MemBERT: Foundation model for memory forensics

Christofer Fellicious
University of Passau
Germany
christofer.fellicious@uni-passau.de

Mehdi Ben Amor
University of Passau
Germany
mehdi.benamor@uni-passau.de

Johannes Garstenauer
Friedrich-Alexander-Universität
Germany
johannes.garstenauer@fau.de

Jelena Mitrović
University of Passau
Germany
Institute for AI R&D of Serbia
Serbia
jelena.mitrovic@uni-passau.de

Hans P. Reiser
Reykjavik University
Iceland
hanr@ru.is

Michael Granitzer
University of Passau
Germany
michael.granitzer@uni-passau.de

Abstract

Foundation models have demonstrated significant advancements in natural language processing and computer vision, yet their potential in cybersecurity is unexplored. Current memory forensics tools and machine learning models often need more versatility and adaptability, presenting a crucial research gap. To address this, we introduce **MemBERT**, a foundation model designed explicitly for memory forensics. MemBERT is trained on extensive process dump data, with and without metadata inclusion, to capture intricate patterns present in main memory. Its potential impact on cybersecurity practices could be significantly similar to the effects of foundation models in natural language processing. We aim to streamline memory forensics by reducing the manual effort and coding traditionally required by cybersecurity practitioners. Through comprehensive experimentation, we demonstrate MemBERT's efficiency in a downstream task of extracting OpenSSH encryption keys and other memory structures from raw process dumps. The results reveal that the robust embeddings generated significantly help identify structures within memory. Additionally, we demonstrate that our model's embeddings can be compressed with minimal loss of accuracy, further highlighting its efficiency. Our findings with MemBERT go beyond just its performance in a specific task. The findings also indicate MemBERT substantially advances memory forensics, providing a versatile and powerful tool for cybersecurity professionals. This research addresses the limitations of the current forensics process model and sets the stage for the broader application of foundation models in the cybersecurity domain. Our results, code and models are available at HuggingFace and https://github.com/padas-lab-de/memBERT.

SAC'25, March 31 –April 4, 2025, Sicily, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3708009

CCS Concepts

• **Applied computing** → **Evidence collection, storage and analysis**; • **Computing methodologies** → **Knowledge representation and reasoning**; *Unsupervised learning*; Neural networks.

Keywords

Foundation model, memory forensics, heap dumps

ACM Reference Format:
Christofer Fellicious, Mehdi Ben Amor, Johannes Garstenauer, Jelena Mitrović, Hans P. Reiser, and Michael Granitzer. 2025. MemBERT: Foundation model for memory forensics. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3672608.3708009

1 Introduction

Digital Forensics is defined as the "process of employing scientific principles and processes to analyze electronically stored information" [13]. Digital forensics plays a crucial role in modern cybersecurity practices. "The objective of forensic science is to determine how digital evidence can be used to recreate, identify suspects to analyze or diagnose the victim machines" [12]. A key aspect of modern digital forensics is examining a computer's volatile memory. The volatile memory of a computer, commonly known as Random Access Memory (RAM), contains a wealth of information. The memory contains information about the running processes, session keys, encryption keys, user information, and much more. We could inspect this memory and analyze its information using digital forensic techniques if we have the appropriate access rights. In the current landscape of memory forensics, no general-purpose machine-learning models could work on raw bytes in memory. Traditional digital forensic techniques involve a lot of manual interventions, rule-based approaches, and complex tool chains [12, 22].

Machine learning applications are slowly proliferating in the memory forensic domain, but they specialize in a single task. We must train such models for each task, requiring more effort and time. The better approach is to train a large general-purpose model known as a foundation model [2]. The idea of a foundation model is that having models that work across domains and problems is more manageable and cost-effective than training a model every time for each specialized task. Foundation models are usually pre-trained on unlabeled datasets in an unsupervised setting and then fine-tuned

for specific applications. Text-based foundation models or large language models(LLMs) such as Bidirectional Encoder Representations from Transformers (BERT), GPT, Llama, and Mixtral, to name a few, use the transformer architecture [6, 9, 19, 20].In short, foundation models revolutionized machine learning by being effective across multiple domains. These models can generate coherent text and images, translate text from one language to another, summarize content, generate software code or algorithms, and even detect hate speech [4, 5, 21]. These capabilities across different domains can also be seen in many popular models such as DALL-E, GPT, SORA[1] [1, 3]. A foundation model for a main memory-specific application would benefit many practical applications, such as malware detection, intrusion detection, and even virtual machine introspection (VMI), highlighting the potential impact of our study. However, foundation models currently available for other domains, such as language or images, are unavailable for digital forensics. In our work, we aim to close the existing gap by introducing the first foundation model trained on memory dumps, which is a novel and intriguing development in digital forensics.

Our main contributions are

- the first foundation models for main memory data with and without metadata information.
- a method of extracting embeddings from the trained models and compressing the embeddings for downstream tasks.
- a use case of the foundation model on extracting SSH encryption keys from process heap dumps.

2 Related Work

Bommasani et al. coined the term foundation model [2]. Foundation models are powerful architectures primarily trained on text data for natural language processing, images for computer vision-based applications, and graphs. The parameter weights of such pretrained foundation models are available, and we can fine-tune the architecture for specific tasks based on the individual application scenario. Most foundation models rely on the transformer architecture as the building block. The transformer architecture relies on the "attention" mechanism, where it models the local and global context dependencies of inputs [20]. Therefore, the attention mechanism gives some parts of the input sequence more importance than others. This attention mechanism solves the issue where earlier parts of the sequence in long sequences were not weighted less even though they were important. Devlin et al. introduced the Bidirectional Encoder Representations from Transformers(BERT) model [6]. The authors improved upon the work of Vaswani et al. by incorporating bidirectional attention [20]. The idea was that representations generated by looking only from left to right were unsatisfactory in many scenarios, and having a bidirectional language model helps better retain the context. Sahn et al. distilled the BERT model to create DistilBERT and reduced the size of the BERT model by approximately 40% [14]. DistilBERT has the same general architecture as BERT with half the number of layers. The authors say that "variations on the last dimension of the tensor have a smaller impact on computation efficiency for a fixed parameters budget than variations on other factors like layers." This is the primary reason for reducing the number of layers in the DistilBERT architecture. Even with the parameters reduced, DistilBERT retains 97% of the performance of BERT. Meta introduced the Large Language Model Meta AI (LLaMA), a 65 billion parameter open source model available at smaller sizes, even going down to 7 billion [19]. The authors claim that "LLaMA requires far less computing power and resources to test new approaches, validate other' work, and explore new cases." The authors trained the model on 1.4 trillion tokens of text across 20 different languages, focusing on those with Latin and Cyrillic alphabets. One of the most famous currently available models is the GPT model from OpenAI. Radford et al. introduced one of the first models known as GPT [11]. The authors attained significant performance gains by generative pre-training of the language model on a diverse corpus. A fine-tuning step follows this pre-training step, where the authors use task-aware input transformations.

We then look at the research for our downstream task, which is extracting SSH encryption keys. Sentanoe and Reiser developed a tool that extracts SSH Keys based on the decryption of SSH network traffic [16]. According to the authors, the algorithm detects establishing SSH connections and extracts information from the cryptographic algorithms. The algorithm also passively captures the network traffic into a PCAP file. A plug could decrypt this captured communication later. The authors also use apriori knowledge about user-lever dataset structures in this method. Sentanoe et al. also developed an entropy method to extract SSH Keys and TLS keys [15]. The method extracted keys of different lengths based on the OpenSSH protocol. The authors then transferred this knowledge to extract TLS keys, even though the algorithm was never explicitly trained on TLS keys. The dataset used by the authors is open source and publically available on Zenodo. Taubmann et al. developed a method to extract keys using knowledge of the structures within the heap combined with the pointers found within the heap itself [18]. Their method requires root permissions to disable SELinux. Other than these, the method does not require any additional modifications.

3 Method

We aim to train a foundation model (*MemBERT*) for memory forensics, which entails training a model on unsupervised data with raw main memory data. We then would fine-tune for the downstream task of extracting SSH encryption keys. Secure Shell (SSH) is a widely used protocol for server connections, and OpenSSH is notable software based on it. The SSH protocol connects to a remote server from a client machine. This popularity also means malicious actors use the SSH protocol to connect to remote machines. Retrieving communication data between servers and malicious actors is crucial for analysis and future prevention of intrusions [17, 23]. Moreover, identifying and monitoring changes in data structures within process heaps can provide valuable insights into thwarting future attacks. Our source code is open source[2] and all models are on HuggingFace[3].

[1] https://openai.com/research/video-generation-models-as-world-simulators

[2] https://github.com/padas-lab-de/memBERT/

[3] https://huggingface.co/johannes-garstenauer

3.1 Dataset

We require a large dataset of raw memory data with labels and good metadata information for the intended downstream task of extracting SSH encryption keys to train MemBERT. Since we train MemBERT on unlabeled data, a larger dataset will include more variety and help the model generalize better. We found a publicly available dataset[4] [7] that contains approximately 100k different OpenSSH heap dumps, where each heap is around 100KB in size. The authors split the dataset into different SSH scenarios: base, client-side, secure copy (SCP), and port forwarding. Multiple versions of SSH are available for each scenario; the heaps are organized based on each version's encryption key length. Each scenario has four different lengths: 16, 24, 32, and 64, and these lengths are based on the encryption algorithm used. This dataset is a perfect candidate for our task as it has over 15GB of raw process dumps.

3.2 Using no metadata information

We intend to train a model that can generate representations on a general level. Therefore, we create a fixed-size raw-byte sequence from the dataset. To do this, we create 128-byte slices with 64-bytes of overlap between them. We consider 128-byte slices because it is a good balance between containing more structural information in the slice. If we had a smaller size, it would mean that some structural information could be lost, and having a larger size would mean that multiple semantic structures might be present in the same slice. Therefore, 128-byte is a sweet spot between having unique local structures and maintaining enough semantic information from larger structures. For this, we start at the beginning of the raw heap data and increment the index by 64 bytes while extracting 128-byte chunks for training. If the extracted slice contains only zeros, we discard that data. We refer to these 128-byte blocks of raw data as chunks or slices and MemBERT trained on this data as the *slice-based model* because we train the model on chunks or slices of raw process heap data.

3.3 Using metadata about pointers and malloc headers

The second method we implement uses metadata information we could extract from the heap data. Taubmann et al. showed that there are semantic structures within the heap [18]. The authors show that extracting keys using similar semantic information and knowledge about the underlying structures is possible. To identify and generate the semantic structures, we make some generalized assumptions about the heap. Furthermore, we outline the assumptions in subsubsection 3.3.1. Since we use metadata information to identify structures and use those structures to train another version of MemBERT, we refer to the model trained in this way as the *structure-based model* or *metadata-based model*. We use the word struct and structure interchangeably here.

3.3.1 Assumptions. We make certain intentionally broad and versatile assumptions about the heaps for the extraction process. These assumptions facilitate an adaptable method that we could apply to various systems and contexts. Firstly, we assume that a heap dump of the target system can be acquired. We need to reconstruct the virtual address mappings of a specific process to obtain a heap dump from the raw physical memory. All the modern forensic memory tools do satisfy this fundamental requirement. The second point of assumption is that the heaps contain the cryptographic keys. Although there are ways for processes to store key material elsewhere, we rarely see such scenarios in practice. Thus, this requirement is only a minor limitation as well. We also assume that the process, OpenSSH in our case, stores the cryptographic keys within data structures that other data structures cross-reference. This cross-reference enables us to follow pointers to extract the cryptographic keys. Lastly, we require that a memory allocator be used to allocate data structures by the process to the heap. The `malloc` function, for instance, generates `malloc headers` containing required metadata like the allocation size of data structures. The size information lets us limit the search space for the end of the structure, thus we make the assumption that the size information is present in the malloc header. Importantly, we do not require specific assumptions about the hierarchy or layout of data structures, which makes our approach broadly applicable to various processes.

3.3.2 Extraction of structures and labeling. We consider a *structure* or *struct* as any data object allocated on the heap. It can be a string, object, structure, array, or integer. We do not differentiate between the data types and consider everything allocated as a structure. The first step is to identify the pointers among the raw data. For this, we know that pointers are 8-byte aligned and have a specific pattern. We identify pointers by checking if the addresses, they point to, fall within the heap address range. We do not require additional mechanisms for address translation [5], given that the mapping of the virtual addresses to physical addresses was resolved by Fellicious et al. in the dataset [7]. Next, we look for a valid malloc header which should be present right before the allocated block and should contain the block size. By having the starting address (obtained from the pointer) and size (obtained from the malloc header), extracting the structure should be straightforward. These identified structures of raw memory are then collated into a dataset and used to train the structure-based MemBERT.

3.4 Pretraining

Our methods of using structures or slices differ only in how we process the data for training MemBERT. All the other steps in training are identical, including the hyperparameters for tuning the model. Our structure-based MemBERT is trained on raw data from semantic structures, and slice-based MemBERT is trained on chunked raw data with overlap. The selection of a model is significant because it requires enough expressive power to learn from the different patterns present in the data. We want to use the smallest possible model that aligns with our dataset. The smallest possible model is more accessible to train without requiring high-end computing resources. The Distilbert model thus satisfies our requirements and has a proven track record of learning different patterns [14].

[4] https://zenodo.org/records/6537904

[5] https://tc.gtisc.gatech.edu/cs6265/2022-summer/refs/amd64-vol2-sys.pdf

We train a tokenizer from scratch on the memory data. A tokenizer converts the data, raw memory in our case, to a machine-readable form. We use the Byte-Pair-Encoding(BPE) tokenizer to tokenize the raw data sequences in our dataset [10]. The advantage of the BPE tokenizer is that it looks at the byte level, which is beneficial in our scenario as we have raw byte data in our dataset. The BPE-based tokenization also allows for merging tokens that frequently appear together. Our dataset has multiple zeros within the structures. These could provide valuable information to the model when clustered together. We expect the tokenizer to generalize to such patterns within the data and merge the frequently occurring tokens. We then pad or truncate the input sequence to a fixed length. For padding, we can use a special token. The padding tokens do not carry any inherent meaning during training. For both slice-based MemBERT and struct-based MemBERT, we tokenize the data at 2 bytes each. This means that we consider four characters a single token, given a single byte (i.e.,00-FF) is two characters. We could also do the tokenization at the byte level, but then it would be challenging to capture the two-byte dependencies, especially for pointers and malloc headers. Going up to four bytes could be beneficial, but the vocabulary size would be too large as there will be 2^{32} possible combinations. Two bytes offer the best balance between a moderately large vocabulary of approximately 30k tokens and the ability to capture the sub-word dependencies.

We pretrain our model in an unsupervised way using masked language modeling objective, by masking tokens randomly and having the model guess the masked tokens. Even though the distribution of tokens is seemingly random at a byte level, there should be meaning within the adjoining tokens at a semantic level. For example, pointers follow a specific pattern, and an allocated structure keeps identical semantics whenever the system allocates the structure, even though the pointers and other run-time arguments could change every time. We hope the model can learn this type of high-level semantics through the masked training approach. Using the Distilbert model, we limit the input sequence length to 512, given that most of the structures in our dataset do not exceed the 512 sequence length. We compare the models based on the perplexity metric during the pretraining stage [8].

3.5 Downstream task and Embedding compression

3.5.1 Encryption Key Extraction. After training MemBERT, we test the trained models' versatility on a downstream task. We extract slices and structures the same way we did for the pre-training dataset, and label them with two and five classes, respectively. We have two classes for the slice-based dataset: non-relevant and relevant. For the structure-based dataset, we created five different classes, and they are

- Class 0: Structures that are not interesting to the current setting
- Class 1: SESSION STATE STRUCTURE
- Class 2: NEW KEYS STRUCTURE
- Class 3: Encryption algorithm name
- Class 4: Encryption key itself

The accompanying metadata consists of the addresses for the five structures that interest us in this context.

The next step is to extract the embeddings for the training and testing datasets for both the slice-based MemBERT and struct-based MemBERT. For this, we consider the sequence embedding for a slice or a structure by taking the output encoding of **[CLS]** token. We train a fully connected neural network classifier C, with a hidden layer of size h, on the output embeddings of MemBERT for three epochs. After completing the training, we test the network on the generated embeddings of the test set.

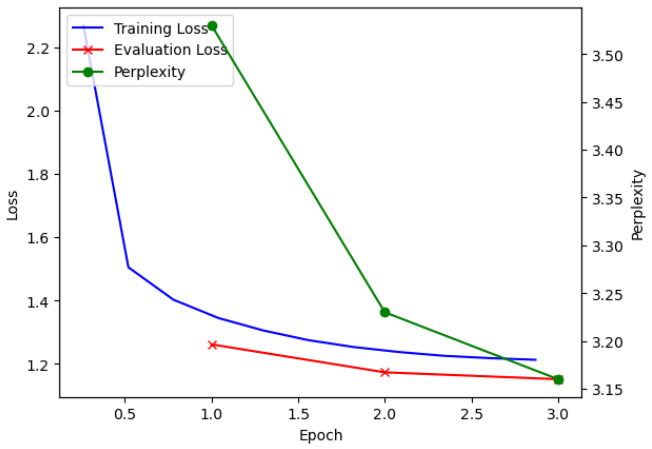
3.5.2 Latent Space compression. We vary compression ratios by modifying $h \in \{512, 256, 128, 64, 32, 16, 8, 4, 2, 1\}$. We do this to investigate how much we can compress the embeddings without significant performance loss.

4 Results

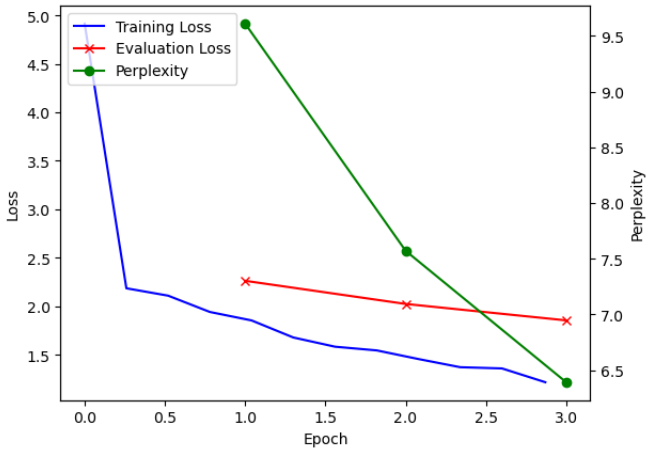
For the pretraining without considering any metadata information, we obtain a perplexity of **2.45** Figure 1a. Furthermore, for the structure-based model, the perplexity is **6.39** Figure 1b. Perplexity is an uncertainty measure that indicates how "perplexed" the model is about the next token. A lower perplexity would indicate that the model understands the structure and rules in the given dataset. However, one has to be aware that the perplexity could often be a weak proxy for the quality of a model and that we need to test the model on the corresponding downstream tasks to ascertain the quality of the trained model. We see that the loss for both models flattens out around three epochs Figure 1. We test our model's performance on SSH encryption key extraction downstream tasks. We extract the raw representations of the [CLS] token to run our downstream tasks. These vectors are extracted from the last layer of MemBERT and have a length of 768. The implemented fully connected network C then compresses this representation to lengths of 512, 256, 128, 64, 32, 16, 8, 4, 2, and 1. For the slice-based downstream task, the idea is to predict the presence of an encryption key within a slice. We explain the results of this downstream task in subsection 4.1. We can extract more semantic information using the metadata, pointers, and malloc headers. Therefore, the struct-based downstream task involves identifying the encryption key, the encryption algorithm, the NEWKEYS structure, and the SSH session state structure. The results of this downstream task are in subsection 4.2.

4.1 Results without any metadata

There are only two classes for the dataset without metadata information: relevant(an SSH encryption key is present) and non-relevant (Absence of SSH encryption key). Since we do not consider the metadata information in this case, we train our foundation model on slices or chunks of an arbitrary size (we chose 128 bytes with an overlap of 64 bytes). We have 4267456 negative (Class 0/non-relevant) instances and 104977 positive (Class 1/relevant) instances. A partial key in a chunk does not count as a relevant class. Having a partial key in a slice could cause problems because a 32-byte partial key from a 64-byte key would look the same (in randomness) as an encryption key of size 32. However, we still do not label these slices with partial keys as relevant. From Table 1, the precision and recall are above 90% for almost all compressed embedding sizes. We see that from Figure 2 that the fine-tuned embeddings form a linear strip with the positive class (Class 1, also the relevant class), having



(a) Training, evaluation loss, and perplexity curves for slice-based MemBERT.



(b) Training, evaluation loss and perplexity curves for structure-based MemBERT.

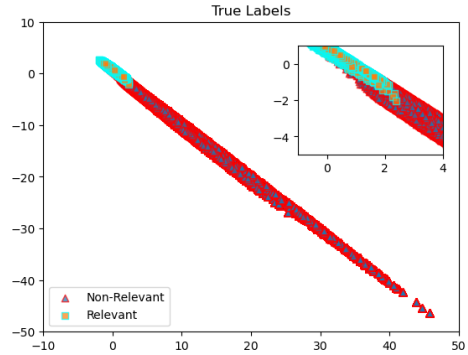
Figure 1: Loss curves and perplexity for the foundation model on both data with and without metadata information for three epochs. We see that the perplexity for the generalized foundation model is comparatively lower that the model trained on the structures.

embeddings that are closer to zero in both dimensions. The non-relevant class (Class 0) is more widely spread, which is expected behavior. This spread of the non-relevant class is due to the wide variety of data content in the class itself. Also, the SSH encryption key could be present anywhere within the 128-byte slice, along with other data. Still, the classifier could identify the slices that contained an encryption key, further proving the robustness of the embeddings themselves.

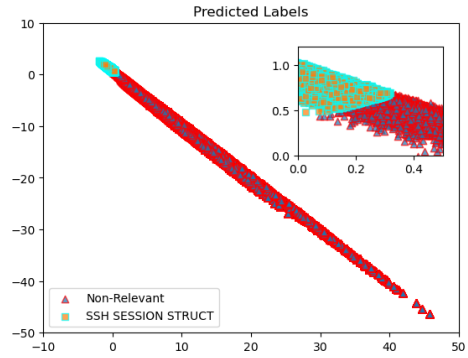
Next, we look at the misclassification examples and we include the false positives and false negatives for each class and not just the positive class. The misclassifications are shown in Figure 3 and the erroneous classifications occur at the decision boundary. We expect some non-relevant, Class 0, slices to have similar content to

Table 1: Metrics for different embedding sizes without metadata. Values in bold show the best performing embedding sizes for each metric with the standard deviation in parenthesis.

Size	Accuracy	Precision	Recall	F1-Score
512	99.09 (0.09)	91.60 (1.35)	89.09 (3.77)	90.17 (1.64)
256	99.14 (0.07)	91.77 (1.46)	90.06 (3.55)	90.77 (1.25)
128	99.10 (0.09)	91.86 (1.67)	89.34 (4.18)	90.38 (1.72)
64	99.17 (0.03)	91.50 (0.46)	92.16 (1.54)	**91.50** (0.46)
32	99.16 (0.05)	91.16 (1.61)	91.53 (3.05)	91.23 (0.90)
16	99.14 (0.07)	**92.10** (1.46)	89.55 (3.09)	90.68 (1.15)
8	99.13 (0.08)	91.50 (1.65)	91.24 (4.03)	90.89 (1.27)
4	**99.21** (0.01)	91.35 (0.68)	**92.52** (1.09)	91.91 (0.23)
2	99.15 (0.07)	91.65 (1.76)	90.74 (3.67)	91.04 (1.28)
1	99.16 (0.04)	90.75 (1.57)	92.24 (1.96)	91.42 (0.39)

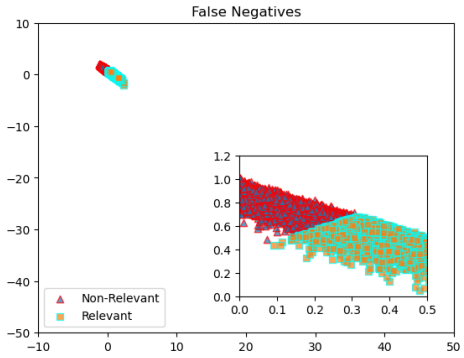


(a) True labels of compressed embedding of size 2.

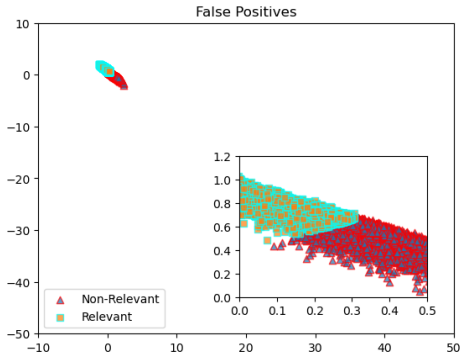


(b) Predicted labels of compressed embedding of size 2.

Figure 2: Compressed embeddings of size 2. We use the test set to generate the embeddings. We see that the true labels in the test set, slices of data that contain full encryption keys, are located towards the top left, implying they have values close to zero for both embedding dimensions. The scale on the plots is identical across all plots for results on slices (without metadata).



(a) False negatives for the relevant and non-relevant classes on predictions of size 2.



(b) False positives for the relevant and non-relevant classes on predictions of size 2.

Figure 3: Plot of false positives and false negatives on embeddings of size 2. We use the test set to generate the embeddings. The false negatives of the relevant class (Class 1) are the false positives of the non-relevant class (Class 0). We use the same scale as Figure 2 for better comparison. The inset images are zoomed-in versions of the plots.

that of the relevant class, or the slices could contain partial keys that cause these false positives.

Our method performs better when comparing our downstream task results with that of a specialized task such as the SSH Key Extraction by Fellicious et al. [7]. The authors report the best F1-Score of *88.55*, while with our method, our F1-Score across all embedding sizes has a minimum of **88.81** and a maximum of **92.89**. Although this might seem like an increase of only 4% points, we need to remember that the dataset is heavily unbalanced and that our current method is better at detecting the relevant slices of data.

4.2 Results with metadata

For the dataset labelled using the metadata information, we classify each identified into five categories: irrelevant, SSH Session struct, new keys, encryption algorithm name, and encryption key itself. We compute the metrics for the structure-based method using the macro average setting. We do not use the weighted metric because our test dataset is too unbalanced and favors the irrelevant class. We see an imbalance of almost 99:1 when comparing the irrelevant class to the relevant classes as shown in Table 2. A weighted metric would skew the metric in favor of the irrelevant class. Therefore, we chose the macro averaged precision to convey our results, as this metric treats all classes equally. And more importantly, the classes that are relevant to us (keys, encryption algorithms) occur only once or twice per heap. This means we need to favour the relevant classes and the macro-averaged metrics help us to do exactly that. From Table 3 except compressing the data to a single dimension, almost all the compression sizes preserve the metrics. We see that all embedding sizes have very good recall with the embedding size of two having the best precision and F1-Score. The metrics gives us an aggregated scalar value which is sometimes quite reductive. Visualizing the data helps us see the decision boundaries where the false positives or false negatives are located, thus giving us a better overall view of the data. For this, we choose the embedding size of two, enabling us to plot the results directly. Choosing a higher embedding size requires compressing the dimensions to display it. We could use the TSNE plot or Principal Component Analysis for dimensionality reduction and plotting. We did not choose either, as compressing a higher-dimensional space to a much smaller dimension could introduce artifacts and might not accurately represent the results visually. We see the true and predicted labels in Figure 4 . The true labels, especially of the non-relevant class, are spread wide, and we expected this as the non-relevant class encompasses many different structures. There is a high probability that some of these structures are similar to those relevant to us. We can see this misclassification for the encryption key class where there are a lot of false positives Figure 4b. This misclassification is due to random byte sequences from non-relevant classes identified as the encryption key. We expected misclassifications for the encryption key as the encryption key is a random string of bytes. So, the model could confuse itself when seeing a random string. We see most of the false positives and false negatives at the decision boundary. We can see the true positives and the decision boundary when looking at Figure 4b. The cluster of embeddings representing the encryption keys is very close to the non-relevant class overall. This close clustering is the reason for the false positives and false negatives, as shown in Figure 5. The best-separated cluster is the encryption algorithm name cluster, which has a demarcated boundary. This encryption algorithm's names are comprised of text values and have only a few values, so the boundaries can be demarcated. When looking at the true positives in Figure 4b, we see the model's well-defined decision boundaries.

4.3 Comparison of embeddings from both foundation models

We compare embeddings generated by fine-tuning the foundation models trained with (structure-based MemBERT) and without metadata information (slice-based MemBERT). We choose only the true predictions for this, as they show us the decision boundaries of the respective classifiers. From Figure 2 and Figure 4, we see that the plot looks very different, although the color scheme of the classes is kept consistent across all plots. The encryption key class is colored in cyan, while the non-relevant class is red. In Figure 4b, each class has its own data cluster. We see that the clustering of the encryption

Table 2: Label counts for structure classifying task in the test set.

–	Non-relevant	Session struct	New keys	Encryption Algorithm	Encryption Key
Count	14305261	12523	25046	25046	51618

Table 3: Mean macro averaged metrics for different embedding sizes (Dim) on structures (using metadata to create structures). The standard deviation is given in parentheses and the values in bold show the best-performing embedding size.

Dim	Accuracy	Precision	Recall	F1-Score
512	99.78 (0.02)	88.68 (1.71)	99.91 (0.03)	93.39 (0.92)
256	99.76 (0.04)	87.90 (3.12)	99.92 (0.02)	92.90 (1.84)
128	99.77 (0.03)	88.93 (1.90)	**99.93** (0.01)	93.51 (1.08)
64	99.78 (0.02)	89.91 (1.43)	99.91 (0.05)	94.06 (0.83)
32	99.78 (0.03)	89.25 (3.08)	99.92 (0.03)	93.66 (1.81)
16	**99.79** (0.02)	90.49 (1.14)	**99.93** (0.01)	94.43 (0.65)
8	99.78 (0.03)	89.79 (2.11)	**99.93** (0.01)	94.01 (1.19)
4	**99.79** (0.01)	90.23 (0.80)	**99.93** (0.03)	94.27 (0.41)
2	99.78 (0.02)	**90.77** (1.42)	99.91 (0.02)	**94.48** (0.77)
1	99.64 (0.16)	81.30 (7.99)	99.50 (0.38)	88.36 (5.72)

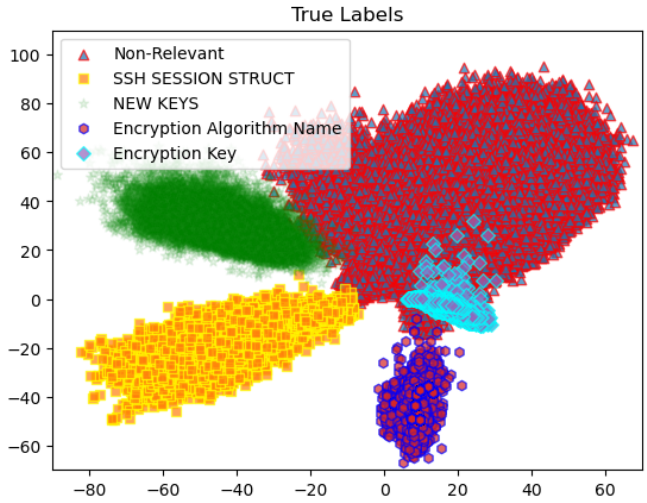
key class is very close to the non-relevant class. We expect this because the non-relevant class contains a lot of random data, and encryption keys are random data of arbitrary length. We can also see the same behavior in Figure 2b. The clustering of the encryption key is very close to that of the non-relevant class. Even though the key starts at any arbitrary position within the slice along with other non-relevant data in other positions, we still see that it is possible to extract encryption keys from the embeddings with high precision and recall. We could create larger networks or use different methods to optimize the performance of the downstream task, but that is not our primary goal. Our goal is to create MemBERT and validate its out-of-the-box performance on a downstream task, which, based on the results shown above, works very well.

5 Ethical Considerations

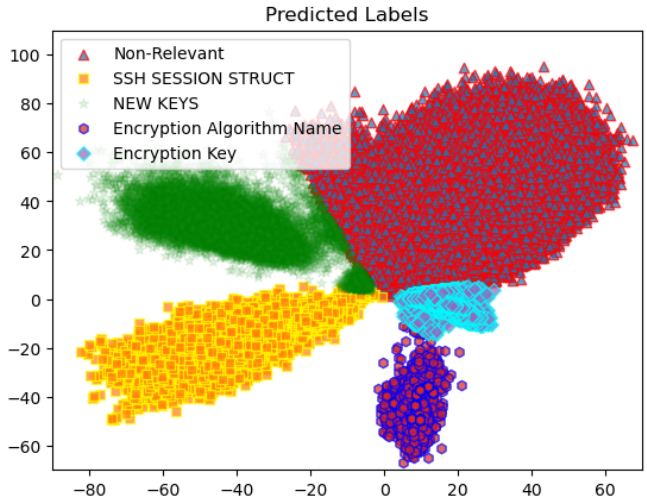
We consider the ethical side of our experiments. There are no privacy violations with the data, as the data creation process was completely synthetic. Our method does not open up a new attack vector, as MemBERT requires direct access to the memory and information about process heap boundaries. It is essential to emphasize that our method and its corresponding models are solely intended for research purposes and must not be utilized for malicious activities.

6 Conclusion

Our work shows that the foundation models trained on raw memory data with and without metadata can learn the underlying data distributions and create embeddings that generalize well to downstream tasks. The results on the downstream tasks for retrieving SSH encryption keys show the robustness of the embeddings created by the foundation models. The results also outperform the previous results on the same datasets. The foundation models created here are the first step in the creation of generalized tools for digital forensics. Our results also show that we could compress the data to two dimensions without losing performance on the downstream task.

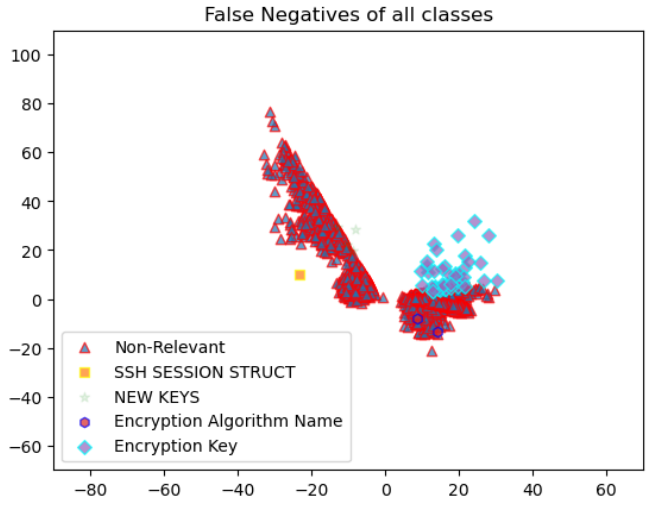


(a) True labels of compressed embedding of size 2.

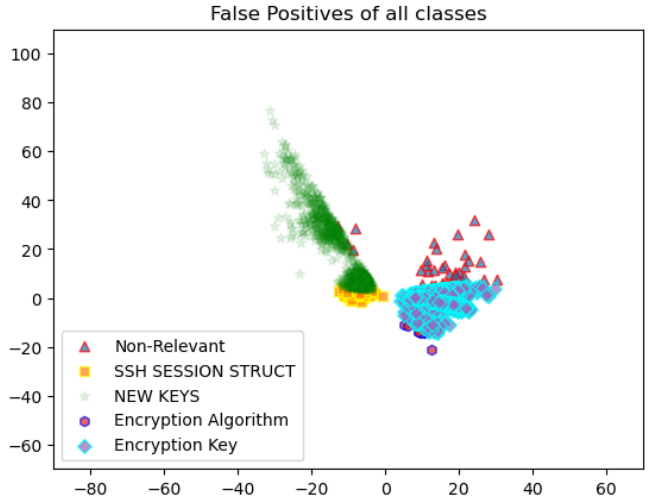


(b) Predicted labels of compressed embedding of size 2.

Figure 4: Compressed embeddings of size 2. The test set is used to generate the embeddings. The encryption keys are spread out more into the non-relevant class cluster while other classes have clearly defined regions. We can also see the decision boundaries among the classes in Figure 4b.



(a) False negatives on predictions of embedding of size 2. It is mostly non-relevant classes and encryption keys that are misclassified.



(b) False positives on predictions of embedding of size 2. Here, the non-relevant classes cross into the space of relevant classes.

Figure 5: Plot of false positives and false negatives on embeddings of size 2. The test dataset is used to generate the embeddings. Most false predictions occur due to the non-relevant class crossing the decision boundaries of the model. A few occur due to the encryption keys being detected as part of the non-relevant class. The scale on all the plots is the same for all structure-based (using metadata) plots.

7 Acknowledgement

The work is funded by the German Federal Ministry of Education and Research (BMBF) under the projects DeepWrite (Grant. No. 16DHBKI059) and CAROLL (01|S20049).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Tommaso Caselli, Valerio Basile, Jelena Mitrović, and Michael Granitzer. 2020. Hatebert: Retraining bert for abusive language detection in english. *arXiv preprint arXiv:2010.12472* (2020).

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Christofer Fellicious, Stewart Sentanoe, Michael Granitzer, and Hans P Reiser. 2022. SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump. *arXiv preprint arXiv:2209.05243* (2022).

[8] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America* 62, S1 (1977), S63–S63.

[9] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[10] Eugene Kharitonov, Marco Baroni, and Dieuwke Hupkes. 2021. How bpe affects memorization in transformers. *arXiv preprint arXiv:2110.02782* (2021).

[11] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[12] Mamoona Rafique and MNA Khan. 2013. Exploring static and live digital forensics: Methods, practices and tools. *International Journal of Scientific & Engineering Research* 4, 10 (2013), 1048–1056.

[13] Sriram Raghavan. 2013. Digital forensic research: current state of the art. *Csi Transactions on ICT* 1 (2013), 91–114.

[14] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[15] Stewart Sentanoe, Christofer Fellicious, Hans P Reiser, and Michael Granitzer. 2022. "The Need for Speed": Extracting Session Keys From the Main Memory Using Brute-force and Machine Learning. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 1028–1035.

[16] Stewart Sentanoe and Hans P Reiser. 2022. SSHkex: Leveraging virtual machine introspection for extracting SSH keys and decrypting SSH network traffic. *Forensic Science International: Digital Investigation* 40 (2022), 301337.

[17] Ren Rui Tan, Simon Eng, Kiam Cheng How, Yongqing Zhu, and Paul Wu Horng Jyh. 2023. Honeypot for Cybersecurity Threat Intelligence. In *IRC-SET 2022: Proceedings of the 8th IRC Conference on Science, Engineering and Technology, August 2022, Singapore*. Springer, 587–598.

[18] Benjamin Taubmann, Omar Alabduljaleel, and Hans P Reiser. 2018. DroidKex: Fast extraction of ephemeral TLS keys from the memory of Android apps. *Digital Investigation* 26 (2018), S67–S76.

[19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[21] Jeff Wu, Long Ouyang, Daniel M Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively summarizing books with human feedback. *arXiv preprint arXiv:2109.10862* (2021).

[22] Tina Wu, Frank Breitinger, and Stephen O'Shaughnessy. 2020. Digital forensic tools: Recent advances and enhancing the status quo. *Forensic Science International: Digital Investigation* 34 (2020), 300999.

[23] Zhenxin Zhan, Maochao Xu, and Shouhuai Xu. 2013. Characterizing honeypot-captured cyber attacks: Statistical framework and case study. *IEEE Transactions on Information Forensics and Security* 8, 11 (2013), 1775–1789.