An Empirical Study of Compression Strategies for Tricorder: From General-Purpose Compressors to Shannon Entropy

Caio Guimaraes Herrera(✉), Hugo H. Nakamura, Isaac S. Soares, Simone R. S. Souza, and Paulo Sergio Lopes de Souza

University of Sao Paulo (ICMC/USP), Sao Carlos, SP, Brazil
caio.herrera@outlook.com

Abstract. Software Reliability Engineering (SRE) requires robust mechanisms to detect anomalies in production systems. The Tricorder methodology addresses this need by monitoring resource consumption and applying the DAMICORE clustering algorithm, which relies on the Normalized Compression Distance (NCD). However, computing the NCD is an expensive step, and its output is ultimately discarded, motivating the investigation of more efficient alternatives. This paper evaluates a set of compression strategies for the NCD metric in the context of Tricorder: seven general-purpose and multimedia compressors (ZLIB, GZIP, PPMD, BZ2, PNG, WEBP, JP2, HEIF), as well as an entropy-based approach (ENTROPY) that estimates compressed size from Shannon Entropy without producing compressed output. We conducted experiments using monitoring profiles from the Decision Tree application, with seven types of injected software faults evaluated against standard classification metrics (accuracy, precision, recall, and F1-score) and response time. Results show that ZLIB, GZIP, PPMD, PNG, and ENTROPY all achieve maximum effectiveness, correctly detecting all anomalies without false positives. ENTROPY additionally presents response times and per-iteration times consistently below the median across all anomaly types, matching or outperforming traditional compressors in efficiency. Most multimedia compressors introduced false positives in the control group, limiting their applicability. These findings establish Shannon Entropy as a viable, low-overhead alternative to traditional compression for NCD-based anomaly detection, particularly in resource-constrained or operationally restricted environments.

Keywords: Software reliability · Compression strategies · Anomaly detection

1 Introduction

Software Reliability Engineering (SRE) aims to ensure that production systems maintain high levels of availability, performance, and resilience. Several failures,

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16769, pp. 243–260, 2027.
https://doi.org/10.1007/978-3-032-30494-0_16

however, do not manifest in conventional tests carried out in development environments, emerging only under real operational conditions [3].

The **Tricorder** methodology contributes to SRE in this context, as it enables the detection of anomalies during the execution of production systems [20]. **Tricorder** has two main phases: first, it monitors the software in production; then, it analyzes the collected consumption profiles through unsupervised machine learning. The latter is done by using **DAMICORE**, a methodology that relies on compressors, or compression strategies, to cluster monitoring profiles and determine a possible anomaly [23]. This step is of great importance, since the cost of compressors may hinder the widespread use of **Tricorder**.

In the **DAMICORE** methodology, compressors are used to calculate the Normalized Compression Distance (**NCD**) from the size of the compressed data. This is the most costly step in its execution. In contrast, the resulting compressed content is discarded as it has no utility in the anomaly detection process. This highlights an opportunity to reduce or eliminate the overhead of generating compressed output through alternative compression strategies.

This paper presents a comprehensive evaluation of alternative compressors for computing the **NCD** metric, as well as an entropy-based strategy for estimating the size of compressed data without generating the output. It compares these approaches with the ones used in previous studies, considering both effectiveness and efficiency in anomaly detection.

The adopted methodology consists of evaluating different compression strategies in the context of **Tricorder**, based on monitoring data previously collected from executions of an application with and without injected faults. The consumption profiles containing the monitoring data were clustered using **DAMICORE**, and each execution's results and response times were stored for analysis. The evaluation comprised both standard classification metrics (accuracy, precision, recall, and F1-score) and response time.

This paper makes two main contributions to unsupervised anomaly detection in the SRE domain. First, it provides a systematic comparison of nine compression strategies for NCD-based anomaly detection, from general-purpose and multimedia to entropy-based approaches; Second, it presents empirical evidence that *Shannon Entropy* is a viable alternative to traditional compressors, maintaining full detection accuracy while reducing computational overhead.

This paper is organized as follows. Section 2 describes the Tricorder methodology, used as the basis for anomaly detection. Section 3 presents the related work from the literature. Section 4 details the evaluated compression strategies, including traditional compressors and the alternative approach based on entropy calculation. Section 5 presents the experimental design, Sect. 6 shows the experimental results, and Sect. 7 analyzes and interprets the findings. Finally, Sect. 8 discusses the findings and limitations of this work, as well as directions for future research.

2 Background

Tricorder is a methodology for detecting software anomalies in production, based on the analysis of resource consumption profiles [20]. These profiles are represented by CSV files, created during the monitoring of an application's execution, containing resource usage metrics such as CPU and GPU utilization, RAM usage and I/O usage.

An initial version of the application, with no known faults, is considered the baseline for the methodology. From this version onward, resource consumption profiles are collected as a reference for anomaly detection. Figure 1 illustrates the behavior of **Tricorder** in this context.

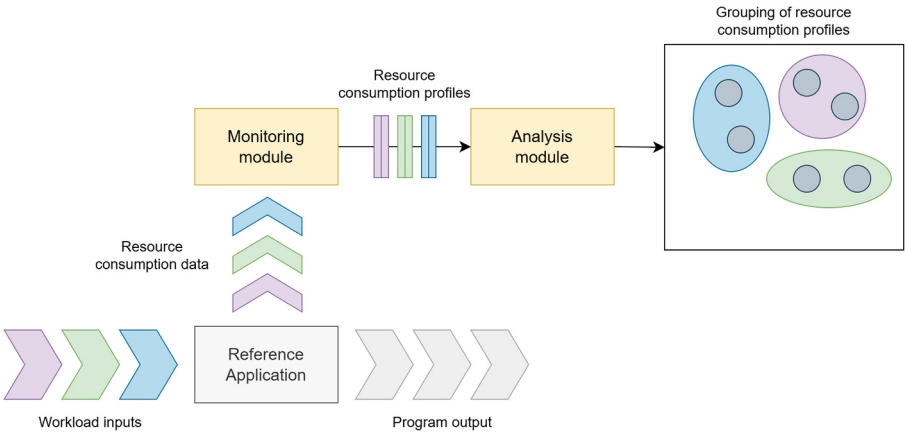

Fig. 1. Tricorder expected behavior when running a reference application. Adapted from Herrera et al. [13].

Then, profiles of a new version of the application, obtained after a maintenance cycle or a new release, are collected and grouped with their predecessors from the baseline version, generating clusters of application behaviors. The **Tricorder** methodology indicates a possible anomalous execution when a cluster made exclusively of profiles from the new version of the application is found. This process is illustrated in Fig. 2.

The clustering approach used by the **Tricorder** methodology is **DAMICORE** [23], a methodology to cluster generic data, through its *Python* implementation, *DamicorePy* [8]. This approach consists of three steps:

1. Construction of a distance matrix for each pair of inputs;
2. Transformation of the distance matrix into a phylogenetic tree;
3. Hierarchical clustering from the phylogenetic tree.

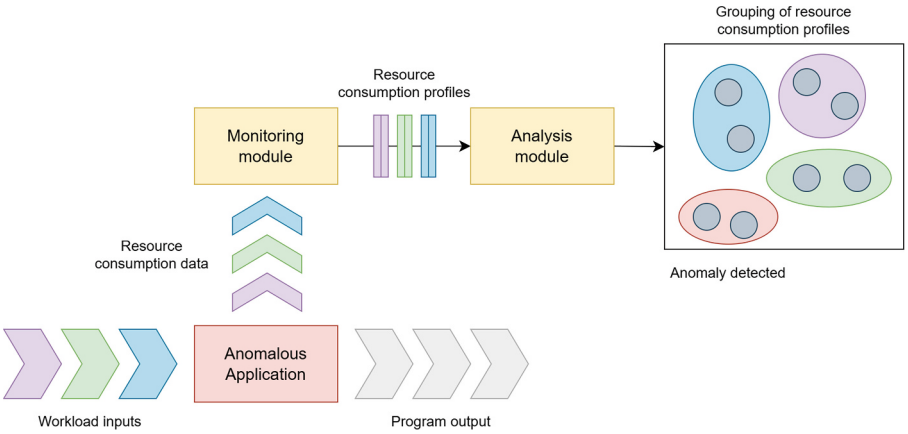

Fig. 2. Tricorder expected behavior when running an anomalous application. Adapted from Herrera et al. [13].

Constructing the distance matrix is the most time-consuming step. It is performed by calculating the **Normalized Compression Distance** (**NCD**) [9,18] between each pair of profiles. The **NCD** is a quantitative similarity metric, calculated between each pair of files, based on their compression. It operates by separately computing the compressed sizes of both files and their concatenation, as presented in Eq. 1.

$$NCD_z(x,y) = \frac{Z(xy) - \min\{Z(x), Z(y)\}}{\max\{Z(x), Z(y)\}} \tag{1}$$

Here, x and y represent the input files, xy their concatenation, and $Z(\alpha)$ the function that returns the compressed size of a file α. The resulting value $NCD_z(x,y) \in [0,1]$ where 0 indicates maximum similarity and 1 indicates minimum similarity.

The performance of the **Tricorder** methodology is closely associated with the choice of compressor used to compute the **NCD** metric, since it determines the generation of the distance matrix and, consequently, the quality of the clusters, directly affecting the detection of software anomalies.

The **Tricorder** methodology, described here, is different from the Tricorder platform proposed by Sadowski et al. [21]. The latter is a supporting platform for developers when conducting program static analysis, and such a project is not related to the methodology we use; it shares only the name.

3 Related Work

The scientific literature establishes that selecting a data compressor is a complex task with no global optimal solution. Performance is dictated by a fundamental trade-off between compression ratio and response time for compression [14],

and it is highly dependent on the intrinsic characteristics of the data, such as repetitiveness and block size [29].

This complexity is also present in the use of compressors in similarity metrics, such as the **NCD**, which is central to this study. Theory suggests that properties of compressors, such as idempotency ($C(xx) \approx C(x)$), are crucial for the robustness of **NCD**. However, practical limitations, such as the compression window size, may introduce significant deviations [7]. Indeed, empirical investigations demonstrate that good theoretical performance of a compressor does not guarantee the best results in practice, underlining the need for contextualized validation [1].

The theoretical pillar of **NCD** is its role as a practical approximation of *Kolmogorov Complexity*, which describes the information contained in an object by the size of the smallest program that generates it. The fundamental relationship between this theory and *Shannon Entropy* is explored in depth by Teixeira et al. [27]. Their work reinforces the theorem that establishes an equivalence between the expected value of *Kolmogorov Complexity* and *Shannon Entropy* for a computable distribution, formalized by the relation presented in Eq. 2. The main conclusion of their study is that this elegant equivalence is an exclusive property of *Shannon Entropy*, not holding for other generalizations such as Rényi and Tsallis entropies, which provides theoretical support for the approach proposed in this paper.

$$0 \leq \sum_{x} P(x)K(x) - H(P) \leq K(P) \tag{2}$$

These challenges have a direct practical impact on anomaly detection. The works of Braga [5] and Durães [11], for example, apply the **Tricorder** methodology with different compressors, highlighting how the choice of the compression algorithm directly impacts performance. The analysis of different compressors for applications such as the *Decision Tree* [24], used in Braga's experiments, provided the direct motivation for the in-depth comparative investigation carried out in this paper.

In this context, the investigations presented here differ from related work by conducting an in-depth empirical analysis. This paper investigates the impact of a set of compressors, lossless and lossy, as well as a new approach, which employs the calculation of *Shannon Entropy* as a replacement for traditional compressors, in computing the **NCD**. Additionally, the effectiveness and efficiency of the **Tricorder** methodology are evaluated to fill a gap in its validation and to expand the understanding of the impact of different approaches on anomaly detection.

4 Compression Strategies

For the experimental stage of this work, a set of compressors was defined, based on a selection that prioritized two fundamental criteria: the representativeness of the main algorithmic paradigms (dictionary, block transformation, and statistical [22]) and the relevance of each compressor as a *de facto* standard in

computational practice [14]. Both lossless and lossy compressors were chosen to investigate whether lossy strategies could also be viable for **Tricorder** and to determine the benefits of their use in terms of effectiveness and efficiency.

Within the dictionary-based compression paradigm, the *DEFLATE* algorithm [17] operates by combining the *LZ77* technique [30] with *Huffman coding* [15]. *LZ77* replaces repeated data sequences with references to previous occurrences, using a sliding window to keep track of recently processed data. When it encounters a previously seen sequence, the algorithm replaces it with a pair of values indicating the distance and length of the original sequence [30]. Then, it compresses the remaining symbols, the literal characters, and the references generated by *LZ77* using *Huffman coding*, assigning variable-length codes based on frequency of occurrence [15]. Its main implementations, the **ZLIB** library and the **GZIP** tool [10], consolidated this approach as a *de facto* standard, being fundamental in countless contexts, from web data transfer to the structuring of different file formats [12]. The **PNG** format [4], a lossless compression format also based on the *DEFLATE* algorithm, applies predictive filters as preprocessing that transform image data to highlight repetitive patterns, thereby improving its compression efficiency.

The **WEBP** compressor supports two modes: lossy, based on the intra-frame prediction of the *VP8* codec [2], and lossless, which uses spatial prediction, color cache, and blocks. In lossless mode, these techniques reduce image-specific redundancies before the final compression with *DEFLATE*, similar to **ZLIB**. Spatial prediction estimates pixels from their neighbors, the cache reuses recent colors, and blocks capture local patterns, achieving higher compression than simple repeated-sequence substitution.

In the block transformation paradigm, Bzip2 (**BZ2**) [25] serves as a reference for its high compression ratios [14]. Its approach consists of applying the Burrows-Wheeler Transform (BWT) [6], a reversible permutation that reorders the characters of a block based on the lexicographic ordering of all its cyclic rotations. This process, although it does not compress the data directly, effectively groups identical symbols, creating high locality of reference. The transformed block is then processed by the Move-to-Front (MTF) transformation [6], which generates a sequence of low-value integers, efficiently compressed at the end using *Huffman coding*.

As a representative of the statistical paradigm, the Prediction by Partial Matching algorithm (**PPM**) [19] operates predictively, building a dynamic probabilistic model from the data stream. To encode a symbol, **PPM** analyzes the immediately preceding characters (the "context") to calculate the probability of the next symbol, which is then encoded with fewer bits by an arithmetic coder. One of its best-known implementations is **PPMd**. The effectiveness of **PPM** on data with clear statistical structure, such as text, makes it a key reference for evaluating contextual information modeling [22].

Beyond the paradigms described above, several image compressors rely on transform-based approaches. The **JPEG 2000** (**JP2**) compressor is an evolution of **JPEG**, offering both lossy and lossless compression, support for larger

images, progressive decoding, greater error tolerance, and processing efficiency. It uses the *Discrete Wavelet Transform* (DWT) for lossy compression and the *Integer Reversible Wavelet Transform* for lossless compression, allowing it to switch between maximum quality and high compaction. After the transformation, the coefficients are quantized (in lossy mode) and encoded by *EBCOT* (Embedded Block Coding with Optimal Truncation), which organizes independent blocks and generates a scalable stream [16].

The **HEIF** (High Efficiency Image File Format) uses the *HEVC/H.265* codec for lossy compression, segmenting images into blocks, applying transforms to remove redundancies, and encoding coefficients by entropy [28]. Unlike **ZLIB**, a generic lossless compressor, **HEIF** is optimized for images, achieving much higher compression ratios with controlled loss while also supporting multiple images, transparency, and metadata.

In contrast to compressors that estimate complexity through algorithmic heuristics, this work evaluates another approach, here referred to as **ENTROPY**, which represents a purely statistical measure of complexity. Grounded in Claude Shannon's seminal work on Information Theory [26], this measure corresponds to the calculation of zero-order *Shannon Entropy*. Entropy, $H(X)$, quantifies the uncertainty or "information content" of a data source based solely on the probability distribution $p(x_i)$ of its symbols, as expressed in Eq. 3.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{3}$$

Unlike compressors, which search for specific patterns (such as literal repetitions or contexts), **ENTROPY** provides a mathematical measure that summarizes the information content of the file, without producing compressed output. Table 1 summarizes the evaluated compressors discussed in this section.

Table 1. Selected compression strategies to evaluate against the **NCD** metric.

Name	Used for	Paradigm	Data loss
ZLIB	Generic data	Dictionary-based	Lossless
GZIP	Generic data	Dictionary-based	Lossless
PPMD	Generic data	Statistical	Lossless
BZ2	Generic data	Block transformation	Lossless
PNG	Multimedia	Dictionary-based	Lossless
WEBP-LS	Multimedia	Dictionary-based	Lossless
WEBP-LY	Multimedia	Block transformation	Lossy
JP2-LS	Multimedia	Block transformation	Lossless
JP2-LY	Multimedia	Block transformation	Lossy
HEIF	Multimedia	Block transformation	Lossy
ENTROPY	Theoretical approximation	Statistical measure	N/A

5 Experimental Studies

We conducted experiments inside an Ubuntu 22.04.5 LTS environment running via WSL2 (Windows Subsystem for Linux 2). It was hosted on a computer with an 8-core Intel Core i7-10510U @ 1.80 GHz, 8 GB DDR4 RAM, and Windows 11 (24H2).

The experiment conducted in this paper used monitoring data already made available by Braga [5]. The profiles refer to the *Decision Tree* application, loaded with the *Breast Cancer* and *Phoneme* datasets [24]. The *Decision Tree* application was chosen because it has heavy workloads (longer response times) and presents stochastic behavior, which makes the clustering performed by Tricorder non-trivial [5]. From the original *Decision Tree* code, different faulty versions were created, each with the insertion of a specific software fault:

- **LOGIC**: logic errors. Low impact on resource consumption;
- **MODEL**: machine learning configuration problems. Low impact on resource consumption;
- **MEMORY**: memory usage problems. Low impact on resource consumption;
- **API**: missing or misplaced API calls. Medium impact on resource consumption;
- **TRAIN**: missing training data. Medium impact on resource consumption;
- **CONC**: synchronization problems. High impact on resource consumption; and
- **PROCESS**: problems in object initialization or method usage. High impact on resource consumption.

For clustering to be performed appropriately, it was necessary to obtain consumption profiles that reflect the expected execution of the selected applications. Therefore, the **CONTROL** and **ANOTHER-CONTROL** types were defined, where **CONTROL** is used as a baseline, and **ANOTHER-CONTROL** is used to verify false positives in anomaly detection. The expected result is that **ANOTHER-CONTROL** is not flagged by **Tricorder** as an anomaly.

In the context of **Tricorder**, the anomaly detection process is executed incrementally. Each profile is inserted one at a time, and **DAMICORE** performs clustering after each insertion. This process continues until a group consisting exclusively of consumption profiles from an anomaly type is identified, or until all available profiles have been included. Thus, the experiment consists of the following steps:

1. Selection of consumption profiles of a given anomaly type;
2. Generation of reference groups with control consumption profiles;
3. For each new anomalous consumption profile loaded, clustering is executed again to update the groups;
4. If an exclusive group is found, i.e., an anomaly detected, the process is interrupted, and the current iteration is saved, along with the system's total response time;

5. Steps 1–4 are repeated until all profiles of each type have been used, and the full experiment is executed 30 times per anomaly type.

The evaluation of effectiveness regarding the selected compression strategies considered four main metrics:

1. **Accuracy:**
$$\frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$
2. **Precision:**
$$\frac{TP}{TP + FP} \tag{5}$$
3. **Recall:**
$$\frac{TP}{TP + FN} \tag{6}$$
4. **F1-Score:**
$$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

Table 2 presents the definition of the confusion matrix for the effectiveness metrics. The evaluation of efficiency regarding compression strategies measures the total response time of the detection process, independently of whether the detection was successful. Furthermore, to evaluate the set of compressors in terms of effectiveness, we considered the number of necessary iterations to detect an anomaly.

Table 2. Confusion matrix for the anomaly detection task performed by **Tricorder**.

	Predicted Anomalous	Predicted Normal
Actual Anomalous	True Positive (TP)	False Negative (FN)
Actual Normal	False Positive (FP)	True Negative (TN)

To properly use the image compressors, however, the input data must be transformed, since the original monitoring files are binary and not suitable for these compressors. The files were then transformed into non-compressed BITMAP files, where each set of 3 bytes represents a pixel's color: the first byte is the red component, the second is the green component, and the third is the blue component. It is worth noting that the chosen encoding is arbitrary with respect to image structure: the resulting bitmaps carry no spatial semantics, which may affect the behavior of compressors designed to exploit two-dimensional visual redundancies.

6 Results

The experimental results are presented below, organized into graphs and tables to highlight the performance of the compression strategies and the differences among them. Table 3 shows the median number of iterations required to detect each anomaly type per compression strategy. The control group is also shown, identified as **A.C.** (from **ANOTHER-CONTROL**) due to space limitations. Cells highlighted in green have a number of iterations that is less than or equal to the median value for the anomaly type across all compression strategies. The median for the control group was not considered, as any detection on the **A.C.** represents a false positive.

Table 3. Median number of iterations for each compressor to signal the detection of an anomaly.

Compressor	A.C.	MEMORY	MODEL	LOGIC	TRAIN	API	CONC	PROCESS
ZLIB	-	27	24	8	17	16	4	4
PPMD	-	17	24	16	6	4	17	2
GZIP	-	29	20	7	17	16	4	2
BZ2	-	27	17	7	9	-	10	2
PNG	-	18	10	9	8	8	5	2
WEBP-LS	16	-	-	9	20	25	11	7
WEBP-LY	19	18	13	25	4	9	3	4
JP2-LS	12	14	5	16	13	7	5	7
JP2-LY	12	14	5	16	13	7	5	7
HEIF	11	2	6	7	4	5	15	1
ENTROPY	-	15	7	24	6	24	11	4
Median	-	**17.5**	**11.5**	**9**	**9**	**8.5**	**5**	**4**

Almost all compression strategies detected all anomalies, with the exception of **BZ2**, which did not detect the **API** anomaly, and **WEBP-LS**, which failed to detect both **MEMORY** and **MODEL** anomalies. Additionally, compressors **WEBP** (both lossy and lossless), **JP2** (both lossy and lossless), and **HEIF** detected anomalies in the control group, resulting in false positives.

Regarding the iterations required to detect an anomaly, the **HEIF** compressor required the fewest iterations for anomalies **MEMORY**, **MODEL**, **LOGIC**, **TRAIN** and **PROCESS**. For **API**, the **PPMD** compressor took the minimum amount of iterations; for **CONC**, it was **WEBP-LY**.

Figure 3 presents the bar charts of the previously defined effectiveness metrics for each compression strategy applied to the anomaly detection task. We observe that the compression strategies **ZLIB**, **PPMD**, **GZIP**, **PNG**, and **ENTROPY** achieved the best results across all metrics, indicating they correctly detected the anomalous profiles and distinguished them from the control profiles. Image compressors, with the exception of **PNG**, are shown to be affected by false positives, leading to reduced accuracy, precision, and F1-score.

WEBP-LS had the lowest results across all metrics due to a combination of detecting a false positive and missing two true positives.



Fig. 3. Bar plots of the effectiveness metrics for each compression strategy in the Decision Tree application.

The efficiency counterpart of this experiment is shown in Table 4, which highlights the median response time for each compression strategy to detect an anomaly. The results shown are only for the used anomalies, and do not include the control group. This metric considers the total execution time for *Damicorepy*, including the time required to compress and group the profiles. For the cases where an anomaly was not detected, the table presents the total execution time. In the same way, cells in which the response time was less than or equal to the median response time for that anomaly are highlighted in green.

Table 4. Median response time (seconds) for each compressor per anomaly type.

Compressor	MEMORY	MODEL	LOGIC	TRAIN	API	CONC	PROCESS
ZLIB	4.28	3.67	1.06	2.33	2.23	0.60	0.58
PPMD	3.47	5.59	3.27	1.13	0.83	3.48	0.52
GZIP	11.88	6.71	2.05	5.45	5.13	1.27	0.78
BZ2	6.22	3.26	1.31	1.61	7.08	1.85	0.50
PNG	14.24	6.82	6.13	5.24	5.45	3.44	1.76
WEBP-LS	38.42	38.30	8.37	21.22	31.02	10.71	6.21
WEBP-LY	18.46	12.14	29.42	3.81	8.16	3.18	3.74
JP2-LS	16.53	5.70	19.46	14.39	7.68	6.15	7.19
JP2-LY	16.30	5.54	19.23	14.16	7.58	5.65	7.15
HEIF	12.66	28.64	33.60	19.79	24.63	82.60	9.01
ENTROPY	2.67	1.21	4.91	1.03	4.91	1.90	0.74
Median	**12.66**	**5.70**	**6.13**	**5.24**	**7.08**	**3.44**	**1.76**

We can observe that compressors **ZLIB**, **BZ2** and **ENTROPY** had faster response times than the median of all compressors for every type of anomaly. **PPMD**, **GZIP** and **PNG** also had response times faster than the median for the majority of the anomalies. Overall, **ENTROPY** was the fastest compression strategy for anomalies **MEMORY**, **MODEL** and **TRAIN**. Compressor **ZLIB** was the fastest for **LOGIC** and **CONC**. **PPMD** had the shortest response time for **API** and **BZ2** was the fastest for the **PROCESS** anomaly.

Going further into this analysis, Table 5 accounts for the time spent per iteration, including a single round of compression and grouping. In this analysis, we can see a more consistent behavior across compression strategies, where the **PNG** compressor is always at the median time, where compressors **ZLIB**, **PPMD**, **GZIP**, **BZ2** and **ENTROPY** are faster than the median iteration time for all evaluated anomalies. **HEIF** is the compressor with the highest iteration time for all scenarios, despite requiring fewer iterations to detect the majority of anomalies.

From these tables, we can begin to understand the differences in response time across compressors. However, it is essential to perform a statistical analysis to detect significant differences in the data. For this reason, the Shapiro-Wilk test was used to verify the normality of the data. Since the majority of pairs of anomaly type and compressor follow nonparametric distributions, the nonparametric Wilcoxon test was applied to confirm significant differences in response times across compression strategies.

All comparisons with $p < 0.05$ were considered statistically significant. Table 6 presents the compression strategy pairs that resulted in non-significant differences in response time according to the Wilcoxon test, after normalization through *Holm's method* to reduce the false discovery rate due to the number of conducted experiments. All other compressor combinations, which accounted for the majority of the analysis, showed significant differences.

Table 5. Median iteration time (seconds) for each compressor per anomaly type.

Compressor	MEMORY	MODEL	LOGIC	TRAIN	API	CONC	PROCESS
ZLIB	0.16	0.15	0.13	0.14	0.14	0.15	0.15
PPMD	0.20	0.23	0.20	0.19	0.21	0.20	0.26
GZIP	0.41	0.34	0.29	0.32	0.32	0.32	0.39
BZ2	0.23	0.19	0.19	0.18	0.24	0.18	0.25
PNG	0.79	0.68	0.68	0.66	0.68	0.69	0.88
WEBP-LS	1.28	1.28	0.93	1.06	1.24	0.97	0.89
WEBP-LY	1.03	0.93	1.18	0.95	0.91	1.06	0.93
JP2-LS	1.18	1.14	1.22	1.11	1.10	1.23	1.03
JP2-LY	1.16	1.11	1.20	1.09	1.08	1.13	1.02
HEIF	6.33	4.77	4.80	4.95	4.93	5.51	9.01
ENTROPY	0.18	0.17	0.20	0.17	0.20	0.17	0.19
Median	**0.79**	**0.68**	**0.68**	**0.66**	**0.68**	**0.69**	**0.88**

Table 6 should be analyzed in conjunction with the time distributions presented in Fig. 4, that provides a visual representation of the response time data for each compression strategy, when looking for three representative anomaly types: **API**, **MEMORY** and **PROCESS**. It visually confirms the statistical similarities found in the Wilcoxon test results.

Table 6. Results for non-significant differences in response time among compression strategies.

Comparison	Anomaly	P-Value
BZ2/ENTROPY	ANOTHER-CONTROL	0.314166
BZ2/JP2-LS	API	1.000000
BZ2/JP2-LY	API	1.000000
BZ2/PPMD	ANOTHER-CONTROL	0.113038
GZIP/HEIF	MEMORY	0.464772
GZIP/PNG	API	0.538887
GZIP/PNG	MODEL	0.334234
GZIP/PNG	TRAIN	0.396577
JP2-LS/JP2-LY	ANOTHER-CONTROL	1.000000
JP2-LS/JP2-LY	CONC	0.132791
JP2-LS/JP2-LY	LOGIC	0.526366
JP2-LS/JP2-LY	PROCESS	1.000000
JP2-LS/WEBP-LY	MEMORY	0.497101
JP2-LS/PPMD	MODEL	1.000000
JP2-LY/PPMD	MODEL	0.243841
PNG/PPMD	CONC	1.000000

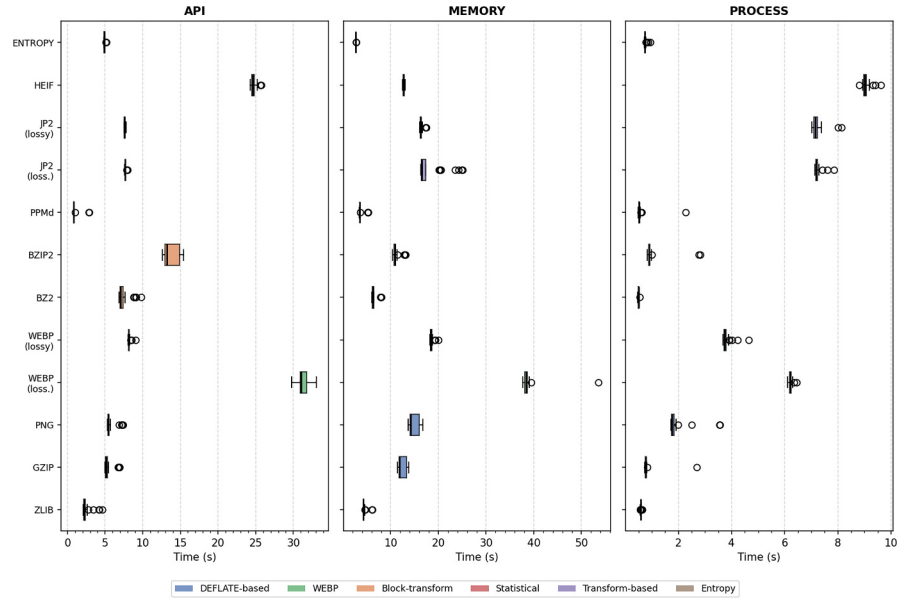

Fig. 4. Boxplots for response time of each compressor against three types of anomalies.

7 Discussion

Results shown in Sect. 6 reveal a consistent correlation between the impact level of a fault and the number of iterations required to detect it. Low-impact anomalies (**MEMORY**, **MODEL**, and **LOGIC**) required an average of 14.7 iterations to be detected, medium-impact anomalies (**TRAIN** and **API**) an average of 11.3, and high-impact anomalies (**CONC** and **PROCESS**) only 6, as the larger changes they introduce in resource consumption profiles make clusters more distinct earlier.

These results reiterate those from previous works regarding the efficiency and effectiveness of both **ZLIB** and **PPMD** compressors, as discussed by Braga [5] and Durães [11]. Moreover, the **PNG** compressor showed positive results: both in effectiveness, by detecting all anomalies without false positives, and in efficiency, with response times below the median for five out of seven anomalies. When looking at iteration time, **PNG** was exactly at the median time for all anomalies.

Some of the non-significant pairs have algorithmic explanations. The similarity between **JP2-LS** and **JP2-LY** is natural, as they are two modes of the same codec and their execution cost difference is negligible. Similarly, **GZIP** and **PNG** both rely on *DEFLATE*, so comparable performance is expected.

WEBP, **JP2** and **HEIF** detected false positives for the control group, while **WEBP-LS** failed to detect two anomalies. A likely contributing factor is the BITMAP transformation: image-specific codecs exploit spatial redundancies in

visual data, and when applied to arbitrarily encoded monitoring bytes, they may perceive distinct consumption profiles as structurally similar, causing the NCD to underestimate distances and triggering spurious detections. This suggests that the preprocessing step is a key limitation for these compressors in the **Tricorder** context, and both the encoding scheme and alternative representations should be explored in future work. Despite that, the **HEIF** compressor required the minimum amount of iterations for four out of seven anomalies, and had the number of iterations below the median in two other anomalies, which indicates that fewer profiles are required for it to detect an anomaly, even if this detection takes a longer time.

The **ENTROPY** compression strategy yielded positive results by detecting all anomalies without generating false positives, while also maintaining total response time and iteration time below the median for all anomalies. This combination of results is matched only by the **ZLIB** compressor, which supports the use of *Shannon Entropy* as an alternative to traditional compression strategies. Furthermore, although it requires more iterations to detect **LOGIC**, **API**, and **CONC** anomalies, its short iteration times keep the total response time below the median.

8 Conclusion and Future Work

This paper evaluated alternative compression strategies for computing the **NCD** metric in the context of the **Tricorder** methodology, including an entropy-based approach (**ENTROPY**) that estimates compressed size without generating actual compressed output. Seven general-purpose and multimedia compressors (**ZLIB**, **GZIP**, **PPMD**, **BZ2**, **PNG**, **WEBP**, **JP2**, and **HEIF**) were compared against **ENTROPY** with respect to both effectiveness and efficiency in anomaly detection. The experiments used monitoring profiles from the *Decision Tree* application, covering seven types of injected software faults. They evaluated results using standard classification metrics (accuracy, precision, recall, and F1-score) and response-time analysis.

Results indicated that the **ENTROPY** compression strategy is efficient and effective in detecting anomalies, comparable to other compression approaches explored in the paper. It correctly identified all anomalies and performed similarly to previously adopted strategies, such as **ZLIB** and **PPMD**.

Regarding efficiency, the **ENTROPY** strategy consistently achieved among the lowest iteration times across all anomaly types, ranking second only to **ZLIB** for most scenarios. This characteristic makes it a promising alternative, especially for detecting low-impact anomalies, as the number of iterations required to detect an anomaly is directly related to the anomaly's impact on resource consumption. Thus, low-impact anomalies benefit more from strategies with low iteration times.

In addition to its effectiveness and efficiency results, the simplicity of implementation and independence from external libraries make **ENTROPY** a low-cost and promising alternative for resource-constrained or operationally restricted environments.

From the perspective of image compressors, **PNG** performed well, detecting all anomalies without false positives. Moreover, it showed response times below the median for five of seven anomalies, while its iteration time was equal to the median for all seven anomalies. These results are not as good as those of established compressors, such as **ZLIB** and **PPMD**, but they are a viable alternative.

Although less precise and with higher execution time, **HEIF** showed an advantage by requiring fewer iterations, making it promising in applications where the number of operations is more critical than total time. These findings reinforce that the choice of compressor should be based not only on precision and execution time, but also on the constraints and priorities of each use scenario. Ultimately, the choice of strategy depends on the specific constraints and priorities of the target application.

The consumption profiles used in the experiment were obtained by monitoring a single application with a subset of possible anomalies. The results may not fully reflect the typical behavior of the evaluated compression strategies across other subsets of anomalies, given the variability in the results presented. Future work may strengthen the results of this study by diversifying the application domains to verify the generalization of compression strategies and reduce biases arising from analyzing a single application. It is also important to conduct robust experiments focused on verifying false-positive rates, with a greater number of alternative control profiles.

Acknowledgments. The authors would like to thank CAPES (PROEX), CNPq (306719/2025-8, 311207/2025-1, 406941/2025-4), and FAPESP (2013/07375-0, 2019/06937-0, 2019/26702-8) for their financial support.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Axelsson, S., Bajwa, K.A., Srikanth, M.V.: File fragment analysis using normalized compression distance. In: Peterson, G., Shenoi, S. (eds.) DigitalForensics 2013. IAICT, vol. 410, pp. 171–182. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41148-9_12
2. Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., Xu, Y.: VP8 Data Format and Decoding Guide. Request for Comments RFC 6386, Internet Engineering Task Force (IETF) (2011). https://www.rfc-editor.org/rfc/rfc6386
3. Beyer, B., Jones, C., Petoff, J., Murphy, N.: Site Reliability Engineering: How Google Runs Production Systems. O'Reilly (2016)
4. Boutell, T., Lane, T., et al.: PNG (Portable Network Graphics) Specification 1.0. Request for Comments RFC 2083, Internet Engineering Task Force (IETF) (1997)
5. Braga, D.: Avaliação do uso de agrupamento de dados de desempenho para apoiar o teste de software no domínio de aprendizagem de máquina. Master's thesis, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos (2022). (in Portuguese). https://doi.org/10.11606/D.55.2022.tde-09112022-160513

6. Burrows, M., L, D.J.W.D.I.G.I.T.A., Taylor, R.W., Wheeler, D.J., Wheeler, D.: A block-sorting lossless data compression algorithm (1994). https://api.semanticscholar.org/CorpusID:2167441
7. Cebrián, M., Alfonseca, M., Ortega, A.: Common pitfalls using the normalized compression distance: what to watch out for in a compressor. Commun. Inf. Syst. **5**(4), 367–384 (2005). https://doi.org/10.4310/CIS.2005.v5.n4.a1
8. Cesar, B.K.M.: Estudo e extensão da metodologia DAMICORE para tarefas de classificação. Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, Brasil (2016). https://doi.org/10.11606/D.55.2016.tde-16112016-170837
9. Cilibrasi, R., Vitányi, P.M.: Clustering by compression. IEEE Trans. Inf. Theory **51**(4), 1523–1545 (2005). https://doi.org/10.1109/TIT.2005.844059
10. Deutsch, P.: GZIP file format specification version 4.3. Request for Comments RFC 1952, Internet Engineering Task Force (IETF) (1996). https://www.rfc-editor.org/rfc/rfc1952
11. Durães, T.d.J.O.: Avaliação do uso de perfis de desempenho aplicados no teste e software em diferentes domínios de aplicação. Master's thesis, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos (2022). (in Portuguese). https://doi.org/10.11606/D.55.2022.tde-05042023-144747
12. Gailly, J.l., Adler, M.: Zlib applications (2026). https://zlib.net/apps.html, Accessed 20 Mar 2026
13. Guimaraes Herrera, C., Silva Montes, V., Souza, S.R.S., Ribeiro dos Santos, R., Lopes de Souza, P.S.: An architecture for a reliability tool applied to distributed systems. In: Computational Science and Its Applications – ICCSA 2025, pp. 332–348 (2025)
14. Gupta, A., Bansal, A., Khanduja, V.: Modern lossless compression techniques: review, comparison and analysis. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), pp. 1–8 (2017). https://doi.org/10.1109/ICECCT.2017.8117850
15. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proc. IRE **40**(9), 1098–1101 (1952)
16. ISO/IEC-JTC-1/SC-29: Information technology – jpeg 2000 image coding system: core coding system. Standard ISO/IEC 15444-1:2004, Int. Org. for Standardization (2004)
17. Katz, P.: DEFLATE Compressed Data Format Specification version 1.3. Request for Comments RFC 1951, Internet Engineering Task Force (IETF) (1996). https://www.rfc-editor.org/rfc/rfc1951
18. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M.: The similarity metric. IEEE Trans. Inf. Theory **50**(12), 3250–3264 (2004). https://doi.org/10.1109/TIT.2004.838101
19. Moffat, A.: Implementing the PPM data compression scheme. IEEE Trans. Commun. **38**(11), 1917–1921 (1990)
20. Montes, V.S., Herrera, C.G., Durães, T.J.O., Delbem, A.C.B., de Souza, S.R.S.L., Souza, P.S.L.d.: Software reliability forecasting using anomalies in profiles of resource usage. In: 2025 IEEE 36th International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 189–196 (2025). https://doi.org/10.1109/ISSREW67781.2025.00071
21. Sadowski, C., Van Gogh, J., Jaspan, C., Soderberg, E., Winter, C.: Tricorder: building a program analysis ecosystem. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 598–608 (2015). https://doi.org/10.1109/ICSE.2015.76

22. Sakr, S.: Xml compression techniques: a survey and comparison. J. Comput. Syst. Sci. **75**(5), 303–322 (2009). https://doi.org/10.1016/j.jcss.2009.01.004, https://www.sciencedirect.com/science/article/pii/S0022000009000142
23. Sanches, A., Cardoso, J.M., Delbem, A.C.: Identifying merge-beneficial software kernels for hardware implementation. In: 2011 International Conference on Reconfigurable Computing and FPGAs, pp. 74–79 (2011). https://doi.org/10.1109/ReConFig.2011.51
24. Santos, S.A., Silveira, B., Durelli, V., Durelli, R., Souza, S., Delamaro, M.: On using decision tree coverage criteria for testing machine learning models. In: Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing. SAST '21, pp. 1–9. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3482909.3482911
25. Seward, J.: bzip2 and libbzip2, version 1.0.8. sourceware.org (2019). https://sourceware.org/bzip2/manual/manual.html. Accessed 20 Mar 2026
26. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
27. Teixeira, A., Matos, A., Souto, A., Antunes, L.: Entropy measures vs. Kolmogorov complexity. Entropy **13**(3), 595–611 (2011). https://doi.org/10.3390/e13030595
28. Telecommunication Standardization Sector of ITU: H.265 : High efficiency video coding. Recommendation ISO/IEC 23008-2:2013, ITU-T (2013)
29. Yao, K., Li, H., Shang, W., Hassan, A.E.: A study of the performance of general compressors on log files. Empir. Softw. Eng. **25**(4), 3043–3085 (2020). https://doi.org/10.1007/s10664-020-09822-x
30. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977)