DFRWS 2023 USA - Proceedings of the Twenty Third Annual DFRWS Conference

Combining AI and AM − Improving approximate matching through transformer networks

Frieder Uhlig [a, f, 1, *], Lukas Struppek [a, 1], Dominik Hintersdorf [a, 1], Thomas Göbel [b, 1], Harald Baier [b], Kristian Kersting [a, c, d, e]

[a] Department of Computer Science, Technical University of Darmstadt, Germany
[b] University of the Bundeswehr Munich, Germany
[c] Centre for Cognitive Science, Technical University of Darmstadt, Germany
[d] German Research Center for Artificial Intelligence (DFKI), Germany
[e] Hessian Center for AI (hessian.AI), Germany
[f] SEC Consult Unternehmensberatung GmbH, Germany

ARTICLE INFO

ABSTRACT

Approximate matching is a well-known concept in digital forensics to determine the similarity between digital artifacts. An important use case of approximate matching is the reliable and efficient detection of case-relevant data structures on a blacklist (e.g., malware or corporate secrets), if only fragments of the original are available. For instance, if only a cluster of indexed malware is still present during the digital forensic investigation, the approximate matching algorithm shall be able to assign the fragment to the blacklisted malware. However, traditional approximate matching functions like TLSH and ssdeep fail to detect files based on their fragments if the presented piece is relatively small compared to the overall file size (e.g., like one-third of the total file). A second well-known issue with traditional approximate matching algorithms is the lack of scaling due to the ever-increasing lookup databases. In this paper, we propose an improved matching algorithm based on transformer-based models from the field of natural language processing. We call our approach Deep Learning Approximate Matching (DLAM). As a concept from artificial intelligence, DLAM gets knowledge of characteristic blacklisted patterns during its training phase. Then DLAM is able to detect the patterns in a typically much larger file, that is DLAM focuses on the use case of fragment detection. Our evaluation is inspired by two widespread blacklist use cases: the detection of malware (e.g., in JavaScript) and corporate secrets (e.g., pdf or office documents). We reveal that DLAM has three key advantages compared to the prominent conventional approaches TLSH and ssdeep. First, it makes the tedious extraction of known to be bad parts obsolete, which is necessary until now before any search for them with approximate matching algorithms. This allows efficient classification of files on a much larger scale, which is important due to exponentially increasing data to be investigated. Second, depending on the use case, DLAM achieves a similar (in case of mrsh-cf and mrsh-v2) or even significantly higher accuracy (in case of ssdeep and TLSH) in recovering fragments of blacklisted files. For instance, in the case of JavaScript files, our assessment shows that DLAM provides an accuracy of 93% on our test corpus, while TLSH and ssdeep show a classification accuracy of only 50%. Third, we show that DLAM enables the detection of file correlations in the output of TLSH and ssdeep even for fragment sizes, where the respective matching function of TLSH and ssdeep fails.

1. Introduction

Digital forensics comprises the analysis and interpretation of digital artifacts. One of the biggest challenges facing digital forensic investigation is coping with the vast number of files to be processed. Hashing algorithms have become an indispensable part of computer science, since the computed hashes can be used as

* Corresponding author. Department of Computer Science, Technical University of Darmstadt, Germany.

E-mail addresses: frieder.uhlig@sec-defence.com (F. Uhlig), lukas.struppek@cs.tu-darmstadt.de (L. Struppek), dominik.hintersdorf@cs.tu-darmstadt.de (D. Hintersdorf), thomas.goebel@unibw.de (T. Göbel), harald.baier@unibw.de (H. Baier), kersting@cs.tu-darmstadt.de (K. Kersting).

[1] Equal contribution.

unique identifiers to compare digital artifacts. For traditional cryptographic hashes, such as MD5 and SHA-2, even changing a single bit already alters the hash value significantly due to their diffusion property (intended for one-way functions), which obscures the relationship between the input and its corresponding hash value. Cryptographic hashes are therefore unsuitable for recognizing similar artifacts. Fuzzy hashing, on the other hand, breaks the cryptographic diffusion and thus establishes a relationship between the original file and its corresponding hash. Through approximate matching, the similarity of two fuzzy hashes can be determined.

The concept of approximate matching is not limited to traditional fuzzy hashes. In the image domain, similar approaches based on neural networks to extract meaningful features from images are used for image retrieval (Liu et al., 2016; Zhao et al., 2015) or detecting illegal content (Apple Inc, 2021; Struppek et al., 2022).

A recent survey by Singh (2021) shows that 65% of digital forensic investigators are actively using fuzzy hashes during their investigation processes. Fuzzy hashing is primarily used to identify related documents, but they are also used in other contexts which are based on similarity search. Examples are data loss prevention (Göbel et al., 2021), IoT device identification (Charyyev and Gunes, 2020; Göbel et al., 2022a,b), genome sequencing (Healy and Chambers, 2011) or biometric template protection (Ong et al., 2014). However, one main application for fuzzy hashes remains malware detection and analysis (Diaz, 2020; Lazo, 2021), for example, as used in Google's VirusTotal (Shiel and O'Shaughnessy, 2019).

Traditional approaches compute similarity scores between fuzzy hashes to identify shared fragments of different files. However, such similarity scores are limited in their expressiveness and, as shown by Göbel et al. (2022a,b), frequently fail to detect shared fragments or even state misleading similarities between files. The fact that fuzzy hashes translate bytes into computable strings makes them especially interesting for processing on the bases of machine-learning techniques. With our approach, which we call *Deep Learning Approximate Matching (DLAM)*, we aim to bridge recent advances in deep learning and fuzzy hashing and demonstrate large performance improvements for the task of fragment detection by combining both concepts.

More specifically, we introduce a transformer-based (Vaswani et al., 2017) approach to detect parts of file fragments in their corresponding fuzzy hashes. We empirically show that DLAM improves fragment detection in fuzzy hashes produced by the most commonly used algorithms `ssdeep` (Kornblum, 2006) and `TLSH` (Oliver et al., 2013), whereas traditional similarity measures fail to detect the fragments. In contrast to state-of-the-art fuzzy hashing algorithms and previous deep learning approaches, DLAM consistently classifies the fragments with an accuracy of over 90% on multiple tested file types, namely JavaScript, PDF and XLSX.

We point out to restrict our evaluation on approximate matching schemes, which are actually used and provide an executable implementation. In summary, we make the following contributions:

- We introduce Deep Learning Approximate Matching, or DLAM in short, that combines efficient fuzzy hashing algorithms with transformer-based models from natural language processing.
- We show that DLAM is superior to the most common conventional approximate matching schemes `ssdeep` and `TLSH` and even works for small fragment sizes where traditional similarity measurements fail completely.
- We further demonstrate that DLAM achieves comparable results to state-of-the-art multi-resolution fuzzy hashing algorithms,

such as `mrsh-v2` and `mrsh-cf` in the case of fragment detection.
- We provide the complete source code of DLAM on GitHub: https://github.com/warlmare/DLAM.

2. Background and related work

In the following, we introduce the basics of fuzzy hashing, including `TLSH` and `ssdeep` as conventional fuzzy hashing algorithms on which our DLAM approach is based. We further describe the characteristics of transformer networks, which build the foundation of our machine-learning approach.

2.1. Fuzzy hashing and approximate matching

Fuzzy hashing was first introduced in the mid-2000s in order to perform altered document and partial file matching by Kornblum (2006). Generally, fuzzy hashing algorithms H are functions that compute a dense hash representation $H(x)$ of a file x. The goal is to provide similar hashes, which are also called digests, for similar inputs: $H(x) \approx H(y) \Leftrightarrow x \approx y$. This property distinguishes them from traditional cryptographic hashes for which small input changes lead to strong output changes, also known as cryptographic diffusion or avalanche effect. Then a similarity function $sim(H(x), H(y))$ is applied, which computes the similarity between two files x and y based solely on their fuzzy hashes. This procedure of assessing the similarity of artifacts or patterns is known as approximate matching. Approximate matching algorithms all work in a three-step manner: they first select features from a digital artifact and generate a hash digest from these features, which then can be compared to other hashes in a third step. As for the comparison step, there are three ways in which current algorithms achieve their goal: either based on plain byte, syntactic or semantic commonalities. The matching mechanism is chosen based on the digital artifact that must be processed. As an example, malware developers often obfuscate their payload deliberately to prevent detection. Benign files might have been further compressed or rearranged through software, which makes it hard to distinguish between obfuscated malware and benign files.

To deal with these issues, approximate matching research has come up with two approaches: The first approach is to create new algorithms that are more resilient and attest the similarity on a more granular level (Azarafrooz and Brock, 2018). The other approach is to pre-process the artifacts before hashing them and focusing on selected parts that hold vital information (Shiel and O'Shaughnessy, 2019). With our work, we propose a third option and detect shared commonalities in fuzzy hashes using machine learning approaches. We note that Microsoft pursues a similar direction (Lazo, 2021). However, this direction has so far only barely been formalized in academic research. Fig. 1 illustrates the differences in the approaches of fragment detection of conventional approximate matching in Fig. 1a and our proposed DLAM in Fig. 1b.

Overall, the combination of fuzzy hashes and machine learning is still in its early stages. Raff et al. (2018) first proposed to train neural networks for malware detection from raw byte sequences. However, training networks with sufficient capacity on entire files is time- and resource-consuming. Peiser et al. (2020) used simple feed-forward neural networks to detect malware in JavaScript files through static analysis based on features computed by fuzzy hashing algorithms and compared their results with conventional JavaScript malware classifiers, none of which rely on fuzzy hashes. The authors showed that simple models, trained on a small dataset with only 5000 samples, already achieved a high detection accuracy. However, Peiser et al. (2020) limited their investigation to
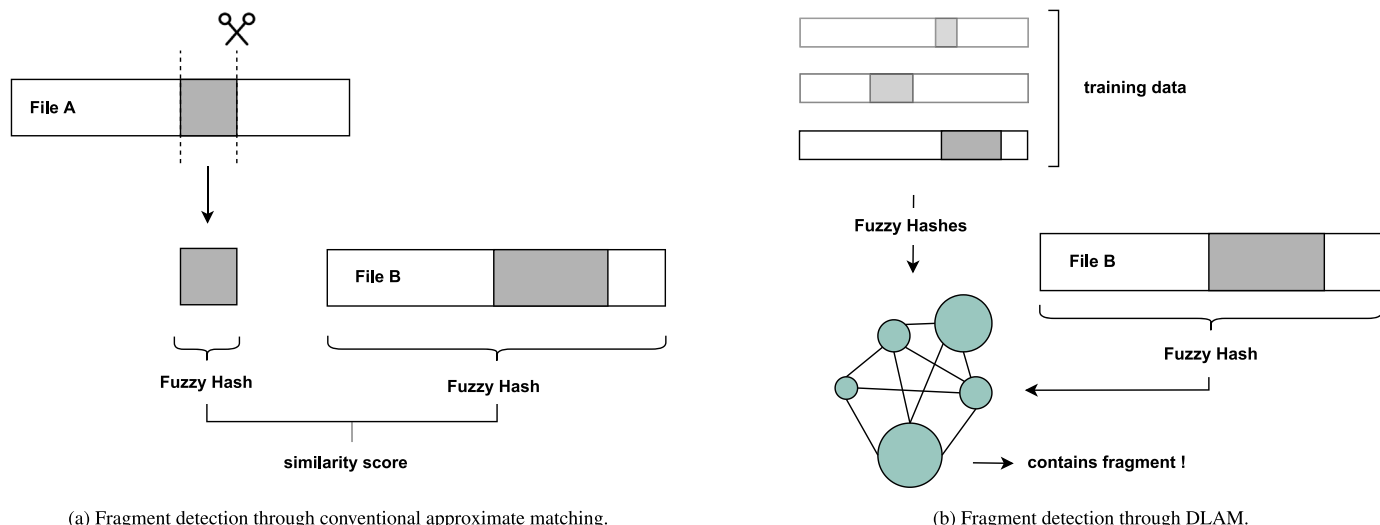
(a) Fragment detection through conventional approximate matching.

(b) Fragment detection through DLAM.

Fig. 1. Fragment detection through conventional approximate matching and through DLAM. In conventional approximate matching, the exact fragment that needs to be detected has to be extracted first in a costly operation. Through DLAM, models are trained to perceive the fragment as a common denominator and classify files based on its presence.

JavaScript malware detection and only trained their models on JavaScript files that either contained malware or did not.

In our work, we take a more holistic view and contextualize the fusion of deep learning and digital forensics based on fuzzy hashing. We further evaluate our approach on a set of different file types by injecting synthetic fragments into real-world files. We use real-world files and synthetic fragments because they allow us to study the influence that the total fragment content in a file has on the classification accuracy. Malware samples rarely come with the information how much malware is contained in a sample.

Throughout this paper, DLAM is used in combination with the fuzzy hashes TLSH and ssdeep, which we now introduce in more detail. However, we emphasize that DLAM is, in principle, also compatible with other fuzzy hashing algorithms.

Raff and Nicholas (2018) explained that LZJD could be suitable particularly for the application of machine learning. They highlight the metric space and kernel properties of the Jaccard distance that LZJD inherits.

TLSH (Trend Micro Locality Sensitivity Hash) is a hashing algorithm first presented by Oliver et al. (2013). It is specifically intended for malware detection and clustering. TLSH scans the byte code of a file with a sliding window and combines 5 bytes at a time into a unit using the Pearson hash function (Pearson, 1990), a fast non-cryptographic hashing procedure. These units are then mapped into an array of so-called buckets. Next, the digest body, a hexadecimal string, is constructed from the array of buckets by dividing it into 3 distinct quartiles q1, q2 and q3. The first three bytes of the hash are the digest header, which primarily consists of checksums, and forms together with the body a 70-character hexadecimal string. The final TLSH hash is a fixed-length string with 72 characters. To compare two hashes, TLSH calculates an approximation to the hamming distance between two digest bodies. The relation of two artifacts is expressed through a distance score between 0 (very similar) and 1000 (not similar).

ssdeep was introduced by Kornblum (2006). The algorithm breaks a file up into pieces using a rolling hash function, which hashes an input in a sliding window approach. Then, a non-cryptographic hash function is used to hash every piece that has been created by the rolling hash function. These smaller hashes are concatenated and form the hash signature for the whole input file. To compare two hashes, a distance measure is used that determines

the correlation between the two hashes. The similarity is indicated by a score between 0 (not similar) and 100 (very similar). As the produced hashes can vary in size, the final hash contains information about the total size at the beginning of the hash. Even though the overall hash is variable in its length, the size is bounded to 148 encoded characters.

2.2. Language processing with transformers

To perform similarity analyses on fuzzy hashes, we interpret the binary strings as a language representation and process these strings using transformer networks. Since their introduction in 2017 by Vaswani et al. (2017), transformers are the dominating architecture for natural language processing and are even increasingly used for image processing (Dosovitskiy et al., 2021a). Unlike previous state-of-the-art models for sequence modeling, transformers rely on the attention mechanism and remove the recurrent components of previous approaches, such as LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014). More specifically, they rely on the multi-head self-attention mechanism, which enables the models to process different representations at different positions by running an attention mechanism several times in parallel.

The multi-head attention mechanism consists of three learned matrices: queries Q, keys K, and values V. The attention A is computed by the dot-product $A = \frac{QK^T}{\sqrt{d_k}}$, where d_k is a scaling factor based on the dimension of keys. The attention output is then computed by the weighted sum $Att(Q, K, V) = softmax(A) \cdot V$, with the attention weights computed by the softmax function and V representing the input sequence. In other words, the softmax of the attention $softmax(A)$ is weighing the importance of the tokens in the input sequence to the task at hand. Since detecting relations between different positions in the hashes is essential for fragment detection, transformers are a natural architecture choice for this task.

Transformers usually need large amounts of data to be trained on. Therefore, pre-training plays a crucial role in their performance. Devlin et al. (2019) proposed bidirectional encoder representations from transformers (BERT), a masked language model, which introduced a novel pre-training procedure. To train the masked language model, randomly selected words in the input sequences of

the training corpus are masked out. The model then has to predict the masked word based on its surrounding context. A pre-trained model can then be fine-tuned for other NLP tasks with only a single additional output layer added.

After demonstrating impressive results in natural language tasks, the transformer architecture has been transferred to other domains as well. These include computer vision with vision transformers (Dosovitskiy et al., 2021b), speech recognition with convolutional-augmented transformers, called conformers (Gulati et al., 2020), and reinforcement learning with decision transformers (Chen et al., 2021). With our work, we take the application range of transformers one step further and combine them with fuzzy hashing algorithms to improve approximate matching.

3. Transformer-Boosted approximate matching

In machine learning assisted common fragment detection, the model should learn what constitutes a regular file and what constitutes a file containing a particular element we want to detect. In this work, we introduce DLAM to improve the detection behavior of traditional fuzzy hashing algorithms. Our approach consists of two steps. For each input file *x* to check, we first compute a dense hash representation $H(x)$ using a fuzzy hash algorithm *H*. The fuzzy hashes serve in this case as an intermediate representation of the files and can be seen as an input pre-processing step. The hashes reduce the complexity and length of the original files and make them processable by deep learning models.

We investigate supervised fragment detection, which comes down to a binary classification of files to predict if some fragment is present in an input file. The hashes and binary labels, which indicate whether a fragment was hidden in the input file or not, are then provided to a deep learning model, in our case a transformer, for training. Since the hashes are in a byte string representation, they are tokenized and interpreted as a sequence to train the model. Because the order of the bytes is crucial to the classification task, we added a simple absolute positional encoding to the sequence to indicate the sequence order to the transformer model after tokenizing the hash strings.

As stated in Section 2.1, ssdeep produces hashes of variable length. As Fig. 2b illustrates, all hashes produced by ssdeep are padded with zeros to unify the length of the hashes, allow parallel training of the models, and apply attention masking to train the transformer model. Attention masking is used to prevent the model from attending to the padding tokens and only paying attention to the informative tokens of the hash values. As different training samples have different sizes, masking out the padding tokens prevents the model from overfitting the training data and predicting the appearance of the fragment based on the number of padding tokens.

3.1. Experimental protocol

In the following, we will describe the experimental setting we used to evaluate our approach. The pipeline for our experiments is visualized in Fig. 3.

Fuzzy Hashing Algorithms: In our experiments, we focus on ssdeep and TLSH, as both are widely used and form the de-facto standard for malware detection. As such, they are both part of Google's VirusTotal, Maltego and other actively used virus detection platforms. Thanks to the fact that transformers can process inputs of arbitrary length, our approach is also applicable to fuzzy hashes of arbitrarily length, such as those generated by sdhash or mrsh-v2 (mrsh-cf is currently unable to provide its fuzzy hashes as readable strings). However, multi-resolution hashes are many times longer than TLSH and ssdeep, which are limited in their

maximum length. This potentially unlimited length is a drawback when processing fuzzy hashes by machine learning. In their paper, Peiser et al. (2020) also apply a simple feed forward network to the sdhash algorithm, which is also theoretically unbounded in length. However, they use the method of count vectorization, which means that important information such as the relative position of the segments, which we consider crucial, is lost and therefore abandon the approach adopted by Peiser et al. (2020).

The feature extraction pipeline for both fuzzy hashes ssdeep and TLSH is visualized in Fig. 2. According to Peiser et al. (2020), when using a rolling-window approach for tokenizing the fuzzy hashes produced by TLSH, the classification accuracy is increased with a feed-forward network. In our experiments, the rolling-window approach did not influence the classification accuracy with a transformer. However, to ensure comparability to Peiser et al. (2020), we still used a rolling-window approach for tokenizing the fuzzy hashes produced by TLSH.

Contrary to the findings of Peiser et al. (2020), who used a similar rolling window approach on ssdeep for training their feed-forward network as well, we could not confirm that such an approach improved the classification performance of the feed-forward models when working with ssdeep. Therefore, we applied a simple per-char integer encoding on the ssdeep hash, as can be seen in Fig. 2b. We also removed the chunk size and colons from the ssdeep hashes, for both of the model architectures. During our experiments, we noticed that the retention of these elements leads to a worse classification accuracy by the models.

Testbed: The test environment in which all our tests were conducted is as follows. We use a virtual machine equipped with 16x Intel Xeon Platinum 8280 CPU @ 2.70 GHz, 2 × 16 GB DDR4-3200 RDIMM ECC, 800 GiB SSD, Ubuntu 21.10 Linux 5.13.0−40-generic x86_64. In addition, three Nvidia Tesla V100s with 32 GB VRAM are used for the training of the deep learning assisted approximate matching models. All modules for the creation of the training data as well as the machine learning models and the evaluation were realized in Python 3.8. Although this setup is quite extensive, it is still not sufficient when dealing with fuzzy hashes of variable length. We suggest that this should be looked into in future research with more powerful hardware.

Data Selection: To investigate the extent to which the classification performance of the models depends on the type of input file, we selected common file types, namely JavaScript, PDF and XLSX, for training and evaluation. Göbel et al. (2021) showed that the accuracy of approximate matching indeed varies depending on the file type. Therefore, we trained and evaluated one model per file type. Since no single file corpus exists that is big enough to allow for training and evaluation of models on all file types, multiple corpora were used. We combined the *Govdocs corpus* (Garfinkel et al., 2009), *SRILabs JavaScript corpus* (Raychev et al., 2016) and the *FUSE corpus* (Barik et al., 2015) and used 100,000 samples for each of the file types. This number of samples per file type represents the maximum amount of files that could be assembled with a reliable provenience, meaning that they originate from corpora that have been screened for duplicates and are balanced in size.

For evaluation of the models, 5000 samples for each file type from the *NapierOne corpus* (Davies et al., 2022) were used. For training and evaluation, only files that were large enough to be reliably hashed by TLSH were selected. Since the files for training and evaluation were collected from different sources, the risk of having duplicate files in the training and test sets is minimal.

Evaluation Metrics: We used various standard metrics for evaluating our experimental results. We measured the accuracy (Acc), the false-positive rate (FPR), the true-positive rate (TPR), the false-negative rate (FNR), and the true-negative rate (TNR). Regarding the application of our approach in digital forensics for
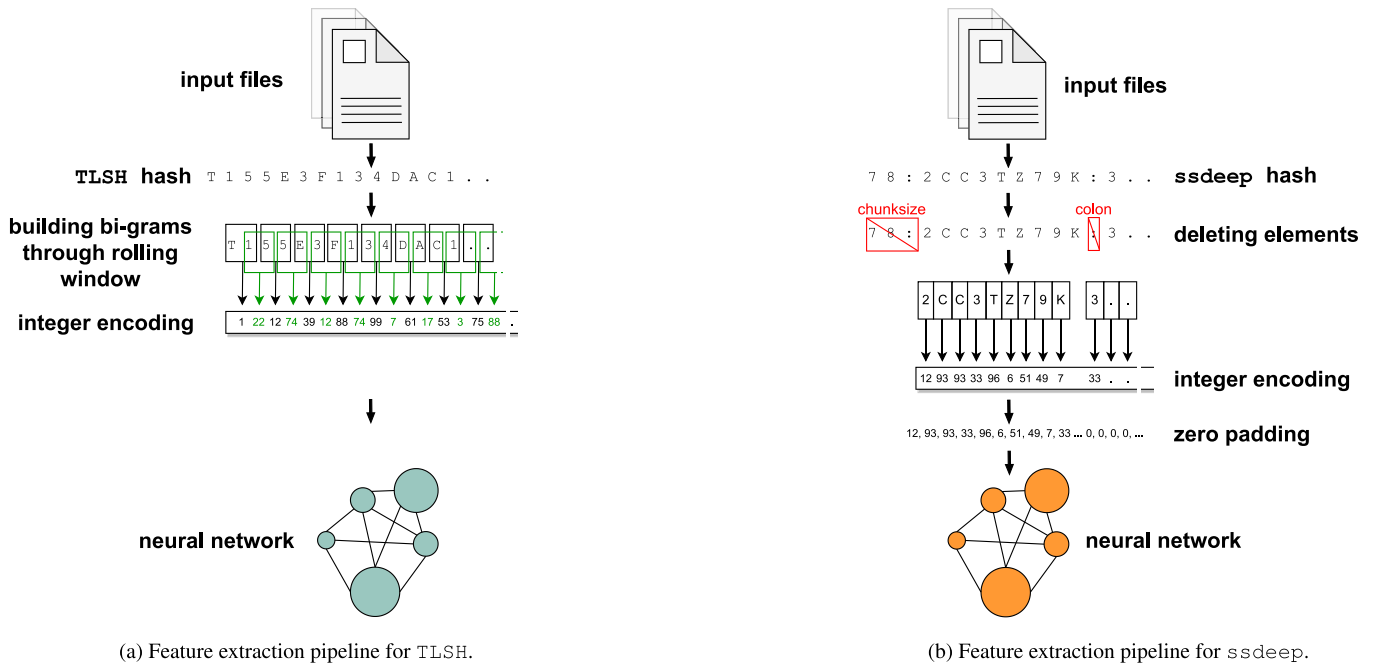
(a) Feature extraction pipeline for `TLSH`.

(b) Feature extraction pipeline for `ssdeep`.

Fig. 2. Feature extraction and prediction pipeline. On the left, the process for `TLSH` hashes is depicted as it is used for both models (feed-forward and transformer). On the right, the process for `ssdeep` hashes is depicted as it is used for creating embeddings for transformer and feed-forward models.
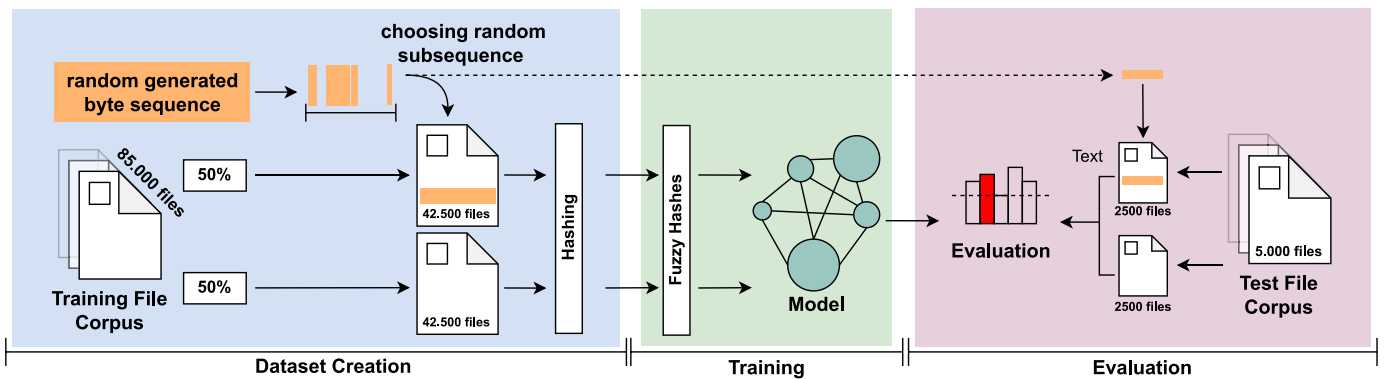zusätzliche Affiliation.

Fig. 3. The three stages of our training and evaluation pipeline.

malware or sensitive file detection, false-positive and false-negative predictions are the most relevant performance indicators. False-negative predictions represent all files that a filter failed to detect. Any false-negative prediction can mean a compromise of systems with potentially devastating consequences. False-positive predictions, in turn, are fatal misclassifications in the context of malware detection because they impair processes for no reason and lead to a sustained loss of confidence in the technology by the users.

Training Setup: The architecture of the transformer model relies on the *TinyBERT* architecture by Jiao et al. (2020). The model is a smaller, distilled version of the widely used, pre-trained language model BERT Devlin et al. (2019). Compared to BERT, TinyBERT models have significantly fewer number of parameters, which improves training and inference time noticeably. All our experiments were implemented in the deep-learning framework PyTorch (Paszke et al., 2019). We note that a single GPU, as, for example,

provided by Google Colab for free, is sufficient to train our models on 100,000 files in 20 min. Training on more powerful, data-center-grade GPUs, e.g., an NVIDIA A100 GPU, can speed up the training time significantly. Furthermore, training can also be scaled across multiple GPUs, allowing for larger batch sizes to accelerate the training process even more. Adaptive sampling strategies to create a representative subset of available data might also be used to reduce the effective number of samples in the training data.

Following the original TinyBERT architecture, our transformer encoder model has 12 attention heads and 4 hidden layers. The intermediate feed-forward layer size in the encoder is 1200. The dimensionality of the encoder and pooling layers is 312. This model has a sufficient size for our purposes, as it needs less training data than larger transformers. We further initialized our model with pre-trained weights trained on a natural language task and found that it boosts the performance even if fine-tuned on abstract hashes compared to training a randomly initialized model from scratch.
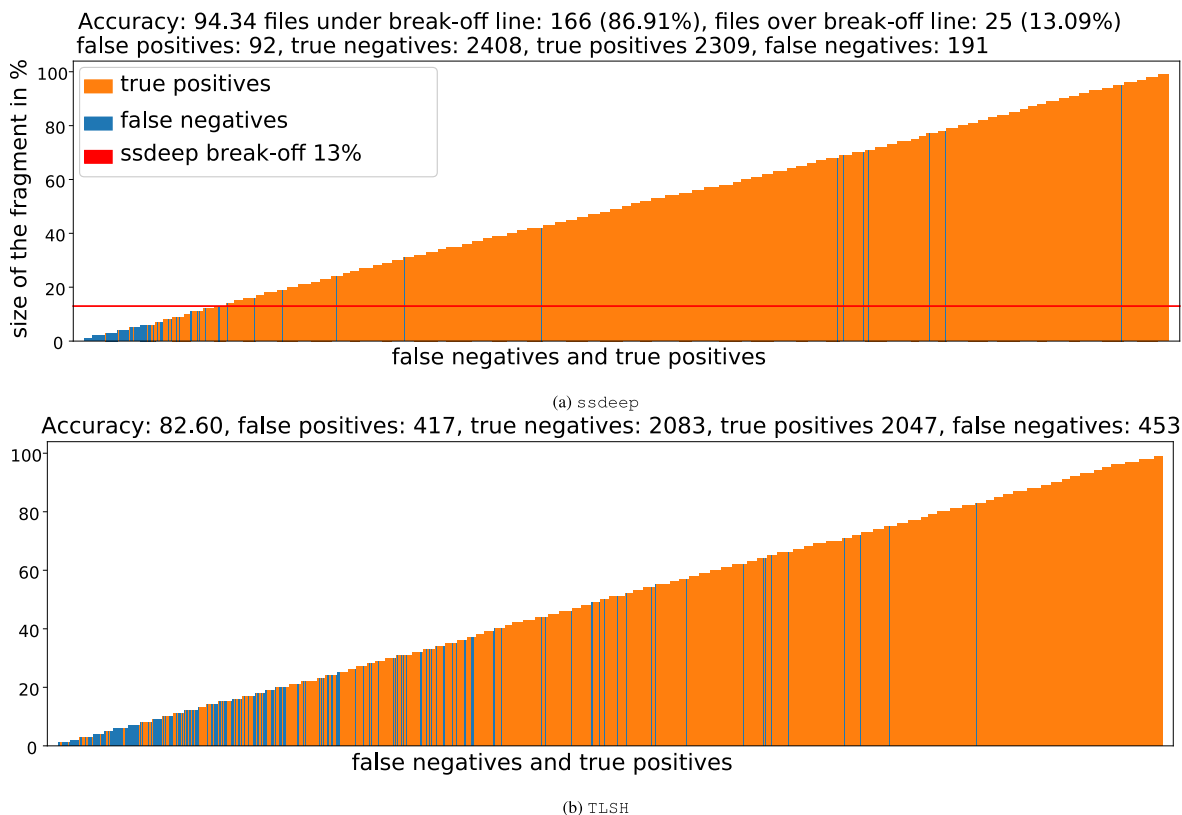
Accuracy: 94.34 files under break-off line: 166 (86.91%), files over break-off line: 25 (13.09%)
false positives: 92, true negatives: 2408, true positives 2309, false negatives: 191

(a) `ssdeep`

Accuracy: 82.60, false positives: 417, true negatives: 2083, true positives 2047, false negatives: 453

(b) `TLSH`

Fig. 4. Classification results of DLAM with a transformer model for all PDF files that carry an to-be-detected fragment within them. The blue bars symbolize false negatives, while the yellow ones symbolize true positives. The height of each bar is proportional to the size of the to-be-detected fragment within the file. There are 2500 bars in each graph. The red line in Fig. 4a symbolizes the so-called *ssdeep-break-off*. It corresponds to the minimum amount of relation that `ssdeep` can conventionally approximately match.

The output layer is a standard feed-forward layer with two output neurons.

For comparison reasons, we also trained a simple feed-forward neural network, whose architecture was inspired by the work of Peiser et al. (2020). The model consists of three fully-connected layers, followed by batch normalization layers (Ioffe and Szegedy, 2015) and ReLU activations. It also contains a dropout layer with dropout probability $p = 0.125$ after the first block.

We trained all networks using the *Adam optimizer* (Kingma and Ba, 2015) with a fixed learning rate of 10^{-3}, a batch size of 1024 tokenized hashes for our transformer model (TinyBERT), and a smaller batch size of 512 for the feed-forward network. We further used 15% of the training data as a validation set and performed early stopping, meaning that we stopped training after the validation loss started to increase. We then used a standard cross-entropy loss function for optimization:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{n} y_i \cdot \log(\widehat{y_i}).$$

Here, $y_i \in 0, 1$ denotes the ground-truth binary indicator if the corresponding file contains a to-be-found fragment, and $\widehat{y_i} \in [0, 1]$ denotes the predicted probability of $y_i = 1$. We emphasize that the model could also be designed with a single output and optimized with a binary cross-entropy loss. However, by designing the model for multi-classification, we could easily extend it for additional classes, if necessary.

For each of the three file type settings, the models were trained on 85,000 files and evaluated on 5000 files. This is visualized in Fig. 3. Note that the training and evaluation files come from different datasets. Into half of the training and evaluation files, a distinct fragment in the shape of a randomly generated byte sequence was inserted. Even though the same randomly generated byte sequence was used for creating all the files containing a fragment, the specific elements that were copied were chosen at random. In addition, the inserted fragment could occupy between 1% and 99% of the original size of the file.

To investigate how traditional fuzzy hashes perform the approximate matching task without the help of machine learning, the common fuzzy hashing algorithms `ssdeep`, `TLSH`, `mrsh-cf`, and `mrsh-v2` were used to compare each evaluation file with the entire randomly generated to-be-found fragment sequence. If their resulting similarity scores were greater than zero, they were considered a positive prediction. If the similarity score was zero, this was considered a negative prediction.

3.2. Experimental results

We now want to empirically investigate the following research questions:

Q1: Can DLAM detect file fragments more precisely than traditional approximate matching?
Q2: Does the type of file containing the fragment affect detection with DLAM?
Q3: Does DLAM with transformer networks perform better than with simple neural networks?
Q4: Can DLAM compensate for weaknesses in conventional approximate matching with fuzzy hashing?

Q1: DLAM meets accuracy of state-of-the-art fuzzy hashing algorithms. We state our experimental results, which compare our DLAM approach to common and state-of-the-art fuzzy hashing algorithms in Table 1. The table includes the accuracy, false-positive rate, true-positive rate, false-negative rate, and true-negative rate in percentage. While ssdeep and TLSH on their own completely fail in detecting the files with injected fragments for all file types, the state-of-the-art hashing algorithms mrsh-cf and mrsh-v2 are still able to reliably identify the fragments in most cases. This is especially interesting since the files to be classified have the same length as the fragment, which in theory is an easy setting for ssdeep and TLSH as was shown by Martínez et al. (2020). Unlike ssdeep and TLSH, mrsh-cf and mrsh-v2 are multi-resolution hashes, which can map many more features into the hashes. As a result, the length of the hashes is not limited to a maximum size, which increases accuracy for similarity matching since less information about the files is lost.

As for the deep learning approaches, our approach is on par with the feed-forward model of Peiser et al. (2020) for JavaScript files, but significantly outperforms the feed-forward model on PDF and XLSX files by 15.24% and 3.56%, respectively. Our transformer-based approach generally performs better when using the hashes of ssdeep than with using the TLSH hashes. The superiority of ssdeep over TLSH for approximate matching with real-world file types was also pointed out by Göbel et al. (2022a,b).

Our approach also outperforms the state-of-the-art multi-resolution fuzzy hashing functions mrsh-cf and mrsh-v2 on the JavaScript files. While the accuracy of the approach of Peiser et al. (2020) is 18.2% lower than the state-of-the-art hashes, the accuracy of our approach is only 2.96% below the mrsh-cf hash on PDF files. On XLSX files, the transformer-based DLAM approach even outperforms the mrsh-cf hash and achieves an accuracy of 97.36%. This is especially interesting since our transformer-based approach enables fragment detection based on fuzzy hashes like ssdeep and TLSH, which scale much better than the mentioned multi-resolution hashes with only moderate accuracy losses when using DLAM. As mentioned at the beginning, ssdeep and TLSH are used for malware detection in practice. Their short length means that they map large amounts of data to a fraction of the disk space that mrsh-v2-hashes would take up for the same data.

Fig. 4a and b show the false negatives (as blue bars) and the true positives (yellow bars). The height of the bars represents the proportion of the fragment in the total file. In both figures, the accumulation of blue bars in the left half of the graphs shows that the smaller the proportion of to-be-found fragment in a file, the higher the risk of a false-negative classification by the model.

With ssdeep, the model detects the to-be-found fragments in the hashes much more consistently, even for small fragment sizes. The red line in Fig. 4a is an approximation of the minimum amount of similarity that two files need to have in common, in order for ssdeep to approximately match them. Göbel et al. (2022a,b) discovered that between 9% and 13% was the minimum amount of common content that two randomly generated files of size 2048 KB needed to possess in order for ssdeep to find any relation. Even though this minimum was never confirmed for PDF files specifically, our experimental results support this theory.

The fact that almost 87% of all false negatives have a smaller fragment than 13% means that the smaller the fragment, the less likely it is to be classified correctly. The false negatives in Fig. 4b are more widespread and cannot primarily be attributed to a small fragment size, as for ssdeep in Fig. 4a. Göbel et al. (2022a,b) stated that TLSH was less precise than ssdeep when tested on various file types in their needle in a haystack test case. An observation that we could confirm with our tests. The classification accuracy of any model was consistently lower for TLSH than that of ssdeep. Our experiments show that working consciously with DLAM and ssdeep within these boundaries — only classifying files with to-be-found content of more than 13% — even leads to a higher classification accuracy than with mrsh-cf or mrsh-v2.

Q2: DLAM reliably detects fragments for all file types. Our results in Table 1 further illustrate that the accuracy with which DLAM detects fragments in files seems to be more stable in face of different file types. Using a feed-forward network leads to highly fluctuating accuracy values for each file type. The same can be seen for the multi-resolution hashes. Even though their accuracy is very high on PDF and XLSX files, there is a high loss of accuracy on JavaScript files, indicating that the file types have a high impact on the performance of state-of-the-art fuzzy hashes. Our transformer-based DLAM approach, on the other hand, is much more stable and the accuracy for the fragment detection is consistently above 90% for all three file types when using ssdeep hashes. It, therefore, offers higher reliability in its predictions for different domains than fuzzy hashing algorithms on their own.

Q3: Transformers outperform simple neural networks for

Table 1
Comparison of our results for fragment detection in JavaScript (JS), PDF, and XLSX files for traditional similarity-based fuzzy hashing algorithms (left side) and DLAM with feed-forward (FF) and transformer (TF) models (right side). Whereas traditional similarity metrics fail for the fixed-sized fuzzy hashes generated by TLSH and ssdeep, our DLAM approaches significantly increase the performance and are even more accurate than state-of-the-art fuzzy hashing algorithms with variable hash sizes for JavaScript and XLSX files. We highlighted the best 3 accuracy values for each file type.

		Traditional Fuzzy Hashing				DLAM (Ours)			
		mrsh-cf	**MRSH-v2**	**ssdeep**	**TLSH**	**ssdeep (FF)**	**TLSH (FF)**	**ssdeep (TF)**	**TLSH (TF)**
JS	Accuracy (%)	81.54	67.38	50.02	50.02	**92.70**	82.08	**92.70**	**87.90**
	TPR (%)	0.63	0.83	0.00	0.00	0.86	0.75	0.86	0.84
	FPR (%)	0.00	0.05	0.00	0.00	0.01	0.11	0.01	0.08
	TNR (%)	1.00	0.95	1.00	1.00	0.99	0.89	0.99	0.92
	FNR (%)	0.37	0.17	1.00	1.00	0.14	0.25	0.14	0.16
PDF	Accuracy (%)	**97.3**	**95.44**	50.04	49.98	79.10	78.42	**94.34**	82.60
	TPR (%)	0.95	0.93	0.00	0.00	0.75	0.74	0.93	0.83
	FPR (%)	0.00	0.02	0.00	0.00	0.16	0.17	0.04	0.17
	TNR (%)	1.00	0.98	1.00	1.00	0.84	0.83	0.96	0.83
	FNR (%)	0.05	0.07	1.00	1.00	0.25	0.26	0.07	0.17
XLSX	Accuracy (%)	**96.78**	88.22	52.54	51.17	93.80	90.28	**97.36**	90.74
	TPR (%)	0.94	0.77	0.05	0.03	0.89	0.89	0.96	0.92
	FPR (%)	0.00	0.00	0.00	0.04	0.01	0.08	0.01	0.11
	TNR (%)	1.00	1.00	1.00	0.96	0.99	0.92	0.99	0.89
	FNR (%)	0.06	0.23	0.95	0.97	0.11	0.11	0.04	0.08

fragment detection. As can be seen in Table 1, our DLAM approach with `ssdeep` consistently outperforms the neural networks with `TLSH` hashes. Using `TLSH`, the model produces more false-negatives than with `ssdeep`, as can be seen in Fig. 4b and a. The results in Table 1 also show that a combination of transformers and `ssdeep` consistently outperforms all approaches that are based on simple feed-forward networks. So choosing a more complex architecture with a much higher number of parameters pays off in this case.

However, even if feed-forward networks perform worse than transformer models in most cases, they still achieve a decent performance. Consequently, for applications with limited computational power available or to speed up inference time, feed-forward networks might still be a reasonable architecture to choose.

Q4: DLAM increases the robustness when working with fuzzy hashes. One way to degrade the performance of traditional fuzzy hashes is to concatenate the same file multiple times before feeding them into the algorithms. We show that our DLAM approach strongly increases the detection robustness of fuzzy hashes. We concatenated the same file a pre-defined number of times and then calculated the hashes for the composed file. Table 2 states our results and illustrates that DLAM can indeed compensate for the known low resilience against repetitive content.

As also shown in previous research (Göbel et al., 2022a,b), the similarity scores of `ssdeep` and `TLSH` could be lowered to 0 when comparing a single instance of a file with another file that consisted of multiple concatenations of the first one. In the case of `TLSH`, a 16-fold concatenation of the same 5000 bytes was sufficient to lower the similarity score to 0, leading to a prediction accuracy of 50%, which comes down to random guessing in a binary prediction problem. However, our results further show that `TLSH`, in combination with a transformer network, is still able to detect files with injected fragments and benign files with higher accuracy than `mrsh-cf` and `mrsh-v2` in face of a repetition. These results also indicate that there is much more information present in `TLSH` hashes than can be processed by simple similarity measurements.

4. Discussion and limitations

With this work, we demonstrated the large potential that lies in the combination of traditional fuzzy hashing algorithms and recent advances in deep learning. For the task of supervised fragment detection, our DLAM approach achieves accuracy values close to those of multi-resolution fuzzy hashes. Furthermore, the fuzzy hashes `ssdeep` and `TLSH` used in combination with DLAM are far more scalable due to more effective compression procedures. For example, an unbounded `mrsh-v2` hash compresses 68x worse than a `ssdeep` hash bounded to 148 bytes in length. With our DLAM, a comparatively small hash length can then still be classified with high accuracy, comparable to conventional approximate matching

with `mrsh-cf` and `mrsh-v2`. For the supervised fragment detection in JavaScript, DLAM achieves even higher accuracy values than all state-of-the-art `mrsh-cf` and `mrsh-v2`. Furthermore, DLAM also increases the resilience against file repetition, where `ssdeep` and `TLSH` alone fail to provide meaningful similarity scores.

Based on our empirical results, we recommend combining DLAM with `ssdeep` over `TLSH`, since `ssdeep`'s weaknesses can be narrowed down much more precisely. Similarities smaller than 13% are no longer safely detectable and should therefore be avoided when classifying with DLAM and `ssdeep`. `TLSH`, on the other hand, is generally less precise, and possible false predictions are much more difficult to predict. A way to overcome the limitations of `ssdeep` with regard to smaller similarities could lie in the work of Shiel and O'Shaughnessy (2019) on improving file-level fuzzy hashes for malware variant classification. Section-level hashing means that digital artifacts are not hashed and compared in their entirety, but only in sections. By compartmentalization — dividing the digital artifacts into equally sized blocks and hashing each one — a much more granular search for possible similarities can be performed. So one input file is no longer represented by a single hash but by multiple ones, each representing the exact same amounts of bytes.

We limited our analyses to algorithms that produce hashes with a fixed-sized (`TLSH`) or a maximum length (`ssdeep`). Other algorithms, such as `mrsh-v2`, produce large hashes with potentially infinite lengths. Since multi-resolution hashes can theoretically have any length, transformers that can process inputs of any length are applicable in theory. Peiser et al. (2020) were able to process, `sdhash` which produces a limitless hash with a bag of words approach. However, this has the disadvantage that all positional information on where sequences are located in a hash is lost. Which is why it was not adapted for `mrsh-v2` in this work.

4.1. Practical application

The main practical applicability of this work is illustrated here once again with an example. DLAM can be understood as a building block to gain significantly more visibility in the context of incident response or endpoint detection and response. As Lazo (2021) describes, files can be hashed on endpoints and these hashes are matched by the EDR solution with known hashes, i.e., malware IOCs. DLAM now enables the targeted training of machine learning networks to detect specific malware signifiers in fuzzy hashes (i.e., ransomware, adware, etc.). Until now, this required either a large number of cryptographic hashes, which must cover all polymorphic forms of a malware, or fuzzy hashes, which can only express a shared content with known malware files by the similarity score. However, this is rather unreliable as the shared content might just be coincidental noise. With DLAM, machine learning models can be trained to classify files based only on specific malware sequences in

Table 2
Prediction accuracy for fragment detection with traditional fuzzy hashing algorithms with variable (`mrsh-cf` and `mrsh-v2`) and fixed (`ssdeep` and `TLSH`) hash sizes, and our DLAM approach combining `ssdeep` and `TLSH` with a transformer network. Per row, 5000 files with a file size of 5000 bytes were created. Half of them contained between 1% and 99% of fragment bytes within them. All files were concatenated according to the respective multiplication factor and then had to be classified. The results are averaged over 10 test runs.

Multiplication factor	Traditional Fuzzy Hashing				DLAM (Ours)	
	mrsh-cf	**MRSH-v2**	**ssdeep**	**TLSH**	**ssdeep (TF)**	**TLSH (TF)**
x 1	97.54	76.13	50.0	50.0	91.09	94.94
x 2	92.19	74.37	50.0	50.0	81.68	93.57
x 4	91.99	72.67	50.0	50.0	63.56	93.25
x 8	91.69	76.47	50.0	50.0	53.55	93.07
x 16	91.49	73.73	50.0	50.0	53.55	93.95
x 32	88.21	71.11	50.0	50.0	50.0	93.59

the fuzzy hashes. The models are trained to distinguish between benign files and malware. Our results show that this is already possible with the common fuzzy hashes `ssdeep` and `TLSH`. However, this is all the more impressive as these two hashing algorithms produce very short hash strings on which DLAM can be trained. Applied onto `ssdeep` and `TLSH`, DLAM can achieve classification accuracies that match and even surpass those of fuzzy hashing algorithms that are unlimited in length (like `mrsh-cf` and `mrsh-v2`). As Lazo (2021) has shown, an approach like DLAM can be used to discover a previously unknown strand of malware altogether.

5. Conclusion and future work

The goal of our work is to open up the research on machine learning and fuzzy hashes to the broader scientific community. With deep learning approximate matching, or DLAM in short, it is now possible to contextualize a promising fusion of the two research areas of deep learning and digital forensics based on fuzzy hashing. Our application of transformer models in this context leads to higher classification accuracy than other current approaches. More precisely, we empirically demonstrated that applying DLAM to efficient standard fuzzy hashes, such as `ssdeep` and `TLSH`, creates a powerful application that rivals the best-performing fuzzy hashes `mrsh-v2` and `mrsh-cf` in terms of supervised file fragment detection. By adjusting the size of the inserted fragment and the file type, we were able to show the impact of these two factors, both of which are important when using DLAM in practical applications for data loss prevention or malware detection. The full potential that DLAM offers for digital forensics and cybersecurity becomes even more apparent, recalling that, `ssdeep` and `TLSH` are very compact and many times smaller than multi-resolution hashes. This paper proves that machine learning is a powerful enabler that enhances the practical applications of fuzzy hashes well beyond the approximate matching.

As was explained previously, we could not apply DLAM (Transformer) on fuzzy hashes with potentially unlimited length due to hardware constraints. This does not mean that future research should shy away from exploring this area, we leave the realization open for future research.

Another interesting avenue is to extend DLAM beyond the binary classification of anomalies, which we explored in this work. We expect DLAM to also improve results for unsupervised fragment and outlier detection, content retrieval, and multi-class file classification based on fuzzy hashes, but leave the empirical proof for future work.

Acknowledgements

References

Apple Inc, 2021. CSAM detection - technical summary. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf. (Accessed 13 October 2022).

Azarafrooz, A., Brock, J., 2018. Fuzzy Hashing as Perturbation-Consistent Adversarial Kernel Embedding. CoRR abs/1812.07071.

Barik, T., Lubick, K., Smith, J., Slankas, J., Murphy-Hill, E.R., 2015. Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In: Working Conference on Mining Software Repositories. IEEE Computer Society, pp. 486–489.

Charyyev, B., Gunes, M.H., 2020. Iot traffic flow identification using locality sensitive hashes. In: International Conference on Communications (ICC). IEEE, pp. 1–6.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., Mordatch, I., 2021. Decision transformer: reinforcement learning via sequence modeling. In: Advances in Neural Information Processing Systems. NeurIPS), pp. 15084–15097.

Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y., 2014. On the properties of neural machine translation: encoder-decoder approaches. In: Proceedings of SSST@EMNLP 2014, Workshop on Syntax, Semantics and Structure in Statistical Translation, pp. 103–111.

Davies, S.R., Macfarlane, R., Buchanan, W.J., 2022. Napierone: a modern mixed file data set alternative to govdocs1. Forensic Sci. Int.: Digit. Invest. 40, 301330.

Devlin, J., Chang, M., Lee, K., Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT, pp. 4171–4186.

Diaz, V., 2020. Why is similarity so relevant when investigating attacks. https://blog.virustotal.com/2020/11/why-is-similarity-so-relevant-when.html, 2022-07-20.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N., 2021a. An image is worth 16x16 words: transformers for image recognition at scale. URL:. In: International Conference on Learning Representations https://openreview.net/forum?id=YicbFdNTTy.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N., 2021b. An image is worth 16x16 words: transformers for image recognition at scale. In: International Conference on Learning Representations (ICLR).

Garfinkel Jr., S.L, F, P.F., Roussev, V., Dinolt, G.W., 2009. Bringing science to digital forensics with standardized forensic corpora. Forensic Sci. Int.: Digit. Invest. 6, 2–11.

Göbel, T., Uhlig, F., Baier, H., 2021. Evaluation of network traffic analysis using approximate matching algorithms. In: Advances in Digital Forensics XVII. Springer, pp. 89–108.

Göbel, T., Uhlig, F., Baier, H., 2022a. Find my iot device – an efficient and effective approximate matching algorithm to identify iot traffic flows. In: Gladyshev, P., Goel, S., James, J., Markowsky, G., Johnson, D. (Eds.), Digital Forensics and Cyber Crime. Springer International Publishing, pp. 72–92.

Göbel, T., Uhlig, F., Baier, H., Breitinger, F., 2022b. FRASHER - a framework for automated evaluation of similarity hashing. Forensic Sci. Int.: Digit. Invest. 42, 301407.

Gulati, A., Qin, J., Chiu, C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., Pang, R., 2020. Conformer: convolution-augmented transformer for speech recognition. In: Annual Conference of the International Speech Communication Association, pp. 5036–5040.

Healy, J., Chambers, D., 2011. Fast and accurate genome anchoring using fuzzy hash maps. In: 5th International Conference on Practical Applications of Computational Biology & Bioinformatics. Springer, pp. 149–156.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 1735–1780.

Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. ICML), pp. 448–456.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., Liu, Q., 2020. Tinybert: distilling BERT for natural language understanding. In: Findings of the Association for Computational Linguistics. EMNLP, pp. 4163–4174.

Kingma, D.P., Ba, J., 2015. Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR).

Kornblum, J.D., 2006. Identifying almost identical files using context triggered piecewise hashing. Forensic Sci. Int.: Digit. Invest. 3, 91–97.

Lazo, E.G., 2021. Combing through the fuzz: using fuzzy hashing and deep learning to counter malware detection evasion techniques. https://www.microsoft.com/security/blog/2021/07/27/combing-through-the-fuzz-using-fuzzy-hashing-and-deep-learning-to-counter-ma lware-detection-evasion-techniques/, 2022-07-20.

Liu, H., Wang, R., Shan, S., Chen, X., 2016. Deep supervised hashing for fast image retrieval. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2064–2072. https://doi.org/10.1109/CVPR.2016.227.

Martínez, V.G., Hernández-Álvarez, F., Encinas, L.H., et al., 2020. An improved bytewise approximate matching algorithm suitable for files of dissimilar sizes. Mathematics 8, 1–37.

Oliver, J., Cheng, C., Chen, Y., 2013. Tlsh - a locality sensitive hash. In: 4th Cybercrime and Trustworthy Computing Workshop.

Ong, T.S., Teoh, A., Khoh, W.H., 2014. Improved Fuzzy Hashing Technique for Biometric Template Protection.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 8024–8035.

Pearson, P.K., 1990. Fast hashing of variable-length text strings. Commun. ACM 33, 677–680.

Peiser, S.C., Friborg, L., Scandariato, R., 2020. Javascript malware detection using locality sensitive hashing. In: ICT Systems and Privacy Protection, pp. 143–154.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole EXE. In: The Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence. AAAI Press, pp. 268–276.

Raff, E., Nicholas, C., 2018. Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. URL: Digit. Invest. 24, 34–49. https://doi.org/10.1016/j.diin.2017.12.004 https://www.sciencedirect.com/science/article/pii/S1742287617302566.

Raychev, V., Bielik, P., Vechev, M.T., Krause, A., 2016. Learning programs from noisy data. In: Symposium on Principles of Programming Languages (POPL), ACM, pp. 761–774.

Shiel, I., O'Shaughnessy, S., 2019. Improving file-level fuzzy hashes for malware variant classification. Forensic Sci. Int.: Digit. Invest. 28 (Suppl. ment), S88–S94.

Singh, M., 2021. Essential Characteristics of Approximate Matching Algorithms: A Survey of Practitioners Opinions and Requirement Regarding Approximate Matching, 10087. CoRR abs/2102.

Struppek, L., Hintersdorf, D., Neider, D., Kersting, K., 2022. Learning to break deep perceptual hashing: the use case neuralhash. URL:. In: 2022 ACM Conference on Fairness, Accountability, and Transparency. Association for Computing Machinery, New York, NY, USA, pp. 58–69. https://doi.org/10.1145/3531146.3533073.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems. NeurIPS), pp. 5998–6008.

Zhao, F., Huang, Y., Wang, L., Tan, T., 2015. Deep semantic ranking based hashing for multi-label image retrieval. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).