It Works on My Machine: A Systematization of Knowledge on Reproducibility and Replicability in ML-Based Ransomware Detection

Bruno Pereira
INESC TEC & U. Minho
bruno.f.pereira@inesctec.pt

Beatriz Cepa
INESC TEC & U. Minho
beatriz.cepa@inesctec.pt

Cláudia Brito
INESC TEC & U. Minho
claudia.v.brito@inesctec.pt

João Marco Silva
INESC TEC & U. Minho
joaomarco@di.uminho.pt

Vera Rimmer
DistriNet, KU Leuven
vera.rimmer@kuleuven.be

João Paulo
INESC TEC & U. Minho
joao.t.paulo@inesctec.pt

Tânia Esteves
INESC TEC & U. Minho
tania.esteves@inesctec.pt

***Abstract*—Machine Learning (ML) has emerged as a promising technique to make detection tools more accurate and adaptable to the ever-evolving, increasingly voluminous threat of ransomware attacks. In this paper, we study ML-based ransomware detection solutions grounded in behavioral analysis, from the perspective of reproducibility and replicability, as these properties are key for validating, reusing, and comparing existing work. Our study explores how these properties evolved over the past 10 years for the full ML pipeline workflow (*i.e.*, sample collection and analysis, feature extraction/selection, model configuration, and explainability). We discuss often overlooked yet critical challenges and provide guidelines to improve reproducibility and replicability in the field.**

***Index Terms*—ransomware detection, machine learning, reproducibility, replicability, systematization of knowledge**

1. Introduction

Cryptographic ransomware is a recognized and nefarious type of malware with unique behavior (*e.g.*, data encryption, file system traversal, and multiple accesses) when compared to other malware attacks [37]. Over the years, it has been responsible for high-profile attacks that have resulted in huge financial penalties for victims. For instance, in 2025, the SafePay ransomware group claimed responsibility for the attack on Ingram Micro, which resulted in severe operational impact, including outages in order processing, cloud licensing, and hardware distribution systems across multiple regions. This incident resulted in estimated daily revenue losses of $136 million, standing out as an example of ransomware being used to disrupt global commerce at scale [46].

The persistent increase and evolution of ransomware attacks, coupled with the escalating financial gains for perpetrators, have spurred significant research on ransomware detection techniques [37]. In many of these works, Machine Learning (ML) models are leveraged for learning complex behavioral patterns of ransomware attacks, which are then used to detect their presence and block them at the victim's system. As ML becomes commonplace across many security tools, including ransomware detection tools, it is critical to assess the reproducibility and replicability of research in this field. Without these properties, it becomes increasingly hard to reason about and compare different works, blocking the path to meaningful research and more secure production systems.

Goals. For the above reason, we explore how research on ML-based ransomware behavioral detection has evolved over the past 10 years, discussing the methodologies followed by existing work and providing guidelines for better reproducibility and replicability of future works.

Related Work. Several works have documented the reproducibility crisis in machine learning [16, 39, 40, 47], including issues such as insufficient reporting and limited artifact sharing. In the cybersecurity field, studies have addressed general challenges in learning-based security [7] and on the experimental evaluation of solutions tackling generic malware [43]. However, assessing reproducibility and replicability at the level of the specific ransomware subdomain remains essential. In ransomware detection, experimental outcomes depend on domain-specific factors—ransomware family selection, behavioral analysis environment configurations, and system realism—that are not captured by general-purpose ML guidelines or cybersecurity/malware-based ones.

At the same time, due to differing primary focuses, existing ransomware studies and surveys do not offer a sufficiently comprehensive view of the broader reproducibility challenge in ML-based ransomware detection. While some works touch on this problem without providing an in-depth analysis [5, 21, 37], others focus only on the quality and reproducibility of a single stage of the detection pipeline (*e.g.*, dataset creation [42]) or only conduct checks on papers with strong evidence of reproducibility [15], which remain a minority.

Contributions. To address these gaps, we build upon the existing work and provide the first systematization of knowledge targeted at reproducibility and replicability[1] of ML-based ransomware detection based on behavioral analysis[2], covering a wide range of papers in the field. Further, we analyze common pitfalls, their causes, and the challenges to be overcome at each stage of the ML pipeline. With this study, we aim to provide a common-

1. Reproducibility means that the experiments and results can be recreated by a different team in a similar experimental environment. Replicability means that experiments and results can be recreated by different teams in a different environment.

2. Behavioral analysis is also referred to as dynamic analysis.

DOI 10.1109/EuroSPW72509.2026.00032

place for researchers and practitioners to improve their future work. Our main contributions are as follows:

- **Comprehensive classification of 32 research articles (2016-2025):** We systematically examine the methodologies followed by existing ransomware detection solutions across the various phases of the ML-based pipeline, introducing a taxonomy based on key reproducibility and replicability characteristics (§3).
- **Consolidation of reproducibility guidelines:** Leveraging from the proposed taxonomy, we then discuss open challenges and propose 20 guidelines that overcome these while promoting more transparent and reproducible research in the field (§4).

2. Background

This section overviews key ransomware characteristics relevant to attack detection (§2.1), the type of datasets used in ML development (§2.2), and the typical stages of an ML workflow development (§2.3).

2.1. Ransomware Characteristics

Cryptographic ransomware attacks are characterized by their distinctive operational behavior. Although their infection and propagation mechanisms resemble those of a typical malware attack, once active in the system, they encrypt the victim's data and exploit the resulting inaccessibility to coerce ransom payments [37].

Therefore, a distinctive characteristic of ransomware is its intensive storage I/O behavior. To encrypt files, many families (*i.e.*, variants of a given ransomware attack) interact extensively with the OS storage stack, such as the file system, resulting in high storage and system call activity (*e.g.*, read and write operations) in a short period. Additionally, to locate target files, these attacks typically traverse the victim's directories, exhibiting a scan pattern over the full file system and/or disk [37].

Encryption may be applied to all user data or to specific files. While encrypting the entire data maximizes the attack surface, it also requires more time and extra system resources, which can be more easily detected by security tools. Consequently, some families adopt selective encryption, targeting specific file types/extensions (*e.g.*, documents – .pdf, .doc, .pptx, .xlsx; photos – .jpeg, .png; videos – .avi, .mp4) to compromise the most valuable data while accelerating and hiding the encryption process [20].

Regardless of the strategy, data encryption transforms file data into a high-entropy format, another key indicator of ransomware activity [38, 50]. These attacks may also create previously unseen file extensions (*e.g.*, *.ecrypt* in EREBUS, *.avoslinux* in AVOSLOCKER) to mark encrypted files [20]. Finally, ransomware often leaves ransom notes across directories to inform the victims.

The aforementioned characteristics lead to uncommon OS behavior, including directory traversal, changes in file type, high-volume and intensive read/write operations, and increased data entropy, among others, which are commonly used by detection tools to identify such attacks.

2.2. Dataset Types

When developing ML-based detection solutions, researchers may interact with three types of datasets: the *Sample*, the *Behavioral*, and the *Feature* datasets.

The ***Sample Dataset*** is a collection of executable files (*i.e.*, binaries), including both benign and ransomware samples. The samples can be labeled by class (benign or ransomware) and family, and serve as the source of information for analysis, which can be conducted either statically by examining the binaries themselves or dynamically by observing their behavior during execution. Our study focuses on solutions following dynamic analysis.

The ***Behavioral Dataset*** consists of data obtained from the dynamic analysis of each binary in the *Sample Dataset*, typically comprising execution traces of system calls, file operations, and API calls. The type of behavioral data can vary depending on the tools used for dynamic analysis, the underlying operating system, and the specific operations monitored (*e.g.*, network, file system).

Finally, the ***Feature Dataset*** corresponds to the features extracted from the *Behavioral Dataset*. It is usually obtained by reducing dimensionality and selecting features that best discriminate malicious and benign software.

We argue that this distinction between datasets is fundamental because conflating them into a single dataset introduces ambiguity about which data is actually used, especially when multiple kinds of datasets are involved.

2.3. ML Workflow Development

A typical development workflow for an ML-based ransomware detection solution is depicted in Figure 1 [21].

1. Sample Dataset Collection. The first step involves collecting representative samples of ransomware and benign software. This is usually done by resorting to public malware repositories (*e.g.*, VirusTotal, MalwareBazaar) for ransomware and legitimate software sources (*e.g.*, official app stores) for benign samples.

2. Behavioral Dataset Collection. Each sample in the *Sample Dataset* is subsequently executed, usually within a controlled environment (*e.g.*, isolated virtual machine), to extract its dynamic behavior (*e.g.*, I/O operations) [5].

3. Feature Dataset Creation. The next step consists of extracting features from the *Behavioral Dataset* and selecting the most relevant ones to construct the final feature dataset used for model training [54].

4. ML Model Development. Based on the extracted features, the ML-based detection model is designed and implemented. This process is usually carried out by using established Python libraries (*e.g.*, scikit-learn) and ML/DL frameworks (*e.g.*, Weka, PyTorch).

5. ML Model Training and Testing. To train and test the developed models, the *Feature Dataset* is split into two subsets: a *training set* and a *test set* [21]. The former is used to learn the model parameters, while the latter evaluates the model on previously unseen data. This process may be repeated multiple times, with iterative fine-tuning to improve performance.

6. ML Model Explainability. Although a relatively recent practice, some developers employ explainability mechanisms to understand the decisions made by ML models and how input data influences those decisions.

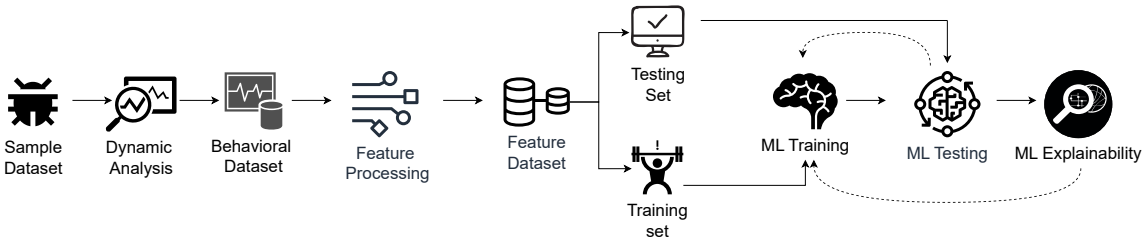

Figure 1. Illustration of the ML development pipeline for detection solutions, spanning sample collection, behavioral analysis, feature processing, model training and evaluation, and explainability. Solid arrows represent the standard workflow, while dashed arrows indicate iterative steps. For example, unsatisfactory testing results may require retraining the model. Likewise, insights from explainability may prompt further retraining.

This step is important, as it can reveal spurious correlations [7]. Based on these insights, developers may refine the model or adjust the features selected from the *Behavioral Dataset*.

Although the ML pipeline resembles the ones used for malware detection, developing ransomware-oriented solutions requires accounting for specific characteristics of these attacks (*e.g.*, storage-intensive behavior, selective targeting of file types, delayed execution to evade detection, constant and rapid emergence of new families) that can significantly affect system design and evaluation. In this work, we study the extent to which prior work accounts for these factors and discuss their impact on the reproducibility of existing solutions.

3. Study and Taxonomy

To assess the reproducibility and replicability of the ML-based ransomware detection field, we analyzed a curated set of 32 research papers published between 2016 and 2025 (Table 1). This set is intended to provide a broad and representative snapshot of research practices in the field, offering meaningful insights into prevailing trends and approaches without aiming to be an exhaustive collection of all published works.

We selected candidate papers using the following methodology. First, we considered publications from the past five years in leading security, systems and machine learning venues, including *USENIX Security Symposium*, *NDSS*, *OSDI*, *IEEE S&P*, *EuroS&P*, *ACM SIGSAC CCS*, *AsiaCCS*, *NeurIPS*, and *MLSys*, as well as renowned journals such as *ACM TOPS*, *IEEE TIFS*, and *IEEE TDSC*. From these, we selected papers with titles containing the keywords *ransomware*, *malware*, or *malicious*. After reviewing the resulting 343 papers, we applied three inclusion criteria: *(1)* the work must evaluate the proposed solution using ransomware samples; *(2)* it must propose an AI-based classification or detection approach for cryptographic ransomware; and *(3)* it must incorporate behavioral analysis and use storage I/O features (*e.g.*, file operations). This process yielded 4 papers. To expand this set, we also included relevant works cited in the most recent ransomware survey published in *ACM Computing Surveys* [37]. Applying the same criteria, we added 28 additional papers published between 2016 and 2020.

Table 2 classifies the analyzed articles according to the information reported for each workflow phase described in §2.3. This section details the different approaches followed by the reviewed work across our taxonomy's classification axes. In §4, we then discuss open challenges and guidelines based on our observations.

TABLE 1. LIST OF SURVEYED PAPERS IN CHRONOLOGICAL ORDER

Solution	Year	Venue	Citations	Top
ElderRan [48]	2016	arXiv	463	
ShieldFS [14]	2016	ACSAC	433	
RansHunt [25]	2017	ICCIT	80	
Homayoun et al. [28]	2017	IEEE TETC	260	✓
Chen et al. [12]	2017	RACS	155	
Maniath et al. [35]	2017	RDCAPE	133	
DNA-Droid [23]	2017	NSS	112	
Ferrante et al. [22]	2018	FPS	108	
Saleh Al-rimy et al. [45]	2018	IJIE	82	
RWGuard [36]	2018	RAID	211	
SSD-Insider [10]	2018	ICDCS	106	
Takeuchi, Sakai, and Fukumoto [51]	2018	ICPP	138	
RansomBlocker [38]	2019	ACM/IEEE DAC	43	
Ashraf et al. [8]	2019	arXiv	36	
Chen et al. [11]	2019	SciSec	59	
Egunjobi, Parkinson, and Crampton [19]	2019	IDEAL	27	
Goyal et al. [24]	2019	ICRESH	19	
Maigida et al. [33]	2019	JSE	5	
Al-rimy, Maarof, and Shaid [41]	2019	FGCS	126	✓
Bae, Lee, and Im [9]	2020	CC:PE	185	
Abbasi, Al-Sahaf, and Welch [1]	2020	EvoApplications	29	
Jethva et al. [30]	2020	JCS	70	
Hwang et al. [29]	2020	WPC	173	
Zuhair, Selamat, and Krejcar [56]	2020	Applied Sciences	35	
Sharmeen et al. [49]	2020	IEEE Access	97	✓
Ahmed, Kocer, and Al-rimy [3]	2020	TIIS	51	
Ahmed et al. [4]	2020	JNCA	120	✓
Zhou et al. [55]	2020	ICISSP	8	
CanCal [52]	2024	ACM SIGSAC	9	✓
DeftPunk [53]	2024	USENIX OSDI	11	✓
Sloun et al. [50]	2025	NSDI	1	✓
Minerva [27]	2025	ACM AsiaCCS	19	✓

** The Top column highlights publications in top-tier venues, i.e., classified as A/A* by the CORE Ranking (conferences) or Q1 by SCImago (journals).*

3.1. Dataset Usage/Creation

As previously mentioned, the development workflow of ML-based detection solutions involves three datasets (Sample, Behavioral, and Feature), which researchers may either create from scratch or derive from existing ones.

Our analysis shows that all reviewed papers build feature datasets from scratch. Further, most of these works build new behavioral datasets, with only four leveraging publicly available ones [1, 27, 33, 50]. Building a behavioral dataset requires both a well-curated sample dataset and a realistic analysis environment to execute samples and record their behavior, a process that can be time-consuming. As reported by Maniath et al. [35], even with a fully automated pipeline, analyzing and extracting features from just 157 ransomware samples required 56 hours. Regarding sample datasets, all works build these from scratch or from subsets of previous datasets [22, 23].

This pattern of incrementally constructing all three datasets from scratch, rather than building on prior work,

TABLE 2. CLASSIFICATION OF ML-BASED RANSOMWARE DETECTION RESEARCH BASED ON ITS REPRODUCIBILITY AND REPLICABILITY.

	Dataset			Tested Samples											Behavioral Analysis							Feature Extraction					ML Training					Availability				
				Sample Size			Ransomware					Benign											Input		Techniques		Output						Dataset			Code	
Papers	Sample	Behavioral	Feature	# Ransomware	# Malware	# Benign	# Families	Families Name & Distribution	Collection Date	Source	Samples Hash	Apps class	Apps name	Similar RW behavior	Source	Environment	Operating System	Network	File System	User Behavior	Analysis Tools	Execution Time	Groups	Operations	Feature Engineering	Feature Selection	Data Type	Data Format	Train/Test set	Model Types	Libraries/Frameworks	(Hyper-)parameters	Explainability	Sample	Behavioral	Feature	Pseudo-code	Artifacts
[48]	○	○	○	582	—	942	11	✓	✓	✓	+/-	✓	✗	✓	✓	S	W XP	✓	✓	✗	AP	✓	✓	+/-	K	K	B	✓	✓	ST	✓	✓	✗	✗	✓	✗	✗	✗
[14]	○	○	○	688	—	2245	11	✓	✓	✓	+/-	✗	✗	✓	✗	SR	W 7, 8.1, 10	✓	✓	✓	C	✓	✓	✓	K	—	N	✓	✓	ST	✗	+/-	✗	?	✓(∗)	?	PC	✓(∗)
[25]	○	○	○	360	532	460	21	✓	✗	✓	✗	✗	✗	✗	✗	S	W XP W 7	✗	✗	✗	AP	✗	✓	✗	✗	K	✗	✗	✓	ST	✓	+/-	✗	✗	✗	✗	✗	✗
[28]	○	○	○	1624	—	220	3	✓	✓	✓	✗	✗	✗	✗	✓	S	W 10	✗	✗	✓	C	✓	✓	✓	KC	K	N	✓	✓	MT	✓	✗	✗	✗	✗	✗	PC	+/-
[12]	○	○	○	83	—	85	4	+/-	✗	✓	✗	✓	✗	✗	✓	S	W 7	✗	✗	✗	MT	✓	✓	✗	KC	K	N	✓	✓	MT	✓	✗	✗	✗	✗	✗	✗	✗
[35]	○	○	○	157	—	?	✗	✗	✗	✓	✗	✗	✗	✗	✗	S	W ?	✗	✗	✗	AP	✓	✓	✗	K	—	N	✓	✗	ST	✓	✓	✗	✗	✗	✗	✗	+/-
[23]	◐	○	○	1928	—	2500	8	✓	✓	✓	✗	✗	✗	✗	✓	S	A 4.1	✗	✗	✓	C	✗	✓	✗	K	—	N	✗	✓	ST	✓	✗	✗	✗	✗	✗	✗	+/-
[45]	○	○	○	38152	—	1000	?	✗	✓	✓	✗	✗	✗	✗	✓	S	W XP	✗	✗	✗	AP	✓	✓	✗	K	K	N	✓	✓	ST	✓	✗	✗	✗	✗	✗	PC	✗
[36]	○	○	○	14	—	?	14	+/-	✗	✓	✗	✗	✓	✓	✗	S	W ?	✗	✗	✗	MT	✗	✓	✓	✗	✗	N	✓	✓	MT	✗	✗	✗	✗	✗	✗	✗	✗
[10]	○	○	○	10	—	15	8	✓	✗	✓	✗	✓	✓	✓	✗	S	I	✗	✗	✗	✗	✗	+/-	+/-	C	—	N	✗	✗	ST	✗	✗	✗	✗	✗	✗	✗	✗
[51]	○	○	○	276	—	312	?	✗	✗	✓	✗	✓	✗	✗	✓	S	W 7	✗	✓	✗	AP	✗	✓	✗	K	—	N	✓	✓	ST	✗	✗	✗	✗	✗	✗	F	✗
[22]	◐	○	○	672	—	2386	5	+/-	✓	✓	✗	✗	✗	✗	✓	S	A 4.0	✗	✗	✓	MT	?	✓	✗	✗	✗	N	✓	✓	MT	✗	✗	✗	✗	✗	✗	✗	✗
[38]	○	○	○	✗	—	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	I	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	ST	✗	✗	✗	✗	✗	✗	✗	✗
[9]	○	○	○	942	830	292	✗	✗	✗	✗	✗	✗	✗	✗	✓	S	W 7	✗	✗	✗	MT	✓	✓	✗	KC	—	N	✓	✓	MT	✓	✗	✗	✗	✗	✗	F	✗
[8]	○	○	○	∗	—	∗	✗	✗	✗	✓	✗	✗	✗	✗	✓	S	W 7	✗	✗	✗	AP	✗	✓	✗	K	K	✗	✓	✓	MT	✗	+/-	✗	✗	✗	✗	F	✗
[11]	○	○	○	7	—	✗	7	✓	✓	✗	✗	✗	✗	✗	✗	S	W ?	✗	✗	✓	AP	✗	✗	✗	K	K	N	✓	✗	ST	✓	✗	✗	✗	✗	✗	✗	✗
[19]	○	○	○	200	—	200	?	✗	✗	✓	✗	✗	✗	✗	✓	S	W 10	✓	✗	✗	AP	✗	✗	✗	✗	✗	✗	✗	✓	MT	✓	✗	✗	✗	✗	✗	✗	+/-
[24]	○	○	○	67	—	?	?	✗	✗	✓	✗	✗	✗	✓	✗	✗	W ?	✗	✗	✗	✗	✗	+/-	✗	✗	✗	✗	✓	✗	ST	✗	✗	✗	✗	✗	✗	✗	✗
[33]	—	●	○	582	—	942	11	+/-	✗	—	—	✓	✗	✗	—	—	W ?	—	—	—	—	—	✓	✗	✗	C	✗	✓	✓	ST	✗	✗	✗	—	—	✗	PC	✗
[41]	○	○	○	8152	—	1000	15	✓	✓	✓	✗	✓	✗	✓	✓	S	W ?	✗	✗	✗	AP	✗	✓	✗	KC	KC	N	✓	✓	MT	✓	✗	✗	✗	✗	✗	PC	✗
[1]	—	●	○	582	—	942	11	✓	✗	—	—	✗	✗	✗	—	—	W ?	—	—	—	—	—	✓	✗	✗	K	B	✓	✓	MT	✓	✓	✗	—	—	✗	✗	✗
[30]	○	○	○	668	—	103	21	✓	✗	✓	✗	✗	✗	✓	✓	S	W 7	✓	✓	✗	AP	✓	✓	✗	K	K	NB	✓	✓	MT	✓	✗	✗	✗	✓∗	✗	PC	✗
[29]	○	○	○	∗	—	∗	✗	✗	✗	✓	✗	✗	✗	✗	✓	S	W ?	✗	✗	✗	AP	✗	✓	+/-	✗	✗	N	✓	✓	ST	✗	✗	✗	✗	✗	✗	✗	✗
[56]	○	○	○	35015	500	500	14	✓	✓	✓	✗	✗	✗	✗	✓	✗	W ?	✗	✗	✗	✗	✗	✓	✗	C	—	B	✓	✓	MT	✓	✗	✗	✗	✗	✗	PC	✗
[49]	○	○	○	483	—	754	14	✓	✓	✓	✗	✗	✗	✗	✓	S	W ?	✗	✗	✗	AP	✓	✓	+/-	K	K	B	✓	✓	ST	✓	✓	✗	✗	✗	✗	PC	✗
[3]	○	○	○	673	—	742	14	✓	✓	✓	✗	✓	✓	✓	✓	S	W XP	✓	✓	✓	AP	✓	✓	✗	K	K	B	✓	✓	ST	✗	✗	✗	✗	✗	✗	PC	✗
[4]	○	○	○	1354	—	1358	14	✓	✗	✓	✗	✓	✗	✓	✓	S	W XP	✓	✓	✓	AP	✓	✓	+/-	K	C	✗	✓	✓	MT	✗	✗	✗	✗	✗	✗	PC	✗
[55]	○	○	○	899	—	241	9	✓	✓	✓	✗	✗	✗	✗	✗	S	W ?	✗	✗	✗	AP	✗	✓	✗	K	—	N	✓	✓	ST	✓	✓	✗	✗	✗	✗	✗	✗
[52]	○	○	○	1761	—	1768	?	✗	✓	✓	✗	✓	✗	✗	✗	S	W ?	✗	✗	✗	✗	✗	+/-	+/-	C	—	✗	✓	✗	ST	✗	✗	✗	✗	✗	✗	PC	✗
[53]	○	○	○	13	—	16	10	✗	✗	✗	✗	✗	✓	✓	✗	?	WS 16, 19, 22, C	✗	✗	✗	C	✗	+/-	+/-	✗	✗	✗	✗	✓	MT	✗	✗	✗	✗	✓	✗	✗	✗
[50]	—	●	○	383	—	?	5	✓	✗	—	—	✗	✗	✗	—	—	—	—	—	—	—	—	✓	✓	K	—	✗	✗	✓	ST	✓	✗	✗	—	—	✓	✗	✓
[27]	○	◐	○	476	—	2245	10	✓	✓	✗	✗	✓	✗	✓	—	S	W 11	✗	✓	✗	✗	✗	+/-	+/-	✗	✗	N	✓	✓	ST	✗	✗	✓	✗	✗	✗	✗	✗

Global
— Not applicable
? - Unclear
∗ - Inconsistent
✓ - Provided
✗- Not provided
+/- - Partially provided

Dataset
○ New dataset
◐ Partially new
● Reuse dataset

Environment & OS
S - Sandbox
R - Real
W - Windows
WS - Windows Server
A - Android
C - CentOS

Analysis Tool
AP - Analysis Platform
MT - Monitoring Tool
C - Custom

Feature
K - Known
C - Custom
B - Binary
N - Numeric

Model Types
ST - Single
MT - Multiple

Code
PC - Pseudo-code
F - Formulas
✓(∗) - Unavailable

is time-consuming and often results in redundant effort across research teams.

3.2. Tested Samples

Training effective detection models requires datasets that include ransomware (from diverse families) and benign samples. Ideally, the benign set would also include applications that exhibit ransomware-like behaviors (*e.g.*, data encryption or compression) so that the model does not incorrectly associate certain traits (*e.g.*, generating high-entropy data) exclusively with ransomware.

Sample selection and size. Almost all analyzed works report the total number of considered ransomware and benign samples. However, some papers provide inconsistent values throughout the text [8, 29], while others do not mention the exact number of benign samples [24, 35, 36, 50]. In RansomBlocker [38], three in-house ransomware variants are used, but it is unclear whether they were used for model training or solely for evaluation. SSD-Insider [10] and Minerva [27] also develop in-house ransomware variants for testing specific ransomware traits.

While most works rely on ransomware-based datasets, three papers also include other types of malware [9, 25, 56]. The inclusion of additional malware may affect the learning process and the evaluation of detection models, making it crucial for papers to report this information.

Train and test samples. Only four works explicitly report the number of samples allocated to model training and testing [10, 14, 19, 52]. This is an important detail to understand if the model is tested on unseen samples (*e.g.*, new ransomware families not used during model training). A related critical aspect is the discrepancy between the number of samples selected and those actually used for training and testing. Several works may leave readers with the impression that the proposed solutions were tested on the full set of selected samples. However, later in the paper, it is noted that several samples are excluded due

to various reasons (*e.g.*, non-functional binaries, inability to connect to Command and Control (C&C) servers) [3, 9, 24, 30, 49]. Such discrepancies may lead to incorrect assumptions about the dataset size and affect the reproducibility and replicability of reported results.

Ransomware families and hashes. When considering further details about ransomware samples, such as families, ten papers do not fully report the number of families, their names, or the sample distribution, sometimes providing only a few illustrative examples [8, 9, 19, 24, 29, 35, 38, 45, 51, 52]. Among the remaining works, four mention only the family names without detailing the distribution of samples per family [12, 22, 33, 36].

Notably, ShieldFS [14] is the only work to report sample hashes within the paper. However, these are reported only for the three samples used to evaluate the proposed system, not for the other samples used in the ML workflow. In EldeRan [48], the samples' hashes are disclosed alongside the publicly available dataset, but neither the dataset nor the hashes are referenced in the paper. For the remaining works, no sample hashes are reported, making it impossible to know the exact sample binary used.

Ransomware sources and temporal information. Regarding the sources of ransomware samples, most works rely on public malware repositories such as VirusTotal and VirusShare. Three papers [22, 23, 27] draw samples from datasets used in prior works, including R-PackDroid [34], HelDroid [6], Contagio [13] and Koodous [31]. Five papers do not report their sources [9–11, 27, 53].

Only fourteen papers describe the temporal information of their samples, using varying strategies. Some specify the dates when samples were collected, others report the dates on which samples were added to public malware repositories (*e.g.*, VirusTotal), while others indicate when the corresponding ransomware family was first seen.

Benign sample description and selection. Only four papers specify the names of benign application samples [3, 10, 36, 53]. Eight papers report solely the application categories (*e.g.*, office suite applications, development tools, media players, internet browsers), whereas the remaining works provide no information in this regard. Importantly, none of the works report the versions of the benign applications, making it impossible to reproduce the exact version of the binaries.

Only eleven papers select benign applications exhibiting behaviors similar to ransomware, such as data compression or encryption utilities. Interestingly, in CanCal [52], benign samples doing data compression, encryption, and backup are intentionally excluded to avoid behaviors similar to ransomware. However, no discussion or justification for this decision is provided.

Benign sample sources. Most works rely on software repository websites such as *Software Informer* and *Portable Apps* for fetching benign samples. Other works collect binaries directly from Windows system directories (*e.g.*, the *System32* folder) [3, 4, 8, 9, 49]. For Android environments, Google Play and alternative marketplaces such as F-Droid, Mumayi, and AnZhi are commonly used sources [22, 23]. Additionally, one paper collects benign samples from Softonic [29], two others use applications from real machines [14, 53], and eight other works do not report the source of their benign samples.

3.3. Behavioral Analysis

To perform behavioral analysis, researchers typically execute samples in a controlled environment, using monitoring tools to collect data on their interactions with the network, file system, and OS.

Execution environment. The majority of works on ransomware analysis and detection rely on sandboxed environments to execute ransomware samples, ensuring that the underlying infrastructure remains protected from infection or damage [37].

The Cuckoo Sandbox, an open-source dynamic malware analysis system, is often used to provide a virtualized, controlled environment for executing malware samples and extracting behavioral information. Alternatively, Egunjobi et al. [19] use virtualized environments provided by VirusTotal.[3] DNA-Droid uses a custom Android sandbox that can hook and report Android API calls.

A few works collect behavioral data from real environments. ShieldFS [14] captures benign application behavior from 11 operational desktop computers, while malicious behavior is collected from a controlled, sandboxed environment. DeftPunk [53] gathers benign data from EBS virtual disks deployed in Alibaba Cloud. However, it is unclear how ransomware data is gathered. Unlike prior works, four of the papers do not provide any information about their analysis environment [24, 36, 38, 56].

Operating System. The prevalence of ransomware attacks and availability of samples targeting the Windows OS make it a common choice for research works. Only two works focus on Android [22, 23], and another two propose OS-independent solutions [10, 38]. Several works do not specify the targeted OS and version, leaving readers to infer the OS from the type of data collected (*e.g.*, Windows API features). Notably, ShieldFS [14], RansHunt [25], and DeftPunk [53] are the only ones that analyze samples across multiple OS versions.

Among the reported OS versions, the majority target older systems such as Windows XP and Windows 7, with only four papers focusing on more recent versions (Windows 10 or 11) [14, 19, 27, 28]. Some works justify the use of Windows XP due to its weaker security protections, which allow for the observation of a broader range of ransomware behavior [3, 48].

Network setup. Whether the analysis environment allows external communication can influence the behavior or even the ability to execute ransomware samples (*e.g.*, due to some samples' reliance on communication with C&C servers). Among the analyzed works, only six report their network setup. Of these, one work disables all network connectivity [19], while the others allow controlled network access to enable communication with C&C servers [3, 4, 14, 30, 48].

File system state. An often-overlooked but crucial detail, particularly due to ransomware's file-intensive nature, is how the targeted file system is populated with realistic files and directories. In our analysis, we observe that only seven papers provide details about the state of the file system. The strategies described include populating multiple directories with common user files, especially those frequently targeted by ransomware (*e.g.*, images,

3. https://docs.virustotal.com/docs/in-house-sandboxes

office documents, and text files), across locations such as *My Documents* and *My Pictures and Videos* [3, 4, 48]; constructing large-scale file datasets (*e.g.*, 10 folders each containing 1,000 subfolders [51], around 33,000 files distributed across numerous subdirectories [27], or collections totaling 1.51 GB [30]); and recreating realistic user environments using real files that contain typical user data, such as credentials and browsing history, combined with directory structures observed on a clean OS [14].

Notably, Jethva et al. [30] report the number and distribution of files per file type and describe how files are distributed across the entire file system by following prior studies on file organization within directories [26] and across file systems [18].

User behavior simulation. Another important aspect concerns simulating user behavior and interactions with the system, as benign applications may require it to operate in realistic conditions, and some ransomware families expect it to trigger attacks [3, 28]. Despite this, only seven of the reviewed papers describe how user behavior is simulated in their experiments.

Among these works, two provide no detailed reproduction information, merely reporting the presence of generic user activity [14, 23]. For example, ShieldFS [14] reports that benign samples are analyzed on real machines with genuine user interactions, whereas ransomware samples are executed in a sandboxed environment where basic user activity, such as mouse movement and application launches, is artificially emulated. However, no further details on the emulation mechanism are provided.

One work adopts Monkey [17], an automated input-generation tool that injects a pseudo-random stream of user events into the system [22]. The remaining works use custom scripts to reproduce user behavior, simulating activities such as file creation and deletion, web browsing, email exchange, and interaction with online services [3, 4, 11]. In Homayoun et al. [28], the authors develop PyWinMonkey, a Monkey-inspired tool for Windows that automates user interactions. From the above works, this is the only tool made publicly available [44].

Analysis tools. Most papers that collect new behavioral data specify the tools used, with only six exceptions [10, 24, 27, 38, 52, 56]. The majority rely on existing dynamic analysis platforms such as Cuckoo Sandbox, while one paper uses a modified version that provides enhanced anti-anti-sandbox features [35]. Similarly, one work leverages dynamic analysis from the VirusTotal platform [19]. Four papers develop custom tools [14, 23, 28, 53], whereas others rely on other existing monitoring tools, including the PIN tool [9], strace and tcpstat [22], API Monitor [12], and the IRPLogger [36].

Sample execution time. The duration of a sample's execution directly affects the amount and type of behavioral information collected, making it an important reproducibility metric. Only eleven papers report execution times, which vary widely, from as little as 5 seconds [45] to up to 90 minutes [14]. The latter follows the best practices for malware experimentation as suggested in [43], since malware often exhibits additional behavior when allowed to execute for extended periods. In summary, four works analyze samples for less than 1 minute, five for 1–30 minutes, and two for 1 hour or more.

3.4. Feature Processing

The feature engineering phase involves parsing and preprocessing raw dynamic analysis data and then applying feature extraction and selection techniques to construct the final feature dataset for model consumption.

Input data. Most works describe the data used to construct the feature dataset only at a high level, typically by indicating broad groups of dynamically collected information such as API function calls, registry key operations, and file operations. Only four papers mention the specific functions or operations used [14, 28, 36, 50]. Other works either provide partial information or none. Notably, only two papers clearly specify the exact information extracted from the raw data (*e.g.*, retaining only the function name while discarding arguments and return values) [4, 41]. While this may seem like a minor detail, its omission can lead to different feature datasets when solutions are reproduced by independent research teams.

Feature engineering and selection. In terms of feature engineering, fourteen works employ established techniques (*e.g.*, binary encoding, N-grams, and TF–IDF), while eleven propose custom approaches or hybrid methods [9, 35, 41, 56]. A similar pattern holds for feature selection, where most studies reporting its use rely on well-known techniques such as mutual information, with three employing custom or hybrid methods [4, 12, 33, 41].

Although many papers describe the formulas and algorithms of these techniques, especially the custom ones, a recurrent limitation is the lack of detail regarding specific parameter settings. For example, many papers that use n-grams fail to specify the final chosen length, even when multiple values were tested [4, 8]. Likewise, the exact number of selected features is seldom reported.

Finally, eight papers provide little to no information regarding preprocessing, feature engineering, or feature selection procedures [19, 22, 24, 27, 29, 36, 38, 53].

Output. Considering the final format of the feature dataset, most works represent features as vectors or matrices, but seven papers do not clearly specify their output structure [10, 19, 23, 25, 38, 50, 53]. While features are predominantly expressed in binary or numerical form, ten papers do not indicate the data type [4, 8, 19, 24, 25, 33, 38, 50, 52, 53]. Finally, only four papers explicitly note the file format, with three using CSV and one using ARFF [19, 25, 35, 55].

3.5. ML Training

For training, researchers typically split the dataset into training and test sets, apply their selected model(s) using established ML/DL frameworks or toolkits, and then refine the model parameters.

Training/Test Sets. While the majority of works report their data-splitting methodology, six fail to specify this information altogether [10, 11, 24, 35, 38, 52]. Among those that do, 10-fold cross-validation is the most prevalent technique [12, 14, 25, 28–30, 33, 36, 41, 45, 49], with occasional use of 4-fold and 5-fold variations [1, 23]. As a simpler alternative, the 80/20 fixed train-test split is widely adopted [3, 4, 8, 9, 23, 48], followed by the 70/30 split [22, 27, 50, 55, 56].

Models. All reviewed papers explicitly report the types of models employed in their detection systems and experiments. The three most frequently used categories are decision-tree-based models (*e.g.*, Decision Trees, Random Forests), appearing in eighteen papers; kernel-based models (*e.g.*, Support Vector Machines), in fourteen papers; and linear models (*e.g.*, Linear or Logistic Regression), in ten papers. While some works focus on a single model type, nearly half evaluate or combine multiple model types [1, 3, 4, 8, 9, 12, 19, 22, 28, 30, 36, 41, 53, 56].

Parameters/Hyperparameters. Despite clear reporting of models, detailed configuration information is scarce. Only five papers fully document their parameters or hyperparameters [1, 35, 48, 49, 55]. An additional three provide incomplete details (*e.g.*, describing the use of Random Forest with 100 trees without specifying other criteria like maximum depth) [8, 14, 25]. The remaining works provide no information on model parameters whatsoever.

Frameworks/Toolkits. Only about half of the reviewed works report the specific libraries, toolkits, or frameworks used for model training. Among those, ten mention ML libraries such as Scikit-learn or MLlib [1, 9, 11, 23, 30, 41, 45, 50, 55, 56], six cite ML toolkits like WEKA [12, 19, 25, 28, 48, 56], and three reference DL frameworks such as TensorFlow or PyTorch [23, 35, 49].

Explainability. Given the critical role of ransomware detection systems, the ability to interpret and explain model decisions is highly important. Despite this, only Minerva [27] mentions the use of explainability techniques, specifically SHAP, to assess feature contributions to the final classification.

3.6. Artifacts

Sharing research artifacts is a cornerstone of open science, directly enabling the community to verify results, avoid redundant work, and build upon existing solutions.

Datasets. Most works create their own datasets, yet only four provide a direct link to them within the paper [14, 30, 50, 53]. The dataset from Elderan [48], although not directly referenced in the paper, is available in a GitHub repository.[4] Of these five datasets, only three remain accessible, as the links provided by Jethva et al. [30] and Continella et al. [14] are no longer operational. Furthermore, only Sloun et al. [50] supplies the final feature dataset (the authors provide a script to generate it rather than the dataset itself), whereas the other works make only the corresponding behavioral datasets available.

Source Code. A similar lack of availability is observed regarding the source code of the proposed solutions. While eleven studies provide pseudocode [3, 4, 14, 28, 30, 33, 41, 45, 49, 52, 56] and three include mathematical formulations [8, 9, 51] to clarify parts of their methodology, only two papers share source code [14, 50]. Of these, only the artifacts from [50] remain accessible. Notably, this work stands out as an exception by providing complete, well-documented source code to reproduce every stage of its pipeline (from feature dataset generation to the final detection system) and by having undergone successful artifact evaluation, achieving an *available*, *functional*, and *reproducible* classification.

Three additional papers share only partial components or auxiliary tools: a tool for simulating user interactions on Windows [28], a modified behavioral analysis tool [35], and a script for extracting features from VirusTotal [19]. DNA-Droid [23] provided its behavioral analysis tool as a service to the community, where users could submit samples for classification. However, the service link is no longer operational. Furthermore, although a GitHub repository for the tool exists, it is not cited in the paper, nor does it reference the publication.[5]

4. Best-Practice Recommendations

Based on the aforementioned observations, we provide guidelines to address key challenges in the validation, reproducibility, and replicability of current ML-based ransomware detection research. Our guidelines build on existing security and malware research principles [7, 15, 42, 43], extending them to ransomware detection where they are often overlooked, and introducing ransomware-specific considerations to address critical gaps.

4.1. Dataset Collection

Sample selection. Our observations show that recent research does not sufficiently account for general guidelines for building correct datasets, while these guidelines are often oblivious to ransomware-specific details.

Guideline 1. Clearly define the research objective (*e.g.*, binary detection, family classification, clustering) and justify the dataset scope accordingly, explaining the rationale for using only ransomware, a ransomware–benign mixture, or a broader dataset that includes other malware.

Guideline 2. Describe the dataset's composition, reasoning on its inter-class balance. For instance, using a balanced class distribution (*e.g.*, 50% ransomware, 50% benign) ensures the model learns effectively from all classes [43]. For evaluation, testing on realistic, imbalanced distributions ensures the model's performance reflects real-world conditions, where benign activity vastly outnumbers ransomware attacks [7, 42].

Guideline 3. Explain how intra-class diversity is addressed, including the selection criteria and distribution of samples within ransomware families, as different families may exhibit distinct operational traits (*e.g.*, differing propagation methods, encryption patterns, or target profiles) as well as evolving behaviors (*e.g.*, data exfiltration) that may impact model learning and generalization.

Guideline 4. Document and justify the selection of benign samples to reflect the intended deployment environment. Include typical applications for that context (*e.g.*, office tools and databases in an enterprise, virtual machines in a cloud data center) and a diverse set of software to ensure generalizability. It should also incorporate applications that exhibit ransomware-like behaviors (*e.g.*, intensive file access or encryption) to adequately validate the model.

Guideline 5. Disclose the specific samples used by: *(i)* listing the ransomware families and providing their collection date for timely relevance, along with sample identifiers (*e.g.*, MD5 hashes); *(ii)* specifying the exact

4. https://github.com/rissgrouphub/ransomwaredataset2016

5. https://github.com/amirhgharib/dnadroid

names and versions of benign applications; and *(iii)* citing the sources from which all samples were obtained [43].

Behavioral analysis. Several works that use their own selected samples to generate a custom behavioral dataset do not provide sufficient information for reproducing the environment to collect behavioral data.

Guideline 6. Fully document the experimental analysis environment, including hardware specifications (*e.g.*, CPU, RAM, storage, network), OS distributions and versions, and the virtualization or sandboxing technology used [43]. These factors directly influence ransomware runtime behavior, such as the number of threads spawned or the speed of file encryption, and therefore determine the behavioral data collected for analysis.

Guideline 7. Clearly describe both the file system environment and the network configuration used during behavioral analysis. To elicit representative I/O behavior (*e.g.*, ransomware may target specific file types or sizes), the file system environment should mimic a typical host, including realistic hierarchies, file extensions, and content. Clearly document the use of existing tools (*e.g.*, Impressions [2]) or novel strategies to generate such states, along with the steps taken to ensure realism, thereby enabling validation and reproducibility. Additionally, specify which network communications were blocked or allowed [43], as these settings directly influence ransomware execution and observed behavior (*e.g.*, enabling or preventing contact with C&C servers for key retrieval or data exfiltration).

Guideline 8. Ensure samples run in environments configured to simulate realistic user usage [43]. This is necessary to capture authentic behavior, as benign software often requires user interaction or realistic workloads for full operation, and many ransomware variants employ anti-analysis checks triggered by user activity (*e.g.*, mouse movement) before deploying their payload.

Guideline 9. Identify the tools used to automate dataset generation, produce workloads, and monitor or trace samples, specifying their configurations and usage steps. Report the type of monitored data (*e.g.*, API calls, network packets, file operations), the sampling time windows (*e.g.*, per event, at fixed intervals), and the output data format.

Guideline 10. Ensure that the sample execution time is sufficient to capture its behavior, as assuming a uniform execution duration across all samples can lead to incomplete or misleading observations. As shown in [32], in many cases shorter execution times (around two minutes) are sufficient and more efficient, enabling the analysis of a larger number of samples. However, longer executions (up to one hour [43]) may be necessary in scenarios where benign applications perform operations gradually, or ransomware introduces execution delays to evade detection.

4.2. Feature engineering, ML training and testing

Feature engineering. While standard feature engineering and selection techniques are commonly applied, the reporting of specific parameters and the final selected features is frequently incomplete or omitted.

Guideline 11. Describe the feature engineering pipeline, including the source of features (*e.g.*, API calls, file system operations), aggregation methods, temporal granularity, and any normalization or encoding steps applied. When applicable, also detail any feature selection or dimensionality reduction techniques used (*e.g.*, PCA, mutual information, model-based selection), including their motivation, configuration parameters, and their impact on the size and composition of the final feature set.

Guideline 12. Specify the final set of selected features, their data format (*e.g.*, binary, numeric), and the structure of the resulting dataset (*e.g.*, a matrix of feature vectors).

ML training. Although model training is a standard phase for all solutions, critical details about parameter settings and hardware are often under-reported, compromising reproducibility and fair comparisons.

Guideline 13. Report the full training configuration, including the model architecture, hyperparameter values, optimization algorithm, random seeds used for model initialization and training, training duration, and the hardware (*e.g.*, RAM, GPU, CPU, disk) used [15].

Guideline 14. Define and justify the dataset splits (training, validation, testing). Prefer temporal or family-wise splits over random splits. Temporal splits should use older samples for training and newer samples for testing, simulating deployment on future, unseen ransomware. Family-wise splits should hold out entire ransomware families to evaluate generalization to unseen variants. This is important because random splits can mix highly correlated samples across splits (*e.g.*, executions of the same family or collection period), leading to data leakage and overly optimistic performance estimates that do not reflect real-world detection scenarios [7, 43].

ML testing and explainability. Testing strategies often lack detail, and the use of explainability techniques is nearly absent, undermining the credibility and interpretability of detection models.

Guideline 15. Justify and describe the experimental evaluation in detail, specifying its goals, test environment, and performance metrics. Performance must be reported using multiple complementary metrics (*e.g.*, precision, recall, F1-score), with explicit attention to false positives and false negatives due to their critical operational impact [7, 15]. In addition, the inference time should be measured and discussed. Ideally, final testing should be conducted on distinct datasets and in varied environments to assess the solution's robustness under unpredictable conditions.

Guideline 16. Comparisons of model performance with prior work must be explicitly categorized as theoretical or experimental. Direct comparison with results reported in prior publications should be avoided when experimental setups differ, as this can lead to invalid or misleading conclusions [7, 42]. When experimental comparisons are performed, the full setup must be reported, including datasets, model configurations, and execution environment.

Guideline 17. Whenever possible, incorporate model interpretability or explanation techniques to justify model decisions, assess whether learned patterns align with meaningful behavioral indicators, and support trust, debugging, and reproducibility of results [7].

TABLE 3. GUIDELINES COMPLIANCE ACROSS WORKS.

Papers	Year	Sample Selection					Behavioral Analysis					Feature Engineering		ML Training		ML Testing & Explainability			Artifacts		
		G1	G2	G3★	G4★	G5	G6★	G7★	G8★	G9	G10★	G11	G12	G13	G14★	G15	G16	G17	G18	G19	G20
[48]	2016	F	P	P	F	P	P	F	N	P	F	P	F	P	P	P	P	N	P	N	N
[14]	2016	F	P	P	F	P	P	F	F	P	F	P	F	P	P	P	–	N	P	P	N
[25]	2017	F	P	P	N	P	P	N	N	P	N	P	N	P	P	P	P	N	N	N	N
[28]	2017	F	P	P	N	P	P	N	F	P	F	F	F	P	P	P	–	N	N	P	N
[12]	2017	F	P	N	P	P	P	N	N	P	F	P	F	P	P	P	–	N	N	N	N
[35]	2017	P	N	N	N	P	P	N	N	P	F	P	F	P	N	P	–	N	N	P	N
[23]	2017	F	P	P	N	P	P	N	F	P	N	P	P	P	F	P	P	N	N	P	N
[45]	2018	F	P	N	N	P	P	N	N	P	P	P	F	P	P	P	–	N	N	N	N
[36]	2018	F	N	N	P	P	N	N	N	P	N	P	F	P	P	P	F	N	N	N	N
[10]	2018	F	P	N	F	P	P	N	N	N	N	N	P	P	N	P	–	N	N	N	N
[51]	2018	F	P	N	P	P	P	N	N	P	N	P	F	P	P	P	P	N	N	N	N
[22]	2018	F	P	P	N	P	P	N	F	P	N	P	F	P	F	P	–	N	N	N	N
[38]	2019	F	P	N	N	N	N	N	N	P	N	N	N	P	N	P	F	N	N	N	N
[9]	2019	F	P	N	N	P	P	N	N	P	F	P	F	P	P	P	N	N	N	N	N
[8]	2019	F	P	N	N	P	P	N	N	P	N	P	P	P	P	P	–	N	N	N	N
[11]	2019	F	P	N	P	P	N	N	F	P	N	P	F	P	N	P	–	N	N	N	N
[19]	2019	F	P	N	N	P	N	P	N	P	N	P	N	P	P	P	–	N	N	P	N
[24]	2019	F	N	N	P	P	N	N	N	N	N	N	P	P	N	P	–	N	N	N	N
[33]	2019	F	P	N	P	P	–	–	–	–	–	P	P	P	F	P	N	N	N	N	N
[41]	2019	F	P	F	P	P	P	N	N	P	N	F	F	P	P	P	P	N	N	N	N
[1]	2020	F	P	P	N	P	–	–	–	–	–	P	F	P	P	P	P	N	N	N	N
[30]	2020	F	P	P	F	P	F	F	N	P	F	F	F	P	P	P	–	N	P	N	N
[29]	2020	F	P	N	N	P	N	N	N	P	N	P	F	P	P	P	–	N	N	N	N
[56]	2020	F	P	F	N	P	N	N	N	P	N	P	F	P	P	P	N	N	N	N	N
[49]	2020	F	P	P	N	P	N	N	N	P	P	F	F	P	P	P	–	N	N	N	N
[3]	2020	F	P	P	P	P	N	F	F	P	P	F	F	P	P	P	N	N	N	N	N
[4]	2020	F	P	P	P	P	N	F	F	P	F	F	P	P	P	P	N	N	N	N	N
[55]	2020	F	P	P	N	P	N	N	N	P	N	P	F	P	P	P	N	N	N	N	N
[52]	2024	F	P	P	P	P	N	N	N	P	N	F	P	P	N	P	N	N	N	N	N
[53]	2024	F	F	P	F	P	P	N	N	P	N	P	N	P	P	P	N	N	P	N	P
[50]	2025	F	P	P	P	P	–	–	–	–	–	P	N	P	P	P	–	N	F	F	F
[27]	2025	F	P	P	F	P	N	P	N	P	N	P	F	P	P	P	P	F	N	N	N

F Full compliance P Partial compliance (*i.e.*, meets at least one aspect of the guideline) N No compliance – Not applicable
★ Highlights guidelines that require special attention to the specific characteristics of ransomware

4.3. Artifacts

The public release of research artifacts is a critical step for ensuring reproducibility. Our observations show this practice remains exceptionally rare in the field.

Guideline 18. Thoroughly document and accurately label all datasets (sample, behavioral, and feature) to enable reuse. Where possible, share the original sample datasets or provide a list of sample hashes for reconstruction. Share raw behavioral datasets before preprocessing to preserve critical information and enable the creation of multiple feature datasets [15]. Share the final feature datasets to maximize reproducibility and reduce redundant effort.

Guideline 19. Release the solution code along with comprehensive documentation of the required hardware, dependencies, and external tools needed to reproduce the full pipeline [15]. To maximize reproducibility, automate the entire development and evaluation process using executable scripts, provisioning tools (*e.g.*, Ansible, Terraform), or containerized environments (*e.g.*, Docker).

Guideline 20. Use a permanent public archive with irrevocable versioning and long-term storage, such as Zenodo, to deposit the final artifact (code, datasets, documentation). This ensures long-term availability and citability through a persistent DOI. Cross-reference the paper in the artifact's documentation and the artifact's DOI in the paper.

4.4. Discussion

Table 3 correlates the guidelines with the analyzed works, providing an overview of how existing solutions conform to these recommendations in practice.

The research objective and scope (G1) and the final features format (G12) are mostly described across all analyzed works, *i.e.*, the works are fully compliant with these guidelines. In contrast, the reasoning about inter-class balance (G2), the sample disclosure and provenance (G5), the tooling and data collection pipeline (G9), the training configuration (G13), and the evaluation methodology and metrics (G15) are poorly attained, *i.e.*, while most papers provide some details, none fully satisfy all requirements of these guidelines.

For file system state and network configuration used during behavioral analysis (G7), realistic user interaction conditions (G8), and the reasoning about sample execution time (G10), the results are more polarized, with works either fully compliant or not compliant, suggesting a clear distinction between works that explicitly address experimental realism and those that omit it entirely. A similar pattern is observed for intra-class diversity (G3) and behavioral analysis environment specification (G6), which are mostly partially achieved or not achieved, reflecting inconsistent reporting of dataset diversity and experimental infrastructure.

Finally, most papers do not comply with the model explainability (G17) and dataset/code sharing (G18–G20) guidelines, with only two papers (one for each) achieving full compliance. Notably, these are the only guidelines in which a (albeit slight) temporal trend can be observed, likely driven by increasing community attention to model interpretability and reproducibility, as well as by growing encouragement from conferences and journals for artifact submission and open science practices. In contrast, no clear temporal evolution is observed for the remaining guidelines.

5. Conclusion

We reviewed ML–based ransomware detection research proposed over the past decade and found that many lack sufficient methodological detail and artifact availability, which substantially limit the reproducibility and reusability of the proposed solutions. By proposing new guidelines and synthesizing existing ones from related fields, while adapting them to the specific challenges of ransomware detection, we aim to steer future research toward more reproducible and replicable practices.

We recognize that achieving full transparency and data sharing in this domain can be non-trivial due to possible operational considerations and infrastructure constraints surrounding malware handling and data distribution. Nevertheless, improved methodological documentation, at least partial artifact release, and adherence to common guidelines remain feasible steps that can substantially enhance reproducibility and replicability.

Further, ensuring that all guidelines are met can be time-consuming and error-prone, especially when done manually. Thus, given the significant redundancy observed across independent research efforts, we argue that developing dedicated, reusable tools is essential to advancing the field. Future work should focus on developing or exploring new frameworks to systematically share, extend, curate, and properly document existing datasets and ML pipelines. Such shared resources would accelerate progress and markedly improve the rigor and overall quality of research in this critical domain.

Acknowledgment

We thank our shepherd and the anonymous reviewers for their insightful comments and feedback. This work was co-funded by the European Regional Development Fund (ERDF) through the NORTE 2030 Regional Programme under Portugal 2030, within the scope of the project Rescueware, reference 21746 (NORTE2030-FEDER-02306900) (*João Paulo*); by national funds through FCT - Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (https://doi.org/10.54499/UID/50014/2025) (*Tânia Esteves and Bruno Pereira*) and within the Ph.D. grants 2025.00966.BD (*Bruno Pereira*) and 2025.03394.BDANA (*Beatriz Cepa*); and by the Internal Funds KU Leuven and by the Cybersecurity Research Program Flanders (*Vera Rimmer*).

References

[1] Muhammad Shabbir Abbasi, Harith Al-Sahaf, and Ian Welch. “Particle swarm optimization: A wrapper-based feature selection method for ransomware detection and classification”. In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer. 2020, pp. 181–196. DOI: 10.1007/978-3-030-43722-0_12.

[2] Nitin Agrawal, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. “Generating realistic impressions for file-system benchmarking”. In: *ACM Transactions on Storage (TOS)* 5.4 (2009), pp. 1–30. DOI: 10.1145/1629080.1629086.

[3] Yahye Abukar Ahmed, Baris Kocer, and Bander Ali Saleh Al-rimy. “Automated analysis approach for the detection of high survivable ransomware”. In: *KSII Transactions on Internet and Information Systems (TIIS)* 14.5 (2020), pp. 2236–2257. DOI: 10.3837/tiis.2020.05.021.

[4] Yahye Abukar Ahmed et al. “A system call refinement-based enhanced Minimum Redundancy Maximum Relevance method for ransomware early detection”. In: *Journal of Network and Computer Applications* 167 (2020), p. 102753. DOI: 10.1016/j.jnca.2020.102753.

[5] Saleh Alzahrani et al. “A Survey of Ransomware Detection Methods”. In: *IEEE Access* 13 (2025), pp. 57943–57982. DOI: 10.1109/ACCESS.2025.3556187.

[6] Nicoló Andronio, Stefano Zanero, and Federico Maggi. “HelDroid: Dissecting and Detecting Mobile Ransomware”. In: *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, 2015, pp. 382–404. DOI: 10.1007/978-3-319-26362-5_18.

[7] Daniel Arp et al. “Dos and Don’ts of Machine Learning in Computer Security”. In: *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022, pp. 3971–3988. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/arp.

[8] Arslan Ashraf et al. “Ransomware Analysis using Feature Engineering and Deep Neural Networks”. In: *CoRR* abs/1910.00286 (2019). DOI: 10.48550/arXiv.1910.00286.

[9] Seong Il Bae, Gyu Bin Lee, and Eul Gyu Im. “Ransomware detection using machine learning algorithms”. In: *Concurrency and Computation: Practice and Experience* 32.18 (2020). e5422 cpe.5422, e5422. DOI: https://doi.org/10.1002/cpe.5422.

[10] SungHa Baek et al. “SSD-Insider: Internal Defense of Solid-State Drive against Ransomware with Perfect Data Recovery”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 875–884. DOI: 10.1109/ICDCS.2018.00089.

[11] Qian Chen et al. “Automated Ransomware Behavior Analysis: Pattern Extraction and Early Detection”. In: *Science of Cyber Security*. Ed. by Feng Liu et al. Springer International Publishing, 2019, pp. 199–214. DOI: 10.1007/978-3-030-34637-9_15.

[12] Zhi-Guo Chen et al. “Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph”. In: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems (RACS ’17)*. Association for Computing Machinery, 2017, pp. 196–201. DOI: 10.1145/3129676.3129704.

[13] Contagio. *Contagio Malware Dump*. Accessed: April 2026. URL: https://contagiodump.blogspot.com/.

[14] Andrea Continella et al. “ShieldFS: a self-healing, ransomware-aware filesystem”. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC ’16’)*. Association for Computing Machinery, 2016, pp. 336–347. DOI: 10.1145/2991079.2991110.

[15] Nadia Daoudi et al. “Lessons Learnt on Reproducibility in Machine Learning Based Android Malware Detection”. In: *Empirical Software Engineering* 26.4 (2021), p. 74. DOI: 10.1007/s10664-021-09955-7.

[16] Abhyuday Desai, Mohamed Abdelhamid, and Nakul R. Padalkar. “What is reproducibility in artificial intelligence and machine learning research?” In: *AI Magazine* 46.2 (2025), e70004. DOI: 10.1002/aaai.70004.

[17] Android Developers. *UI/Application Exerciser Monkey*. Accessed: April 2026. URL: https://developer.android.com/studio/test/other-testing-tools/monkey.

[18] John R. Douceur and William J. Bolosky. “A large-scale study of file-system contents”. In: *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS ’99)*. Vol. 27. 1. Association for Computing Machinery, 1999, pp. 59–70. DOI: 10.1145/301453.301480.

[19] Samuel Egunjobi, Simon Parkinson, and Andrew Crampton. “Classifying Ransomware Using Machine Learning Algorithms”. In: *Intelligent Data Engineering and Automated Learning (IDEAL 2019)*. Springer International Publishing, 2019, pp. 45–52. DOI: 10.1007/978-3-030-33617-2_5.

[20] Tânia Esteves et al. “CRIBA: A Tool for Comprehensive Analysis of Cryptographic Ransomware’s I/O Behavior”. In: *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*. 2023, pp. 46–58. DOI: 10.1109/SRDS60354.2023.00015.

[21] Jannatul Ferdous et al. “AI-Based Ransomware Detection: A Comprehensive Review”. In: *IEEE Access* 12 (2024), pp. 136666–136695. DOI: 10.1109/ACCESS.2024.3461965.

[22] Alberto Ferrante et al. “Extinguishing Ransomware - A Hybrid Approach to Android Ransomware Detection”. In: *Foundations and Practice of Security*. Springer International Publishing, 2018, pp. 242–258. DOI: 10.1007/978-3-319-75650-9_16.

[23] Amirhossein Gharib and Ali Ghorbani. “DNA-Droid: A Real-Time Android Ransomware Detection Framework”. In: *Network and System Security*. Springer International Publishing, 2017, pp. 184–198. DOI: 10.1007/978-3-319-64701-2_14.

[24] Parth S. Goyal et al. “Crypto-Ransomware Detection Using Behavioural Analysis”. In: *Reliability, Safety and Hazard Assessment for Risk-Based Technologies*. Springer Singapore, 2020, pp. 239–251. DOI: 10.1007/978-981-13-9008-1_20.

[25] Md Mahbub Hasan and Md. Mahbubur Rahman. “RansHunt: A support vector machines based ransomware analysis framework with integrated feature set”. In: *2017 20th International Conference of Computer and Information Technology (ICCIT)*. 2017, pp. 1–7. DOI: 10.1109/ICCITECHN.2017.8281835.

[26] B. J. Hicks et al. “Organizing and managing personal electronic files: A mechanical engineer’s perspective”. In: *ACM Transactions on Information Systems (TOIS)* 26.4 (2008). DOI: 10.1145/1402256.1402262.

[27] Dorjan Hitaj et al. “Minerva: A File-Based Ransomware Detector”. In: *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (ASIA CCS ’25)*. Association for Computing Machinery, 2025, pp. 576–590. DOI: 10.1145/3708821.3733867.

[28] Sajad Homayoun et al. “Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence”. In: *IEEE Transactions on Emerging Topics in Computing* 8.2 (2020), pp. 341–351. DOI: 10.1109/TETC.2017.2756908.

[29] Jinsoo Hwang et al. “Two-stage ransomware detection using dynamic analysis and machine learning techniques”. In: *Wireless Personal Communications* 112.4 (2020), pp. 2597–2609. DOI: https://doi.org/10.1007/s11277-020-07166-9.

[30] Brijesh Jethva et al. “Multilayer ransomware detection using grouped registry key operations, file entropy and file signature monitoring”. In: *Journal of Computer Security* 28.3 (2020), pp. 337–373. DOI: 10.3233/JCS-191346.

[31] Koodous. Accessed: April 2026. 2015. URL: https://koodous.com/?tab=koodous.

[32] Alexander Küchler et al. “Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes”. In: *NDSS 2021, Network and Distributed Systems Security Symposium*. ISOC. Internet Society, 2021. DOI: 10.14722/ndss.2021.24475. URL: https://hal.science/hal-04611612.

[33] Abdullahi Maigida et al. “An Intelligent CryptoLocker Ransomware Detection Technique Using Support Vector Machine Classification And Grey Wolf Optimization Algorithms”. In: *i-manager’s Journal on Software Engineering* 13 (Mar. 2019). DOI: 10.26634/jse.13.3.15685.

[34] Davide Maiorca et al. “R-PackDroid: API package-based characterization and detection of mobile ransomware”. In: *Proceedings of the Symposium on Applied Computing*. SAC ’17. Association for Computing Machinery, 2017, pp. 1718–1723. DOI: 10.1145/3019612.3019793.

[35] Sumith Maniath et al. “Deep learning LSTM based ransomware detection”. In: *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*. 2017, pp. 442–446. DOI: 10.1109/RDCAPE.2017.8358312.

[36] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. “RWGuard: A Real-Time Detection System Against Cryptographic Ransomware”. In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Michael Bailey et al. Springer International Publishing, 2018, pp. 114–136. DOI: https://doi.org/10.1007/978-3-030-00470-5_6.

[37] Harun Oz et al. “A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions”. In: *ACM*

Comput. Surv. 54.11s (2022). DOI: 10.1145/3514229.

[38] Jisung Park et al. "RansomBlocker: a Low-Overhead Ransomware-Proof SSD". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Association for Computing Machinery, 2019. DOI: 10.1145/3316781.3317889.

[39] Joelle Pineau et al. "Improving Reproducibility in Machine Learning Research(A Report from the NeurIPS 2019 Reproducibility Program)". In: *Journal of Machine Learning Research* 22.164 (2021), pp. 1–20. URL: http://jmlr.org/papers/v22/20-303.html.

[40] Edward Raff et al. "What Do Machine Learning Researchers Mean by "Reproducible"?" In: *Proceedings of the AAAI Conference on Artificial Intelligence* 39.27 (2025), pp. 28671–28683. DOI: 10.1609/aaai.v39i27.35093.

[41] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. "Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection". In: *Future Generation Computer Systems* 101 (2019), pp. 476–491. DOI: https://doi.org/10.1016/j.future.2019.06.005.

[42] Emilio Rios-Ochoa et al. "Ransomware Family Attribution With ML: A Comprehensive Evaluation of Datasets Quality, Models Comparison, and a Simulated Deployment". In: *IEEE Access* 13 (2025), pp. 108108–108126. DOI: 10.1109/ACCESS.2025.3580395.

[43] Christian Rossow et al. "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook". In: *2012 IEEE Symposium on Security and Privacy*. 2012, pp. 65–79. DOI: 10.1109/SP.2012.14.

[44] Sajadhomayoun. *PyWinMonkey: Performing random activities on a windows program*. Accessed: April 2026. URL: https://github.com/sajadhomayoun/PyWinMonkey.

[45] Bander Ali Saleh Al-rimy et al. "Zero-Day Aware Decision Fusion-Based Model for Crypto-Ransomware Early Detection". In: *International Journal of Integrated Engineering* 10 (Nov. 2018). DOI: 10.30880/ijie.2018.10.06.011.

[46] Security Boulevard. *Inside the Ingram Micro Ransomware Attack: Lessons in Zero Trust*. Accessed: April 2026. 2025. URL: https://securityboulevard.com/2025/11/inside-the-ingram-micro-ransomware-attack-lessons-in-zero-trust/.

[47] Harald Semmelrock et al. "Reproducibility in machine-learning-based research: Overview, barriers, and drivers". In: *AI Magazine* 46.2 (2025), e70002. DOI: https://doi.org/10.1002/aaai.70002.

[48] Daniele Sgandurra et al. "Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection". In: *CoRR* abs/1609.03020 (2016). arXiv: 1609.03020. URL: http://arxiv.org/abs/1609.03020.

[49] Shaila Sharmeen et al. "Avoiding Future Digital Extortion Through Robust Protection Against Ransomware Threats Using Deep Learning Based Adaptive Approaches". In: *IEEE Access* 8 (2020), pp. 24522–24534. DOI: 10.1109/ACCESS.2020.2970466.

[50] Christian van Sloun et al. "Detecting Ransomware Despite I/O Overhead: A Practical Multi-Staged Approach". In: *Network and Distributed System Security Symposium*. Vol. 978. 3543-3545. Internet Society. 2025, pp. 3543–3545. DOI: https://dx.doi.org/10.14722/ndss.2025.240764.

[51] Yuki Takeuchi, Kazuya Sakai, and Satoshi Fukumoto. "Detecting Ransomware using Support Vector Machines". In: International Conference on Parallel Processsing (ICPP) Workshops '18. Association for Computing Machinery, 2018. DOI: 10.1145/3229710.3229726.

[52] Shenao Wang et al. "CanCal: Towards Real-time and Lightweight Ransomware Detection and Response in Industrial Environments". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Association for Computing Machinery, 2024, pp. 2326–2340. DOI: 10.1145/3658644.3690269.

[53] Zhongyu Wang et al. "Ransom Access Memories: Achieving Practical Ransomware Protection in Cloud with DeftPunk". In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024, pp. 687–702. URL: https://www.usenix.org/conference/osdi24/presentation/wang-zhongyu.

[54] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. 1st. O'Reilly Media, Inc., 2018. URL: https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/.

[55] Jiaxing Zhou et al. "Evaluation to Classify Ransomware Variants based on Correlations between APIs". In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,* INSTICC. SciTePress, 2020, pp. 465–472. DOI: 10.5220/0008959904650472.

[56] Hiba Zuhair, Ali Selamat, and Ondrej Krejcar. "A Multi-Tier Streaming Analytics Model of 0-Day Ransomware Detection Using Machine Learning". In: *Applied Sciences* 10.9 (2020). DOI: 10.3390/app10093210.