Agility for Steganalysis: Dealing with Distribution Drift in JPEG File Structures

Matěj Zorek
Czech Technical University in Prague
Prague, Czech Republic
matej.zorek@fel.cvut.cz

Tomáš Pevný
Czech Technical University in Prague
Prague, Czech Republic
pevnak@protonmail.ch

Rainer Böhme
University of Innsbruck
Innsbruck, Austria
rainer.boehme@uibk.ac.at

***Abstract*—Most research on image steganalysis focuses on pixel-level manipulations, even though many steganographic tools, and almost all stego-malware, embed data in the file structure of image formats, like JPEG. Such payloads are challenging to detect at scale because legitimate JPEGs differ widely across encoders, compression libraries, and social media processing pipelines. This causes distribution drift that inflates errors unpredictably. Since JPEG files can be viewed as a hierarchical tree of markers, we propose to use structure-aware methods, specifically a neural network for hierarchical data (HMill) and hierarchical tree distance (HTD), and compare them with conventional feature engineering (HashTrick). In all tested scenarios, the structure-aware models, especially HMill, achieved the highest detection accuracy and high robustness with respect to distribution drift.**

***Index Terms*—Steganography in file headers, Steganalysis, JPEG, File format forensics, Classification**

1. Introduction

Steganography is the art of hiding information within seemingly ordinary data, ensuring that the act of secret communication itself remains invisible. A steganographic system modifies a cover object, for example, an image or audio file, to embed a secret payload while preserving the cover's original appearance and statistical properties [1]. Its adversary, steganalysis, faces a binary decision problem: distinguishing objects carrying hidden payloads from pristine cover objects. Unlike cryptography, which protects the content of a message, steganography conceals its very existence. Success is therefore measured not by the secrecy of the message itself but by the undetectability of stego objects within the distribution of normal files [2]. This tension defines one of the oldest problems in information security [3].

In the context of digital images, steganography is most often studied at the signal level, where the payload is embedded directly into the pixel values or transformed coefficients, often using the least significant bits. This paradigm has dominated the field for more than three decades, inspiring a number of embedding schemes [4]–[7] and machine learning-based detectors [8], [9].

However, image files contain more than pixels and coefficients; they also exhibit a complex internal structure. These non-visual components, the syntax elements such as markers, metadata fields, and auxiliary segments, offer ample space for concealing data without affecting the visual content. Despite being routinely exploited by malware [10]–[12], steganalysis of file structure manipulations remains mostly overlooked in research. The only work known to us, Maljpeg [10], relies on hand-crafted features that project the rich structure of the JPEG syntax elements into a vector embedding.

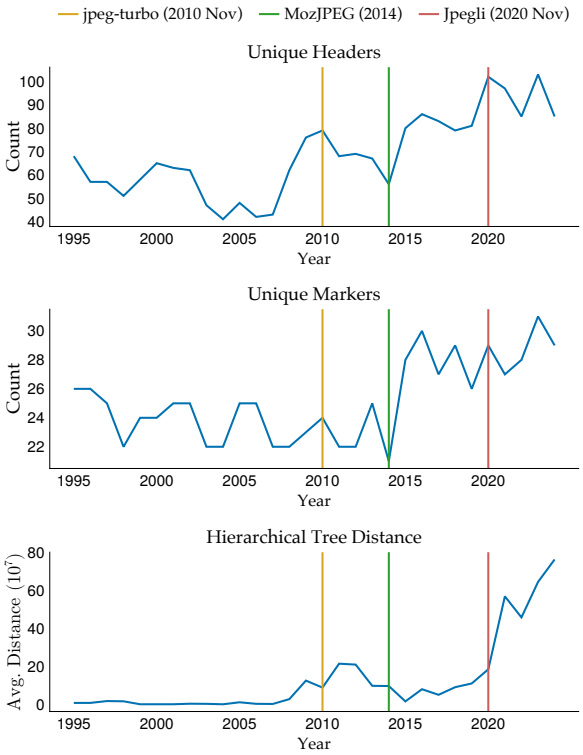


Figure 1: Evolution of the JPEG format over time. The plot shows the number of unique markers, unique header structures, and the average pairwise HTD distance across years, aligned with the release dates of major compression libraries. (See Secion 3.1 for the sampling method.)

A likely reason for this lack of attention is that detecting such manipulations appears straightforward at first glance, unlike pixel-level steganography. However, this intuition is misleading in practice. In reality, common image formats exist in countless different implementations, each producing distinct marker layouts, metadata formats, and segment orderings [13]. Some differences are

DOI 10.1109/EuroSP68448.2026.00085

stark while others are subtle [14]. The syntax structure of JPEG files is therefore far from static. Furthermore, conventions evolve continuously as new libraries are released and improved [15]. Even different versions of the same library may emit files with distinct structures [16]. Finally, large web publishers, like social media platforms, tend to optimize the settings of common libraries for their purposes or use custom libraries [17].

To illustrate this phenomenon, Figure 1 shows how the structures of JPEG files have changed between 1995 and 2024. With time, the number of unique markers and their compositions have steadily increased, confirming that variability is a real challenge for reliable learning-based steganalysis. This variability directly translates into distribution drift, making reliable detection across different encoders a fundamentally challenging problem. Addressing this challenge requires agility, in the sense that detectors must adapt to new encoder conventions without requiring redesign or even retraining.

In this work, we focus on steganography embedded in the structural components of JPEG files, such as metadata segments and marker layouts, rather than signal-domain manipulations. Although detecting such manipulations may appear easier than signal-domain steganalysis, this setting remains practically relevant and insufficiently studied. We therefore study methods that operate directly on the hierarchical syntax of JPEG headers and compare them with flattened feature-based baselines.

We evaluate all models under three scenarios designed to mimic distribution drift encountered in practice. The first scenario studies the transition between sequential and progressive JPEG encoding. The second simulates the emergence of new compression libraries that share common markers but differ in internal structure and segment layout. The third introduces drift caused by images originating from different social media platforms, each applying proprietary recompression pipelines that produce platform-specific syntax.

For each scenario, we consider three variants: (i) a reference with no drift, (ii) drift affecting only cover images, and (iii) drift affecting both covers and stego images. Across all experiments, methods that leverage structural information achieve the highest accuracy and demonstrate superior robustness to distribution drift.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the datasets, the preprocessing pipeline, the payload embedding methods, the models evaluated, and the experimental setup. Sections 4–6 present our experiments and results. Section 7 discusses the findings and the broader implications. Finally, Section 8 concludes the paper.

2. Related Work

Conventionally, image steganography research has focused primarily on manipulations of the signal, such as modifying the least significant bits of pixel values or DCT coefficients. Surveys of these approaches summarize numerous techniques in the spatial and transform domains and their trade-offs between invisibility, payload capacity, and robustness [18]. Although the surveyed studies established the foundation for image-level steganalysis, they overlook the information contained in file structures. Early investigations into the weaknesses of practical tools demonstrated that many off-the-shelf steganographic encoders leak structural traces through fixed-length fields, magic prefixes, or appended trailers [19]. These findings motivated the development of structure-based steganalysis, where the detector analyses metadata, marker sequences, or file headers instead of image content.

Building on this idea, research on the use of steganography in malware extended structural analysis to the security domain, showing that malicious payloads can be hidden within ordinary media containers. Structural inspection of files has proven effective in detecting such threats, as anomalous marker sequences and segment sizes often expose embedded data [12]. Similar concepts have also emerged in multimedia forensics, where metadata irregularities and encoder signatures are used to identify tampering in other formats [20], reinforcing the broader relevance of structure-based methods for integrity and anomaly detection.

Within the JPEG domain, MalJPEG [10] introduced a machine-learning approach that classifies files based on structural statistics, such as marker frequencies and segment lengths. Although it achieves high accuracy within its training distribution, its fixed feature representation limits adaptability to unseen encoders or recompression schemes. More general approaches, such as detectors inspired by artificial immune systems [21], aim to achieve independence from the embedding scheme, but still rely on hand-crafted structural features. Complementary work represents malware binaries as grayscale images and applies convolutional neural networks for classification [22]. While these methods operate on visualized bytes rather than image containers, they demonstrate the potential of data-driven models for learning discriminative representations directly from the raw file structure.

Comprehensive surveys of the use of steganography in malware detection confirm that most existing detection systems concentrate on signal-domain analysis, whereas structural indicators in headers and metadata remain underexplored [11], [23]. They also emphasize the need for methods that remain robust under evolving encoder implementations and platform-specific recompression pipelines, conditions that continue to challenge the practical deployment of steganalysis and malware detection systems. Therefore, the development of robust structure-based models represents an important direction for future research on detecting hidden payloads in complex file formats.

3. Research Method

Our workflow proceeds in five stages. First, we collect heterogeneous JPEG data and verify the structural integrity of each file. Second, we preprocess the data by converting each file into a structured representation suitable for downstream analysis. Third, we generate stego samples by embedding payloads into predefined header locations. Fourth, we construct and train the models on a designated training subset. Finally, we evaluate their precision and reliability under several testing conditions.

3.1. Data

We collected three datasets to support different parts of our study. Each dataset serves a distinct purpose, ranging from measuring the general variability in JPEG structures to modeling platform-specific and temporal distribution drifts.

The first dataset, *Web-10K*, consists of 10,000 heterogeneous JPEG images randomly sampled from the internet. This dataset was derived from the publicly available image URLs provided by Dornauer and Felderer [24]. It is designed to maximize syntax and source diversity rather than the diversity of the image content. As the images originate from a wide range of web domains, software libraries, and encoding pipelines, the dataset exhibits substantial variability in header structures and metadata layouts. Although we do not claim that this collection perfectly reflects the global distribution of JPEGs on the internet, its heterogeneous composition captures a broad spectrum of encoder implementations likely encountered in practice, making it a valid foundation for evaluating robustness across encoding conventions.

The second dataset, *Social-6*, contains 600 JPEG images (approximately 100 per platform) collected from six major social media platforms: Facebook, Instagram, BlueSky, Pinterest, X, and LinkedIn. The images were collected from publicly accessible content on each platform. Despite its smaller size, it captures the distinctive header structures characteristic of each platform, where each exhibits two to three consistent encoding templates.

Finally, we compiled a temporal reference dataset, *JPEG-Timeline* (1995–2024), consisting of 30,000 images scraped from the Wayback Machine [25] (1,000 per year). For each year, we queried the archive for files labeled as JPEG images and randomly sampled 1,000 that, according to their metadata, were stored in their original form and not post-processed. Although this dataset is not used for model training or evaluation, it serves to illustrate the evolution of JPEG headers and encoding practices over time and supports the motivation for our experimental design (see Figure 1).

3.2. Preprocessing

All images were first validated to ensure correct JPEG syntax. The structural integrity of each file was inspected by verifying the presence of all mandatory JPEG markers: Start-of-Image (`SOI`), Quantization Tables (`DQT`), Start-of-Frame (`SOF`), Huffman Tables (`DHT`), Start-of-Scan (`SOS`) and End-of-Image (`EOI`). Files missing any of these markers or containing corrupted segments were excluded from further processing. The number of excluded samples was negligible, far below one percent.

The validated files were then parsed using a custom parser implementation developed for this work. The code, including the parser and scripts used to run the experiments in this paper, is publicly available at https://github.com/aicenter/AgilityForSteganalysis.jl

The parser iterates through the binary representation of each JPEG file (see Figure 2a), detects all marker segments, and organizes them into a Hierarchical Structured Tree (HS-Tree) representation [26] as shown in Figure 2b.

In the HS-Tree, each node represents either a JPEG marker or a subcomponent of a marker. The root node is virtual; it anchors the hierarchy but does not carry any intrinsic value. Child nodes of the root correspond to top-level JPEG markers, while children of these markers capture structured subcomponents when applicable (e.g., within a quantization table, nodes may represent the table destination or table data). In some segments, such as embedded thumbnails, nodes may themselves contain nested JPEG markers, reflecting a JPEG file within the parent JPEG.

Each node is annotated with its position in the file, raw content (typically a byte string or vector), and for variable-length segments, the segment length. This preserves the sequential and structural information of the original JPEG file. Note that the HS-Tree itself does not impose any ordering on nodes; storing position and length ensures that sequential information is preserved for downstream processing.

Signal data is intentionally ignored, as our analysis focuses solely on marker and header composition. Importantly, we do not interpret or decode marker contents, but rather treat them as symbolic and positional elements that capture the organization of the file.

3.3. Payload Embedding Methods

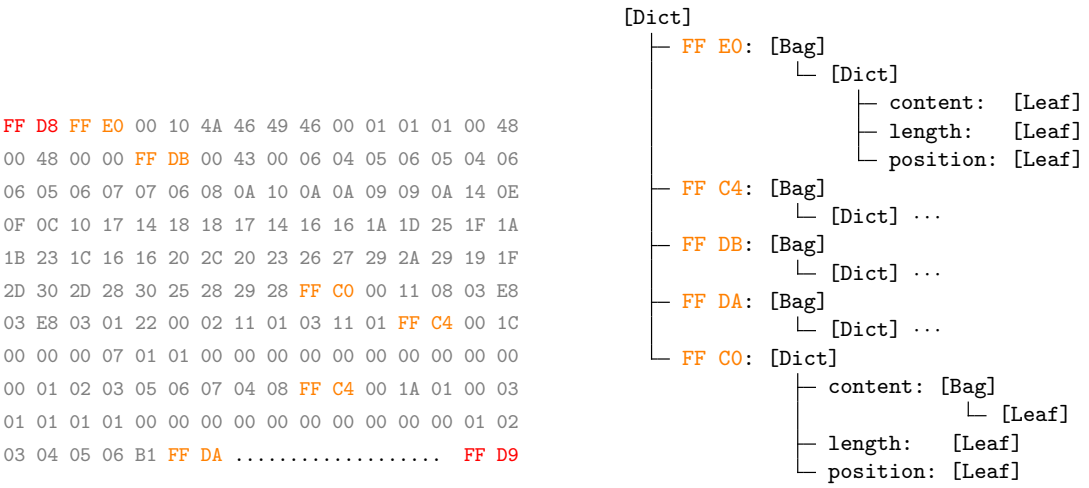
Prior to conducting the steganalysis experiments, we systematically examined potential locations within JPEG files suitable for embedding hidden payloads without affecting image reconstruction. The investigation focused on non-essential but commonly used marker segments, which are ignored by standard decoders and thus provide plausible opportunities for covert data insertion. The main findings are summarized below.

Behind End of Image (`EOI`). The data attached after the `EOI` marker can, in principle, conceal additional information. However, its reliability depends on the JPEG decoder implementation. Some decoders ignore trailing bytes entirely, while others may misinterpret or reject such files. Our custom parser flags any data following the `EOI` marker as an error to ensure strict conformance. Moreover, such payloads are trivially detectable by structural inspection and therefore unsuitable for steganographic purposes.

Comment (`COM`). The Comment segment provides a natural and flexible channel for hidden data. Arbitrary payloads of unrestricted length can be embedded, and multiple comment segments may appear within a single file. Because these segments are ignored by all standard decoders, they represent a safe and non-destructive option for information hiding.

Application Segments (`APP0`–`APP15`). Application markers typically store metadata generated by encoders or software frameworks (e.g., EXIF, JFIF, or thumbnail data). Their internal structure is not standardized, and multiple segments with identical marker identifiers may appear within a file. Consequently, these segments can also serve as containers for arbitrary payloads without affecting image decoding.

Quantization and Huffman Tables. Although it is theoretically possible to embed information by modifying entries in the quantization or Huffman tables, this approach is highly constrained and fragile. Even minor



(a) JPEG file.

(b) HS-Tree representation schema.

Figure 2: Hex dump of the JPEG file (a) and its structural representation as an HS-Tree (b) with some branches omitted for compactness.

TABLE 1: Payload embedding configurations used for generating stego samples.

Location	Lengths (bytes)	Content
COM	$\{8, 16, 32, 64, 128, 256, 512, 1024\}$	random / text
APP2	$\{8, 32, 564, 612\}$	random / text
APP8	$\{8, 32, 564\}$	random / text

perturbations can cause decoding errors or visible artifacts, and the overall storage capacity is extremely limited. Therefore, this technique was excluded from practical consideration.

In the experiments reported in this paper, we focus on payload insertion into Comment and Application segments because they are non-essential for image reconstruction and commonly occur in natural JPEGs. Concretely, we use COM segments and two application markers: APP2, which is widely used, and APP8, which is really rare. The payload content is either random byte sequences or text-like bytes sampled from a corpus, and lengths are varied to probe detectability as a function of payload size. Table 1 lists the specific embedding configurations used to generate stego samples.

A recent large-scale survey [27] of real steganography software confirms that the vast majority of publicly available tools embed data into JPEG header segments such as COM or APP markers, matching the embedding mechanisms used in our experiments and lending external validity to our design. Our evaluation does not model an adaptive adversary that deliberately minimizes structural anomalies to evade detection, as explored in adversarial embedding strategies [28], [29].

3.4. Steganalysis Based on Hierarchical Structure

Structure-aware classifiers are far less developed than methods for tensors of fixed size, graphs, and sequences. In this work, we therefore focus on two representative approaches that operate directly on the hierarchical structure of JPEG files: Hierarchical Multi-Instance Learning (HMill) and Hierarchical Tree Distance (HTD).

3.4.1. Hierarchical Multi-Instance Learning.

HMill [30], [31] is a state-of-the-art discriminative model for structured tree data. Alternative architectures include recursive neural networks [32] and transformer-based models [33]. However, according to the latest comparison in [33], HMill provides the strongest overall performance while maintaining a relatively low computational complexity. Furthermore, HMill has been successfully applied in the security domain [30], [34].

HMill creates a neural network with an architecture that directly mirrors the structure of the data. The architecture is based on the recursive principle that lower-level nodes project their data into a fixed-size vector, assuming that their child nodes have already been projected in the same way—with the exception of leaves, which have no children. This leads to an architecture composed of three distinct structural blocks.

Leaf nodes are embedded into a fixed-size vector by an appropriate neural network chosen for the specific leaf type. For example, strings can be represented as a histogram of tri-grams projected by a feed-forward neural network (FFNN), or by a more expressive architecture such as BERT [35]. Cardinal types can be represented by one-hot encoding and then projected by an FFNN, and real-valued attributes are processed in an analogous way.

Dict nodes are inner nodes that assume that all child nodes have already been projected onto fixed-size vectors. They therefore apply an FFNN to each child projection, concatenate the results, and project this concatenated vector using another FFNN. Since the number of child nodes is fixed, all operations are over known dimensions and thus well defined.

In contrast, *bag nodes* must accept a variable number of children, but by the recursive construction, each child has already been projected into a fixed-size vector. They are processed by first projecting all child vectors with an

FFNN, then taking the element-wise mean and element-wise maximum, concatenating these two summary vectors, and projecting the result with another FFNN. As proved in [36], the above construction is a universal approximator of the data-generating process in the space of JSONs.

As an example, consider the schema of a JPEG file in Figure 2b. For simplicity, we focus only on the branch `FF E0`. Leaves `content`, `length`, and `position` are projected onto fixed-dimensional vectors of dimension d using an FFNN, which can be either shared between leaf types or specialized for each. These three vectors are then concatenated into a vector of dimension $3d$ and projected by another FFNN into a vector of dimension d_d. At this point, the `Dict` node is represented as a single vector. This procedure is applied to every item in the `Bag`, after which all items are represented by vectors of dimension d_d. These vectors are then aggregated by element-wise mean and element-wise maximum to vectors, which are then concatenated to a vector of dimension $2d_d$ and projected by FFNN to a single vector of dimension d_b representing the `Bag`. The remaining samples are processed in the same recursive fashion.

The HMill architecture is parametrized by the FFNNs that perform these projections within each structural block. Since the architecture is fully differentiable and the output for each sample is a single vector, it can be trained like any standard FFNN using established optimization methods for neural networks. In this work, the classifiers are trained using cross-entropy loss and optimized with Adam, exploring multiple batch sizes and learning-rate configurations.

3.4.2. Hierarchical Tree Distance. HTD [26] is a differentiable metric for the space of hierarchically structured trees. Like HMill, it decomposes the problem into elementary building blocks and applies a recursive formulation in which the distances between child nodes are computed first and then aggregated. For leaf data, the distance function depends on the modality: strings may use a string distance or an n-gram vector representation, cardinal types use the discrete metric, and vectors or numbers can use any L_p norm. To compute distances between *Dict nodes*, HTD employs a weighted product metric, while for *Bag nodes* it uses either set distances (Hausdorff distance [37]) or multi-set distances (Wasserstein distance [38]).

Similarly to HMill, the entire construction is differentiable, allowing all parameters (in particular, the weights of the product metric) to be learned. We optimize these parameters using a triplet-loss objective [39], which aligns the metric with the target classification task and yields a domain-specific notion of similarity between JPEG structures. Once trained, the learned metric can be used within standard classifiers such as k-Nearest Neighbors [40] (KNN) and Support Vector Machines [41] (SVM), where it replaces the conventional Euclidean distance.

As an example, we again consider the sub-schema of the key `FF E0` in Figure 2b. The computation of the HTD distance between samples x_1 and x_2 begins by computing the distances between the leaves `content`, `length`, and `position` in the two samples. This yields three pairwise distance matrices, d_c, d_l, and d_p, each of identical shape. Using the weighted product metric, these matrices are combined into a single matrix d_d, which represents the distances between the `Dict` items in this sub-schema for samples x_1 and x_2.

The distance between the corresponding `Bag` nodes in x_1 and x_2 is then computed using either the Hausdorff or Wasserstein distance, with the matrix d_d serving as input. This produces a single value measuring the distance between the `FF E0` subtrees in the two samples. Distances for all other sub-schemas are computed in the same way, and the resulting distances are finally aggregated via another weighted product metric to obtain the full HTD distance between x_1 and x_2.

The resulting HTD metric can be used directly as a distance function in KNN. For SVM, we instead incorporate HTD as the core component of a kernel function, for example $\kappa(x, y) = \exp\left(-d_{\mathrm{HTD}}(x, y)^2/\gamma\right)$, allowing the classifier to operate in a distance-induced feature space.

3.5. Steganalysis Based on Features

The second group of methods projects the structured data onto flat vectors, allowing the use of conventional machine-learning algorithms.

The first approach, HashTrick [42], [43], is a general technique widely used in cybersecurity applications for efficient feature hashing. It encodes structural information by hashing node identifiers together with contextual attributes into a sparse feature vector, effectively forming a histogram of structural tokens. This embedding enables the use of standard classifiers without modification; we therefore evaluate it with KNN and SVM, in the same configuration as for HTD. Embedding dimensionality and token-encoding schemes are treated as tunable hyperparameters, controlling the trade-off between representation compactness and information preservation.

The second approach is MalJPEG [10], a feature-engineering method originally proposed for malware detection and retrained here under identical conditions for steganalysis. It relies on hand-crafted features derived from the frequency and layout of JPEG markers and segment metadata. Like HashTrick, MalJPEG features are evaluated using KNN, and additionally with XGBoost [44] to remain consistent with the original proposal. Together, these feature-based baselines provide a point of comparison for assessing the benefits of the proposed structure-aware methods.

3.6. Evaluation Setup

Detectors are evaluated under conditions similar to real-world deployment, where JPEG files originate from heterogeneous sources and continuously evolve over time. We measure both their detection accuracy and robustness to distributional drift caused by the introduction of new encoders or changes in compression settings. All experiments are formulated as binary classification problems, distinguishing between cover and stego images.

All models and every experiment use the same protocol. The available data are split into training and test sets, ensuring that no image in any form (cover or stego) appears in both splits. The experiments are repeated four times with different random seeds to reduce stochastic effects. The first run is used for the selection of hyperparameters, performed by grid search over the ranges listed

in Table 6. The best configuration from the first run is used in the remaining three runs, which are then used to measure performance. We chose this protocol to maximise the number of hyperparameters tested within a reasonable computational budget, while also enabling us to run all combinations in parallel.

Performance is measured using several complementary metrics. The most relevant is the balanced accuracy (BACC), defined as the arithmetic mean of the true-positive and true-negative rates,

$$\mathrm{BACC} = \frac{1}{2}\Big(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} + \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}\Big),$$

which remains informative even under class imbalance. We also report the Area Under the ROC Curve (AUC), which reflects the ranking capacity of the model between thresholds.

Finally, robustness to distribution drift is measured by the Drop in Balanced Accuracy (ΔBACC), defined as

$$\Delta\mathrm{BACC} = \mathrm{BACC}_{\text{out of distribution}} - \mathrm{BACC}_{\text{in distribution}},$$

which quantifies the change in accuracy when detectors are exposed to previously unseen data sources.

In this context, distribution drift refers to any systematic change in the joint distribution of cover or stego samples that arises from new encoder implementations, compression libraries, or platform-specific recompression pipelines. Formally, a model is trained on data drawn from distributions $p_{train}(x, y)$ and tested on drifted distributions $p_{test}(x, y) \neq p_{train}(x, y)$. The magnitude of the resulting ΔBACC indicates how well the detector generalizes beyond the training conditions.

4. Experiment 1: Sequential vs Progressive

4.1. Setup

The first experiment simulates the adoption of a different encoding mode, which represents a natural and realistic form of distribution drift in JPEG data. The JPEG standard supports four principal encoding modes. Two are almost never used, and the other two are *sequential* and *progressive*.

Sequential JPEGs store image data in a single top-to-bottom scan, whereas progressive JPEGs encode multiple scans of increasing detail. Progressive encoding was rare in early web content, but became widespread after the release of MozJPEG in 2014 and its subsequent adoption by major web platforms [15]. Consequently, newer datasets contain a mixture of both modes, which differ substantially in header composition, marker ordering, and segment density.

Specifically, sequential JPEGs contain a single `SOS` segment and may include restart markers (`RST`), whereas progressive JPEGs contain multiple `SOS` segments and no `RST`. Such structural variations may mislead the detectors despite not being related to the hidden payload.

The experiment uses the Web-10K dataset and partitions it by mode into 6,600 sequential and 3,400 progressive samples. For each location listed in Table 1, we generate 100 stego samples, half containing random byte sequences and half carrying ASCII-encoded text strings, resulting in 1,500 stego images for the in-distribution (sequential) setting. These samples are evenly and randomly split into 750 training and 750 test images, ensuring no overlap between the two subsets. To evaluate drift on progressive JPEGs, we apply the same generation protocol and produce an equivalent set of 1,500 stego samples, allowing a fair comparison across distributions.

TABLE 2: Experiment 1: Sequential vs Progressive images. Reported values represent the mean and standard deviation computed over three independent runs. Structural / feature-based detectors are printed in black/gray.

Sc	Method	BACC [%]	TPR [%]	FPR [%]	AUC [%]
In-Dist	MalJ.-KNN	53.8 ± 4.4	7.7 ± 8.8	0.1 ± 0.1	66.2 ± 0.4
	MalJ.-XGB	83.9 ± 1.7	68.5 ± 3.7	0.8 ± 0.4	96.4 ± 0.1
	HT-KNN	56.9 ± 0.6	13.8 ± 0.3	0.4 ± 0.4	76.0 ± 0.6
	HT-SVM	69.7 ± 0.6	39.5 ± 1.3	0.1 ± 0.1	90.8 ± 0.1
	HTD-KNN	77.1 ± 2.7	54.6 ± 5.6	0.5 ± 0.2	90.6 ± 4.2
	HTD-SVM	71.9 ± 2.3	49.7 ± 4.5	6.0 ± 0.1	75.1 ± 1.8
	HMill	99.2 ± 0.1	98.7 ± 0.4	0.3 ± 0.1	99.6 ± 0.4
O-Dist (C)	MalJ.-KNN	50.0	0.0	0.0	70.8 ± 2.8
	MalJ.-XGB	72.0 ± 2.6	68.5 ± 3.7	24.5 ± 3.2	84.0 ± 1.1
	HT-KNN	59.4 ± 1.3	17.7 ± 1.1	0.1 ± 0.1	69.3 ± 4.6
	HT-SVM	50.0	0.0	0.0	72.1 ± 6.9
	HTD-KNN	79.8 ± 8.7	64.1 ± 19.0	0.2 ± 0.1	88.3 ± 0.3
	HTD-SVM	72.1 ± 4.6	48.5 ± 7.5	4.3 ± 1.6	82.4 ± 1.5
	HMill	99.2 ± 0.2	98.7 ± 0.4	0.4 ± 0.1	99.5 ± 0.6
O-Dist (C+S)	MalJ.-KNN	52.0 ± 3.3	4.1 ± 6.6	0.0	67.2 ± 0.9
	MalJ.-XGB	67.2 ± 0.4	59.0 ± 2.5	24.5 ± 3.2	77.0 ± 1.2
	HT-KNN	52.5 ± 0.7	4.6 ± 0.6	0.2 ± 0.1	61.1 ± 0.6
	HT-SVM	60.0 ± 7.4	20.0 ± 15.0	0.1 ± 0.1	73.8 ± 1.8
	HTD-KNN	74.8 ± 1.6	49.7 ± 3.1	0.2 ± 0.1	78.0 ± 0.6
	HTD-SVM	63.9 ± 3.5	31.6 ± 9.4	3.7 ± 2.4	77.1 ± 1.2
	HMill	99.3 ± 0.2	98.9 ± 0.4	0.4 ± 0.1	99.5 ± 0.5

The in-distribution setting (*In-Dist*) serves as the baseline, with both training and testing performed on sequential JPEGs. The out-of-distribution (cover-drift) setting, denoted *O-Dist (C)*, introduces progressive cover images at test time while keeping the stego samples sequential. Finally, the out-of-distribution setting (cover-and-stego-drift), denoted *O-Dist (C+S)*, uses progressive covers and progressive stego samples in the test set, simulating a scenario in which the steganographer adapts to new encoders. We emphasize that only the testing conditions change; detectors are always trained on sequential images. Robustness is quantified by the drop in balanced accuracy, ΔBACC, between the *In-Dist* baseline and the corresponding *O-Dist* scenarios.

4.2. Results

Table 2 summarizes the results of the first experiment. The overall trends reveal a clear hierarchy of detector performance and robustness.

The two simplest approaches, MalJPEG–KNN and HashTrick–KNN, perform poorly across all scenarios. Even in the *In-Dist* setting, their balanced accuracies of 53.8% and 56.9%, respectively, indicate near-random behavior. Analysis of the true- and false-positive rates confirms that these models classify nearly all samples as covers, producing trivial detectors with limited discriminative capacity. Their performance remains at the chance level in both *O-Dist* settings as well.

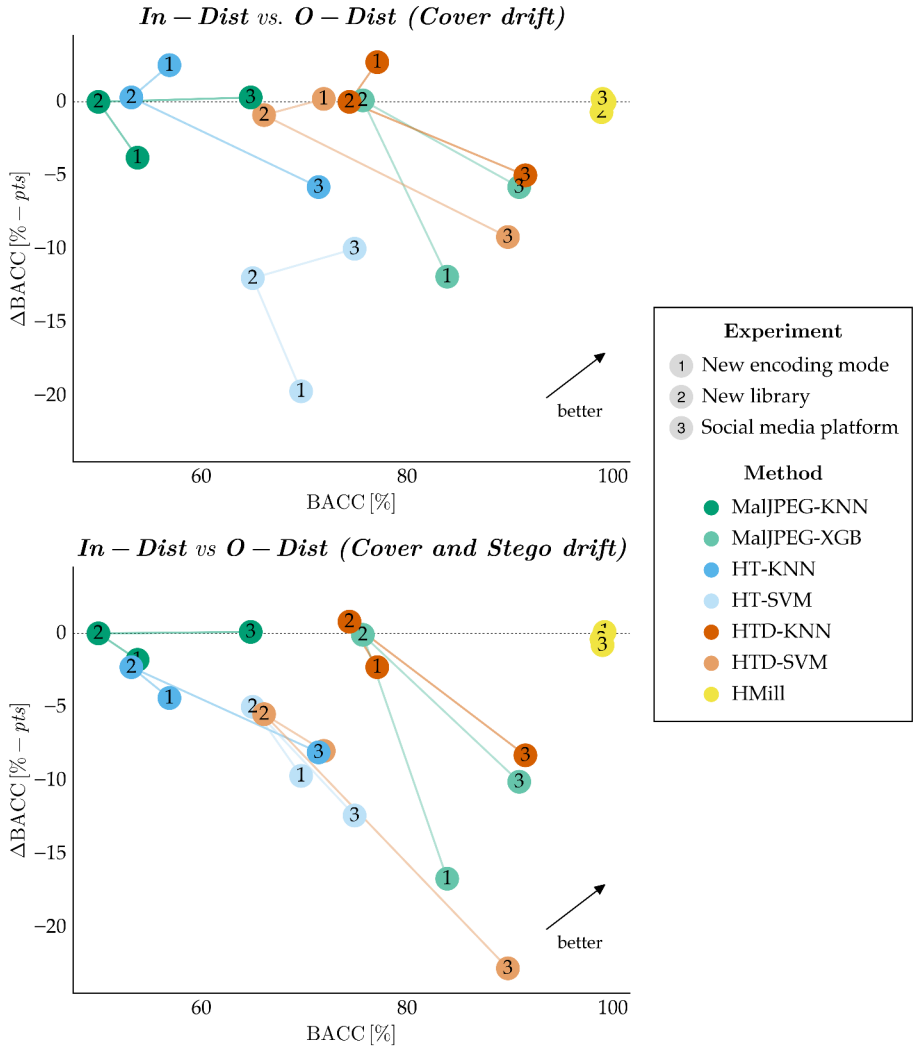


Figure 3: Relationship between baseline performance and robustness under distribution drift. The two panels show (top) cover-only drift and (bottom) combined cover and stego drift. The horizontal axis shows performance in the *In-Dist* setting, while the vertical axis reflects the performance change under drift. Colors identify methods, and numbers denote experiments. Points belonging to the same method are connected into trajectories showing how each method responds across experiments. The upper-right region corresponds to a more desirable combination of high accuracy and robustness.

The MalJPEG–XGB model, although originally designed for malware detection rather than steganalysis, achieves surprisingly competitive results with 83.9% balanced accuracy in the *In-Dist* setting. This behavior can be explained by the fact that one of the hand-crafted features used in MalJPEG explicitly captures the *maximum length of comment segments* (COM) within a JPEG file. Since several of the embedded payloads were inserted into COM markers with fixed sizes, the XGBoost classifier can easily exploit this feature as a direct indicator of steganographic content. In contrast, payloads embedded in APP segments do not affect this feature and therefore largely escape detection. This also explains the substantial performance drop under drift as shown in Table 3, where BACC decreases by 11.9%-pts in *O-Dist (C)* and by 16.7%-pts in *O-Dist (C+S)*. Although MalJPEG-XGB achieves a relatively high true-positive rate, its false-positive rate increases sharply after drift, indicating poor generalization and limited robustness to unseen header structures.

The model HashTrick–SVM performs moderately better than its counterpart KNN, but still exhibits significant sensitivity to drift. Its initial accuracy of 69.7% suggests that vector embeddings capture some discriminative information, yet the model collapses to random guessing once progressive JPEGs appear in the test set. This instability confirms that flattening hierarchical information into fixed-length vectors fails to capture the structural regularities required for robust detection.

Methods based on the Hierarchical Tree Distance demonstrate higher stability across all scenarios. Both KNN and SVM classifiers maintain balanced accuracies above 70% in all settings, with only modest degradation under drift (ΔBACC $\approx 2-8$%-pts). Their consistently high AUC values ($80-90$%) confirm that the learned distance metric captures meaningful structural similarities between the JPEG headers.

TABLE 3: Summary of ΔBACC from all three experiments. Structural / feature-based detectors are printed in black/gray.

Sc	Method	ΔBACC [%-pts]		
		Exp 1	Exp 2	Exp 3
O-Dist (C)	MalJ.-KNN	−3.8	0	0.3
	MalJ.-XGB	−11.9	0.1	−5.8
	HT-KNN	2.5	0.3	−5.8
	HT-SVM	−19.7	−12.0	−10.0
	HTD-KNN	2.7	0	−5.0
	HTD-SVM	0.2	−0.9	−9.2
	HMill	0	−0.7	0.2
O-Dist (C+S)	MalJ.-KNN	−1.8	0	0.1
	MalJ.-XGB	−16.7	−0.1	−10.1
	HT-KNN	−4.4	−2.3	−8.1
	HT-SVM	−9.7	−5.0	−12.4
	HTD-KNN	−2.3	0.8	−8.3
	HTD-SVM	−8.0	−5.5	−22.8
	HMill	0.1	−0.4	−0.8

Finally, the HMill neural network achieves nearly perfect performance across all scenarios (≈ 99% BACC and AUC ≈ 99.5%), showing no measurable degradation under drift. Its consistent combination of high TPR and low FPR confirms that end-to-end hierarchical representation learning is highly effective for detecting payloads across diverse encoding environments.

The superior performance of HMill across both drift scenarios is further illustrated in Fig. 3, which summarizes the relationship between in-distribution BACC and robustness across all experimental settings. The trajectories reveal that the methods respond differently to distinct sources of drift, yet their relative ordering is largely preserved. This suggests that robustness is primarily determined by the underlying representation rather than by the specific form of distribution drift.

5. Experiment 2: New Compression Library

5.1. Setup

While the previous experiment simulated temporal drift through the historical adoption of progressive JPEGs, the second experiment targets a different but equally relevant source of distributional change: the emergence of new or modified compression libraries, which keep approximately the same number of markers, but with small variations. This is motivated by an observation that each JPEG encoder implements the standard with small differences in quantization strategies, marker ordering, and auxiliary metadata. Even within the same version family, compiler optimizations or platform-specific forks may introduce subtle structural variations that have little impact on image quality but can affect the statistical properties of JPEG headers. Such changes can easily confuse detectors that have implicitly learned the peculiarities of a particular encoder rather than general features of the format.

We have therefore created synthetic samples (1500 cover and 1500 stego) by modifying syntax elements in ways that mimic plausible future or alternative encoders. These modifications include (i) removing all but the most common application segments (keeping only `APP0`), (ii) adding a custom `APP14` segment with a constant payload to emulate a vendor-specific metadata field, (iii) enforcing a deterministic ordering of markers, `SOF`, `DQT`, `DHT`, `APPn`, `DRI`, `SOS`, `RSTn`, and (iv) substituting standard 8-bit quantization tables with their 16-bit variants, which are rare, but fully valid according to the JPEG specification.

Together, these adjustments create a structurally distinct, yet syntactically valid, JPEG distribution that none of the detectors have encountered during training.

Training and evaluation follow the same protocol as in Experiment 1. Models are trained on the *Web-10K* dataset containing naturally encoded sequential and progressive JPEGs, and evaluated on modified JPEG files.

As before, three scenarios are evaluated: the *In-Dist* baseline using standard JPEGs for both training and testing; an out-of-distribution (Cover-Drift) setting (*O-Dist (C)*) where only synthetically altered cover images appear in the test set; and an out-of-distribution (Cover + Stego-Drift) setting (*O-Dist (C+S)*) that also includes correspondingly altered stego samples.

TABLE 4: Experiment 2: Existing vs Synthetic new compression library. Reported values represent the mean and standard deviation computed over three independent runs. Structural / feature-based detectors are printed in black/gray.

Sc	Method	BACC [%]	TPR [%]	FPR [%]	AUC [%]
In-Dist	MalJ.-KNN	50.0	0.0	0.0	65.9 ± 0.9
	MalJ.-XGB	75.7 ± 2.3	51.7 ± 4.8	0.2 ± 0.1	92.6 ± 0.3
	HT-KNN	53.2 ± 0.8	6.5 ± 1.5	0.2 ± 0.1	71.1 ± 1.2
	HT-SVM	65.0 ± 0.3	30.2 ± 0.7	0.2 ± 0.2	89.3 ± 0.9
	HTD-KNN	74.4 ± 1.6	48.9 ± 3.3	0.0	76.8 ± 0.9
	HTD-SVM	66.1 ± 1.8	37.1 ± 4.0	4.8 ± 0.4	70.9 ± 0.7
	HMill	98.9 ± 0.2	98.0 ± 0.4	0.2	99.8
O-Dist (C)	MalJ.-KNN	50.0	0.0	0.0	69.5 ± 1.3
	MalJ.-XGB	75.8 ± 2.4	51.7 ± 4.8	0.1 ± 0.1	93.4 ± 0.4
	HT-KNN	53.5 ± 0.4	7.2 ± 0.8	0.1 ± 0.1	67.5 ± 1.5
	HT-SVM	53.0 ± 5.3	6.1 ± 5.5	0.0	70.7 ± 9.2
	HTD-KNN	74.4 ± 0.5	49.9 ± 1.9	1.0 ± 0.9	74.6 ± 2.0
	HTD-SVM	65.2±11.4	35.7±30.0	5.3 ± 7.2	82.5 ± 0.4
	HMill	98.2 ± 0.4	98.0 ± 0.4	1.6 ± 1.2	99.6
O-Dist (C+S)	MalJ.-KNN	50.0	0.0	0.0	65.6 ± 0.4
	MalJ.-XGB	75.6 ± 2.5	51.3 ± 5.0	0.1 ± 0.1	92.2 ± 0.1
	HT-KNN	50.9 ± 0.8	1.9 ± 1.6	0.0	56.9 ± 1.2
	HT-SVM	60.0 ± 7.4	20.0±15.0	0.1 ± 0.1	73.8 ± 1.8
	HTD-KNN	75.2 ± 0.6	51.3 ± 1.9	0.9 ± 0.7	77.6 ± 0.9
	HTD-SVM	60.6±11.7	26.4±30.5	5.3 ± 7.2	77.8 ± 1.9
	HMill	98.5 ± 0.5	98.6 ± 0.3	1.6 ± 1.2	99.7

5.2. Results

Table 4 contains the results of the second experiment, which evaluates the detector's robustness to drift caused by a synthetic encoder. The overall performance pattern is similar to that observed in section 4.2, although the extent of degradation varies between methods.

The MalJPEG-KNN again fails to learn any meaningful decision boundary, performing at random across all scenarios (BACC = 50%). In contrast, the MalJPEG-XGB achieves relatively strong and surprisingly stable results, maintaining BACC ≈ 76% and AUC ≈ 93% even under drift. This suggests that despite relying on simple hand-crafted byte-level features, the boosted trees are able

TABLE 5: Experiment 3: Existing vs New Social Media Platform. Reported values represent the average over 15 different splits of platforms and three independent runs. Standard deviations are computed in the same way. Structural / feature-based detectors are printed in black/gray.

Sc	Method	BACC [%]	TPR [%]	FPR [%]	AUC [%]
In-Dist	MalJ.-KNN	64.8 ± 0.7	31.2 ± 2.8	1.6 ± 2.1	64.3 ± 2.1
	MalJ.-XGB	90.9 ± 2.0	91.7 ± 2.9	9.8 ± 4.0	97.1 ± 1.2
	HT-KNN	71.4 ± 2.5	74.0 ± 7.9	31.1 ± 8.6	73.5 ± 3.5
	HT-SVM	74.9 ± 4.4	70.5 ± 8.7	20.7 ± 14.7	81.5 ± 7.0
	HTD-KNN	91.5 ± 2.7	86.4 ± 3.9	3.5 ± 3.5	92.4 ± 4.9
	HTD-SVM	89.8 ± 2.9	90.6 ± 3.8	11.0 ± 5.1	93.8 ± 3.1
	HMill	99.0 ± 0.7	99.0 ± 0.6	0.9 ± 1.1	99.7 ± 0.3
O-Dist (C)	MalJ.-KNN	65.1 ± 1.1	32.3 ± 3.0	2.0 ± 1.4	63.2 ± 5.6
	MalJ.-XGB	85.1 ± 6.0	91.7 ± 2.9	21.5 ± 12.1	93.1 ± 4.5
	HT-KNN	65.6 ± 6.2	68.3 ± 18.3	37.1 ± 17.6	67.7 ± 7.2
	HT-SVM	64.9 ± 4.8	78.6 ± 10.2	48.8 ± 15.6	68.5 ± 10.6
	HTD-KNN	86.5 ± 5.6	84.8 ± 4.0	11.9 ± 11.7	84.1 ± 11.6
	HTD-SVM	80.6 ± 7.2	88.6 ± 5.6	27.4 ± 14.5	84.6 ± 7.4
	HMill	99.2 ± 0.5	99.0 ± 0.6	0.5 ± 0.6	99.8 ± 0.3
O-Dist (C+S)	MalJ.-KNN	64.9 ± 1.0	31.8 ± 2.9	2.1 ± 1.4	65.6 ± 2.1
	MalJ.-XGB	80.8 ± 4.6	83.1 ± 7.5	21.5 ± 12.1	89.6 ± 4.4
	HT-KNN	63.3 ± 3.1	77.3 ± 15.0	50.7 ± 18.8	65.0 ± 5.4
	HT-SVM	62.5 ± 5.0	79.8 ± 8.6	54.9 ± 17.0	69.6 ± 6.7
	HTD-KNN	83.2 ± 7.3	79.4 ± 7.3	13.0 ± 13.2	82.2 ± 12.0
	HTD-SVM	67.0 ± 3.0	82.3 ± 13.3	48.4 ± 14.8	69.5 ± 4.5
	HMill	98.2 ± 0.7	98.5 ± 1.0	2.0 ± 1.4	99.4 ± 0.5

to capture coarse but robust statistical cues that generalize across encoder variants.

Among the baseline models, the HashTrick variants again show weak performance. The KNN classifier remains near the random across all conditions (BACC $\approx$ 53%), confirming our findings from the previous experiment. The SVM variant performs somewhat better *In-Dist* (65%), but loses discriminative power once the test data originate from the modified encoder, dropping to 53% in *O-Dist (C)*. This instability highlights the inherent fragility of vector embeddings when the header layout changes.

The HTD-based classifiers achieve substantially higher and more stable accuracy. HTD-KNN reaches $\approx$ 75% BACC across all conditions with virtually no degradation under drift (ΔBACC $\approx$ 0%-pts), showing that the learned distance metric remains effective even for previously unseen encoder structures. HTD-SVM yields slightly lower scores and exhibits higher variance, particularly in the out-of-distribution settings, suggesting that the SVM kernel may be more sensitive to the non-stationarity of pairwise distances.

The HMill again achieves a near-perfect detection performance, maintaining BACC $\approx$ 99% and AUC $\approx$ 99.7% across all scenarios.

6. Experiment 3: Social Media Platforms

6.1. Setup

The third experiment evaluates robustness under a distribution drift induced by social media processing pipelines. Unlike encoder updates or format revisions, this kind of drift arises from the way online platforms recompress, resize, and re-encode uploaded images. Each platform typically employs its own JPEG encoder configuration, including proprietary quantization tables, metadata policies, and application segments that carry platform identifiers or thumbnails. As a result, images originating from different platforms form several homogeneous but mutually distinct distributions. Therefore, detectors trained on data from one subset of platforms may overfit to platform-specific syntactic patterns rather than to generalizable structural properties.

To study this phenomenon, we used the Social-6 dataset, which contains JPEGs collected from six major platforms: Facebook, Instagram, BlueSky, Pinterest, X, and LinkedIn. Unlike in previous experiments, we generated only 50 stego samples per embedding location, reflecting the smaller size of the dataset. Despite the small sample size, each platform exhibits only two to three consistent header templates, making this dataset an ideal test bed for cross-platform generalization.

We simulate the appearance of a new platform using a *jackknife procedure*, where in each run four platforms are used for training and validation, and the other two are withheld for testing. The evaluation follows the same three-level protocol introduced earlier.

6.2. Results

Table 5 presents the results of the third experiment. Compared to the previous settings, this scenario exhibits greater variability, as evidenced by the wider spread of scores. This reflects the inherent heterogeneity of the *Social-6* dataset, where images originate from multiple platforms with distinct recompression policies.

The two MalJPEG models illustrate the limitations of hand-crafted byte-level features in this setting. The KNN variant performs slightly better than in previous experiments, but remains among the weakest detectors, with BACC $\approx$ 65% across all scenarios. The XGBoost version achieves substantially higher in-distribution performance (BACC = 90.9%, AUC = 97.1%), but degrades notably under drift, dropping to 85% in *O-Dist (C)* and 81% in *O-Dist (C+S)*. The same limitation observed in Experiment 1 appears here as well.

In contrast, HashTrick-based models perform poorly mainly due to their representational rigidity. The HT-KNN variant mirrors the MalJPEG results (BACC $\approx$ 65%), while HT-SVM achieves slightly higher accuracy ($\approx$ 75% in-distribution), but drops sharply on unseen platforms. Both exhibit elevated false-positive rates (30–50%), confirming that vector embeddings derived from hashed representations are highly sensitive to platform-specific artifacts and fail to generalize.

The HTD-based classifiers achieve substantially better and more consistent performance. HTD-KNN reaches balanced accuracies above 90% in-distribution and around 85% under drift, corresponding to moderate degradation (ΔBACC $\approx$ 5%-pts and 8%-pts). HTD-SVM performs similarly but with greater variance, particularly in the *O-Dist (C+S)* scenario, where false-positive rates occasionally exceed 40%. Finally, the HMill model consistently delivers near-perfect performance across all conditions, maintaining BACC $\approx$ 99% and AUC $\approx$ 99.5% with negligible drift impact. Its end-to-end representation learning allows it to adapt to heterogeneous platform structures, achieving both accuracy and robustness that is unmatched by other methods.

7. Discussion

The compared methods differ fundamentally in how flexibly they can project the hierarchical structure of JPEG into a discriminatory representation. MalJPEG does not provide agility, and HashTrick is very small, while HTD allows partial adaptation through learned subtree weights, and finally HMill offers high agility by learning parameterized transformations at every level of the hierarchy. This scale of agility is mirrored in the empirical performance hierarchy observed in the experiments.

Across all three experiments, a consistent trend emerges: robustness to distribution drift in JPEG steganalysis directly correlates with how faithfully a model captures the structure of JPEG syntax. The experimental results reveal a clear hierarchy among methods, hand-crafted, vectorized, metric-based, and fully learned, that correspond to the level of structural awareness that each method preserves.

Hand-crafted and vectorized models (MalJPEG, HashTrick) perform adequately when training and testing conditions are identical, but their accuracy collapses even under modest drift. MalJPEG–XGB, despite achieving high in-distribution accuracy, overfits to superficial statistics that vary between encoders, such as marker frequencies or comment lengths. Similarly, feature hashing in HashTrick loses positional and contextual relations between markers, making it sensitive to unseen syntactic variants. These approaches illustrate a fundamental limitation of static, feature-engineered representations: they implicitly assume a stationary format distribution, an assumption invalidated by the continual evolution of JPEG implementations and recompression pipelines.

Hierarchical Tree Distance offers a middle ground between hand-crafting and full representation learning. By explicitly encoding the tree topology and allowing parameterized weighting of branches, HTD retains meaningful structural information. Its stability across the investigated drifts demonstrates that even partial preservation of hierarchical relationships substantially improves generalization. At the same time, the remaining gap to the performance of neural networks suggests that distance-based classifiers still rely on local alignment of subtrees and cannot learn invariant abstractions of the entire file structure.

HMill neural networks stand out as the only approach that consistently maintains near-perfect accuracy under every tested form of drift. Because HMill jointly learns both feature extraction and classification on the raw hierarchical representation, it can model complex dependencies between markers while being invariant to ordering or metadata variations. The absence of measurable degradation across encoders and platforms demonstrates that end-to-end learning over structured data captures generalizable principles of the JPEG format rather than overfitting to specific encoder artifacts. This property is essential for practical deployment, since retraining for each new encoder version or platform would be infeasible.

Together, these findings highlight two broader insights. First, robustness to distribution drift in structural steganalysis arises from hierarchical fidelity: the more a representation preserves the actual structure of the file, the less the method is susceptible to distribution drift. Second, the success of HMill suggests that structure-aware deep learning can provide dependable detection even in heterogeneous environments that change rapidly. For operational steganalysis systems, this implies that investing in models that directly process hierarchical metadata is more effective than attempting to continually retrain feature-based detectors for every new data source.

Currently, the biggest limitation of HMill is that it can be trained using supervised approaches only, and therefore it cannot be used for anomaly detection or trained in a semi-supervised fashion. This limits its application to security domains with a non-trivial number of labeled examples. The anomaly detection problem can be solved by the distance method utilizing HTD distance, as has been shown in [26]. Nevertheless, HTD distance would still need a few samples to optimize the parameters and, as shown above, its robustness with respect to the distribution drift is limited compared to the HMill. Our results suggest that methods directly exploiting hierarchy in the data are superior, but they can be applied to a limited set of scenarios.

8. Conclusion

This paper addressed a largely overlooked problem in steganalysis: detecting payloads hidden in the structural syntax of JPEG files under evolving real-world conditions. Through three complementary experiments that simulate temporal evolution, encoder diversity, and social media platform recompression, we demonstrated that structure-aware methods, and especially the hierarchical neural model HMill, achieve both superior accuracy and exceptional robustness to distribution drift.

Our results highlight that the agility of a detector is determined not by feature quantity but by structural understanding. While feature-based or hashed embeddings, used in prior art, quickly lose discriminative power when file conventions change, hierarchical approaches maintain stable decision boundaries even across unseen encoders. This robustness is crucial for practical deployment, because models must remain dependable even as underlying distributions drift silently and unpredictably, often in ways that escape the developers' attention.

By modeling image files as trees rather than flat byte streams, we can build detectors that evolve with their ecosystems instead of breaking under distribution drift. Ultimately, our study underscores that in the era of rapidly evolving formats, robust steganalysis requires models that understand structure, not just content. Future work should adapt and validate the insights gained here for use with other structured media data formats, foremost compressed video streams [45].

Acknowledgments

This work has been supported by MPO 60273/24/21300/21000 CEDMO 2.0 NPO and the bilateral project "Fundamental Tradeoffs for Information Hiding in Generated Media" (DETERMINE), co-funded by the Czech GACR (project number 25-17259K) and the Austrian FWF (project number PIN2146324).

References

[1] R. Anderson and F. A. P. Petitcolas, "On the limits of steganography," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 474–481, 1998.

[2] C. Cachin, "An information-theoretic model for steganography," *Information and Computation*, vol. 192, pp. 41–56, 2004.

[3] D. Kahn, "The history of steganography," in *Information Hiding (1st International Workshop)*, ser. LNCS 1174, R. Anderson, Ed. Springer, 1996, pp. 1–5.

[4] C. Kurak and J. McHugh, "A cautionary note on image downgrading," in *Computer Security Application Conference (CSAC)*. IEEE, 1992, pp. 153–159.

[5] N. Hamid, A. Yahya, S. Ahmad, and H. Al-Anees, "Image steganography techniques: An overview," *International Journal of Computer Science and Security (IJCSS)*, vol. 6, no. 3, pp. 168–187, 2012.

[6] R. Crandall, "Some notes on steganography," Posted on the Steganography Mailing List, p. 6, 1998, issue 1.

[7] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, "Digital image steganography: Survey and analysis of current methods," *Signal Processing*, vol. 90, no. 3, pp. 727–752, 2010.

[8] M. Chaumont, "Deep learning in steganography and steganalysis," in *Digital Media Steganography*. Academic Press, 2020, pp. 321–349.

[9] M. A. Wani and B. Sultan, "Deep learning based image steganography: A review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 3, p. e1481, 2023.

[10] A. Cohen, N. Nissim, and Y. Elovici, "Maljpeg: Machine learning based solution for the detection of malicious jpeg images," *IEEE Access*, vol. 8, pp. 19 997–20 011, 2020.

[11] R. Chaganti, V. Ravi, M. Alazab, and T. D. Pham, "Stegomalware: A systematic survey of malwarehiding and detection in images, machine learningmodels and research challenges," *arXiv preprint arXiv:2110.02504*, 2021.

[12] D. Puchalski, L. Caviglione, R. Kozik, A. Marzecki, S. Krawczyk, and M. Choraś, "Stegomalware detection through structural analysis of media files," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–6.

[13] T. Gloe, "Forensic analysis of ordered data structures on the example of JPEG files," in *IEEE Workshop on Information Forensics and Security (WIFS)*. IEEE, 2012, pp. 139–144.

[14] N. Hofer, "Increasing trust in image analysis by detecting trellis quantization in JPEG images," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2024, pp. 3834–3840.

[15] N. Hofer and R. Böhme, "Progressive JPEGs in the wild: Implications for information hiding and forensics," in *Workshop on Information Hiding and Multimedia Security (IH&MMSEC)*. ACM, 2023, pp. 47–58.

[16] M. Beneš, N. Hofer, and R. Böhme, "Know your library: How the libjpeg version influences compression and decompression results," in *Proceedings of the 10th ACM Workshop on Information Hiding and Multimedia Security (IH&MMSEC)*. ACM, 2022, pp. 19–25.

[17] C. Pasquini, I. Amerini, and G. Boato, "Media forensics on social media platforms: a survey," *EURASIP J. Inf. Secur.*, vol. 2021, no. 1, p. 4, 2021.

[18] N. Hamid, A. Yahya, R. B. Ahmad, and O. M. Al-Qershi, "Image steganography techniques: an overview," *International Journal of Computer Science and Security (IJCSS)*, vol. 6, no. 3, pp. 168–187, 2012.

[19] A. Westfeld, "Steganalysis in the presence of weak cryptography and encoding," in *Digital Watermarking, 5th International Workshop, IWDW*, ser. LNCS, Y. Shi and B. Jeon, Eds., vol. 5041. Springer, 2006, pp. 19–34.

[20] Z. Xiang, J. Horváth, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, "Forensic analysis of video files using metadata," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1042–1051.

[21] J. D. J. S. Pérez, M. S. Rosales, and N. Cruz-Cortés, "Universal steganography detector based on an artificial immune system for jpeg images," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 1896–1903.

[22] K. He and D.-S. Kim, "Malware detection with malware images using deep learning techniques," in *2019 18th IEEE international conference on trust, security and privacy in computing and communications/13th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2019, pp. 95–102.

[23] V. Verma, S. K. Muttoo, and V. B. Singh, "Detecting stegomalware: malicious image steganography and its intrusion in windows," in *Security, Privacy and Data Analytics: Select Proceedings of ISPDA 2021*. Springer, 2022, pp. 103–116.

[24] B. Dornauer and M. Felderer, "Web image formats: Assessment of their real-world-usage and performance across popular web browsers – replication package," 2023.

[25] Internet Archive. (2025) Wayback machine. Accessed: Mar. 10, 2025. [Online]. Available: https://archive.org/web/

[26] M. Zorek, T. Pevný, and V. Šmídl, "Differentiable distance between hierarchically-structured data," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2025, to appear.

[27] M. Kombrink, S. van Lierop, D. Stolwijk, M. Worring, D. Vrijdag, and Z. Geradts, "Reveal: A large-scale comprehensive image dataset for steganalysis," *Forensic Science International: Digital Investigation*, vol. 55, p. 302006, 2025.

[28] P. Schöttle and R. Böhme, "Game theory and adaptive steganography," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 760–773, 2016.

[29] S. Bernard, T. Pevnỳ, P. Bas, and J. Klein, "Exploiting adversarial embeddings for better steganography," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, 2019, pp. 216–221.

[30] T. Pevny and P. Somol, "Discriminative models for multi-instance problems with tree structure," in *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, 2016, pp. 83–91.

[31] Š. Mandlík, T. Pevný, V. Šmídl, and L. Bajer, "Malicious internet entity detection using local graph inference," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 3554–3566, 2024.

[32] W. Woof and K. Chen, "A framework for end-to-end learning on semantic tree-structured data," *CoRR*, vol. abs/2002.05707, 2020.

[33] T. Rückstieß, A. Huang, and R. Vujanic, "ORIGAMI: A generative transformer architecture for predictions from semi-structured data," *CoRR*, vol. abs/2412.17348, 2024.

[34] T. Pevný and M. Dedic, "Nested multiple instance learning in modelling of HTTP network traffic," *CoRR*, vol. abs/2002.04059, 2020.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[36] T. Pevný and V. Kovarík, "Approximation capability of neural networks on spaces of probability measures and tree-structured domains," *CoRR*, vol. abs/1906.00764, 2019.

[37] D. Huttenlocher, G. Klanderman, and W. Rucklidge, "Comparing images using the Hausdorff distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 9, pp. 850–863, 1993.

[38] C. R. Givens and R. M. Shortt, "A class of Wasserstein metrics for probability distributions," *Michigan Mathematical Journal*, vol. 31, no. 2, pp. 231–240, 1984.

[39] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[40] M. Agarwal, K. S. Gill, R. Chauhan, A. Kapruwan, and D. Banerjee, "Classification of network security attack using knn (k-nearest neighbour) and comparison of different attacks through different machine learning techniques," in *2024 3rd International Conference for Innovation in Technology (INOCON)*, 2024, pp. 1–7.

[41] K. Ghanem, F. J. Aparicio-Navarro, K. G. Kyriakopoulos, S. Lambotharan, and J. A. Chambers, "Support vector machine for network intrusion and cyber-attack detection," in *Sensor Signal Processing for Defence Conference (SSPD)*. IEEE, 2017, pp. 1–5.

[42] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 1113–1120.

[43] H. S. Anderson and P. Roth, "Ember: An open dataset for training static PE malware machine learning models," 2018.

[44] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. ACM, Aug. 2016, p. 785–794.

[45] V. Lachner and R. Böhme, "A critical survey on video steganography: Trends, pitfalls, and recommendations," *ACM Computing Surveys (CSUR)*, 2026.

Appendix A. Hyperparameters

This appendix lists the hyperparameters used for all models and experiments.

TABLE 6: Training hyperparameters

Metric learning		
Method	Parameter	Set of values
Triplet loss	iterations	{100, 200, 300}
	optimizer	Adam
	learning rate	{0.3, 0.5, 0.7, 1.0}
	batch size	{64, 128}
	bag metric	{Hausdorff, Chamfer} distances
	α	{0, 1.0}
	β	{0, 0.001, 0.1, 1.0 }

Classifiers		
HMill	epochs	{5, 10, 20, 30}
	hidden dim	{16, 32, 64}
	optimizer	Adam
	learning rate	{0.01, 0.03, 0.001, 0.003}
	batch size	{32, 64}
KNN	search	Brute Force
	distance	{Cosine, Euclidean}
	k	$\{2 \dots 750\}$
SVM	kernel	$\kappa(x, y) = e^{-\frac{d(x,y)^2}{\gamma}}$
	γ	$\{0.1, 0.5, 1.0, 1.5, \dots, 20.0\}$

Appendix B. Training and Inference Effort

We report computational effort in terms of preprocessing, training, and inference time, explicitly accounting for structural differences between the evaluated methods. All reported values correspond to Experiment 1 and are summarized in Table 7. The measurements are obtained on the full dataset containing approximately $13,000$ samples, of which $6,380$ are used for training and the remaining samples form the test set, including all *In-Dist*, *O-Dist (C)*, and *O-Dist (C+S)* instances (i.e., all samples evaluated at inference time). All measurements were conducted on a system equipped with an Intel Xeon Silver 4110 CPU.

Preprocessing corresponds to feature extraction and is reported per dataset. This applies to feature-based approaches (MalJPEG, HashTrick), which do not involve parameter learning in the conventional sense. Training time is defined uniformly as a single pass over the training data. In HTD, this corresponds to one pass of triplet-loss optimization used to learn the metric parameters, while in HMill it corresponds to one full training epoch. For conventional classifiers, such as SVM and XGBoost, training time is measured as the time required to fit the model on the entire training dataset.

TABLE 7: Training and inference time for all methods on Experiment 1. Reported values represent the mean and standard deviation computed over all performed runs.

Method	Stage	Time [s]	Unit	Notes
MalJ.-KNN	Feature extraction	123154 ± 152	per dataset[1]	–
	PDM computation	2.70 ± 0.19	per dataset	for KNN
	Training	—	—	no training
	Inference	1.08 ± 0.44	per test set	NN eval
	tmp	0.014 ± 0.018	per test set	–
MalJ.-XGB	Feature extraction	123154 ± 152	per dataset[1]	–
	Training	0.038 ± 0.001	per train set	–
	Inference	0.014 ± 0.018	per test set	–
HT-KNN	Feature extraction	54.86 ± 0.55	per dataset	–
	PDM computation	143.72 ± 1.90	per dataset	for KNN
	Training	—	—	no training
	Inference	1.73 ± 0.11	per test set	NN eval
HT-SVM	Feature extraction	54.86 ± 0.55	per dataset	–
	PDM computation	143.72 ± 1.90	per dataset	for Kernel
	Training	6.45 ± 0.37	per train set	after PDM
	Inference	3.50 ± 0.06	per test set	kernel eval
HTD-KNN	Metric learning	6790 ± 259	per train set	triplet loss
	PDM computation	1346 ± 247	per dataset	for KNN
	Training	—	—	no training
	Inference	4.68 ± 0.10	per test set	NN eval
HTD-SVM	Metric learning	6790 ± 259	per train set	triplet loss
	PDM computation	1346 ± 247	per dataset	for Kernel
	Training	4.19 ± 0.52	per train set	after PDM
	Inference	2.54 ± 0.23	per test set	kernel eval
HMill	Training	611.5 ± 109.9	per train set	end-to-end
	Inference	10.01 ± 0.80	per test set	–

Inference time is reported per test set rather than per sample to ensure comparability across methods. This is particularly important for distance-based approaches, where inference involves computing pairwise distances between sets of samples.

For methods based on pairwise distances (HTD and HashTrick with SVM or KNN), the computation of the pairwise distance matrix (PDM) constitutes a dominant cost. In the case of KNN, inference directly reduces to computing this matrix. For SVM, the PDM is required during training and subsequently reused for kernel evaluation at inference time. Consequently, PDM computation time is reported separately at the dataset level. For HTD, PDM computation is performed using a deduplicated scheme that exploits repeated substructures in the data, reducing the number of distance evaluations. As a result, the total computation time is not proportional to the number of samples, and per-sample normalization would therefore be misleading.

Finally, we report the number of parameters for each method as a measure of model complexity. The notion of a parameter depends on the model class. In SVM, it corresponds to the number of support vectors. In XGBoost, it is defined as the number of internal decision nodes (splits)

1. MalJPEG feature extraction is relatively slow because it relies on sequential byte-level parsing of the JPEG file, as implemented in prior work. This design requires processing the file in a strictly sequential manner, which limits efficiency. To mitigate the overall runtime, we parallelized the extraction procedure.

in the ensemble, i.e., the splitting rules that define tree structure. In HTD, the parameter count includes both the number of support vectors and the learned metric weights. All reported times depend on the dataset, as the underlying schema determines both the number of parameters and the computational cost, and should therefore be interpreted comparatively within the same experimental setting rather than as absolute measures.