An LLM-based Framework for Fingerprinting Internet-connected Devices

Armin Sarabi
University of Michigan
Ann Arbor, MI, USA
arsarabi@umich.edu

Tongxin Yin
University of Michigan
Ann Arbor, MI, USA
tyin@umich.edu

Mingyan Liu
University of Michigan
Ann Arbor, MI, USA
mingyan@umich.edu

ABSTRACT

In this paper we propose the use of large language models (LLMs) for characterizing, clustering, and fingerprinting raw text obtained from network measurements. To this end, We first train a transformer-based masked language model, namely RoBERTa, on a dataset containing hundreds of millions of banners obtained from Internet-wide scans. We further fine-tune this model using a contrastive loss function (driven by domain knowledge) to produce temporally stable numerical representations (embeddings) that can be used out-of-the-box for downstream learning tasks. Our embeddings are robust, resilient to small random changes in the content of a banner, and maintain proximity between embeddings of similar hardware/software products. We further cluster HTTP banners using a density-based approach (HDBSCAN), and examine the obtained clusters to generate text-based fingerprints for the purpose of labeling raw scan data. We compare our fingerprints to Recog, an existing database of manually curated fingerprints, and show that we can identify new IoT devices and server products that were not previously captured by Recog. Our proposed methodology poses an important direction for future research by utilizing state-of-the-art language models to automatically analyze, interpret, and label the large amounts of data generated by Internet scans.

CCS CONCEPTS

• **Networks** → **Network measurement**; • **Computing methodologies** → **Neural networks**; **Natural language processing**.

KEYWORDS

Internet scanning, device fingerprinting, deep learning, large language models

ACM Reference Format:
Armin Sarabi, Tongxin Yin, and Mingyan Liu. 2023. An LLM-based Framework for Fingerprinting Internet-connected Devices. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3618257.3624845

1 INTRODUCTION

The proliferation of Internet-connected devices has given rise to network scanning techniques for giving visibility into the public Internet. Projects and entities such as Censys [13] and Shodan [1] perform regular Internet-wide scans and record snapshots of visible devices on the Internet across many ports. These measurements are widely used for a variety of purposes, including to detect and fingerprint networked devices [5, 11, 17, 27], study trends [16, 19, 20], examine security events [4, 14], and enable various machine learning analysis [22, 26]. However, these Internet-wide scans essentially consist of raw information obtained from protocol handshakes (including banner grabs) with low label/feature coverage, e.g., to identify the underlying hardware/software products, or to facilitate automated analysis. This poses a challenge for researchers, network administrators, and security practitioners that utilize scan data, as one needs to develop their own data processing pipeline to filter for relevant information and make sense of the raw data.

At the same time, recent advances in deep learning have led to the development of large language models (LLMs) for complex text analysis. In particular, transformer-based models [29] such as BERT [12] and GPT [6] have been successfully applied to many natural language processing (NLP) tasks such as language modeling, translation, text classification, and clustering, achieving state-of-the-art performance.

Interestingly and crucially, the large amount of text data generated by Internet scans and the *text*-based nature of the scan data make them suitable for training large language models. Motivated by this observation, in this paper we train and evaluate an LLM on snapshots obtained from Internet scans, distilling raw text into general-purpose embeddings that are amenable to downstream machine learning tasks and analysis. Previous work [26] has used deep learning models to generate numerical embeddings to characterize Internet hosts, but the method relies on first converting the scan data into binary vectors which are then used for training. By contrast, to the best of our knowledge, our study presents the first model that is *directly* trained on raw text without an intermediate feature extraction step (e.g., the bag-of-words model utilized by [26]). It turns out that this allows the model to learn the underlying structure of scan data and support more interpretable analysis (e.g., by directly annotating text).

The output of our model is machine-readable embeddings that encode scan data into numerical vectors, and can be used to characterize the underlying hosts. Furthermore, we train our model to generate high-fidelity and robust embeddings that do not drift over time, by making them less sensitive to dynamic sections such as timestamps and randomized IDs (e.g., cookie IDs) that are regenerated each time a service is probed.

We also provide a preliminary examination of using HTTP banner embeddings generated by our model for clustering and fingerprint generation. These applications are aimed at complementing frameworks built on hand-curated fingerprints such as

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10.
https://doi.org/10.1145/3618257.3624845

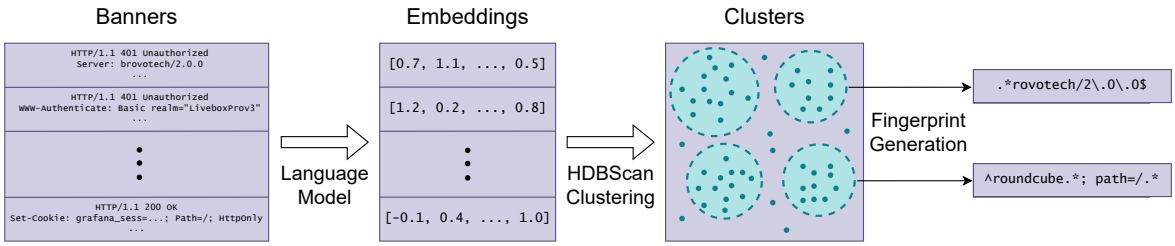

Figure 1: High-level diagram of our fingerprint generation pipeline.

Recog [25]. The final product of this approach is text-based fingerprints (i.e., regular expressions) that can be used to annotate important information in a banner, which in turn lead to identifying software/hardware products deployed on a host. Further examination of our fingerprints shows that we can identify new IoT devices and server products not captured by Recog, thereby increasing label coverage. Extending this study to protocols other than HTTP, and a thorough examination and sanitization of the generated fingerprints is an important part of our ongoing work. Our overall pipeline is illustrated in Figure 1, and our code is available at https://github.com/arsarabi/llm-device-fingerprints.

2 RELATED WORK

General-purpose numerical representations of Internet hosts: Sarabi and Liu [26] train a variational autoencoder (VAE) on a bag-of-words representation of Internet scan data, and use the resulting embeddings for learning tasks such as classification and inference. We on the other hand propose the use of transformer-generated embeddings, which skips the intermediate bag-of-words feature extraction to minimize information loss, as well as allows annotating the original text for better interpretability. Our model is also trained to generate temporally stable embeddings using a contrastive loss function detailed in subsection 4.2, making them better suited for clustering and fingerprint generation.

Product fingerprinting using active network probes: Recog [25] is an open source project that uses hand-curated fingerprints for identifying software/hardware products, services, and operating systems. Feng et al. [17] propose an acquisitional rule-based engine (ARE), an unsupervised approach that can generate rules for discovering IoT devices. Javed et al. [18] attempt to reproduce the implementation of the engine developed in [16], failing to achieve the accuracy reported in the original paper. To address the challenge of reproducibility, we use standard open source libraries for our implementation, allowing researchers to utilize our embeddings and generated fingerprints, as well as reuse models and fine-tune them on downstream tasks.

IoT device identification using passive traffic monitoring: Wang et al. [30] propose IoT-Portrait, a transformer-based model with incremental learning for automatically identifying IoT devices. Meidan et al. [23] propose ProfilIoT, a machine learning approach that leverages decision tree-based models. Aksoy and Gunes [2] develope an automated IoT device identification system using a genetic algorithm, while Aneja et al. [3] explore IoT device fingerprinting by analyzing packet inter-arrival times using deep learning. Msadek et al. [24] present a machine learning-based approach for fingerprinting IoT devices using encrypted traffic analysis. Chowdhury and Abas [9] conduct a survey on device fingerprinting approaches for resource-constraint IoT devices.

3 DATA AND PREPROCESSING

3.1 Raw data

We use 6 different snapshots from the Censys Universal Internet Dataset [8], corresponding to the first Tuesday in each month between July and December 2023. At the time of this writing, Censys contains scans of the entire IPv4 space across 107 protocols and over 3500 ports. From each snapshot, we collect non-empty service banners recorded by Censys, ignoring services that are flagged as truncated.[1] To reduce the size of our dataset, we subsample 10% of all IP addresses from each snapshot, ensuring the same selection of IPs across all snapshots. This results in a total of ~260 million banners across all snapshots.

3.2 Tokenization

To be able to feed text data to a transformer-model, it first needs to be split into tokens. BERT [12] uses subword tokens to limit its vocabulary size while minimizing unknown tokens, generated when encountering sequences not seen during training of the tokenizer. In contrast, RoBERTa [21] uses a byte-level, byte-pair encoding (BPE) tokenizer, which has the advantage of making tokenization lossless by falling back to single-byte tokens when necessary to prevent producing unknown tokens. Therefore, we use the RoBERTa tokenizer due to its losslessness (especially since scans can contain a combination of text and binary data, increasing the likelihood of unknown tokens if a non byte-level tokenizer is used). Note, however, that a pretrained RoBERTa tokenizer would not be optimal due to the different nature of our data. Therefore, we retrain a tokenizer with 50,000 tokens on 100 million randomly selected samples from our dataset, and use it for the remainder of this paper to encode banners into a sequence of tokens.

4 EMBEDDING GENERATION

In this section we go over the steps involved in training a transformer model for generating banner embeddings, including a masked

[1]According to Censys (https://support.censys.io/hc/en-us/articles/4407300349588-Search-2-0-Troubleshooting-Q-A), services flagged as truncated indicate hosts with more than 100 services that are highly likely to correspond to honeypots or firewalled hosts rather than real services.

language model and a supervised model (trained using a contrastive loss function) for generating stable embeddings.

4.1 Masked language model

We first train a masked language model (MLM) on banners in our dataset. MLMs are trained by randomly masking a small percentage of tokens, and then attempting to infer the masked tokens based on the context provided by surrounding text. This incentivizes the model to learn semantic relations, resulting in context-aware embeddings. Note that MLMs are bidirectional models, meaning that a token embedding depends on both preceding and following tokens. MLMs are typically the first step in training a large language model, which are then fine-tuned on downstream tasks, e.g., for classification or machine translation. We train a RoBERTa [21] model with 256-dimensional embeddings, 4 layers, 4 attention heads, and an intermediate layer size of 1024. We train this model for 100,000 iterations with batch size of 1024 and a learning rate of 0.0002, randomly masking 15% of all tokens during training. The above masked language model is trained on all protocols recorded in the Censys dataset, making it a great starting point for studies that aim to process this type of data. However, models and analysis in the remainder of this paper only focus on banners from the HTTP protocol, which constitute ∼70% of banners in our dataset.

4.2 Generating stable banner embeddings

4.2.1 Model description. A banner can contain dynamic sections that vary over time, including timestamps and randomized IDs regenerated every time a host is probed. It is desirable for banner embeddings to be invariant to these changes. Therefore, we use a supervised *contrastive* loss to incentivize the model to generate embeddings that remain stable over time, while maximizing the distance between embeddings of different services. To this end, we first identify banner pairs corresponding to the same IP address and port from two consecutive snapshots (with the associated snapshot denoted using $k \in \{1, 2\}$ from hereon), which is a strong indication that they originate from the same device. We then identify common parts in both banners; this is done by first splitting each banner into its respective headers, and then performing common substring matching to identify common parts in each header.[2] We ignore individual matches less than 3 characters long, and discard pairs where the sum length of all matches is less than 11 tokens. The latter filters out pairs that likely correspond to different devices/services,[3] with the threshold selected by manual inspection of a random set of pairs.

Formally, denote by $t_{i,j}^{(k)}, 1 \le j \le l_i^{(k)}$ the tokenized sequence associated with a banner, with $1 \le i \le n$ specifying a banner pair, and $l_i^{(k)}$ specifying the length of a tokenized sequence. We then generate labels $y_{i,j}^{(k)} \in \{0, 1\}$, indicating whether a token has been matched between two banners as previously described. We use these labels to train a token classification model that can predict stable sections in a banner, which in turn carry important information about the underlying service.

To generate banner embeddings, we need to aggregate the embeddings of all tokens in a sequence. Usually, the embedding of the first token (typically a special beginning-of-sentence token) or the average embedding of all tokens is used as the embedding of the entire sequence. Note, however, that a token with a label of zero should not contribute to the banner embedding, since it can be perceived as noise and does not contain information relevant to the underlying host. We therefore use the weighted average of all token embeddings, with the predicted labels as weights, resulting in embeddings that are robust to the noise present in each banner. Take $e_{i,j}^{(k)} \in \mathbb{R}^m$ and $0 \le \hat{y}_{i,j}^{(k)} \le 1$ to be the m-dimensional token embeddings (from the last layer of the transformer model) and the predicted labels, respectively. We then define the banner embedding $\bar{e}_i^k$ as follows (note that the embedding is normalized to have an ℓ_2-norm of one, which we will justify shortly).

$$\bar{e}_i^{(k)} := \frac{e_i^{(k)}}{\|e_i^{(k)}\|_2}, \quad e_i^{(k)} := \frac{\sum_j \hat{y}_{i,j}^{(k)} e_{i,j}^{(k)}}{\sum_j \hat{y}_{i,j}^{(k)}}.$$

While the above makes embeddings invariant to dynamic portions of a banner (i.e., with a label of 0), token embeddings for static portions (i.e., those with a label of 1) are not guaranteed to remain constant due to their context-aware nature, which may in turn cause embeddings to drift over time. We therefore modify the loss function to force the model to generate similar embeddings for matching pairs, while maximizing distances between randomly selected pairs as follows:

$$\mathcal{L} = -\frac{1}{n_t} \sum_{i,j,k} y_{i,j}^{(k)} \log \hat{y}_{i,j}^{(k)} + (1 - y_{i,j}^{(k)})(1 - \log \hat{y}_{i,j}^{(k)}) + \frac{1}{n} \sum_i \|\bar{e}_i^{(1)} - \bar{e}_i^{(2)}\|_2 - \frac{1}{n} \sum_i \|\bar{e}_i^{(1)} - \bar{e}_{i\oplus 1}^{(2)}\|_2 , \quad (1)$$

where n and n_t are the number of banner pairs and the number of tokens in a mini-batch, respectively. The first term in Equation 1 is the binary cross-entropy loss for token classification. The second term forces the model to generate similar embeddings for matching pairs, while the last term encourages the model to maximize the distance between an arbitrary pair. Note that for the last term, $\oplus$ denotes circular shift for generating non-matching pairs. Further note that we use embeddings with unit norm to prevent them from growing unboundedly large due to the last term.

The above approach utilizes temporal data (via two snapshots of the same device/service roughly one month apart) to supervise and help the model generate higher quality embeddings. Note that the idea of contrastive training using dissimilar/similar pairs has also been used in other domains, see, e.g., [10] for scientific documents, and [28] for neural recordings. The temporal information further incentivizes the model to generate similar embeddings for different versions/configurations of the same hardware or software product. This makes our embeddings suitable for clustering, which we will explore in the next section.

4.2.2 Training and evaluation. We fine-tune the masked language model described in subsection 4.1 using the loss function from Equation 1 for 20,000 iterations, a batch size of 1024 (512 pairs),

[2] Splitting HTTP banners into headers prevents matches between different headers, and further makes the model robust to services that may return headers in a random order that changes between snapshots.

[3] This can happen, e.g., when the underlying device/service is switched out by the operator, or when different physical devices happen to be observed on the same IP/port on two consecutive snapshots due to IP address churn.

```
HTTP/1.1 400 Bad Request
Server: EdgePrism/5.0.2.0
Mime-Version: 1.0
Date: <REDACTED>
Content-Type: text/html
Expires: Tue, 06 Sep 2022 00:38:30 GMT
X-LLID: 6d0829a0cfebc516165dbf208b4d4176
Content-Length: 0
Connection: close
```

Figure 2: An example of a HTTP banner: grayed out areas denote portions predicted to be dynamic. Note that header names (and the starting "HTTP/" for the status line) are ignored and never annotated by the model.

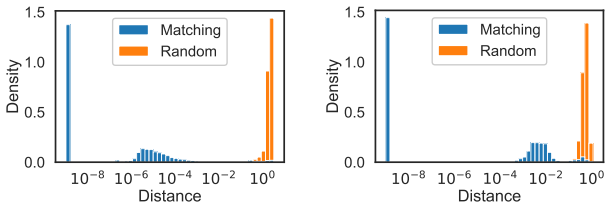

Figure 3: Distribution of ℓ_2 distances between embeddings of matching and random banner pairs for our proposed contrastive model (left) and a vanilla masked language model (right). Contrastive training results in a much more distinctive difference (roughly three orders of magnitude) between the two sets. Note that the x-axis has a logarithmic scale.

and a learning rate of 0.00005. Over a held-out test set of 100,000 pairs, token classification achieves an accuracy of 98.3%, with precision/recall values of 96.9%/98.9% for positive labels and 99.3%/97.9% for negative labels. This indicates that static/dynamic portions of a banner can be predicted with great accuracy. Figure 2 shows an example where the output of the model has been used to annotate static/dynamic portions of the banner (with grayed out areas denoting dynamic portion as predicted by the model). We observe that dynamically generated content (e.g., content of the `X-LLID` header and timestamps) are correctly predicted by the model.[4] Interestingly, the tail of the version is also predicted to be dynamic, which proves to be true in this instance, as the next snapshot has version `5.0.3.0`. This owes to the fact that the model is provided with matching pairs during training, allowing it to observe and subsequently predict frequent changes (e.g., minor version updates).

Figure 2 also demonstrates an important capability of our model by annotating important information in a banner, which in turn results in better interpretability by giving insight into the model's output. This is a direct result of using a transformer-based model which makes it possible to project predictions back onto the original text. This is in contrast to the bag-of-words and autoencoder models used by [26], which would result in a loss of interpretability.

4.3 Examining embeddings

We further examine embeddings generated by our model to show their utility for characterizing scan datasets. Figure 3 displays the distribution of ℓ_2 distances between embeddings of matching and random pairs obtained using the supervised/contrastive model detailed in subsection 4.2, as well as the vanilla masked language model trained in subsection 4.1. Both plots illustrate a separation between the two sets, with contrastive learning achieving a much more distinctive split. For the contrastive (masked language) model, 97.0% (94.9%) of matching pairs have distances less than 0.1, while 97.8% (99.0%) of random pairs have a distance more than 0.1. The jump in the left side of the plots in Figure 3 contains pairs with a distance of zero (due to a perfect match between the two banners), accounting for 52.3% of all pairs. For pairs with a non-zero distance, the average and standard deviation of log-distances is -4.58 ± 1.46 (-2.14 ± 0.70) for matching pairs, and 0.22 ± 0.54 (-0.31 ± 0.21) for random pairs. We therefore see that contrastive training results in a three-order magnitude difference in distances between matching and random pairs.

Pairs in Figure 3 are obtained one month apart. We also repeat the same experiment for the beginning/end of our observation window, resulting in pairs that are five months apart. For this longer window, the percentage of pairs with the exact same banner drops to 16.6%, while the percentage of matching pairs with a distance less than 0.1 is 94.7% (91.1% for the masked language model). This shows that supervised training can help attain robust embeddings that are stable over long observation windows

We also inspect 100 randomly selected banners pairs with a distance larger than 0.1. For all inspected examples, we observe that they either correspond to different hardware/software (due to entirely different header content, especially the `Server` header), or an updated configuration (due to changes in header content beyond simple version updates, e.g, the addition/removal of headers). We have included examples of these two cases in Figure 4. This shows that distances in the embedding space can be used to detect major/anomalous changes in a host, e.g., due to configuration updates, or when observations correspond to completely different devices. Note that the contrastive loss in Equation 1 reduces the number of false positives (e.g., as compared to a naive approach such as edit distances) by minimizing distances due to expected changes in dynamic portions of the banner, i.e., the grayed out sections in Figure 2. To further elaborate, the example in Figure 2 has a distance of 3.6e−4 from its subsequent snapshot despite a normalized edit (Levenshtein) distance of 0.166.[5]

5 FINGERPRINT GENERATION

The banner embeddings detailed in section 4 capture characteristics of the underlying service and are robust to arbitrary noise that may be present in a banner. To determine whether similar embeddings can be used to identify similar physical devices (i.e., with the same manufacturer and/or model) or server software (with the same

[4]Note that the year portion of the timestamp is still predicted to be static since all of our data was obtained from the same year. A longer observation window should teach the model to also treat the year as dynamic.

[5]We define the normalized edit distance as the absolute edit distance (the number of operations required to transform one string into the other) divided by the maximum length of the two strings.

```
HTTP/1.1 200 OK
Date:  <REDACTED>
Server: Apache
X-Frame-Options: SAMEORIGIN
Last-Modified: Mon, 21 Sep 2020 23:43:29 GMT
ETag: "c445-5afdb6adb2a40"
Accept-Ranges: bytes
Content-Length: 50245
Cache-Control: no-store, must-revalidate
Pragma: no-cache
Expires: 0
Content-Type: text/html; charset=UTF-8
Cache-Control: no-cache
```

```
HTTP/1.1 200 OK
Age:          1
Date:  <REDACTED>
Cache-Control: no-cache,no-store,must-revalidate
Connection: Keep-Alive
Via: NS-CACHE-10.0: 127
ETag: "a717-5ec898f83af80"
Server: Apache
X-Frame-Options: SAMEORIGIN
Last-Modified: Thu, 03 Nov 2022 04:40:46 GMT
Accept-Ranges: bytes
Content-Length: 42775
Feature-Policy: camera 'none'; microphone 'none'; geolocation 'none'
Referrer-Policy: no-referrer
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
```

```
HTTP/1.1 200 OK
Server: nginx
Date:  <REDACTED>
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.4.16
Content-Encoding: gzip
```

```
HTTP/1.1 200 OK
Date:  <REDACTED>
Server: Apache
Last-Modified: Thu, 24 Nov 2022 17:26:40 GMT
ETag: "1cb-5ee3ab54558da"
Accept-Ranges: bytes
Content-Length: 459
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figure 4: Examples of banner pairs (snapshots obtained one month apart from the same IP address and port) exhibiting major configuration changes (top), and corresponding to different server products (bottom). The pair at the top has a distance of 0.3 in the embedding space, while the pair at the bottom has a distance of 2.3.

vendor/product), we automatically generate and examine text-based regular expression fingerprints from clustered service banners, as described in the remainder of this section.

5.1 Clustering

To speed up clustering, we apply PCA to reduce embedding dimensionality from 256 to 64. We train PCA using 5 million randomly selected embeddings. For the selected principal components, the cumulative sum of the explained variance ratio is 99.97%, resulting in minimal information loss.

For clustering we use HDBSCAN [15], a hierarchical variation of DBSCAN [7] for density-based clustering. The original DBSCAN detects areas of high-density surrounded by low-density regions. While DBSCAN requires an `epsilon` value (the maximum distance for two samples to be considered neighbors), HDBSCAN removes this hyperparameter by trying various values and finding a clustering with the best stability over `epsilon`. We choose HDBSCAN since it is designed to handle clusters of varying shapes and sizes (e.g., non-convex clusters that can not be retrieved with centroid-based methods such as K-Means), and does not require knowing the number of clusters beforehand.

We train HDBSCAN using 5 million embeddings and setting `min_cluster_size` = 50 and `min_samples` = 5. HDBSCAN can also be provided with a `cluster_selection_epsilon`, for which clusters below the given threshold are merged; increasing this value decreases the total number of clusters by preventing the algorithm from generating micro-clusters. We generate four different clusterings with varying levels of granularity by setting `cluster_selection_epsilon` $\in \{0.01, 0.02, 0.05, 0.1\}$. The resulting number of clusters and percentage of unclustered outlier are 5452/5.86%, 3989/4.57%, 2138/2.32% and 736/0.63%, respectively

5.2 Fingerprint generation

To generate text-based fingerprints that describe each cluster, we randomly select 10 samples from a cluster and apply longest common substring matching to extract common substrings between all samples. Note that this matching is done on a per-header basis. We then convert the result to a regular expression (composed of a series of substrings and wildcard expressions), resulting in a single regex pattern per header. We repeat this process by selecting 100 different sets of 10 samples from each cluster, and generate patterns from all four clusterings from subsection 5.1 to obtain a larger pool of fingerprints with different granularities. This process yields 15,718 patterns/fingerprints from all headers, likely to capture physical (e.g., IoT) devices and/or server software.

We compare our fingerprints to hand-curated regex patterns from Recog [25], also utilized by Censys for labeling scan data. We first extract Recog fingerprints for the HTTP `Server`, `Set-Cookie`, and `WWW-Authenticate` headers, containing 447, 82, and 77 fingerprints at the time of this writing. We then examine our fingerprints (798, 2478, and 635 for the aforementioned headers), and find fingerprints not captured in Recog. Table 1 includes some such examples,

Table 1: Example hardware/software fingerprints generated from different HTTP headers and not captured in the Recog [25] database. This demonstrates the ability of our automated fingerprint generation technique to complement existing databases and to help keep fingerprints up-to-date.

Header	Regular expression	Description
Server	`.*rovotech/2\.0\.0$`	Brovotech IP Camera
Server	`^ALARM\.COM-HTTP-Server$`	Home Automation/monitoring
Server	`^ZNC .* - http://znc\.in$`	ZNC IRC Network Bouncer
Set-Cookie	`^grafana_sess=.*; Path=/; HttpOnly$`	Grafana Web Application
Set-Cookie	`^interworx-cp=.*; path=/.*`	InterWorx Web Hosting Control Panel
Set-Cookie	`^roundcube.*; path=/.*`	Roundcube Email Client
WWW-Authenticate	`^Basic realm="LiveboxProv3"$`	Sagemcom Livebox Pro V3 Router
WWW-Authenticate	`^Basic realm="ZNID24xx.*-Router"$`	Zhone zNID 24xx Series Router
WWW-Authenticate	`^Digest realm="Wisenet NVR", nonce=".*", qop="auth"$`	Hanwha Wisenet Network Video Recorder

including patterns for identifying IoT devices as well as server products. This shows that our framework can be used to complement existing fingerprint databases, and can be applied to scan data on a regular basis to keep fingerprints up-to-date.

We also examine the number of Recog fingerprints that are recovered using technique. We first match Recog fingerprints to ours by finding the pair with the highest overlap ratio.[6] We then filter for fingerprints with an overlap of at least 90%, resulting in recovering 117, 9, and 22 of Recog fingerprints for the HTTP `Server`, `Set-Cookie`, and `WWW-Authenticate` headers. We further observe that the recovered fingerprints account for 98.1%, 63.2%, and 61.2% of all banners that are labeled by Recog for each header, respectively. Additionally, we also search for partial matches by finding fingerprints that are a subset of a Recog pattern (with >90% coverage). This yields partial matches for 251, 25, and 51 of Recog fingerprints for the aforementioned headers, accounting for 99.3%, 89.6%, and 96.1% of banners labeled by Recog. This analysis shows that our proposed technique is successful at recovering frequent patterns. Improving this method to also recover less frequent patterns is a direction that we will explore in future research.

6 DISCUSSION

This paper demonstrates the potential use of LLMs to enable automated analysis of Internet scan data, especially for the purpose of clustering similar devices, as well as extracting hardware/software fingerprints. The ability of transformers to annotate raw text can also be a useful tool for highlighting important information in a banner as shown in Figure 2. Note, however, that while the generated fingerprints can be used to identify specific products (as shown in Table 1), manual validation is still needed to sanitize the associated regex patterns and remove (near) duplicates. As an example, some patterns in Table 1 also capture device configuration in addition to the underlying product (e.g., through the "`path=/.*`" portion of regexes for the `Set-Cookie` header). In addition, data captured in the wildcard portion of the regexes sometimes contains useful information, e.g., the pattern for Zhone routers can capture different models such as `ZNID24xxA1` or `ZNID24xxB1`.

Given the above limitation, an exhaustive examination of the generated patterns to obtain higher quality fingerprints is an important aspect of our ongoing work. It would be interesting to examine whether fingerprint sanitization can also be automated using machine learning. Further note that while we have only showcased our framework on the HTTP protocol, the same methodology can also be applied to other protocols where existing fingerprints are even more sparse.

The proposed technique focuses on temporal stability and the ability to recognize static/dynamic portions of a banner to ensure embeddings are of high-quality. This is achieved by leveraging matching/similar pairs during training to make embeddings invariant to dynamic portions of a banner, while maximizing distances between random pairs. This follows the idea of contrastive learning to create high-fidelity embeddings, an approach that has also been applied to other domains, e.g., to characterize scientific documents [10] and neural recordings [28]. However, this also subjects embeddings to the quality of matching pairs used during training. We use snapshots obtained from the same IP/port to ensure temporal stability, while omitting pairs that likely correspond to different devices/services as discussed in subsection 4.2. Note, however, that the same approach can be used with other techniques for retrieving similar pairs, to generate high-quality embeddings for other application-driven similarity criteria.

7 CONCLUSION

This paper presents a LLM trained on hundreds of millions of banners obtained from Internet-wide scans, a first such effort to the best of our knowledge. Our preliminary analysis shows that we can generate stable and robust numerical embeddings, capturing information about the underlying software/hardware product deployed on an Internet host. This makes our model and the resulting embeddings a suitable candidate for automated analysis, anomaly detection, clustering, and fingerprint generation, which we will explore in more detail in our future research. Utilizing our approach to capture device/service configurations, process network traffic traces, and further leveraging the generative capabilities of LLMs are other directions for future work.

8 ACKNOWLEDGMENTS

We thank Qingyue Jiao, Rohan Sequeira, Sanjana Prabhu, our shepherd (Nina Taft), and the anonymous reviewers for their contributions and valuable feedback. This work is supported by the NSF under grant CNS-2012001.

[6] We compute the overlap ratio as the number of banners that match both fingerprints, divided by the number that match either of the two.

REFERENCES

[1] [n. d.]. Shodan Search Engine. https://www.shodan.io.

[2] Ahmet Aksoy and Mehmet Hadi Gunes. 2019. Automated IoT device identification using network traffic. In *IEEE International Conference on Communications*. IEEE, 1–7.

[3] Sandhya Aneja, Nagender Aneja, and Md Shohidul Islam. 2018. IoT device fingerprint using deep learning. In *IEEE International Conference on Internet of Things and Intelligence System*. IEEE, 174–179.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai botnet. In *USENIX Security Symposium*. 1092–1110.

[5] Shehar Bano, Philipp Richter, Mobin Javed, Srikanth Sundaresan, Zakir Durumeric, Steven J Murdoch, Richard Mortier, and Vern Paxson. 2018. Scanning the Internet for liveness. *ACM SIGCOMM Computer Communication Review* 48, 2 (2018), 2–9.

[6] 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.

[7] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*. Springer, 160–172.

[8] Censys. [n. d.]. Universal Internet BigQuery Dataset. https://support.censys.io/hc/en-us/articles/360056063151-Universal-Internet-BigQuery-Dataset.

[9] Rajarshi Roy Chowdhury and Pg Emeroylariffion Abas. 2022. A survey on device fingerprinting approach for resource-constraint IoT devices: Comparative study and research challenges. *Internet of Things* (2022), 100632.

[10] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S Weld. 2020. Specter: Document-level representation learning using citation-informed transformers. *arXiv preprint arXiv:2004.07180* (2020).

[11] Nicholas DeMarinis, Stefanie Tellex, Vasileios Kemerlis, George Konidaris, and Rodrigo Fonseca. 2018. Scanning the Internet for ROS: A view of security in robotics research. *arXiv preprint arXiv:1808.03322* (2018).

[12] 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).

[13] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *ACM Conference on Computer and Communications Security*. ACM, 542–553.

[14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The matter of heartbleed. In *Internet Measurement Conference*. ACM, 475–488.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *International Conference on Knowledge Discovery and Data Mining*, Vol. 96. 226–231.

[16] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *USENIX Security Symposium*. 1323–1338.

[17] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. 2018. Acquisitional rule-based engine for discovering Internet-of-Things devices. In *USENIX Security Symposium*. 327–341.

[18] Talha Javed, Muhammad Haseeb, Muhammad Abdullah, and Mobin Javed. 2020. Using application layer banner data to automatically identify IoT devices. *ACM SIGCOMM Computer Communication Review* 50, 3 (2020), 23–29.

[19] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of age: A longitudinal study of TLS deployment. In *Internet Measurement Conference*. ACM, 415–428.

[20] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. 2018. Tracking certificate misissuance in the wild. In *IEEE Symposium on Security and Privacy*. IEEE, 785–798.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[22] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. 2015. Cloudy with a chance of breach: Forecasting cyber security incidents. In *USENIX Security Symposium*. 1009–1024.

[23] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. 2017. ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis. In *Proceedings of the Symposium on Applied Computing*. 506–509.

[24] Nizar Msadek, Ridha Soua, and Thomas Engel. 2019. IoT device fingerprinting: Machine learning based encrypted traffic analysis. In *IEEE Wireless Communications and Networking Conference*. IEEE, 1–8.

[25] Rapid7. [n. d.]. Recog: A Recognition Framework. https://github.com/rapid7/recog/tree/main.

[26] Armin Sarabi and Mingyan Liu. 2018. Characterizing the Internet host population using deep learning: A universal and lightweight numerical embedding. In *Internet Measurement Conference*. ACM, 133–146.

[27] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, Dave Choffnes, Alan Mislove, et al. 2018. A first look at certification authority authorization (CAA). *ACM SIGCOMM Computer Communication Review* 48, 2 (2018), 10–23.

[28] Steffen Schneider, Jin Hwa Lee, and Mackenzie Weygandt Mathis. 2023. Learnable latent embeddings for joint behavioural and neural analysis. *Nature* (2023), 1–9.

[29] 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).

[30] Juan Wang, Jing Zhong, and Jiangqi Li. 2023. IoT-Portrait: Automatically identifying IoT devices via transformer with incremental learning. *Future Internet* 15, 3 (2023), 102.