \tag{1}$$

$$\text{Exp. coverage}: \textbf{Coverage}_{indv} = \frac{1}{N}\sum_{j=1}^{N} p_j \tag{2}$$

◯i 对于组合测试程序，事件e_j 至少被一个测试程序触发的概率为： y p g

$$P_{\text{trigger},j} = 1 - \prod_{t_i \in \mathcal{T}_j} (1 - p_{ij}) \tag{3}$$

所有 N事件中触发的木马的期望数量为：

$$\mathbb{E}[\text{Triggers}] = \sum_{j=1}^{N} \left(1 - \prod_{t_i \in \mathcal{T}_j} (1 - p_{ij})\right) \tag{4}$$

其中， $\{p_{ij}\}_{i=1}^{n}$ 是 $[0, 1$中的一组独立概率。

$$\text{Exp. coverage}: \textbf{Coverage}_{comb} = \frac{1}{N}\sum_{j=1}^{N} \left(1 - \prod_{t_i \in \mathcal{T}_j} (1 - p_{ij})\right) \tag{5}$$

可以证明，当多个 $p_{ij} > 0$时， $1 - \prod_{i=1}^{n}(1 - p_{ij}) \geq \max_i p_{ij}$，且不等式为严格不等式。这意味着 **覆盖率**$_{comb}$ ≥**覆盖率**$_{indv}$。我们在实验中观察到了上述所有趋势。

V. 讨论与结论

我们提出了 COVERT，这是一种实用的解决方案，可用于验证商用现货硬件的可信度，同时具备可扩展性（适用于不同的设计规模）和灵活性（适应微体系结构变体）。COVERT 这一无黄金参考的商用现成组件信任验证框架的核心思想，是利用实现无关的微体系结构事件，并通过一组测试程序多次触发这些事件，从而以统计方式最大化触发木马的概率。据我们所知，这是首次对商用现货微处理器进行信任验证，可供原始设备制造商（OEM）在将不可信商用现成组件集成到系统中之前验证其可信度。

尽管 COVERT 很有前景，但我们注意到，在覆盖率、测试效率和可扩展性方面仍有显著的改进空间。其中包括：(1) 当目标商用现成产品包含内存保护机制或纠错码等未见过的实现特性时，即使指令集架构相同，也需要考虑触发覆盖率的下降；(2) 增加测试程序在指令混合方面的多样性。有多种方法可以实现这一点。例如，在 COVERT 生成的有效测试程序前添加随机前缀代码，可以同时改变指令混合和执行轨迹，从而在统计上提高激活随机触发条件的概率。我们未来的工作将研究这些改进机会。

VI. 致谢

作者感谢 Purdue 安全微电子生态系统中心（CSME）提供的支持–CSME#210205。

参考文献

[1]S. Shivakumar 和 C. Wessner。（2022年6月）半导体与国防：利害关系何在？战略与国际问题研究中心。 [在线]。网址：https://www.csis.org/analysis/semiconductors-and-national-defense-what-are-stakes?utm source=_chatgpt.com [2] K. Xiao、D. Forte、Y. Jin、R. Karri、S. Bhunia 和 M. Tehranipoor，“硬件木马：十年后的经验教训”，《*ACM*电子系统设计自动化汇刊》*(TODAES)*，第22卷，第1期，第6页，2016年。[3] P. P. Sarker、U. Das、N. Varshney、S. Shi、A. Kulkarni、F. Farahmandi 和 M. Tehranipoor，“当日常设备变成武器：深入审视寻呼机和对讲机攻击”，*arXiv*预印本 *arXiv:2501 17405*，2025年。。[4] A. Jain、Z. Zhou 和 U. Guin，“硬件木马检测最新进展综述”，载于*2021*年*IEEE*国际电路与系统研讨会（*ISCAS*）。IEEE，2021年，第1–5页。[5] J. Cruz、A. Gaikwad 和 S. Bhunia，“利用机器学习生成训练数据集，实现硬件木马自动植入”，载于*2022*年*IEEE*国际硬件安全与信任研讨会（*HOST*）。IEEE，2022年，第140–145页。[6] R. S. Chakraborty、F. Wolff、S. Paul、C. Papachristou 和 S. Bhunia，“MERO：一种硬件木马检测的统计方法”，载于密码硬件与嵌入式系统—*CHES 2009*。Springer，2009年，第396–410页。[7] S. Paria、P. Gaikwad、A. Dasgupta 和 S. Bhunia，“Latent：利用自动测试模式生成进行硬件木马检测”，载于*2024*年*IEEE*第*33*届亚洲测试研讨会（*ATS*），2024年，第1–6页。[8] Y. Huang、S. Bhunia 和 P. Mishra，“利用侧信道分析实现可扩展的木马检测测试生成”，《*IEEE*信息取证与安全汇刊》，第13卷，第11期，第2746–2760页，2018年。[9] M. Hasan、J. Cruz、P. Chakraborty、S. Bhunia 和 T. Hoque，“零信任环境下商用现成处理器的木马弹性计算”，《*IEEE*超大规模集成电路系统汇刊》，第30卷，第10期，第1412–1424页，2022年。[10] L. Cassano、M. Iamundo、T. A. Lopez、A. Nazzari 和 G. Di Natale，“Deton：通过软件混淆击败微处理器中的硬件木马”，《系统架构期刊》，第129卷，第102592页，2022年。[11] M. Beaumont、B. Hopkins 和 T. Newby，“Safer Path：采用碎片化执行与复制的安全架构，以防范植入木马的硬件”，载于《欧洲设计、自动化与测试会议论文集》。EDA Consortium，2012年，第1000–1005页。[12] A. Stern、D. Mehta、S. Tajik、F. Farahmandi 和 M. Tehranipoor，“Sparta：一种用于木马检测的激光探测方法”，载于2020年IEEE国际测试会议（ITC）。IEEE，2020年，第1–10页。[13] S. Yang、P. Chakraborty、P. SLPSK 和 S. Bhunia，“采用不可信商用现货产品的可信电子系统”，载于2021年第22届国际电子设计质量研讨会（ISQED）。IEEE，2021年，第198–203页。[14] T. Hoque、P. SLPSK 和 S. Bhunia，“商用现成产品中的信任问题：挑战与新兴解决方案”，载于2020年五大湖超大规模集成电路研讨会论文集，系列 GLSVLSI ’20。美国纽约州纽约市：计算机协会，2020年，第211–216页。[15] S. Paria、A. Dasgupta 和 S. Bhunia，“迈向知识产权核和商用现成产品的自动化验证：利用大语言模型覆盖硅前和硅后阶段”，载于2025年第43届IEEE超大规模集成电路测试研讨会（VTS），2025年，第1–5页。[16] M. Hasan 和 S. Paria，“COVERT工件”，2025年。 [在线]。网址：https://github.com/UF-Nelms-IoT-Git-Projects/COVERT [17] J. Cruz 和 J. Hamlet，“硅前硬件木马的设计、检测与预防综述”，IEEE Access，2025年。[18] Z. Zhang、G. Chadwick、H. McNally、Y. Zhao 和 R. Mullins，“LLM4DV：利用大语言模型生成硬件测试激励”，2023年。[19] C. Xiao、Y. Deng、Z. Yang、R. Chen、H. Wang、J. Zhao、H. Dai、L. Wang、Y. Tang 和 W. Xu，“基于大语言模型的处理器验证：神经形态处理器案例研究”，载于2024年欧洲设计、自动化与测试会议暨展览会（DATE），2024年，第1–6页。[20] S. Paria、A. Dasgupta 和 S. Bhunia，“借助大语言模型引导路径探索片上系统安全”，载于2024年五大湖超大规模集成电路研讨会论文集，系列 GLSVLSI ’24。美国纽约州纽约市：计算机协会，2024年，第252–257页。[21] Y. Deng、R. Chen、C. Xiao、Z. Yang、Y. Luo、J. Zhao、N. Li、Z. Wan、Y. Ai、H. Dai 和 L. Wang，“LLM - tg：利用大语言模型实现处理器自动化测试用例生成”，载于2024年第42届IEEE国际计算机设计会议（ICCD），2024年，第389–396页。[22] OpenRISC，“mor1kx：一个开源的OpenRISC 1000处理器知识产权核”，https://github.com/openrisc/mor1kx，5.1.1版。[23] OpenRISC，“OpenRISC项目概览”，https://openrisc.io，访问日期：2025年9月14日。[24] OpenRISC，“or1k_marocchino：基于Tomasulo算法的OpenRISC处理器知识产权核”，https://github.com/openrisc/or1k_marocchino。[25] YosysHQ，“PicoRV32：一种尺寸优化的RISC-V中央处理器”，https://github.com/ YosysHQ/picorv32，2019年，1.0版。[26] Y. Hara、H. Tomiyama、S. Honda 和 H. Takada，“面向实用C语言高层次综合的CHStone基准程序套件提案及定量分析”，载于IEEE国际电路与系统研讨会（ISCAS）论文集，2009年，第1–4页。[27] M. R. Guthaus、J. S. Ringenberg、D. Ernst、T. M. Austin、T. Mudge 和 R. B. Brown，“MiBench：免费且具有商业代表性的嵌入式基准测试套件”，载于IEEE国际工作负载特征分析研讨会（WWC）论文集，2001年，第3–14页。[28] W. Snyder，“Verilator：快速的开源Verilog/SystemVerilog仿真器”，https://www.veripool.org/verilator/，访问日期：2025年9月14日。[29] S. Takamaeda-Yamazaki，“PyVerilog：用于Verilog硬件描述语言的基于Python的硬件设计处理工具包”，载于第11届IEEE开源系统与技术国际会议（ICOSST）论文集，2015年，第45–51页。[30] L. Team，“LangGraph：面向LangChain的状态驱动工作流”，https://docs.langchain.com/docs/langgraph/，访问日期：2025年9月14日。[31] OpenAI，“OpenAI Assistants API”，https://platform.openai.com/docs/ assistants，访问日期：2025年9月14日。[32] B. Shakya、T. He、H. Salmani、D. Forte、S. Bhunia 和 M. Tehranipoor，“硬件木马与受恶意影响电路的基准测试”，《硬件与系统安全期刊》，第1卷，102，2017 1，85no. pp. – 。