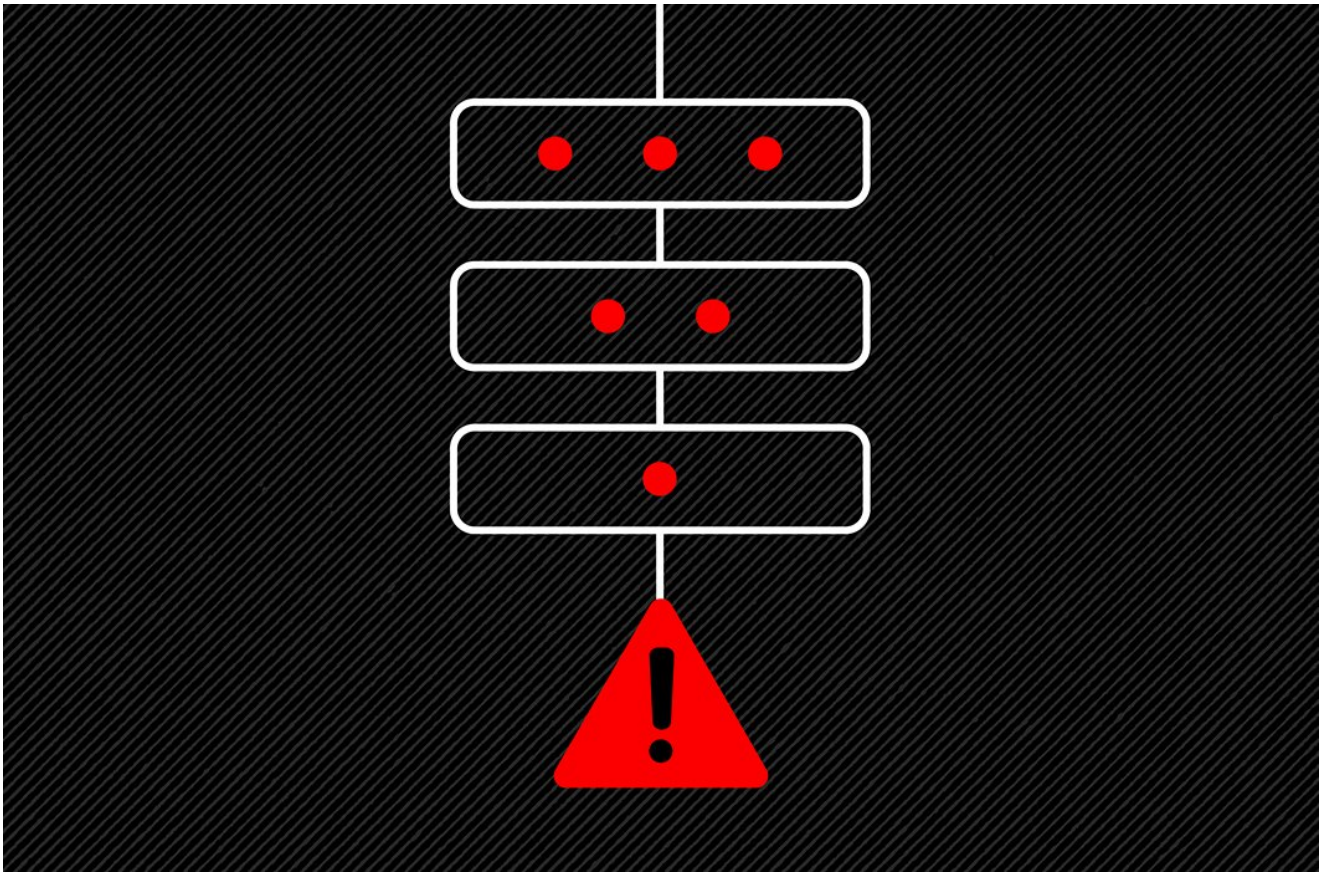
BERT Embeddings Part 2: A Modern ML Approach For Detecting Malware

crowdstrike.com/blog/bert-embeddings-new-approach-for-command-line-anomaly-detection-part-2/

Cristian Popa April 1, 2022



- A novel methodology, BERT embedding, enables large-scale machine learning model training for detecting malware
- It reduces dependency on human threat analyst involvement in training machine learning models
- Bidirectional Encoder Representation from Transformers (BERT) embeddings enable performant results in model training
- CrowdStrike researchers constantly explore novel approaches to improve the automated detection and protection capabilities of machine learning for Falcon customers

CrowdStrike data science researchers recently explored and experimented with the use of Bidirectional Encoder Representation from Transformers (BERT) for embedding command lines, focusing on anomaly detection, but without detailing the model itself. Diving deeper into that research, CrowdStrike researchers explain the reasons for using BERT for command line representation and how to train the model and assess its performance.

The purpose of this experimental research was to leverage self-supervised deep learning methods to obtain better representations of the string fields that show up in CrowdStrike Falcon® telemetry. CrowdStrike constantly tests the latest advancements in the field to see whether they fit the existing machine learning toolkit. This research ultimately demonstrates that a deep learning approach for embedding strings of this nature (command line) is feasible and can produce satisfactory results.

Defining Objectives

An embedding is a representation of an input, usually into a lower-dimensional space. Although embeddings are in scope for various input types, string embeddings are a common choice for representing textual inputs in numeric format for machine learning models. An essential trait of embedding models is that inputs similar to each other tend to have closer latent space representations. This similarity is a consequence of the training objective, but researchers are often interested in lexical similarity when strings are involved.

This experiment aimed to better represent string fields encountered in a large stream of event data. The focus was on two such fields: “command line” and “image file name.” The first field was discussed extensively in the previous blog — it consists of an instruction that starts a process that is then recorded as an event and sent to the CrowdStrike Security Cloud. The second field is the name for the executable starting the process, which is a substring of the corresponding command line. Two main factors dictated the pursuit of such an embedding model: First, we aimed to improve the string processing in our models, and second, we wanted to benefit from the significant developments achieved in the area of natural language processing (NLP) in the last few years.

Building the Model

Data

The experiment started by collecting data for training the model from events related to process creation. To ensure variety, it was collected from all of the supported platforms (Windows, Mac, Linux) and sampled from long spans of time to ensure that the processes are not biased by temporary events (e.g., the Log4j vulnerability).

Model Architecture

For the model architecture, BERT was the primary candidate. The end-to-end model consists of two main components that will be discussed separately: the tokenizer and the neural network (which is what people generally refer to when talking about BERT).

A tokenizer is a simple mapping from strings, called tokens, to numbers. This numeric representation of the inputs is necessary because the BERT neural network, like any other machine learning algorithm, cannot use textual data directly in its computations. The

tokenizer's job is to find the optimal tokens given a vocabulary size, which, in this case, was 30,000 tokens. Another advantage of using a tokenizer is that unknown strings from the training set can still be composed out of tokens learned from other strings. This is important because English has a well-defined set of words, while command lines can theoretically feature any character combination. In Figure 1, there is an example of a tokenized command line in the data set.

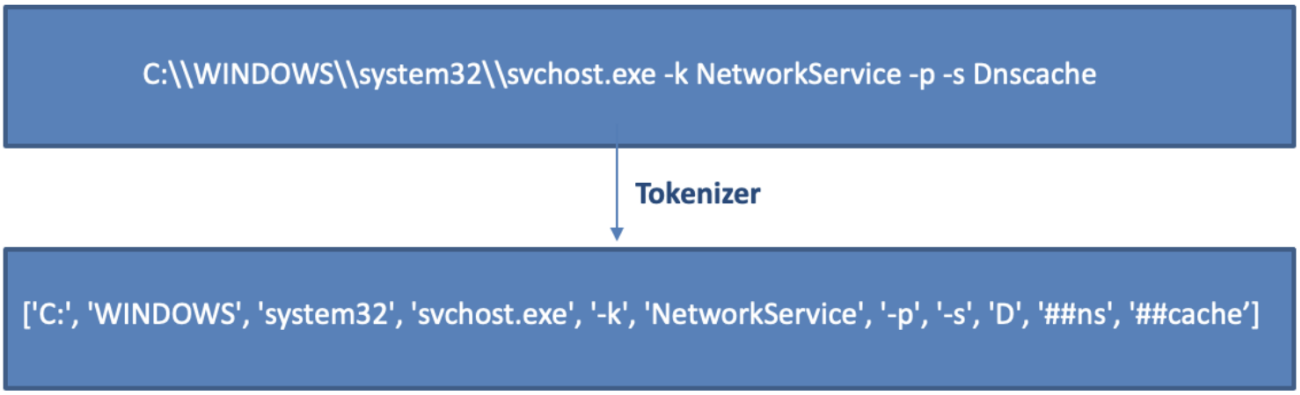


Figure 1. Example of a tokenized command line

The BERT model is an extensive neural network that relies on an NLP concept called "self-attention mechanism." The concept, introduced by Vaswani et al., 2017, has gained significant traction in the research community, to the point where almost all modern NLP solutions use it. To briefly explain how it works, Figure 2 shows how tokens passed through the network pay attention to other relevant tokens in the input. BERT can compute the attention of tokens and use it to build an understanding of the language. Another important concept used by BERT is **masked language modeling (MLM)**. This is the training objective used by the network. Tokens in the input are randomly masked, and the model has to predict the initially masked tokens.

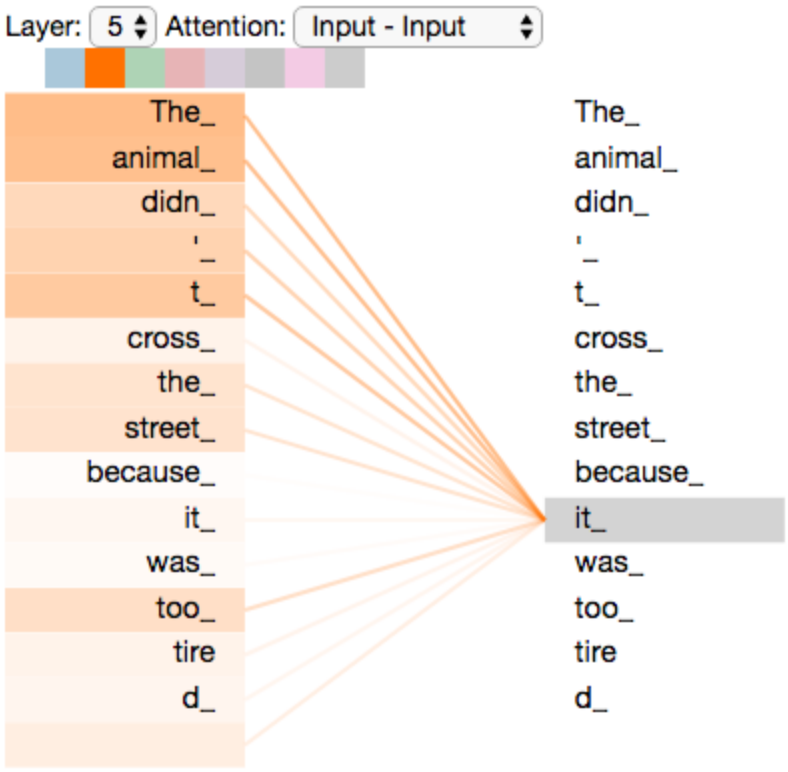


Figure 2. The “it” token in the sentence “The animal didn’t cross the street because it was too tired” comes into focus. The attention is supposed to make sense from a syntactic point of view and is used in machine learning models. Source: https://jalammar.github.io/illustrated-transformer/

The MLM objective allows for *self-supervised learning*, meaning that researchers do not need to explicitly label data, but instead they only need the input strings themselves. This technique presents an advantage for cybersecurity as it removes the need to involve threat analysts in this initial training step.

Diving deeper, there are some appropriate training steps for BERT: **pre-training** and **fine-tuning**. First, the model is pre-trained using MLM on a large dataset consisting of command lines and image file names. The model is then fine-tuned with a malware classification objective, on a different dataset, for example. With this approach for pre-training, it becomes easy to collect a large dataset from which the model will learn the representation of the data. Hence, the second phase requires significantly fewer labeled samples in the dataset. Learning from smaller amounts of labeled data constitutes an obvious advantage that becomes applicable in this case. Additionally, the first step is task-agnostic, meaning that the pre-trained model can be fine-tuned later for any task needed — malware classification, malware family identification, anomaly detection, etc.

Experiments

After getting the model ready for training, one of the first steps is sifting through the data for a diverse subset of samples because the tens of billions of events collected from Falcon telemetry were too many for training BERT from scratch in a reasonable amount of time.

Small embedding models were repeatedly trained briefly to identify the samples they were performing worse on, excluding them from the subset.

Afterward, modeling efforts mainly revolved around finding the right hyper-parameters. Focus was placed on the performance for malware classification using a holdout set from the fine-tuning data, as this was a good measurement of the added benefit of using the embeddings over previous features.

The hyper-parameter that brought the most significant improvement was the change to the maximum number of tokens that can get into the BERT model. This is relevant because while image file name strings, which are shorter, would often fit fine into the default token limit, command lines would not. As a result, a huge chunk of information is lost due to the truncation of the tokens, which was intended to bring all inputs to the same size. Increasing this limit to a calibrated value was crucial in the modeling process. The other experiments focused on the embeddings' size and the number of hidden layers in the neural network. Optimal values were found for them according to the evolution of the classification metrics. Figures 3 and 4 show the fine-tuning process and its rationale.

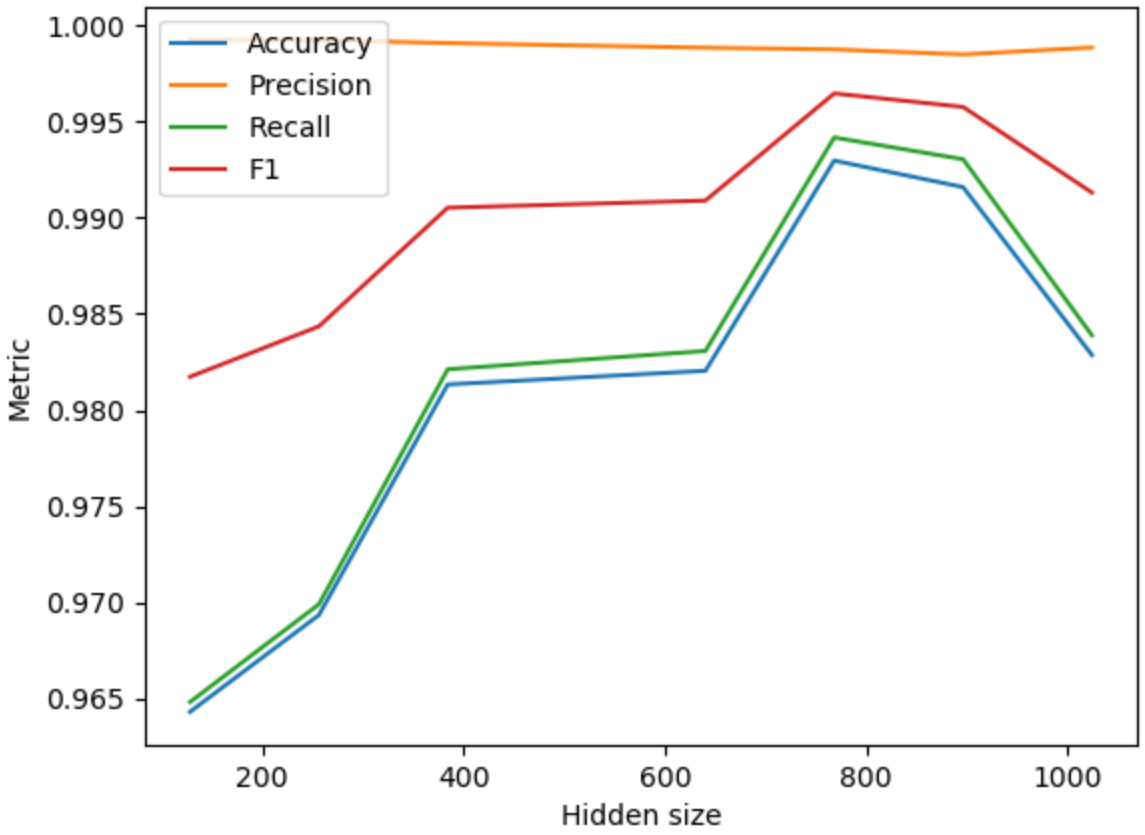


Figure 3. Classification performance while varying the hidden (embedding) size

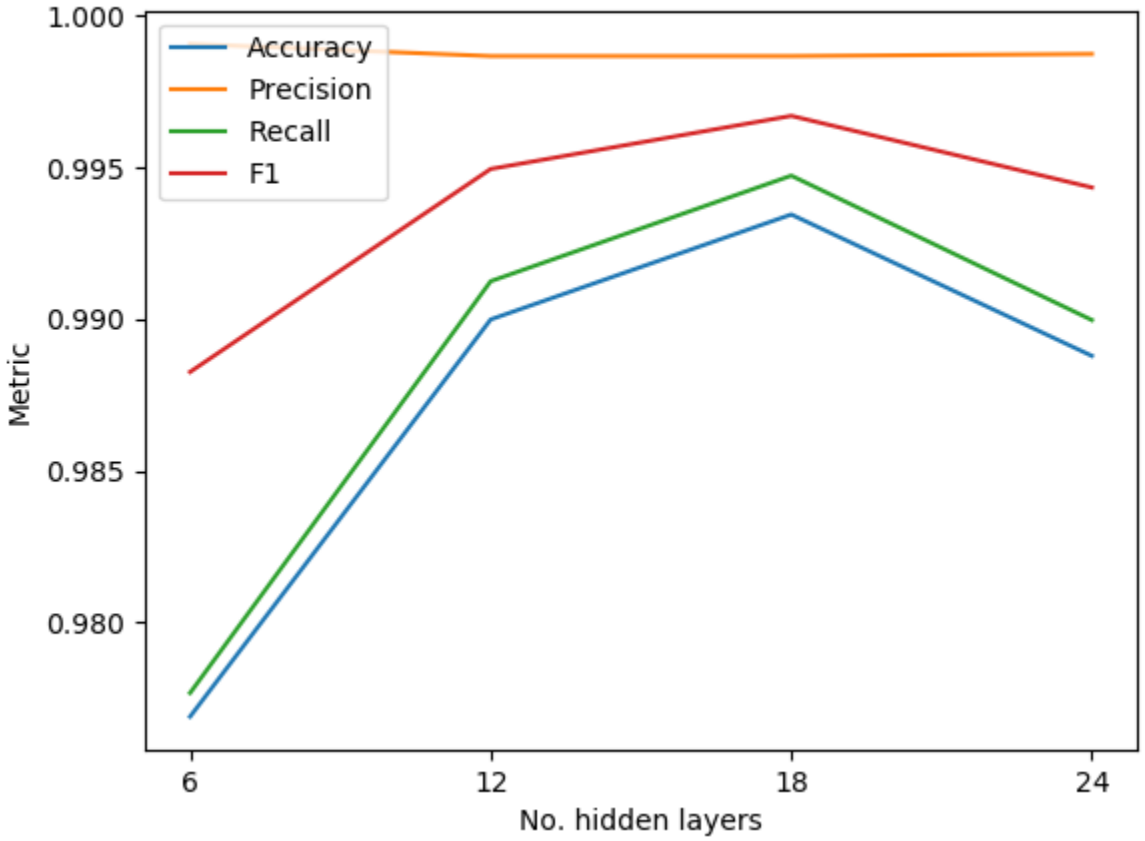


Figure 4. Classification performance while varying the number of hidden layers

In the end, our experiment resulted in an embedding model whose embeddings seem on par with the original features for two of CrowdStrike's existing machine learning models that used strings. These models use not only command lines or image file names for classification, but also other fields coming from the event data, such as execution flags and start times. Results can be observed in Figures 5 and 6.

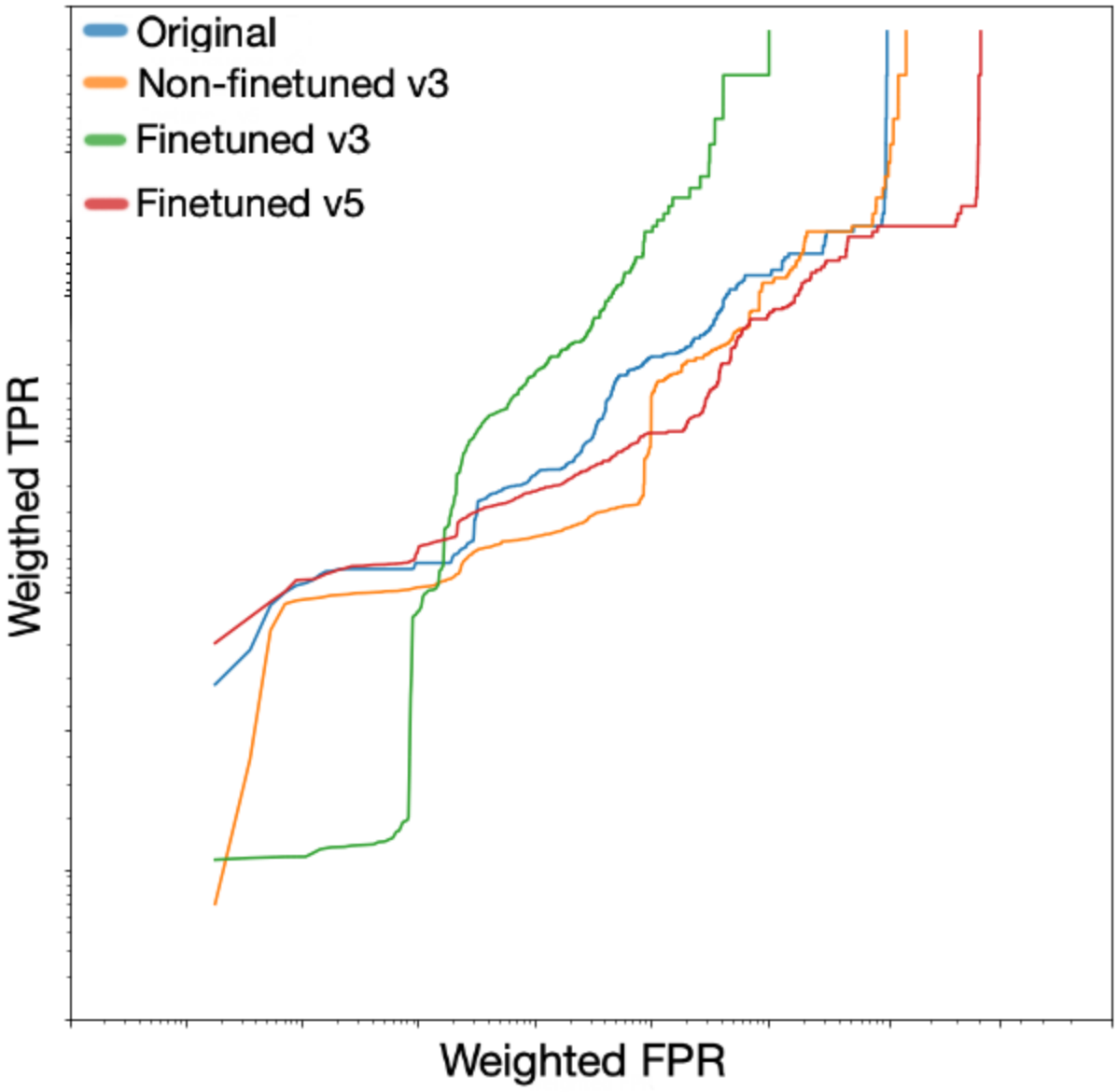


Figure 5. Weighted* true positive rate (TPR) and false positive rate (FPR) for different versions of one of our classification models. “Finetuned_v5” is the version using the embeddings from our latest (and best) fine-tuned BERT model, while “v3” is an earlier version of the model.

* Note: “Weighted” means that the frequency of samples in the dataset is accounted for when computing the metrics, as the data used is guaranteed to contain duplicates

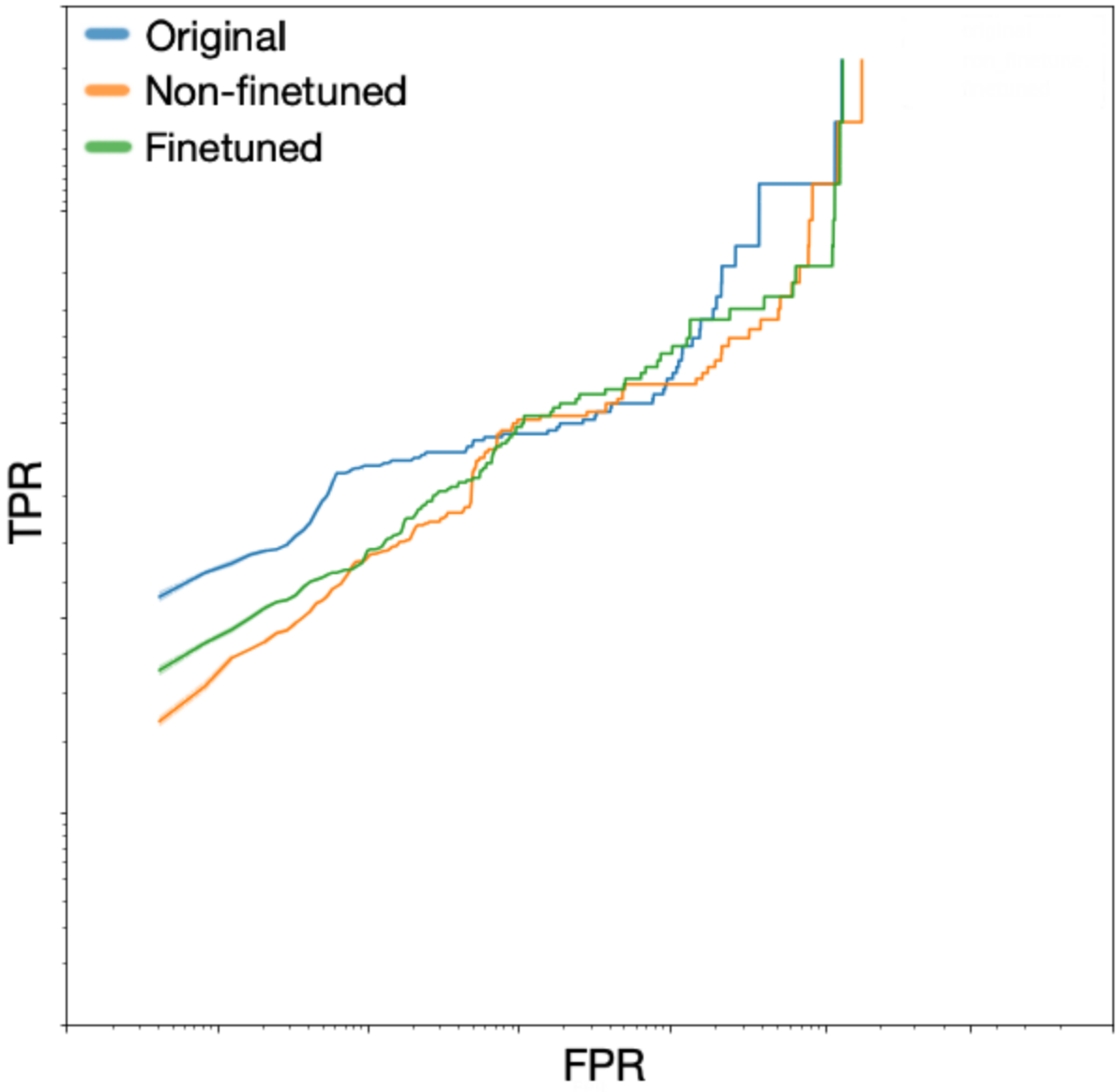


Figure 6. TPR vs. FPR for another one of our classification models

One observation was that the fine-tuned model shows better performance, which makes sense because it was specifically trained to separate between clean and dirty samples.

Future Research Opportunities

Future areas of research interest involve training an end-to-end neural network for classification that incorporates BERT for the string-type features. Currently, the embeddings are computed and used in our gradient boosted tree models. Putting everything together into a single neural network would probably improve the efficiency of the BERT embeddings, as they are trained along with the rest of the features. There is significant potential that the

embeddings would be better used by a deep learning algorithm than a tree-based one, since the latter makes predictions based on higher or lower features than a trained value, while the deep learning algorithm can process the input more freely.

CrowdStrike researchers constantly explore novel approaches to improve the efficacy and the automated detection and protection capabilities of machine learning for customers.

Additional Resources

- *Learn more about the CrowdStrike Falcon® platform by visiting the product webpage.*
- *Learn more about CrowdStrike endpoint detection and response on the Falcon Insight webpage.*
- *Test CrowdStrike next-gen AV for yourself. Start your free trial of Falcon Prevent™ today.*