# 使用协作式大语言模型检测密码学相关软件包

Eduard Hirsch[1] a, Kristina Raab[2] b, Tobias J. Bauer[2] c和 Daniel Loebenberger[2] d
东巴伐利亚技术大学（*OTH*），德国安贝格 *92224*[2]弗劳恩霍夫*AISEC*，德国加兴 *85748*
*e.hirsch@oth-aw.de*，*{kristina.raab, tobias.bauer, daniel.loebenberger}@aisec.fraunhofer.de*

arXiv:2603.07204v1 [cs.CR] 7 Mar 2026

关键词: 密码学清单、依赖项发现、密码学物料清单、协作式大语言模型

摘要: IT系统正面临日益增多的安全威胁，包括高级持续性攻击以及未来可能出现的量子计算漏洞。向密码敏捷性和后量子密码学（PQC）转型，需要在异构IT环境中可靠地清点密码资产。由于软件包数量庞大，人工检测与密码学相关的软件并不可行。此外，静态代码分析流程往往难以应对现代生态系统的多样性。本研究探索将大语言模型（LLM）用作启发式工具，以发现密码学资产。我们提出一种协作框架，利用多个大语言模型评估软件的相关性，并通过多数投票汇总其输出。为保护数据隐私，该方法在本地部署运行，不依赖外部服务器。我们使用超过 65,000 个 Fedora Linux 软件包，通过统计分析、模型间一致性评估和人工验证来评估该方法的可靠性。初步结果表明，大语言模型集成可作为识别密码学软件的高效初筛工具，从而减少人工工作量并助力 PQC 转型。本研究还比较了本地部署与在线大语言模型配置，阐述了自动化密码学资产发现的主要优势、局限性和未来发展方向。

## 1 引言

当前的 IT 基础设施持续面临网络攻击的威胁。此外，量子计算的发展对现有使用经典（非后量子）密码学方案的系统构成了重大风险（Mosca等人，2024）。另外，不应认为任何已部署的协议都能永远安全（Housley，2015）。因此，组织必须做好准备，以便针对新出现的威胁快速调整其密码学机制。这种能力被称为 密码敏捷性 （Näther等人，2025），对于在不断演变的漏洞面前维持稳健的安全服务至关重要，同时也支持密码原语和协议的变更。

实现密码敏捷性的第一步，是系统地识别正在使用的所有密码学资产（密码学资产）。密码学资产包括密钥材料、密码算法以及分布在各 IT 系统中的通信协议。在运行环境中开展这一流程鲜少能够一帆风顺。许多组织甚至尚未建立完整的通用 IT 资产清单，更不用说其密码组件了（Ponemon Institute 和 Entrust，2024）。尽管如此，建立这样的清单至关重要。它为开发密码敏捷基础设施奠定了基础（网络安全与基础设施安全局等，2023）。

创建密码资产清单在大型异构系统中尤其具有挑战性。一个组织的软件组合可能涵盖数万个不同的软件包，其中许多直接或通过传递依赖与密码库相关联。这些软件包可能分布在多个企业系统中，这些系统由数千个相互依赖的软件包、容器层、微服务和嵌入式组件组成，跨越多种操作系统和硬件代际。其中许多工件通过CI/CD流水线或包管理器持续升级。然而，另一些则长期存在于未打补丁的遗留主机或长期使用的固件中。密码功能

a https://orcid.org/0000-0001-9593-2342 b
https://orcid.org/0009-0001-7072-0340 c
https://orcid.org/0009-0006-1073-3971 d
https://orcid.org/0000-0002-7969-6260

可能以显式方式暴露，例如通过对OpenSSL的直接函数调用，也可能以隐式方式深埋在传递依赖、静态库或供应商二进制文件之中。

大规模手动识别与密码学相关的软件包并不可行。传统静态分析流水线难以应对当前软件产品的广度和异质性。软件包数量庞大且版本各异，使加密资产发现成为一项容量挑战。因此，对这些软件包进行（半）自动化分析至关重要。

在此背景下，最先进的大语言模型（LLM）已成为强大的通用代码理解引擎。这些模型经过海量数据训练（OpenAI，2023；Meta AI，2024；Google DeepMind，2025；Anthropic，2023）。这些知识可用于大规模识别使用密码库／协议或实现密码算法的软件包。

本文探讨大语言模型能否通过加快清点工作、减少人为错误，并为后续密码学迁移步骤奠定基础，来协助加密资产发现流程（Loebenberger等人，2025）。我们提出一种基于启发式的方法，采用最先进的提示技术，询问多个大语言模型某个给定软件包是否具有密码学相关性。本研究考察了通过多数投票共识机制汇总响应的做法。近期文献，例如（Davoudi et al., 2025；Choi等人，2025；Li et al., 2024；Kirchhoff和Pavkovic，2025）表明，在缺乏真实标注且无法大规模进行人工验证的情况下，多个大语言模型通过多数投票进行协作式知识验证，能够提供可靠且高效的近似推理。

我们的目标是创建一份初步但可操作的与密码学相关的软件包清单，以便在后续的PQC迁移工作中进行进一步分析。本文探讨以下研究问题：

**RQ1** 如何利用大语言模型，以启发式方法识别实现或依赖密码功能的软件包?

**RQ2** 通过利用多个大语言模型的汇总输出，是否能够提升响应质量?

本文其余部分安排如下：第 2 节回顾该研究领域的相关文献。第 3 节将更详细地指出本研究旨在解决的开放挑战。第 4 节介绍研究方法，包括方法设计、所用大语言模型的选择，以及为回答研究问题而采取的具体步骤。第 5 节呈现对该方法进行统计评估的结果。第 6 节讨论并分析研究发现，同时指出未来研究方向。最后，第 7 节总结全文。

## 2 相关工作

### 2.1 软件包中密码学函数的发现

我们考察了旨在识别软件包中密码功能的各类研究。文献中反复出现三种主要方法：（i）静态代码分析（STATIC）；（ii）知识驱动的模式匹配（KNOW）；以及（iii）基于大语言模型的分析（LLM）——通常以混合形式结合使用。

STATIC 方法会在不执行代码的情况下分析源代码或二进制代码（不同于动态代码分析），通常使用 SonarQube、Cod-eQL、Ghidra 和 IDA Pro 等工具。这些方法可以识别密码学资产（如证书和密钥）或库调用。然而，它们通常针对特定编程语言，这限制了其应用范围。例如，IBM 的 CBOM Kit（IBM Research，2025）专注于 Java 和 Python 代码。

KNOW 方法（如（Wind River Systems，2020）、（Leirimaa，2024）和（Schmitt 等人，2024））通过专家整理的关键词列表、正则表达式规则或分类体系标记密码学函数。这类方法往往存在误报、依赖命名约定以及需要持续更新等问题。此外，这些方法仅适用于特定的语言环境。例如，（Leirimaa，2024）仅适用于实现了 Node.js 加密模块的 JavaScript 软件包。因此，基于知识的密码学资产检测能力有限。

我们发现有两项研究在其方法中结合了 STATIC 和 KNOW 要素。在（Rattanavipanon 等，2025）中，研究人员通过将手动编制的易受量子攻击的库和函数列表与静态调用图分析关联起来，检测 Linux 二进制文件中的密码学函数。值得注意的是，该方法分析的是二进制可执行文件，而不是直接分析源代码；在受限环境中或涉及第三方资产时，这一特点具有优势。

不过，该方法维护成本高且容易出错，需要持续手动更新易受量子攻击的库和函数调用。最后需要指出的是，静态链接的库完全无法被识别。

同样，Meijer et al., 2021 将二进制代码转换为所谓的数据流图（DFG）。随后，优化后的子图同构检验会在数据流图中查找与密码类通用签名（结构模式）相匹配的内容。这些密码类已预先定义为分类体系，例如 Feistel网络或 (N)LFSR，其中通配符用于涵盖未知细节。再者，这种方法需要大量人工工作，因此不适合像我们的方法所追求的那样，快速识别大量与密码学相关的软件包。

结合 STATIC、KNOW 以及大语言模型要素的研究，利用大语言模型补充静态方法和知识驱动方法。Moffie et al., 2025 将解析后的代码语法树与人工整理的密码学函数列表进行比对。为了评估该方法，他们创建了一个基于大语言模型的标注数据集。通过提示预训练模型 ChatGPT-4o 和 Mistral-Large-2407 判断所提供的代码是否包含密码学内容，从而生成标签。尽管这一方法颇具前景，作者指出，大语言模型的标注质量较差，因此仍需人工核验。他们认为，通过更精确地设计提示词，可以提升大语言模型的性能。此外，该方法仅适用于 Java代码。我们想指出，我们无法弄清楚二进制代码是如何分析的（如 Moffie et al., 2025 的图 3 和图 4 所示），因为论文只阐述了对源代码的分析。此外，论文未提供项目代码，进一步妨碍了读者理解。

Shang et al., 2024 的研究在由 ChatGPT 生成的密码二进制摘要上对大语言模型进行微调，提升了二进制代码解读能力，但仍依赖预先编制的密码函数类别关键词列表。

Fan et al., 2024 的一篇综述考察了现有的 STATIC、KNOW 和大语言模型方法，这些方法用于识别密码学函数。综述指出，依赖手工构建的知识库会限制可扩展性和密码敏捷性。作者强调，编译器可变性、混淆与优化策略，以及不同的密码学算法变体或版本，都是长期存在的挑战。他们认为，大语言模型有望推动自适应启发式检测的发展。我们的工作通过使用通用大语言模型进行启发式软件包分类，直接应对了这些局限。此外，我们的方法简单直接，无需深入了解密码学或软件包——分析，因此所有用户都能轻松使用。

## 2.2 大语言模型提示技术

后续两项使用大语言模型检测代码中恶意命令的研究所提出的提示技术，显著影响了我们向大语言模型提问的方法。具体而言，Zahan等人，2025 将迭代式自我改进和“大语言模型作为评判者”框架应用于预训练模型 GPT-3 和 GPT-4。作者提出提示大语言模型以 JSON格式作答；我们发现，这种方法有助于生成高质量的回答。研究结果表明，大语言模型的表现优于基于 CodeQL 的静态分析。

Huang等人，2024 在两阶段流程中结合了静态代码分析与大语言模型推理。第一阶段使用静态代码分析，在图中描绘已知的恶意命令。第二阶段以类似方式在图中描绘新软件包，并提示大语言模型将新生成的图与第一阶段的图进行比较，然后将其分类为恶意或良性。作者采用了先进的提示设计标准，我们在很大程度上借鉴了这些标准。这些标准包括明确的任务界定、上下文依据以及基于 JSON 的输出要求，所有这些都有助于提升不同大语言模型的可解释性和可解析性。

## 2.3 协作式大语言模型推理

当无法获得明确的真实标注时，采用协作策略汇总多个大语言模型的回答是一种有效方法。因此，(Davoudi等人, 2025)通过简单的多数投票汇总多个异构大语言模型的答案；在缺少标注数据或真实标注的情况下，只要提示词清晰且没有歧义，这种方法便能提高可靠性。同样，(Li等人, 2024)表明，对多个模型的回答进行多数投票汇总，能够提升模型在事实性任务和推理任务上的表现。最后，(Choi等人, 2025)在七个自然语言基准测试中区分了多数投票与多智能体辩论，并发现仅投票就几乎带来了全部性能提升，这表明多数投票是一种稳健、可靠且易于采用的协作推理方法。

# 3 我们的贡献

针对 Fan et al., 2024 所指出的问题和未来研究方向，以及从恶意代码检

测研究（Huang等人，2024；Zahan等人，2025）中获得的见解，我们提出了一种利用预训练大语言模型识别与密码学相关的软件包的方法，填补了相关文献中的一项重要空白。为避免违反机密性，尤其是在企业环境中，我们采用离线大语言模型架构，而不是将查询发送至在线服务器。这使组织能够在本地部署环境中使用我们的方法进行密码学资产发现。为提高可靠性，我们进一步采用基于多数投票的协作式大语言模型知识策略，汇总各个大语言模型响应。我们的方法充当初步筛选器，用于在包含大量软件包和依赖项的复杂系统环境中发现密码学资产。

我们的方法具有以下几项优势：

- 它能够概述系统软件栈的密码学态势。
- 该流程仅需少量标注数据即可执行，而这些数据通常只需付出合理的努力便可获得。
- 即使本地大语言模型的准确率低于最新的 GPT 模型，该方法仍然适用，因为多数投票策略会汇总多个模型的结果，从而提高分类的整体可靠性。
- 此外，我们的方法相对容易实现，因此具有较强的实用性。

我们的方法提供由大语言模型生成的软件包信息，可用于构建软件包依赖图，并编制详尽的密码学资产清单（见第 6 节）。这有助于规划 PQC 迁移过程中的算法替换，并通过开展更有针对性的代码审查和依赖扫描予以支持。

除了介绍我们切实可行的方法外，我们还考察了该方法的合理性。在第5.1节中，我们通过统计方法分析大语言模型响应以及各模型之间的一致程度。此外，我们还对390个软件包的样本进行了人工验证，并利用该样本通过统计分析推断各个模型及多数投票的表现。

在本研究中，我们将 Fedora Linux 发行版作为复杂操作系统环境的示例加以关注。Fedora 在企业环境中被广泛使用，并提供丰富且稳定的软件包生态系统，因此是评估我们方法的理想候选对象。不过，我们的方法并不局限于 Fedora，也可适配于具有类似软件包管理系统的其他 Linux 发行版或操作系统。

基于对大语言模型响应的分析，我们制定了应用多数投票策略的指南，以确保基于大语言模型的密码资产发现的可靠性。这些指南见第 5 节和第 6 节，涵盖数据收集、大语言模型选择以及迭代过程中响应验证的最佳实践。

最后，我们将本研究使用的所有代码和数据作为开源资源提供（OTH-AMiQuaSy, 2026），以促进研究的可复现性及该领域的进一步研究。

# 4 方法论

## 4.1 方法概述

本节介绍本研究采用的迭代方法，如图1所示。在1中，我们从选定的包管理器（第4.2节）收集基础包列表，其中包含软件包或模块名称、版本、描述以及一级软件包依赖项。随后，我们将从包管理器收集的软件包详细信息填入2中创建的提示模板。借助此提示，我们向大语言模型发起查询，以判断每个软件包与密码学的相关性——此为步骤3。我们精心设计提示，运用最先进的提示工程技术（如少样本提示和指令提示），确保内容清晰且有效。提示的设计也参考了（Huang等人，2024；Zahan等人，2025）的研究发现。第4.3节将详细讨论最终的提示。

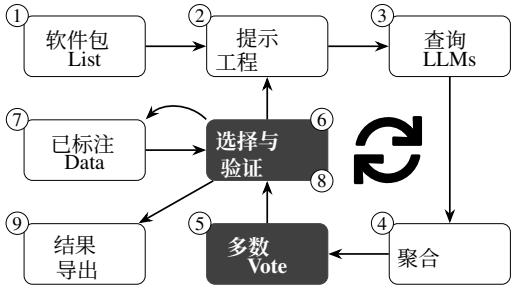

图1：密码学相关性识别方法的关键步骤。

通过纳入包管理器的详细信息为大语言模型提供必要的上下文，可从两个方面使该方法受益：

1. 软件包仓库通常是公开的，而且往往包含软件包的源代码。我们假设，正如引言中所述，这些信息已用于训练常见的大语言模型。

2. 提供描述和依赖项等额外信息，能让大语言模型对软件包有更多背景知识。即使几乎没有公开信息可用，这也能帮助它们做出更明智的判断。

创建提示后，会将其应用于$n$ 个不同的大语言模型，这些模型根据可用性和技术多样性进行选择（见第 4.3 节）。随后汇总各大语言模型的响应4 ，并5 采用多数投票策略。对于一组包含$n$ 个模型的软件包，如果有 $\lfloor \frac{n}{2} \rfloor + 1$ 个或更多模型将其分类为与密码学相关，我们就将该软件包归类为与密码学相关。第 4.4 节将详细介绍这一策略。

接下来，根据大语言模型响应的性能矩阵进行筛选。此过程会确定最终用于多数投票的大语言模型集合，并对一部分软件包进行交叉验证，以验证结果。这些步骤6 、7和8 将在第 5.1 节中说明。如果结果不理想，可以通过优化提示来迭代这一流程。这种迭代方法有助于持续提高分类准确率。最终结果9 将导出以供后续处理，例如构建首个版本的密码学物料清单（CBOM）。

此外，除了工作流本身，我们还会考察模式和相关性（第 5.1.2 节），以评估结果的合理性并发现潜在问题。整体结果见第 5 节，其中详细展示了完整评估。这说明了所选模型的准确率，并对实际应用该方法时预期可达到的质量范围进行了分类。

## 4.2 数据收集

每个软件包的基本规格信息均可公开获取，包括名称、描述和一级依赖项。这些信息对于向大语言模型提供提示、使其能够对软件包的密码学相关性作出有用判断至关重要。常见的包管理器，例如*dnf*、*apt*、*pip*，以及*npm* ，都可以从系统环境中检索这些信息。还有一些工具能够检测操作系统或软件包仓库，并提取所需信息，例如开源工具 Syft[1]。

本研究使用 Fedora Linux 发行版中的 *dnf*[2] 包管理器编制软件包列表。大多数提取的软件包来自 Fedora 和 RPM Fusion 官方仓库，但我们也纳入了来自其他来源的软件包，例如 Docker、VirtualBox 和 Visual Studio Code。我们尽可能涵盖更多软件包，以便全面了解某个特定生态系统的密码学相关性。

导出软件包列表后，我们从包名中移除了版本与架构特定信息（例如，openssl3.2.11.fc40.x86_64被映射为 openssl）。此外，软件包描述取自最新的可用版本。我们假设，就密码学相关信息而言，不同版本的软件包描述基本保持一致。不过，从运行中的系统获取软件包时，最好使用已安装版本的软件包描述。

然而，在后续的加密资产发现阶段，考虑版本详情至关重要，因为这些信息对于确定具体支持或实现了哪些算法，以及识别任何漏洞都必不可少。

总之，最终的去重列表包含 65,295 个软件包，后续步骤将用它们查询大语言模型。

## 4.3 模型使用

我们部署了五种不同的本地大语言模型，它们均可免费使用，并且能够在本地机器上运行，从而保护隐私。不过，为了获得良好性能，建议使用性能强大的 GPU；否则，推理时间可能会很长。为了便于理解——这里不详细介绍机器的规格——据撰写本文时的测量结果，在开发者笔记本上执行单次查询约需 4 分钟，而在性能强大的 DGX 机器上则需约 2 至 20 秒。有关硬件和软件配置的技术细节，请参阅第 4.6 节。

这五种大语言模型因易于获取、访问速度快且技术多样而入选。模型托管在两种不同的基础设施上：*GPT4All*[3] 和 *Ollama*[4]。*GPT4All* 是一个框架，可通过 Python API 查询各种大语言模型，而无需运行专用服务器。相比之下，*Ollama*提供服务器，支持通过 REST API 访问模型。这样的配置使我们能够充分利用这两种基础设施及其模型的优势。

对于*GPT4All* ，包含以下模型：

- GPT4All-13b-snoozy-q4_0
- Meta-Llama-3-8B-Instruct.Q4_0
- Nous-Hermes-2-Mistral-7B-DPO.Q4_0

[1]Syft：https://perma.cc/PXU5 99 2 -SR
DNF：https://perma.cc/W45V-H8QQ

[3]*GPT4All*: https://perma.cc/R3NY-S7AY [4]
*Ollama*: https://perma.cc/Y9JB-CTNQ

- Phi-3-mini-4k-instruct.Q4_0

在*Ollama*服务器上，我们使用了 DeepSeek R1，该模型支持迭代式思考过程，从而提高响应质量。

这些提示文本旨在向大语言模型提供必要的软件包信息。对于每个软件包，都会将提示中的特定占位符替换为软件包列表中的实际值，包括名称、描述和依赖项。然后，向大语言模型询问相关软件包的密码学相关性。为确保大语言模型的响应保持一致，提示要求使用特定的输出格式（JSON）。

最初，所有模型都使用相同的提示。然而，在评估过程中，我们发现修改提示能够改善某些模型的结果。因此，我们针对特定模型调整了提示。

通过这种方法，我们向大语言模型提供了明确的响应指令，并采用了单样本提示和指令提示等技术。我们还尝试了不同的方法，例如提供一个具体软件包作为单样本示例，或创建更长的示例列表。然而，我们发现，使用以描述性名称作为字段规范的输出模板进行单样本提示，能够获得最佳结果。这里所说的“最佳结果”是指大语言模型能够输出可解析的 JSON 响应，这是后续聚合步骤所必需的。总而言之，保持提示简短且精确是成功的关键。

## 4.4 聚合

我们指示所有被查询的大语言模型生成一致的 JSON 响应，然后将其解析并存储在 CSV 文件中。检索到的 JSON 对象包含三个属性：“package”、“crypto-graphic_relevance”和“justification”。需要注意的是，响应并不总是采用正确的 JSON格式。这可能由多种原因造成，例如缺少引号或括号，或者完全缺少答案字段。即使我们尝试进一步优化提示，这些问题仍然存在。因此，我们实现了一个能够处理常见错误并预判此类问题的 JSON 解析器。例如，解析器可以修正缺失或错误的引号和括号，也能处理不同模型略有差异的输出格式。不过，有时响应格式错误，完全无法解析。在这种情况下，我们会直接丢弃该响应，并在 CSV 文件中添加一行，其中包含名称，其他字段留空。在第 5 节中，我们将展示无法解析的大语言模型回答数量与可解析回答数量的对比。

要生成正确且可复现的响应，精心设计提示词和调用参数都很重要。温度、top-p 和 max_词元等参数会显著影响大语言模型的输出。在可能的情况下，我们将温度设为 0，以获得近乎确定性的输出。不过，为了增强模型间的独立性，提高温度可能会有帮助。

其他参数均保留模型默认值。具体而言，我们将 top-p 设为模型默认值（*GPT4All* 为 0.4，*Ollama*为 0.95），以保留预期的确定性与可变性之间的平衡（Arora 等，2024）。我们推测，一定程度的可变性可以降低相关性，并避免不同模型生成几乎相同的响应。这对多数投票策略尤为重要。否则，聚合过程的有效性会降低（见第 5 节）。

CSV 文件中存在少量需要处理的小问题，例如非 ASCII 字符和格式错误的条目。这些问题经过人工修正或删除，最终得到干净的 CSV 文件。最后，我们使用 CSV 检查工具验证了这些文件。每个模型对应的最终 CSV 文件都包含每个软件包的以下信息：软件包名称、密码学相关性分类（字符串“True”或“False”）以及大语言模型提供的理由。

接下来，我们将所有大语言模型的 CSV 文件合并为一个*pandas* 数据框，以便进行评估。合并后的数据框中，每个软件包对应一行，各大语言模型的分类结果对应各列。除了各模型的单独分类外，还在一个额外的列中提供了汇总判定。该汇总采用多数投票策略。如前所述，如果至少有 $\lfloor \frac{n}{2} \rfloor + 1$ 个模型对某项分类达成一致，就按该分类结果对软件包进行分类。例如，如果五个大语言模型中有三个将某个软件包分类为与密码学相关，那么该软件包的多数投票结果就为 true。

## 4.5 筛选与验证

我们选取了 390 个具有代表性的软件包作为真实标注，用于验证我们方法的结果。该样本通过从六个分层中随机抽取软件包生成，这些分层依据真实计数（范围为 0 至 5）划分。这样的分层既确保了软件包种类多样，也考虑到了不同模型之间一致程度的差异。

我们手动检查了软件包的详细信息，例如描述和依赖项，并根据其实际的密码学相关性进行标注。

我们使用已标注样本，借助性能指标对我们的方法进行迭代验证（见第5.2.1节和第5.2.3节）。此外，我们进行了交叉验证，以评估对所选模型的依赖程度以及多数投票策略的整体性能。基于该交叉验证，选出了一组模型（见第5.2.4节）。

## 4.6 实验设置

图1所述流程需要管理三个重要的技术方面：

- 软件包列表的数据收集，
- 模型执行，以查询大语言模型，以及
- 结果聚合与评估大语言模型的响应。

数据收集可以在任何能够访问所选包管理器的机器上进行，无需特殊硬件。软件要求取决于所选择的收集流程；在本研究中，我们在 Fedora Linux 发行版上使用了*dnf* 包管理器。

模型执行需要配备 GPU 的高性能机器，以便高效运行大语言模型，尤其是在处理大量软件包时。不过，如果软件包数量很少，配备轻量级 GPU 的普通机器也可能足够。尽管如此，使用非专用硬件时，预计每个软件包的干扰时间约为四到五分钟。本研究使用一台配备七块 NVIDIA A100 图形处理器的 DGX 机器运行五个大语言模型，每个模型处理约 65,000 至 70,000 条查询。

这些模型在无根 Docker 容器中运行，并通过*GPT4All*的 Python 集成进行调用；DeepSeek R1 模型则通过*Ollama*服务器提供服务。每个*GPT4All* 容器分配了一块 A100 GPU，以便并行执行；*Ollama* 服务器同样运行在一块 A100 GPU 上。对*GPT4All* 的查询通过其 Python API 直接发出，速度显著快于基于 REST 的*Ollama* 接口。因此，所有*GPT4All* 运行均在约一天内完成，而*Ollama* 服务器则耗时约三天。

结果聚合与评估 在 Jupyter Notebook 中进行，我们在数据框中处理了聚合后的 CSV 输出。除了描述性统计（见第 5.1 节）外，我们还使用一组具有代表性的软件包手动验证了结果，并优化了提示的具体内容。我们还比较了离线大语言模型与在线模型的结果（见第 5.2 节）。

# 5 结果与验证

## 5.1 大语言模型的软件包分类概述

### 5.1.1 大语言模型响应质量分析

首先，我们根据各大语言模型响应的可解析性对其进行评估。如果针对某个软件包的回答无法解析为有效的 JSON 对象，即使已考虑缺少引号或方括号等常见错误，也会将其标记为无效。请注意，内容相关的问题，例如字段缺失、属性拼写错误以及完全格式错误的回答，也会计为错误，因此视为无效。表1显示了各模型的响应质量，即有效回答的数量与错误回答的数量。进一步而言，模型规模与错误数量并无直接相关性。例如，规模最小的模型 phi（2.1 GB）产生的错误最少（72 个），而规模最大的模型 GPT4All（6.9 GB）产生的错误最多（1,137 个）。

表1：各模型对有效和无效软件包的响应

| 模型 | size | 有效 | 无效 | 错误率 |
|---|---|---|---|---|
| phi | 2.1 GB | 65,222 | 72 | 0.11% |
| DeepSeek | 5.2 GB | 65,199 | 95 | 0.15% |
| Llama | 4.4 GB | 65,094 | 200 | 0.31% |
| Mistral | 3.9 GB | 64,974 | 320 | 0.49% |
| GPT4All | 6.9 GB | 64,157 | 1,137 | 1.74% |
| agg | — | 63,529 | 1,765 | 2.70% |

需要指出的是，存在错误的软件包总数少于各单个模型之和，因为有些软件包未被多个模型作答。未作答的软件包总数为 1,765（约占全部软件包的 2.70%）。我们从数据集中剔除了大语言模型响应中至少存在一处错误的软件包，以便进一步分析。这确保了仅纳入五个模型均返回有效答案的软件包。因此，清理后的数据集共包含 63,529 个软件包。

对单个模型的考察表明，应答率仍有提升空间。例如，GPT4All 模型的未应答软件包数量最多（1,137）——错误日志显示，许多情况下 JSON 输出无法被解析。然而，其中许多

这些问题源于本可纠正的一致性格式错误。就本文的研究范围而言，我们决定不再进行进一步的错误纠正，并保留原始结果。

还可以采用其他策略。除了改进答案解析器之外，也可以重新向产生错误的软件包对应的大语言模型发送提示。这可能包括使用相同的提示、但略微提高温度来重新查询模型，以引入变化并可能避免重复失败。或者，可以修改提示以提高清晰度并减少歧义。在该方法的重新迭代过程中采用了这些策略，详见第5.2.3节。

### 5.1.2 分析数据分布

根据投票认为结果为真（与密码学相关）的大语言模型数量，对大语言模型的有效响应进行了汇总。如图2中的黑色条形所示。具体而言，$x$ 轴按一致程度分类，范围从 0 到 5。这表示针对某个特定软件包，有多少个大语言模型投票认为结果为真。$y$ 轴显示落在相应一致程度范围内的软件包数量。例如，五个模型均认为有 1,669 个软件包与密码学相关。

为了更好地理解这些结果，首先将真实投票的经验分布与二项分布相匹配。如图2中的白色条形所示，经验分布存在显著偏差。卡方拟合优度检验证明了这一点（合并第4组与第5组后频率大于5），其中 $\chi^2 \approx 612{,}668$，自由度$df = \#\text{number-of-groups} - 1 - \#\text{estimated-parameters} = 5 - 1 - 1 = 3$。在 $\chi^2$ 分布表[5], 中查找 $\chi^2$在$df = 3$ 下的临界值，我们发现计算值 $\chi^2 \approx 612{,}668$远大于显著性水平 $\alpha = 0.001$下分布表中的临界值16.266。因此，我们可以在0.1%的显著性水平上拒绝经验数据服从二项分布的原假设。换言之，随机观察到如此大偏差的概率小于0.1%。

其次，我们使用贝塔-二项分布对经验数据进行了拟合。贝塔-二项分布是一种离散概率分布，当一系列伯努利试验中的成功概率服从贝塔分布而非固定值时，就会产生这种分布。它可以对数据中的过度离散进行建模；我们的数据明显存在过度离散，因为独立同分布试验这一假设并不成立。如图2中的灰色条所示，与基本二项分布相比，贝塔-二项分布似乎能更好地拟合经验数据。

然而，卡方拟合优度检验现在给出的结果是 $\chi^2 \approx 767.63$，自由度为$df = 5 - 1 - 2 = 2$（估计了两个参数）。在与之前相同的 $\chi^2$ 分布表中查找自由度为 $\chi^2\ df = 2$ 对应的临界值后，我们发现，显著性为 $\alpha = 0.001$ 时的临界值 13.82 仍相差甚远。

β-二项分布的两个参数不足以充分刻画实际投票行为；数据仍存在显著的过度离散。这表明，模型投票为真的概率$p$ 并非常数——无论是在不同大语言模型之间，还是在不同软件包之间，都是如此。

持续存在的过度离散意味着二项框架的独立性假设并不成立，这反映了不同大语言模型、软件包之间或二者交互作用所导致的潜在变异性。下一节将详细探讨这些依赖项。

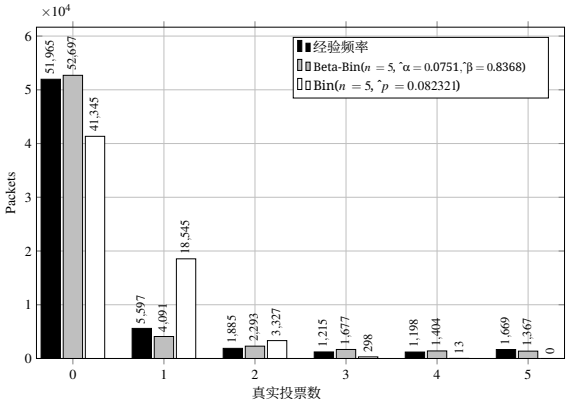

图2：按大语言模型中的真实投票数划分的软件包分布，并与经验分布、二项分布和贝塔二项分布进行比较。

### 5.1.3 大语言模型的独立性假设

要使多数投票策略有效，各个模型必须尽可能相互独立。然而在实践中，这一假设未必成立。模型的回答可能呈现相关性（Kim 等人，2025），原因有以下几种：

- 共享的训练数据来源
- 相似的架构和目标
- 基于人类反馈进行调优
- 共用基准测试和评估循环
- 语言和概念规律
- 大规模训练中的模型平均效应

[5] $\chi^2$ 分布表：https://perma.cc/P2UC-U63Q

- 系统性偏差

为了理解模型之间的依赖项，我们引入设计效应和有效样本量的概念。设计效应（Kish，1965；Donner 和 Klar，2000）表示观测值之间的相关性高于相互独立时，抽样方差的增加幅度。对于一个大小为$n$的簇，若观测值之间的平均类内相关系数为 ρ，则设计效应 $D_{eff}$ 的计算方式如下：

$$D_{eff} = 1 + (n-1)\cdot\rho \tag{1}$$

有效样本量（ESS）定义为：能够提供与相关样本相同精确率的等效独立样本的规模。可通过设计效应按以下方式计算：

$$n_{eff} = \frac{n}{D_{eff}} \tag{2}$$

由于类内相关系数 ρ 尚未定义，我们可以采用近似值。根据（Donner 和 Klar，2000），设计效应可通过观测到的 $Var_{obs}$ 与独立方差 $Var_{ind}$ 之间的关系进行估算：

$$D_{eff} = \frac{Var_{obs}}{Var_{ind}} \tag{3}$$

鉴于我们有五个模型（$n = 5$）、$Var_{ind}$ =0.3777（基于 $Bin(n = 5, \hat{p} = 0.082321)$）以及 $Var_{obs} = 1.17$ （经验数据），我们可以按以下方式估算设计效应和有效样本量：

$$D_{eff} = \frac{1.17}{0.3777} \approx 3.096 \tag{4}$$

$$n_{eff} = \frac{5}{3.096} \approx 1.615 \tag{5}$$

这表明，尽管使用五个模型评估每个软件包，综合响应所提供的信息量相当于约**1.6 次独立评估**。换言之，模型输出呈中度至高度相关（$\rho \approx 0.52$），反映出模型之间存在显著的共同偏差或共识。

## 5.2 结果验证

### 5.2.1 人工评估

如第 5.1.2 节所示，各大语言模型中的真实投票并非均匀分布。因此，有些软件包似乎比其他软件包更难分类。为确保人工评估涵盖所有分类难度等级，我们采用了分层抽样方法。对于从 0 到 5 的每个一致性等级（见图2），随机抽取 65 个软件包的样本，共计 390 个软件包。人工评估通过审查软件包描述和依赖项进行。必要时，我们还审查了源代码和文档。根据这些信息，我们确定了每个软件包的密码学相关性。对于这 63,529 个软件包，390 个软件包的样本量基于 95% 的置信水平和 5% 的误差范围。

表2：所选大语言模型在 Fedora 安全相关性分类中的性能指标。

| 模型 | Acc | Spec | Prec | 召回率 | F1 |
|---|---|---|---|---|---|
| 多数 | 0.73 | 0.77 | 0.74 | 0.70 | 0.72 |
| Llama | **0.76** | 0.69 | 0.72 | 0.83 | **0.77** |
| phi | 0.72 | 0.54 | 0.65 | **0.92** | 0.76 |
| Mistral | 0.75 | 0.87 | 0.82 | 0.62 | 0.70 |
| DeepSeek | 0.67 | 0.71 | 0.67 | 0.64 | 0.66 |
| GPT4All | 0.67 | **0.94** | **0.86** | 0.38 | 0.53 |

表2展示了所选本地模型和多数投票（多数）策略的性能指标，包括准确率（Acc）、特异度（Spec）、精确率（Prec）、召回率和F1分数。在这一轮初始实验中，模型和多数投票策略的表现都不太理想。表现最佳的模型是Llama，其F1分数为77%，优于多数投票策略。这些结果表明，模型的分类性能仍有很大的提升空间。

总之，我们发现，多数投票会将性能平均化。虽然这能减轻表现较弱模型的影响，但也会抑制表现较好模型的优势。

### 5.2.2 外部模型验证

为了将本地模型的结果与最先进的云端模型进行比较，我们使用相同的390个软件包样本集，向知名供应商提供的四个大语言模型发起了查询。其中包括 OpenAI 的 GPT-5、Google 的 Gemini 2.5 Flash 和 2.5 Pro，以及 Mistral AI 的 Codestral-2508 模型。

按照上一节所述完成初步评估后，我们将这390个软件包视为评估在线模型的真实值。我们采用了相同的提示和多数投票策略，并计算了相同的性能指标。结果汇总于表3。

该表显示，在线模型的表现优于本地模型，但在F1分数方面仅高出3%至9%（表现最好的本地模型是Llama，其

表3：所选在线模型的性能指标。

| 模型 | Acc | Spec | Prec | 召回率 | F1 |
|---|---|---|---|---|---|
| 多数 | **0.86** | 0.84 | 0.84 | 0.87 | **0.86** |
| OpenAI | 0.84 | 0.77 | 0.79 | 0.91 | 0.85 |
| Gemini Flash | 0.84 | 0.73 | 0.77 | 0.94 | 0.85 |
| Gemini Pro | 0.81 | 0.65 | 0.72 | **0.97** | 0.83 |
| CodeStal | 0.82 | **0.90** | **0.88** | 0.74 | 0.80 |

F1分数为77%）。此外，多数方法似乎在在线模型上比在本地模型上效果更好。由于我们的方法出于隐私原因更倾向于本地大语言模型，我们仅将在线模型用于外部模型比较。

### 5.2.3 重新迭代与提示优化

第5.2.1节中通过人工验证生成的真实标注被用于执行图1所示的迭代过程。首先，我们改进了解析逻辑，使其能够容忍常见的格式错误。其次，如果响应无法正确解析，我们会提高温度并重新查询模型。第三，我们根据第一轮评估的结果，针对每个本地模型优化提示词。这些优化包括调整措辞、提供更清晰的说明，以及量身定制提示词，使其更好地契合相应模型的优势。较大的模型（如 DeepSeek）得益于更长、更详细的提示词；而较小的模型（如 phi）则更适合较短、更精简的提示词。

表4：提示优化后所选本地模型的性能指标。

| 模型 | Acc | Spec | Prec | 召回率 | F1 |
|---|---|---|---|---|---|
| 多数 | **0.85** | 0.75 | 0.78 | **0.95** | **0.86** |
| DeepSeek | **0.85** | 0.88 | **0.87** | 0.81 | 0.84 |
| Llama | 0.79 | 0.80 | 0.79 | 0.78 | 0.78 |
| phi | 0.72 | 0.55 | 0.65 | 0.90 | 0.76 |
| Mistral | 0.78 | 0.91 | **0.87** | 0.64 | 0.74 |
| GPT4All | 0.67 | **0.94** | 0.86 | 0.39 | 0.53 |

如表4所示，提示优化显著提升了本地模型的性能。多数投票策略优于所有单个模型，F1分数达到86%。目前表现最佳的单个模型是 DeepSeek，F1分数为84%，其次是Llama，为78%。这些结果表明，提示工程在提升大语言模型针对特定任务（如密码学相关性识别）的性能方面发挥着关键作用。离线生成的多数投票目前已可与在线模型相媲美（见表3）。

### 5.2.4 最终优化

最后，基于前几节的研究结果，我们旨在进一步优化多数投票。将表达式$D_{\text{eff}}$代入公式(2)，并使用估计的类内相关性$\rho \approx 0.52$，可得到有效样本量的如下表达式：

$$n_{\text{eff}} = \frac{n}{1+\frac{(n-1)}{2}} = \frac{2n}{n+1} \text{ with} \tag{6}$$

$$\lim_{n\to\infty} \frac{2n}{n+1} = 2 \tag{7}$$

随着集成中添加更多模型，有效样本量仅略有增加。使用三个模型时，有效样本量约为 1.5；使用五个模型时，也仅上升到约 1.67。此后曲线趋于平缓，并接近 2 的最大值。因此，使用三个至五个模型即可获得大部分潜在收益（给定 $\rho \approx 0.52$）。除非新增模型之间具有高得多的独立性，否则添加更多模型几乎没有益处，而在实践中很难实现这一点（参见第5.1.3节）。

回到图1所示的方法，验证步骤首先确定用于最终多数投票策略的三个最佳模型。在本研究中，这是通过从五个本地模型中选取三个模型的所有可能组合并进行评估来实现的。每种组合的性能指标均根据第5.2.1节确立的真实标注计算得出。这可以通过多种方式完成，例如针对 F1分数、召回率进行优化，或使用自定义加权指标。

接下来，我们将选择范围缩减为五个模型中的三个，这仅使$n_{\text{eff}}$ 略有下降，但允许模型集成剔除表现不佳的模型，同时确保有足够的模型形成稳定的多数投票。为聚焦于密码学相关软件包，我们采用加权方法，优先考虑由$r$ 表示的召回率，而非由$s$表示的特异度。因此，我们针对如下定义的得分进行了优化：

$$\text{Score} = w_{\text{r}} \cdot r + w_{\text{s}} \cdot s \tag{8}$$

权重为$w_{\text{r}} = 0.7$ 和$w_{\text{s}} = 0.3$。这一选择体现了正确识别密码学相关软件包（高召回率）的重要性，同时仍保持识别非相关软件包时合理的准确率水平（特异度）。该优化过程中找到的模型为 DeepSeek、phi和Mistral。

为估算多数投票集成的内部泛化性能，我们在带标签的验证集上进行了分层五折交叉验证（CV）。在五次迭代中的每一次，

都会在训练子集上选出最优三模型集成，随后在留出折上进行评估。表5汇总了所有折的平均指标。最优组合根据上述加权得分计算得出。通过交叉验证，该集成的平均 F1分数达到 0.82，召回率始终较高（0.85），且精确率与特异度较为均衡。这表明其在不同数据划分中表现稳定。

表5：多数投票集成的五折交叉验证结果。报告各折的均值$\mu$ 和标准差 σ 。

| 指标 | Acc | Spec | Prec | 召回率 | F1 |
|---|---|---|---|---|---|
| $\mu$ | 0.82 | 0.79 | 0.79 | 0.86 | 0.82 |
| σ | 0.056 | 0.046 | 0.050 | 0.075 | 0.059 |

各折之间的差异较小，表明该集成方法表现稳定，不会过度受到特定数据子集的影响。尽管标注数据数量有限（390 个样本），交叉验证仍是评估这些结果可靠性的实用方法。总体而言，结果表明该集成方法具有良好的内部泛化能力。然而，这些结果仍反映的是内部验证，而非在独立测试集上的评估。

## 6 讨论

第 5 节的结果表明，采用多数投票策略集成本地大语言模型，能够有效识别与密码学相关的软件包。要取得这些结果，几项优化至关重要。首先，提示词工程发挥了重要作用。根据每个模型的优势量身定制提示词，并使其保持简洁，提升了响应质量。其次，改进解析逻辑以处理常见格式错误，从而能够使用更多有效响应。第三，根据模型互补的优势（例如较高的召回率或特异度）来选择模型，提升了多数投票的整体性能。

研究结果还表明，更大的模型并不一定能产生更好的结果。例如，规模最小的模型 phi 在召回率方面优于更大的模型。这表明，仅凭模型规模并不能可靠地反映特定任务的性能，例如识别密码学相关软件包。即便是来自在线提供商的模型，也并未始终优于本地模型，这说明经过适当优化后，本地大语言模型可以具备竞争力。

另一个关键因素是模型的独立性。如第5.1.3节所述，模型之间相关的响应会削弱多数投票的有效性。因此，选择具有多样化架构和训练数据的异构模型有助于缓解这一问题。

本研究采用基于验证精心筛选的带标签数据集的迭代方法，这对于完善初步结果至关重要。通过依据性能指标持续评估和优化提示词、解析逻辑及模型选择，我们显著提升了分类准确率。

然而，这种方法也有其局限性。依赖软件包描述和依赖项，可能会导致对公开信息稀少或含糊的软件包进行错误分类。此外，虽然用于验证的人工评估较为全面，但仍可能无法涵盖所有与密码学相关的细节。

## 7 结论与展望

总而言之，本研究表明，经过适当优化并结合使用后，本地大语言模型能够有效识别与密码学相关的软件包。我们的方法可帮助组织分析其软件技术栈的密码学相关性，同时借助本地部署的离线大语言模型框架保护隐私。研究结果凸显了提示工程、模型独立性和迭代优化方法的重要性。未来的工作将着力进一步改进提示策略，并扩展检测方法，以进一步提取特定的密码原语并生成全面的密码学物料清单。我们正在开发一个原型，将本方法集成到更广泛的密码资产发现框架中，并将通过 GitHub 发布——参见 (OTH-AMiQuaSy, 2026)。

## 参考文献

Anthropic（2023）。Claude 模型的模型卡与评估。Claude 模型使用“互联网上公开可用的信息”、授权数据和人工提供的数据混合训练。Arora, C., Sayeed, A. I., Licorish, S. A., Wang, F., 和 Treude, C.（2024）。优化大语言模型以生成代码：哪些超参数设置能带来最佳结果？载于第*31*届亚太软件工程会议论文集（*APSEC 2024*），第281–290页，中国重庆。IEEE。

Choi, H. K., Zhu, X., 和 Li, Y.（2025）。辩论还是投票：哪种方式能让多智能体大语言模型做出更好的决策? https://arxiv.org/abs/2508. 17536。代码：https://github.com/deeplearning-wisc/debate-or-vote。网络安全与基础设施安全局、美国国家标准与技术研究院和美国国家安全局（2023）。量子就绪：向后量子密码学迁移。建议组织将密码系统和资产列入清单，作为首要步骤。Davoudi, S. P. M., Fard, A. S., 和 Amiri-Margavi, A.（2025）。大语言模型间的集体推理：一种无需真实标注即可验证答案的框架。https://arxiv.org/abs/2502.20758。Donner, A. 和 Klar, N.（2000）。健康研究中的整群随机试验设计与分析。Arnold，伦敦。Fan, Y., Biswas, P., 和 Garman, C.（2024）。R+r：二进制文件中密码函数识别方法的系统性研究。载于*2024*年度计算机安全应用大会（*ACSAC*），第1092–1108页。ISSN：2576-9103。Google DeepMind（2025）。Gemini 2.5 pro——模型卡。预训练数据集包含“公开可用的网络文档”。Housley, R.（2015）。密码算法灵活性及选择强制实施算法的指南。BCP 201（RFC 7696）。Huang, Y., Wang, R., Zheng, W., Zhou, Z., Wu, S., Ke, S., Chen, B., Gao, S., 和 Peng, X.（2024）。SpiderScan：基于图的行为建模与匹配，实用地检测恶意 npm 软件包。载于第*39*届*IEEE/ACM*自动化软件工程国际会议论文集，ASE '24，第1146–1158页。计算机协会。IBM Research（2025）。Cbomkit：用于处理密码物料清单（CBOM）的工具集。https://github.com/IBM/cbomkit。Kim, E. M., Garg, A., Peng, K., 和 Garg, N.（2025）。大语言模型中的相关错误。载于第*42*届国际机器学习会议论文集（*ICML 2025*）。Kirchhoff, M. 和 Pavkovic, R.（2025）。组合并量化大语言模型以提升性能。Inovex GmbH，检索日期：2025-09-08。Kish, L.（1965）。抽样调查。John Wiley & Sons，纽约。Leirimaa, K.（2024）。通过自动生成密码物料清单支持后量子密码迁移和密码灵活性。硕士论文，于韦斯屈莱大学。https://jyx.jyu.fi/jyx/Record/jyx_123456789_98042。Li, J., Zhang, Q., Yu, Y., Fu, Q., 和 Ye, D.（2024）。更多智能体就是你所需要的一切。http://arxiv.org/abs/2402.05120。Loebenberger, D., Gazdag, S.-L., Herzinger, D., Hirsch, E., Näther, C., 和 Steghöfer, J.-P.（2025）。密码迁移的形式化。Meijer, C.、Moonsamy, V. 和 Wetzels, J.（2021）。密码学在哪里？二进制代码中专有密码原语的自动识别与分类。载于第 *30* 届 *USENIX* 安全研讨会论文集（*USENIX Security 2021*），第 555–572 页，加拿大不列颠哥伦比亚省温哥华。USENIX 协会。https: //perma.cc/HZJ2-EVZ8。Meta AI（2024）。Meta Llama 3 — 8B 指令版：模型卡。“Llama 3 使用来自公开可用来源的超过 15 万亿词元数据进行了预训练。”Moffie, M.、Boehm, O.、Koyfman, A.、Bin, E.、Sztokman, E.、Bhattacharjee, S.、Saha, M. 和 McGugan, J.（2025）。Cryptoscope：分析现代软件中的密码学应用。http://arxiv.org/abs/2503.19531。Mosca, M. 等（2024）。2024 年量子威胁时间线报告。技术报告，全球金融服务风险研究所。Näther, C.、Herzinger, D.、Steghöfer, J.-P.、Gazdag, S.-L.、Hirsch, E. 和 Loebenberger, D.（2025）。迈向对密码敏捷性的共同理解——系统综述。OpenAI（2023）。GPT-4 技术报告。*arXiv* 预印本 *arXiv:2303.08774*。报告指出，GPT-4 使用“公开可用数据（例如互联网数据）”和授权数据进行了预训练。OTH-AMiQuaSy（2026）。detecting-crypto-packages：根据名称、描述和一级依赖项检测密码学软件包。https://github.com/OTH-AMiQuaSy/detecting-crypto-packages。GitHub 代码库，访问于 2026 年 1 月 9 日。Ponemon Institute 和 Entrust（2024）。2024 年零信任与加密研究。技术报告，Ponemon Institute / Entrust。44% 的受访者无法全面了解其密码学资产；43% 的受访者表示缺乏清单或发现能力。Rattanavipanon, N.、Suaboot, J. 和 Werapun, W.（2025）。用于协助软件可执行文件迁移至后量子密码学的工具链。*IEEE Access*，13：4368–4380。https://doi.org/10.1109/ACCESS.2024.3524309。Schmitt, N.、Henrich, J.、Heinz, D.、Alnahawi, N. 和 Wiesmaier, A.（2024）。密码学清单的标准与工具。载于 *Sicherheit 2024*，第 P-345 卷，第 49–63 页，德国沃尔姆斯。德国信息学会。https://doi.org/10. 18420/sicherheit2024_003。_Shang, X.、Chen, G.、Cheng, S.、Zhang, Y.、Zhang, W. 和 Yu, N.（2024）。FoC：使用大语言模型在剥离二进制文件中找出密码学函数。http://arxiv.org/abs/2403.18403。Wind River Systems（2020）。Crypto detector：密码学检测工具。GitHub 代码库；Apache-2.0 许可证。Zahan, N.、Burckhardt, P.、Lysenko, M.、Aboukhadijeh, F. 和 Williams, L.（2025）。利用大语言模型检测 npm 恶意软件包。载于*2025 IEEE/ACM* 第 *47* 届国际软件工程大会（*ICSE*），第 2625–2637 页。https: //doi.org/10.1109/ICSE55347.2025.00146。