基于 LLM 的恶意软件分析的语义预处理

Benjamin Marais, Tony Quertier, and Grégoire Barrué

Orange Innovation, Cesson-Sevigne, France
benjamin.marais@orange.com
tony.quertier@orange.com
gregoire.barrue@orange.com

Abstract. 在恶意软件分析的背景下，许多方法依赖于人工智能来处理大量数据。然而，这些技术侧重于数据视图（图像、序列），而不是专家的观点。注意到这一问题，我们提出了一种专注于专家知识的预处理方法，以提高恶意软件语义分析和结果解释性。我们提出了一种新的预处理方法，为可移植执行文件创建 JSON 报告。这些报告汇集了静态和行为分析中的特征，并结合了打包签名检测、MITRE ATT&CK 和恶意软件行为目录（MBC）的知识。该预处理的目的是收集二进制文件的语义表示，以便恶意软件分析师理解，并能够增强用于恶意文件分析的人工智能模型的解释性。使用这种预处理方法来训练一个大型语言模型进行恶意软件分类，在一个代表业务现实的复杂数据集上实现了 0.94 的加权平均 F1 分数。

Keywords: 恶意软件分析 · 数据预处理 · 大型语言模型 · 网络安全。

介绍

虽然人工智能现在是恶意软件检测和恶意软件分类 [7] 领域中的常用工具，但模型的训练仍然是主要问题之一，因为数据表示的问题。一些研究试图通过从样本中提取更多信息来解决这个问题，比如 EMBER[2]，而其他研究则尝试改变文件的表示形式，例如将它们转换为图像 [18]，以便训练卷积神经网络，或者直接分析二进制文件的字节序列 [23]。与其它特征提取方法 [26] 一样，EMBER 为每个样本提供了大量特征。然而，要理解哪些是最具影响力的特性可能很困难 [19]。还有其他特征提取技术 [6]，其中一些结合了静态与行为分析 [32]，但它们似乎都存在难以理解的问题，因此无法提供某些结果的可解释性。请注意 EMBER 最近发布了一个新版本，在 PE 文件格式中提供了更详细的报告 [9]。

这种缺乏相关表示导致恶意软件类别的分类不佳，以及结果解释性差。我们需要在这些报告中添加更多上下文信息。目标是利用大型语言模型的摘要能力从报告中提取最重要信息。事实上，大型语言模型（LLM）为许多领域提供了新的机会，而恶意软件检测和分类显然是其中之一。这些模型学习语义报告的能力与注意力机制相结合非常有用。例如，使用 LLMs 尝试理解代码 [28]，甚至拆解混淆的代码 [25]。

本文旨在为 PE 文件特征提取提供一种新的预处理方法，该方法结合了静态和行为分析以创建包含详细信息的 PE 文件样本 JSON 报告。我们通过使用此预处理方法来借助 LLM 模型对恶意软件类别进行分类，展示了其相关性。实际上，这种分类任务比通常的检测任务更为复杂，一些研究尝试例如使用图像分类 [12]，或从 EMBER 提取的特征 [9]，[13]。虽然 EMBER 为恶意软件分类提供了一组标准化的静态 PE 特征，但它并未包含任何源自已知基于规则的威胁签名的上下文信息。我们的方法通过整合打包器签名检测和 YARA 规则匹配来补充现有的数据集（[8]，[10]，...）。这些额外的特性旨在捕捉更高层次的语义信号并增强可解释性，这对于威胁搜索和高级恶意软件分类至关重要。我们预处理的一个优点是报告可以直接由分析师理解，因为它们包含具体信息。

本文组织如下。在第 1 节中，我们介绍了我们的数据集、文件格式以及用于创建报告的预处理方法。然后，在第 2 节中汇总了我们使用报告进行实验的信息，通过训练两个大型语言模型来执行分类任务，并详细说明了所获得的结果。最后，我们总结了使用完整的预处理过程来训练 LLM 的优势并讨论未来的工作。

1 方法论

在本节中，我们通过描述不同的基本模块来介绍论文的方法。首先，我们详细说明了 PE 格式和我们在工作中使用的数据集。然后，我们描述了为我们的实验提出的预处理方法，以及我们如何从二进制文件中提取所选特征。

1.1 文件格式和数据集

可移植可执行格式 恶意文件仍然是重要的网络安全威胁，在 2024 年占成功网络攻击的 60%以上 [20,11]。其中，基于可移植可执行文件（PE）格式的恶意软件占流通恶意文件的 95% 以上 [4]，并且通常被威胁行为者用于隐藏和传播恶意载荷以及实施漏洞利用 [14]。

PE 格式是从 UNIX 系统主要使用的 COFF（Common Object File Format，通用目标文件格式）继承而来的文件格式。该格式由 Microsoft 于 1993 年创建 [16]，用于 Windows 操作系统文件。它主要用于 DLL 文件 (.dll) 和可执行文件 (.exe)。如图 1 所示，PE 文件分为两部分：头部和节。头部描述了文件及其内容。它包含诸如文件创建日期、加载到内存的信息以及节的数量等信息。每个节都由一个特定的头部描述，其中包括其名称、大小和在虚拟内存中的位置。这些节通常包含可执行代码 (.text)、使用的变量及其默认值 (.data) 或用于执行文件所需的函数 (DLLs) 的信息 (.idata)。

通过研究文件头和各节中包含的信息以及其行为（和目的），可以判断一个文件是否可能具有恶意。

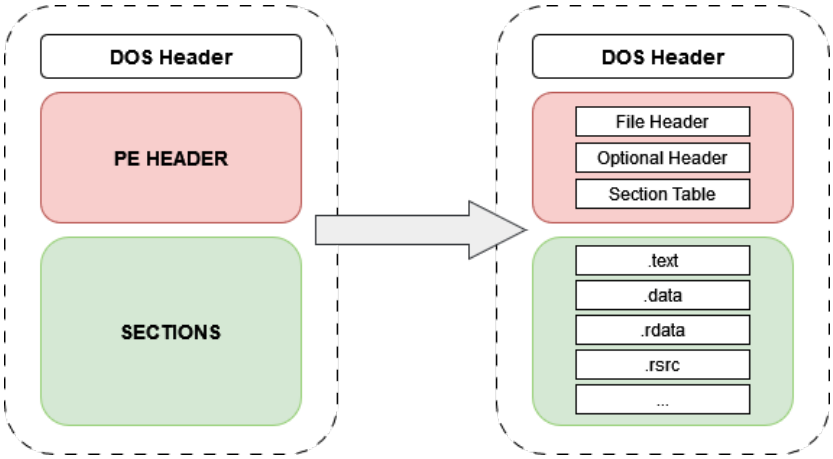


Fig. 1. PE 文件的简化架构

数据集 用于我们工作的文件来自一个开源数据集，该数据集在线上可以获取。我们选择了 BODMAS 数据集 [30] 作为恶意文件的来源。这个数据集的一个特点是，文件类型（或类别）由作者提

供的元数据描述。在这个数据集中，我们隔离出了 8 个类别，这些类别的样本数量最多，其他类别则没有足够的样本来训练一个分类模型。不同的类别包括：特洛伊木马、蠕虫、勒索软件、后门、信息窃取者、下载器、投放器和病毒。它们的区别在于感染计算机后的行为；例如，勒索软件会加密计算机的数据以迫使用户支付赎金，而特洛伊木马则可以隐藏其恶意行为以从受感染的计算机中窃取数据。

我们选择使用一个相对较大的数据集（约 57,000 个样本）相比其他一些工作 [18,24]，以便确保我们的模型能够得到良好的训练。这涉及到代表性不足的恶意软件类别，因为有时候找到某些类型的恶意软件比较困难，而找到另一些类型则容易得多。但依我们看来，这样可以拥有不同类别的实际分布情况。此外，如果我们想要有一个完全平衡的数据集，我们就不得不大幅减少数据集的大小，从而降低训练的质量。

Table 1. BODMAS 的恶意软件重新分配

File Categories	Number of instance
Trojan	29,972
Worm	16,697
Backdoor	7,331
Downloader	1,031
Ransomware	821
Dropper	715
Infostealer	448
Virus	192

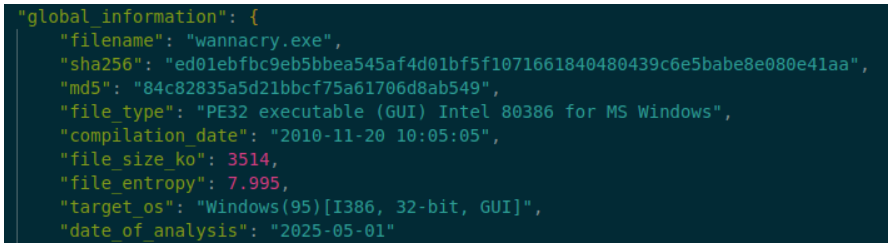
1.2 预处理和特征描述

基于我们的恶意软件分析经验、专家知识，并受 EMBER [2] 的启发，我们提出以下预处理步骤，将二进制文件转换为适合与机器学习或深度学习模型一起使用的便捷形式。我们从文件中提取大量用于静态恶意软件检测和分析的信息，并将其格式化为 JSON 文件的报告形式。如同 EMBER 一样，我们的重点主要在于提取二进制文件中的静态特征，因为这些是该领域恶意软件分析师使用的主要特性。然而，我们也基于规则或签名来提取信息以丰富我们的 JSON 报告。这些规则或签名提供了行为（MITRE ATT&CK 和 MBC）或混淆（打包）信息。在本节的剩余部分中，我们将详细描述从二进制文件中提取的特征。对于 JSON 报告的每个子部分，我们都会提供 Wannacry 勒索软件变种 [17] 的示例，其 sha256 是 *ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa*。

全局信息 JSON 报告的第一部分提供了分析文件的全局信息，包括文件名、签名（sha256, md5）、文件类型、目标操作系统、编译日期、文件大小和文件熵。这些信息描述了文件，例如奇怪的编译日期或高的熵值可以是可疑行为的指标。然而，这不能用来决定一个文件是否为良性文件或恶意软件，需要结合下面解释的所有信息来判断。

章节信息 报告的第二部分提供了与文件各节相关的信息。

在 PE 文件中，节表描述了程序在内存中的组织方式。每个节包含运行时或链接过程中所需的数据或代码。一个常见的节包括。文本（可执行代码）、.数据（已初始化数据）、.数据区（只



```
"global_information": {
    "filename": "wannacry.exe",
    "sha256": "ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa",
    "md5": "84c82835a5d21bbcf75a61706d8ab549",
    "file_type": "PE32 executable (GUI) Intel 80386 for MS Windows",
    "compilation_date": "2010-11-20 10:05:05",
    "file_size_ko": 3514,
    "file_entropy": 7.995,
    "target_os": "Windows(95)[I386, 32-bit, GUI]",
    "date_of_analysis": "2025-05-01"
```


Fig. 2. 全局文件信息

读数据）和. 资源（如图标或对话框等资源）。每个节具有一组特性，其中有一个标志描述了该节的属性，例如它是否可执行、可读、可写或包含代码，具体参见表 2。

Table 2. 某些部分特征

Flag Name	Description
MEM EXECUTRE	Section contains executable code
MEM WRITE	Section is writable
MEM READ	Section is readable.
CNT CODE	Section contains code
CNT INITIALIZED DATA	Section contains initialized data

基于节分析进行恶意软件分析是很常见的 [21,29]。例如，更改节名称是一种常用的规避检测或阻碍逆向工程的技术 [1,22]。另一方面，某些节名称是如 UPX 这样的逃避工具的特征 [1]，这些工具将原始代码和数据压缩到自定义节中，比如. 上 *x1* 或. 上行 *2*。此外，恶意软件可能会给一个节分配不寻常的特点，例如使. 资源可执行，这可能表明该节中隐藏了恶意负载。这些命名和特征上的异常是警示信号，并且常常指示使用打包、混淆或代码注入技术来绕过反病毒引擎或加强逆向工程。

在我们的 JSON 报告中，如图 3 所示，我们保留了包括名称、大小、签名（sha256）和各部分特征在内的各项内容。

导入的函数 导入地址表（IAT）是 PE（便携可执行）文件中的一个关键结构，它列出了程序依赖的所有外部函数，通常来自 Windows DLL，如 kernel32.dll、user32.dll 或 advapi32.dll。

IAT 中的每个条目对应于一个将在运行时由 Windows 加载器解析并加载的函数，并指向相关 DLL 中该函数的实际内存地址。

IAT 包含导入函数的名称以及它们所在的 DLL。

在静态分析期间，IAT 为二进制文件的行为提供了有价值的见解。例如，通过序号而不是名称进行导入可能是可疑的 [3]，因为这是一种常见的逃避检测或混淆恶意意图的策略。分析师还寻找导入函数中的恶意模式，如通常用于代码注入或进程空洞的 API 调用等。

包含的一系列导入强烈暗示该二进制文件可能执行表 3 中描述的一种操作。

因此，在静态恶意软件分析中仔细审查 IAT 是识别潜在威胁的关键步骤。

在下一节中，我们提取与 IAT 相关的信息。如图 4 所示，我们首先给出 imphash，这是一个与导入函数相关的文件签名，以及清楚和按序号导入的函数的数量。然后我们按照库（例如 ker-

[1] https://upx.github.io/

```
"section_information": {
    "number_of_sections": 4,
    "sections_desscription": [
        {
            "name": ".text",
            "sha256_section": "7e02fab4c69f500a381269cf00d273702d910366a6cb8979a0def828
            "id_section": 0,
            "entropy": 6.598759307021768,
            "file_ratio": 0.007698681526806527,
            "raw_address_start": "0x00001000",
            "raw_address_end": "0x000079B0",
            "section_size": 27056,
            "virtual_address": "0x00001000",
            "virtual_size": 27056,
            "characteristics": [
                "CNT_CODE",
                "MEM_EXECUTE",
                "MEM_READ"
            ]
        },
        {
            "name": ".rdata",
            "sha256_section": "7a7c8335535965af42e8e56457d63e07666afef48208f156fd4e81f5
            "id_section": 1,
            "entropy": 6.691364173966162,
            "file_ratio": 0.006952032342657343,
            "raw_address_start": "0x00008000",
            "raw_address_end": "0x0000DF70",
            "section_size": 24432,
            "virtual_address": "0x00008000",
            "virtual_size": 24432,
            "characteristics": [
                "CNT_INITIALIZED_DATA",
                "MEM_READ"
            ]
        },
        {…
        },
        {…
        }
    ]
```


Fig. 3. 文件部分信息

Table 3. 一些来自 [31] 的危险 Windows API 调用

Exploit	Windows API Calls
Code injection	CreateRemoteThread, OpenProcess, VirtualAllocEx, WriteProcessMemory, EnumProcesses
Dynamic DLL loading	LoadLibrary, GetProcAddres
Memory scrapping	CreateToolhelp32Snapshot, OpenProcess RadProcessMemory, EnumProcess
Unpacking/self-injection	ViruatlAlloc, VirtualProtect
Execute a program	WinExec, ShellExecute, CreateProcess
Query artifact	CreateMutex, CreateFile FindWindow, GetModuleHandle, RegOpenKeyE

nel32.dll）排序提供所有导入函数的列表。如果一个函数是通过序号导入的，我们会尽可能检索其名称。

```
"imports_information": {
    "imphash": "68f013d7437aa653a8a98a05807afeb1",
    "number_of_imports": 114,
    "number_of_ordinals": 0,
    "has_no_imports": false,
    "imports": [
        {
            "libname": "kernel32.dll",
            "imports": [
                {
                    "impname": "getfileattributesw",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "getfilesizeex",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "createfilea",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "initializecriticalsection",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "deletecriticalsection",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "readfile",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
                {
                    "impname": "getfilesize",
                    "is_ordinal": false,
                    "ordinal_value": 0
                },
```


Fig. 4. IAT 信息

打包签名 第四部分，如图 5 所示，与混淆有关，特别是当文件被压缩时。压缩代码是威胁参与者和恶意软件编辑者常用的一种技术，用于隐藏恶意负载并禁用或防止静态分析 [27]。即使这是一种出于知识产权原因由软件编辑者使用的技巧，它也是对可疑恶意文件分析的一个指示器。

我们使用开源工具如 DIEC[2]、PEid[3] 或 PyPackerDetecte[4] 来检测打包工具的签名。基于这些不同的工具，我们提取潜在打包程序的签名及其他可能打包的迹象。我们定义打包标签 l_p 如下：

$$l_p = \frac{\sum_{i=1}^{n} w_i l_{pi}}{\sum_{i=1}^{n} w_i} \tag{1}$$

其中，$l_{pi} \in \{0,1\}$ 是每个打包检测工具返回的打包标签，如果工具检测到打包签名或打包指示，则为 $l_{pi} = 1$。每个子标签 l_{pi} 都根据工具在打包检测方面的效率定义的一个权重 w_i 进行加权。

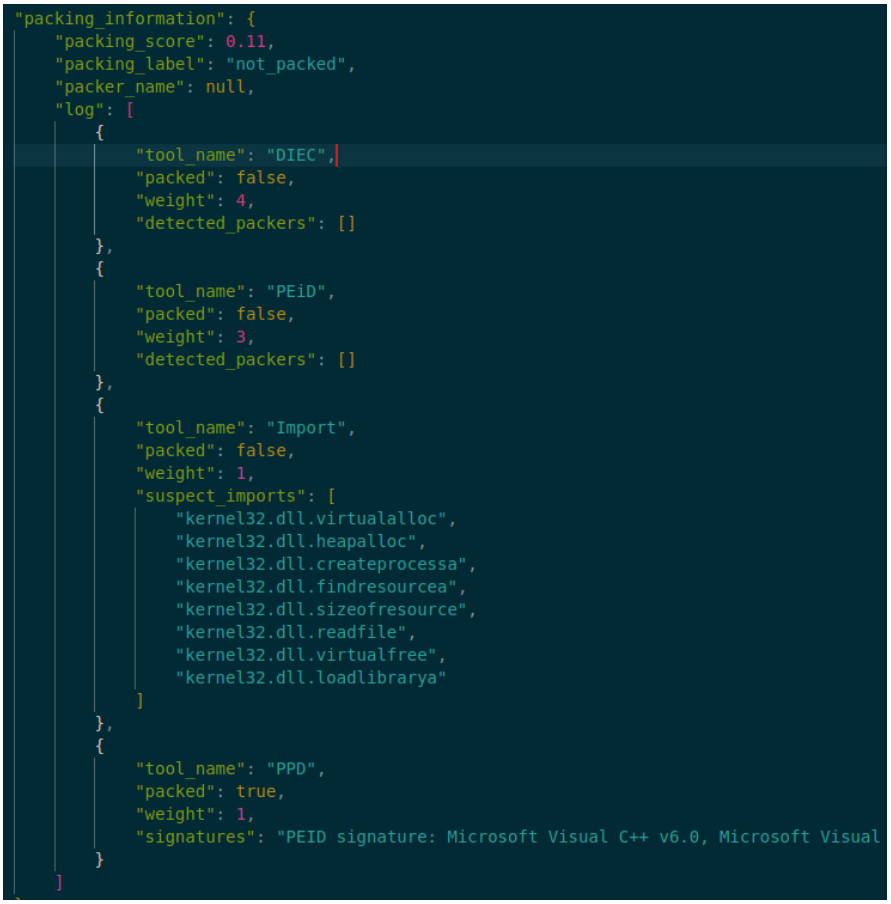
最后，基于打包检测工具返回的签名，我们也返回文件上可能使用的打包软件。

Mitre ATT&CK 和 MBC 信息 然后最后一节与 Mitre Att&ck 和 MBC 识别有关。

[2] https://github.com/horsicq/Detect-It-Easy

[3] https://github.com/packing-box/peid

[4] https://github.com/packing-box/pypackerdetect



```
"packing_information": {
    "packing_score": 0.11,
    "packing_label": "not_packed",
    "packer_name": null,
    "log": [
        {
            "tool_name": "DIEC",
            "packed": false,
            "weight": 4,
            "detected_packers": []
        },
        {
            "tool_name": "PEiD",
            "packed": false,
            "weight": 3,
            "detected_packers": []
        },
        {
            "tool_name": "Import",
            "packed": false,
            "weight": 1,
            "suspect_imports": [
                "kernel32.dll.virtualalloc",
                "kernel32.dll.heapalloc",
                "kernel32.dll.createprocessa",
                "kernel32.dll.findresourcea",
                "kernel32.dll.sizeofresource",
                "kernel32.dll.readfile",
                "kernel32.dll.virtualfree",
                "kernel32.dll.loadlibrarya"
            ]
        },
        {
            "tool_name": "PPD",
            "packed": true,
            "weight": 1,
            "signatures": "PEID signature: Microsoft Visual C++ v6.0, Microsoft Visual
        }
    ]
```


Fig. 5. 打包签名

MITRE ATT&CK[5] 框架是一个基于现实世界观察的对手战术和技术的全球可访问知识库。它提供了一种有结构的方法来理解网络威胁在整个攻击生命周期中的运作方式。对于恶意软件分析，特别是在分类诸如特洛伊木马和勒索软件等威胁时，ATT&CK 帮助分析师将恶意行为映射到具体的技术上，如凭据转储、命令与控制或数据加密以实现影响。通过将观察到的恶意软件行为与 ATT&CK 技术对齐，安全专业人员可以深入了解恶意软件的功能性、目标及潜在缓解措施。这种行为映射对于威胁情报、检测工程和事件响应至关重要，使防御者能够预测攻击者的行动并改善其防御姿态。

恶意软件行为目录（MBC）[6] 是一个结构化的框架，旨在以一致和标准化的方式描述和分类恶意软件的行为。与一般的威胁框架不同，MBC 特别关注恶意软件的操作方式，包括数据窃取、持久性、规避和执行等能力。通过深入研究恶意软件为中心的技术和模式，它补充了 MITRE ATT&CK，对于分类如特洛伊木马、勒索软件和间谍软件等恶意软件类别尤其有价值。通过对观察到的行为进行映射到 MBC 条目，分析师可以更好地理解恶意软件样本的功能性，将其与已知行为进行比较，并丰富威胁情报。这种详细的行为洞察支持逆向工程、检测开发和归因工作，从而提升整个恶意软件分析过程。

[5] https://attack.mitre.org/
[6] https://github.com/MBCProject/mbc-markdown

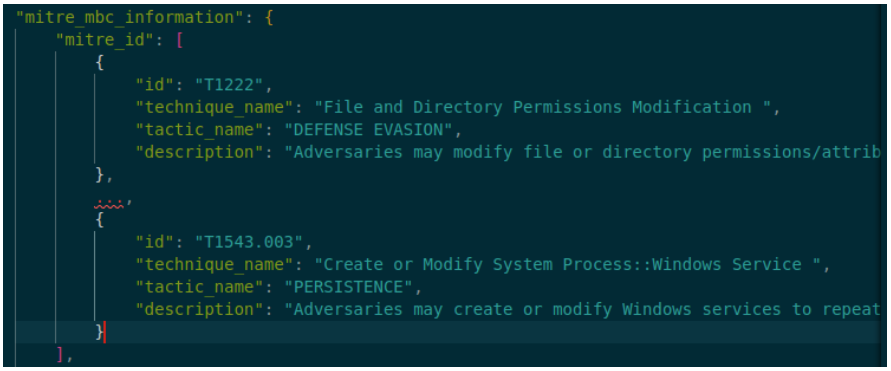


Fig. 6. MITRE ATT&CK 和 MBC

使用 CAPA[7]，我们提取 MITRE ATT&CK 和 MBC 信息并将其整合到我们的 JSON 报告中，如图 6 所示。包含此信息的目的是为恶意软件分析师提供关于恶意文件的行为信息，并通过深度学习模型对文件进行更好的描述来改进恶意软件分析。

2 实验

在本节中，我们展示了基于变压器架构的恶意软件分类方法的结果。这是一个比恶意软件检测更为复杂的用例。我们在这里不涉及检测任务，因为我们认为它不需要使用变压器，并且之前已经在其他工作中得到了很好的解决 [2,7,15]。

2.1 模型和训练

我们选择了 BERT base uncased 模型 [5]，这是一个通过使用掩码建模目标在大量英语语料库上以自监督方式预训练的 transformers 模型，因为它是一个非常灵活的模型，仅有 1.1 亿个参数。Bert 的一个限制在于 512 个标记的数量，但通过对报告的相关部分进行选择，我们最终得到了一种编码，其中平均标记数量为 403，中位数为 350，只有 15% 的数据大于 512 个标记。我们认为这是在 Bert 模型的质量和训练速度之间的一个良好折衷。为了理解这种限制意味着什么，我们也测试了 ModernBert 模型，这是一个具有 1024 个标记限制的类似模型。仅 1.45% 的数据大于 1024 个标记。使用这些小型模型使我们能够在 Apple M4 Pro MacBook 上进行测试。我们也通过在 4090RTX 上对具有 15 亿参数的 deepseek 模型进行微调来进行了一些测试，但计算时间过长且结果不太令人满意。

使用 BERT 模型，我们能够处理和分析与每个恶意软件相关的 JSON 报告。变换器（Transformers）凭借其捕捉文本数据中复杂上下文关系的能力，是此分类问题非常适应的解决方案。以下结果突显了这种方法的有效性，并提供了基于变换器的恶意软件分类的优势见解，前提是具备适当的预处理。由于类别不平衡，我们关注 F1 分数，但我们也提供其他指标以供参考。训练/测试分割比例如常为 80/20，我们在 10 个周期上使用批量大小为 16 来训练我们的模型。我们将 8 类恶意软件进行分类，从 0 到 7 排序如下：特洛伊木马、蠕虫、勒索软件、后门、信息窃取者、下载器、投放器和病毒。

[7] https://github.com/mandiant/capa

2.2 BERT 结果

为了展示我们关于使用 BERT uncased 模型进行分类任务的初步结果，我们在图 7 中计算了训练集的混淆矩阵，在图 8 中计算了测试集的混淆矩阵。我们可以看到该模型在识别数量最多的类别方面非常出色。对于数量较少的类别来说不太明显，但结果仍然不错。唯一分类效率不高的类别是 dropper 类别，其中模型倾向于将 droppers 识别为 trojans。这并不太令人惊讶，并且可以解释为：droppers（也称为 trojan-dropper）是一种特洛伊木马，其设计目的是在计算机上安装恶意软件。例如，F-Secure 或 Kaspersky 将其归类为 trojan-dropper。

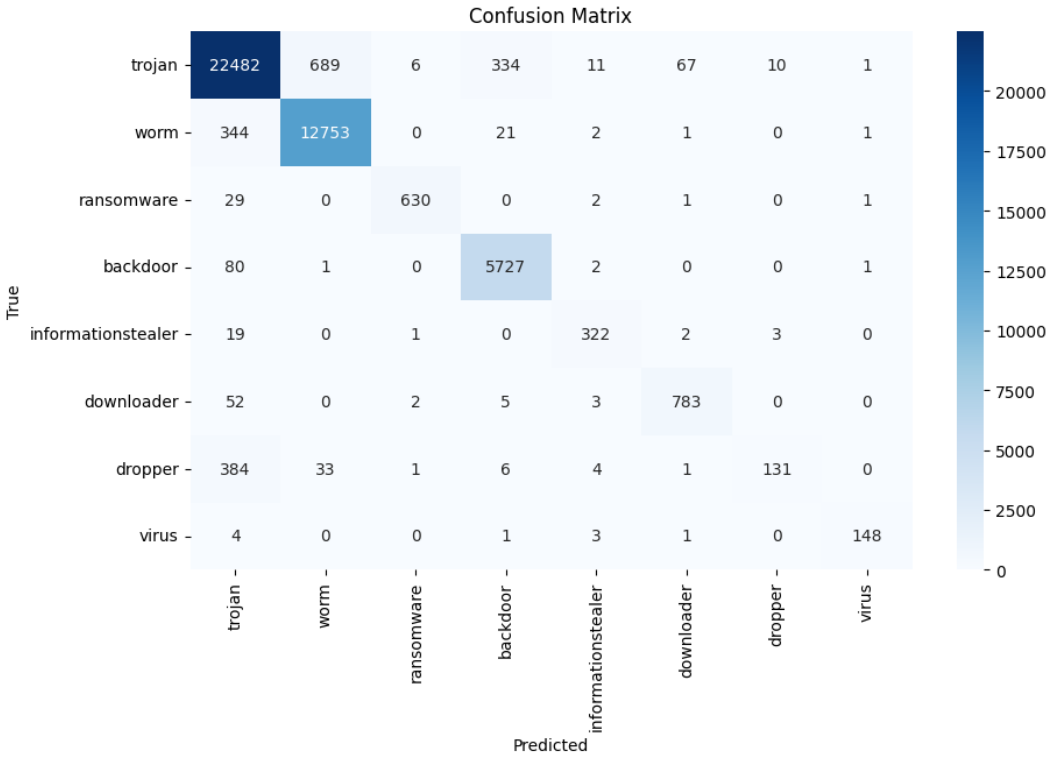


Fig. 7. 训练数据集的混淆矩阵。

为了更精确地了解我们模型的性能，我们在表格 4 和 5 中分别展示了训练集和测试集的准确率、F1 分数和召回率。我们还在这些表格中指定了每个类别的样本数量，并计算了分类任务的平均得分。

使用 BERT 模型获得的分类结果在大多数恶意软件类别中表现出色，即使是代表性不足的类别，其 F1 分数也大于 0.92（或 0.83），除了一个类别外，在训练（或测试）过程中均表现如此。如表 4 和 5 所示，该模型在区分所有类别时达到了非常高的准确率和 F1 分数，并且误分类极少，只有 droppers 除外。不出所料，代表性不足的类别对于模型来说更难识别，但如果能够增加这些类别的样本数量，则可以解决这个问题。

2.3 ModernBERT 结果

在本节中，我们首先通过使用 ModernBERT 来说明分类任务的结果。为此，图 9 显示了训练数据集的混淆矩阵，而图 10 则显示了相同的混淆矩阵，但针对测试数据集。

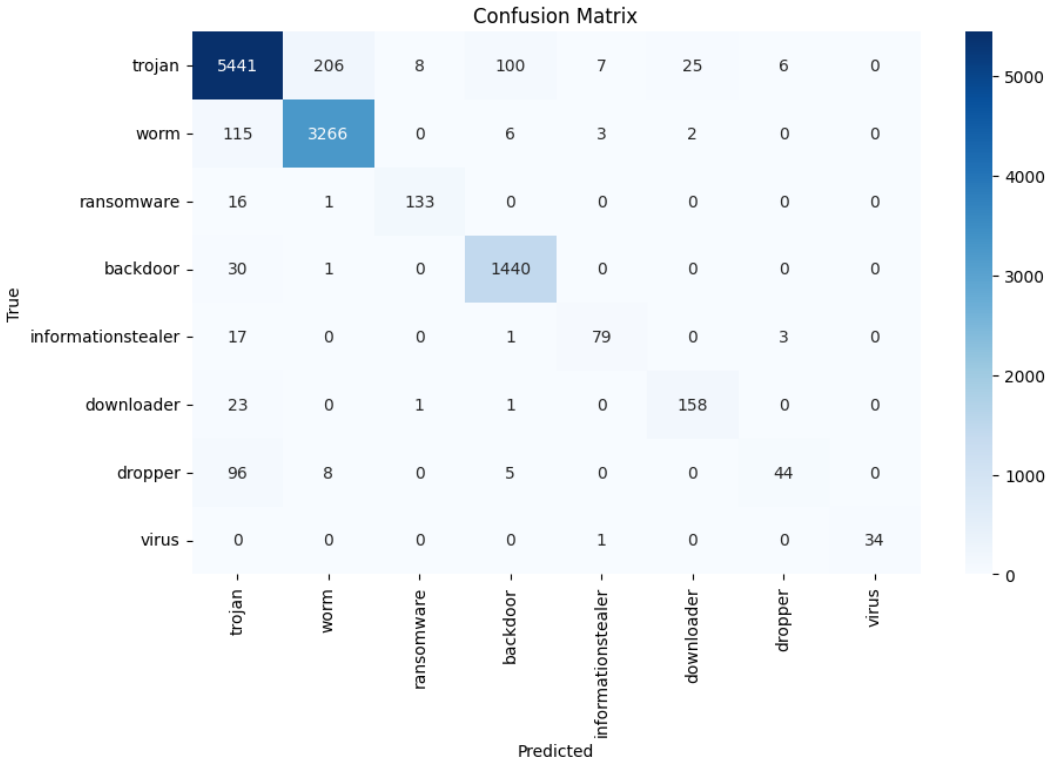


Fig. 8. 测试数据集的混淆矩阵。

Table 4. BERT 的训练分类报告

类	准确率	回忆	F1 分数	支持
0	0.96	0.95	0.96	23600
1	0.95	0.97	0.96	13122
2	0.98	0.95	0.97	663
3	0.94	0.99	0.96	5811
4	0.92	0.93	0.93	347
5	0.91	0.93	0.92	845
6	0.91	0.23	0.37	560
7	0.97	0.94	0.96	157
全球				
准确性			0.95	45105
宏平均	0.94	0.86	0.88	45105
加权平均	0.95	0.95	0.95	45105

增加 ModernBERT 编码器的最大输入长度可改善某些恶意软件家族的分类性能。虽然代表性较强的家族分类依然表现强劲，但延长序列长度显著增强了对释放者程序的检测能力，这是原始 BERT 模型的一个局限性。具体来说，根据表 6，在训练集上释放者的 F1 分数从 0.37 增加到 0.84，并且根据表 7，在测试集上从 0.43 增加到 0.75。尽管表格可能表明病毒的性能有所下降（在表 5 和表 7 之间，从 0.99 降到 0.87），但由于病毒样本数量较少，实际上这仅对应于三个额外的错误。

Table 5. 使用 BERT 测试分类报告

类	准确率	回忆	F1 分数	支持
0	0.95	0.94	0.94	5793
1	0.94	0.96	0.95	3392
2	0.94	0.89	0.91	150
3	0.93	0.98	0.95	1471
4	0.88	0.79	0.83	100
5	0.85	0.86	0.86	183
6	0.83	0.29	0.43	153
7	1.00	0.97	0.99	35
全球				
准确性			0.94	11277
宏平均	0.91	0.84	0.86	11277
加权平均	0.94	0.94	0.94	11277

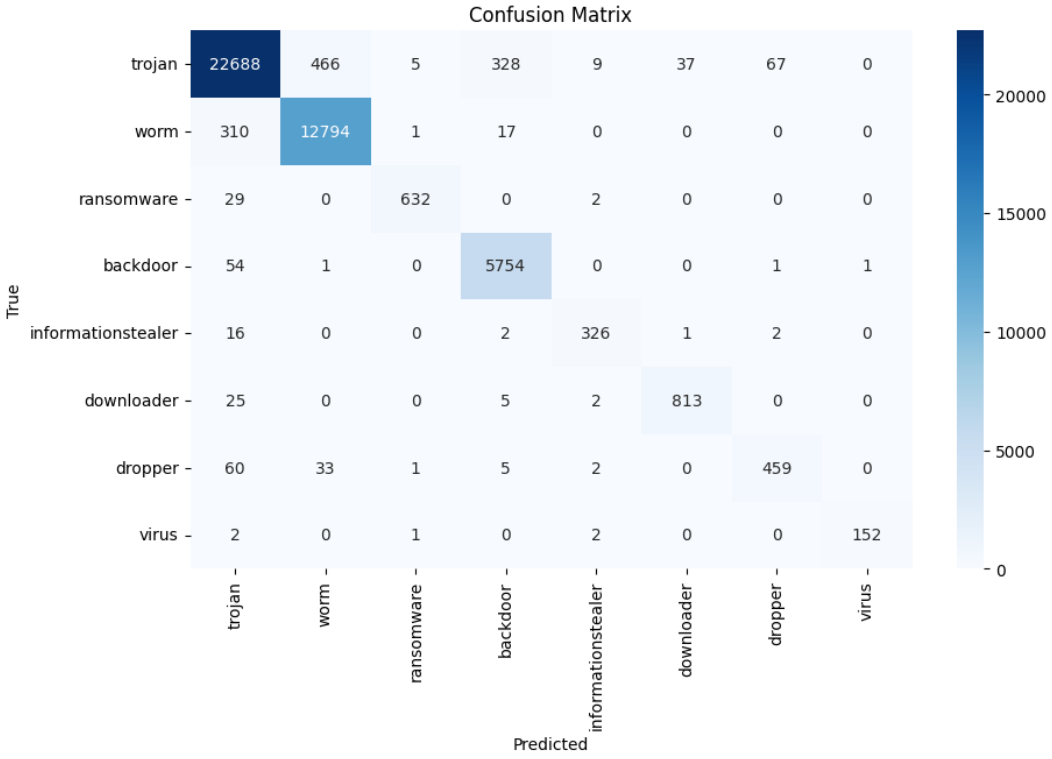


Fig. 9. 训练数据集的 ModernBERT 混淆矩阵。

使用 ModernBERT 模型将最大序列长度从 512 到 1024 并不会明显影响训练速度，但确实需要更多内存。然而，鉴于恶意软件分类结果的改进，这种权衡似乎是合理的。

总体而言，这些结果突出了利用与恶意软件的非常完整的报告相关的语言模型进行自动分类的潜力。因此，创建这样的报告非常重要，尽可能多地收集来自静态和行为分析的信息。

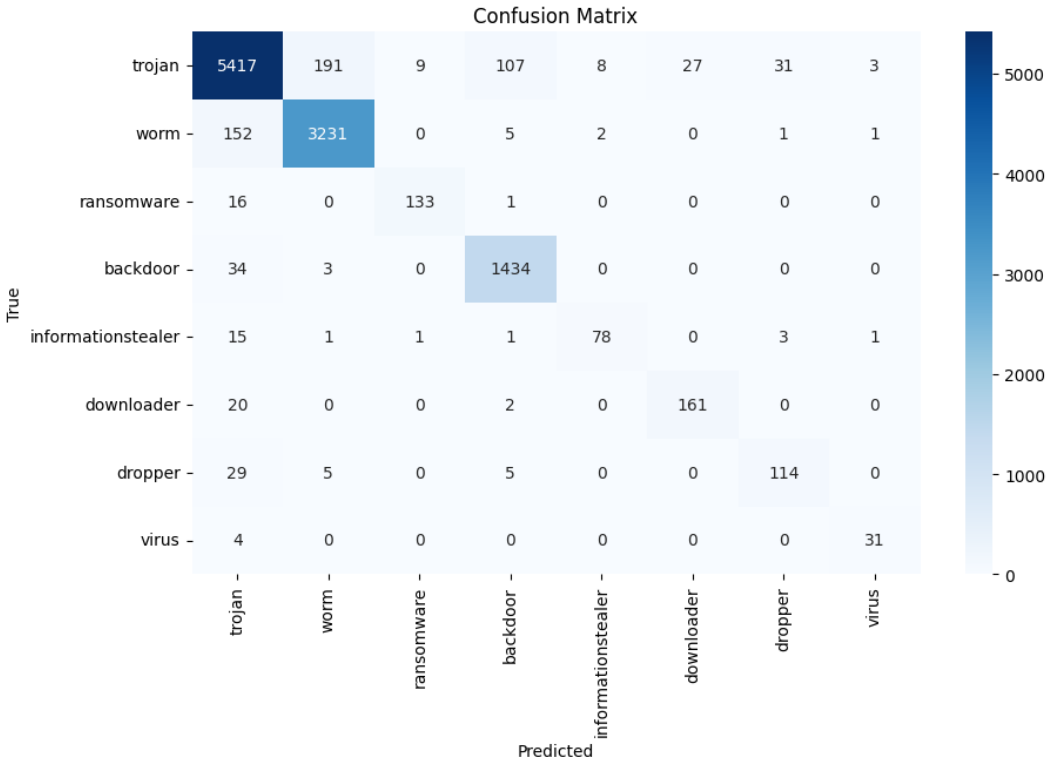


Fig. 10. 测试数据集的 ModernBERT 混淆矩阵。

Table 6. 使用 ModernBERT 的训练分类报告

类	精度	提醒	F1 分数	支持
0	0.98	0.96	0.97	23600
1	0.96	0.98	0.97	13122
2	0.99	0.95	0.97	663
3	0.94	0.99	0.97	5811
4	0.95	0.94	0.94	347
5	0.96	0.96	0.96	845
6	0.87	0.82	0.84	560
7	0.99	0.97	0.98	157
全球				
准确性			0.97	45105
宏平均	0.95	0.95	0.95	45105
加权平均	0.97	0.97	0.97	45105

Table 7. 测试分类报告与 ModernBERT

类	精度	回顾	F1 分数	支持
0	0.95	0.94	0.94	5793
1	0.94	0.95	0.95	3392
2	0.93	0.89	0.91	150
3	0.92	0.97	0.95	1471
4	0.89	0.78	0.83	100
5	0.86	0.88	0.87	183
6	0.77	0.75	0.75	153
7	0.86	0.89	0.87	35
全球				
准确性			0.94	11277
宏平均	0.89	0.88	0.88	11277
加权平均	0.94	0.94	0.94	11277

结论与未来工作

总而言之，本文提出了一种针对 PE 恶意软件文件的新预处理方法，该方法允许通过大型语言模型进行语义分析。我们的预处理生成 JSON 报告，收集静态和行为特征，例如打包签名和 YARA 规则。我们的预处理汇集了具体的特征，这些特征可以直接被分析师阅读和理解。我们使用 BERT 模型对一个现实数据集中的八类恶意软件进行了分类，其中某些类别代表性不足。分类结果非常好且令人鼓舞，表明我们的预处理方法适用于语义分析。

这项工作为我们提供了几个方面来提升性能。首先，静态和行为分析将受益于动态分析的补充，这将在报告中整合，从而为 PE 文件提供更为完整的预处理。我们还想通过检测恶意软件的类别及其家族来细化分类。为此，增加数据集的规模将是合适的，收集更多来自不同类别的样本，因此也包括不同的家族。这可能会提升我们的模型性能，使其能够更精确地区分诸如下载器和特洛伊木马这样的相近类别。我们还可能能够对更多的类别和家族进行分类。最后，大型语言模型的一大优势是使用注意力机制。我们在这一方向上进行了一些测试，但对于分类应用场景来说结果并不明确，但将它用于根据报告总结样本行为将会很有趣。对于分类应用场景，我们将尝试基于对我们关于报告中包含的信息的知识提供一些解释性说明：例如某些 Mitre Att&ck id 可能特定于勒索软件类别。

请注意，我们的特征提取和报告生成方法目前是一个原型，将根据之前的视角进行增强，并公开发布以供开放研究。我们相信这个新的完整的预处理是非常好的用于 AI 辅助恶意软件分类的工具，因为它在同一位置汇集了文件的所有信息，允许非常模块化的分析，如果有人只想使用特定的功能的话。我们期望这一贡献有助于解决恶意软件分类中的不良数据表示问题。

References

1. Anderson, H.S., Kharkar, A., Filar, B., Evans, D., Roth, P.: Learning to evade static pe machine learning malware models via reinforcement learning (2018), https://arxiv.org/abs/1801.08917
2. Anderson, H.S., Roth, P.: Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
3. Aubert, M.: Pestudio standard (2016), https://medium.com/@aubsec/pestudio-standard-f2ada4e8564, accessed: 2025-05-28

4. AV-Test: Av-atlas (2025), https://portal.av-atlas.org/malware, accessed: 2025-05-28
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2019), https://arxiv.org/abs/1810.04805
6. El Neel, L., Copiaco, A., Obaid, W., Mukhtar, H.: Comparison of feature extraction and classification techniques of pe malware. In: 2022 5th International Conference on Signal Processing and Information Security (ICSPIS). pp. 26–31. IEEE (2022)
7. Gaber, M.G., Ahmed, M., Janicke, H.: Malware detection with artificial intelligence: A systematic literature review. ACM Comput. Surv. **56**(6) (Jan 2024). https://doi.org/10.1145/3638552, https://doi.org/10.1145/3638552
8. Harang, R., Rudd, E.M.: Sorel-20m: A large scale benchmark dataset for malicious pe detection (2020), https://arxiv.org/abs/2012.07634
9. Joyce, R., Miller, G., Roth, P., Zak, R., Zaresky-Williams, E., Anderson, H., Raff, E., Holt, J.: Ember2024 – a benchmark dataset for holistic evaluation of malware classifiers (06 2025). https://doi.org/10.48550/arXiv.2506.05074
10. Joyce, R.J., Raff, E., Nicholas, C., Holt, J.: Maldict: Benchmark datasets on malware behaviors, platforms, exploitation, and packers (2023), https://arxiv.org/abs/2310.11706
11. Kapersky: The cyber surge: Kaspersky detected 467,000 malicious files daily in 2024 (2024), https://www.kaspersky.com/about/press-releases/the-cyber-surge-kaspersky-detected-467000-malicious-files-daily-in-2024, accessed: 2025-05-28
12. Khadilkar, A., Stamp, M.: Image-based malware classification using qr and aztec codes (2024), https://arxiv.org/abs/2412.08514
13. Loi, N., Borile, C., Ucci, D.: Towards an automated pipeline for detecting and classifying malware through machine learning (2021), https://arxiv.org/abs/2106.05625
14. Mandiant: Portable executable file infecting malware is increasingly found in ot networks (2021), https://cloud.google.com/blog/topics/threat-intelligence/pe-file-infecting-malware-ot/?hl=en, accessed: 2025-05-28
15. Marais, B., Quertier, T., Morucci, S.: Ai-based malware and ransomware detection models (2022), https://arxiv.org/abs/2207.02108
16. Microsoft Corporation: Inside windows: An in-depth look into the win32 portable executable file format (2019), https://learn.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail, accessed: 2025-05-28
17. Mohurle, S., Patil, M.: A brief study of wannacry threat: Ransomware attack 2017. International journal of advanced research in computer science **8**(5), 1938–1940 (2017)
18. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th international symposium on visualization for cyber security. pp. 1–7 (2011)
19. Oyama, Y., Miyashita, T., Kokubo, H.: Identifying useful features for malware detection in the ember dataset. In: 2019 seventh international symposium on computing and networking workshops (CANDARW). pp. 360–366. IEEE (2019)
20. Positive Technologie: Cybersecurity threatscape: Q2 2024 (2024), https://global.ptsecurity.com/analytics/cybersecurity-threatscape-2024-q2, accessed: 2025-05-28
21. Quertier, T., Barrué, G.: Use of multi-cnns for section analysis in static malware detection. arXiv e-prints pp. arXiv–2402 (2024)

22. Quertier, T., Marais, B., Morucci, S., Fournel, B.: Merlin – malware evasion with reinforcement learning (2022), https://arxiv.org/abs/2203.12980
23. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe (2017), https://arxiv.org/abs/1710.09435
24. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge (2018), https://arxiv.org/abs/1802.10135
25. Rong, H., Duan, Y., Zhang, H., Wang, X., Chen, H., Duan, S., Wang, S.: Disassembling obfuscated executables with llm. arXiv preprint arXiv:2407.08924 (2024)
26. Sun, B., Li, Q., Guo, Y., Wen, Q., Lin, X., Liu, W.: Malware family classification method based on static feature extraction. In: 2017 3rd IEEE International Conference on Computer and Communications (ICCC). pp. 507–513. IEEE (2017)
27. Vigna, G., Balzarotti, D.: When malware is packing heat. ENIGMA 2018, January 16-18, 2018, Santa Clara, CA, USA (2018), copyright Usenix. Personal use of this material is permitted. The definitive version of this paper was published in ENIGMA 2018, January 16-18, 2018, Santa Clara, CA, USA and is available at :
28. Wang, Y., Le, H., Gotmare, A.D., Bui, N.D., Li, J., Hoi, S.C.: Codet5+: Open code large language models for code understanding and generation. arXiv preprint arXiv:2305.07922 (2023)
29. Xiao, M., Guo, C., Shen, G., Cui, Y., Jiang, C.: Image-based malware classification using section distribution information. Computers & Security **110**, 102420 (2021). https://doi.org/https://doi.org/10.1016/j.cose.2021.102420, https://www.sciencedirect.com/science/article/pii/S0167404821002443
30. Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., Wang, G.: Bodmas: An open dataset for learning based temporal analysis of pe malware. In: 2021 IEEE Security and Privacy Workshops (SPW). pp. 78–84. IEEE (2021)
31. Zeltser, L.: Tips for reverse-engineering malicious code, https://zeltser.com/reverse-engineering-malicious-code-tips/, accessed: 2025-05-28
32. Zhang, J., Qin, Z., Yin, H., Ou, L., Zhang, K.: A feature-hybrid malware variants detection using cnn based opcode embedding and bpnn based api embedding. Computers & Security **84**, 376–392 (2019)