Addressing Underspecification with Trustee, netReplica, and LLMs

Kate Larrick, Colter Sirlin, Michelle Zimmermann

Abstract

While tools like Trustee are able to produce human-readable representations of a black-box model's hidden decisions, analyzing these reports requires a domain expert who understands the model being represented, the features of the dataset used to train the model, and the details of the network that the dataset was collected from.

The goal of our project is to streamline the process of identifying shortcut learning in a Trustee trust report decision tree with the use of LLMs. To test the feasibility of using an LLM to replace a domain expert in analyzing a model's decision tree, we took datasets collected with netReplica and Puffer, purposely skewed them to introduce biases, trained a model on these skewed datasets, and developed an interface that allows an LLM (specifically Google AI's Gemini model) to process the decision tree produced by Trustee from the given model and dataset. We then instructed the LLM to provide possible modifications to the dataset to mitigate any bias it discovered. Finally, we analyzed the LLM's output by determining if it identified the bias that we had purposely introduced.

1 Introduction

Machine learning models have made significant progress detecting complex patterns in network traffic and addressing network security problems. However, there is still a limited amount of data for these models to use, and existing datasets have been demonstrated to have significant issues [1]. These problems can be difficult to identify, as models trained on biased data may perform exceptionally well on that dataset–sometimes even better than if the data were more realistic, uniform, and random. However, these biased models fail when used in other contexts.

Researchers have created tools to highlight such issues. For example, Trustee [1] makes black-box models interpretable by creating a decision tree that closely mirrors the underlying model's decisions. This helps people identify underspecification issues such as shortcut learning, out-of-distribution issues, and spurious correlation. netReplica [2] can generate fine-tuned network configurations. These networks can be simulated to explore the QoE of custom network environments. In tandem, Trustee and netReplica can be used to identify underspecifications in a model and generate network configurations where the model performs poorly. Then a new iteration of training can incorporate data from these configurations, resulting in a more generalizable and accurate model. This process can be repeated, improving the model with each iteration. This plan is promising but is limited by a reliance on human interpretation to find shortcomings in the model and create test cases that expose them.

We propose automating this iterative process by replacing the human-in-the-loop with an LLM agent. Although we narrowed the scope of our project to analyzing the ability of an LLM to identify and correct bias in a dataset used to train a machine learning model, the next step would be to modify the biased dataset with the LLM's suggestions and retrain the model on the adjusted dataset in a loop until an appropriately high confidence is achieved.

In this paper, we show that an LLM, when prepared with the proper context, can identify bias in a training dataset to a degree of accuracy that allows someone who is not a domain expert to understand the modifications that must be made. While netUnicorn has the same goal of iteratively improving model generalizability using explainability tools like Trustee, it requires real-world data collection and a human-in-the-loop to determine what new training data to collect for each iteration [3]. Although we are yet to prove that an LLM agent can fully replace the human-in-the-loop for the analysis of the Trustee trust report, our work is the first step in allowing iterative model improvement to proceed completely autonomously.

The remainder of this report is structured as follows: we provide a high level overview of the scope of our problem, with a definition of its goals and limitations, and then we detail our experiment design approach and implementation along with the challenges we encountered in skewing datasets, developing our LLM interface, and analyzing the LLM's output. Lastly, we illustrate how we evaluated the LLM's output to determine the quality of its suggestions.

2 Background and Motivation

Analyzing the decision tree in the trust report produced by Trustee requires a true domain expert, which we experienced ourselves as we attempted to identify bias in our baseline decision tree (produced by running Trustee on the Fugu machine learning model and our NetReplica dataset). The pruned version of this tree had 19 levels and dozens of nodes, making it difficult for an untrained viewer to follow every decision path and draw conclusions about underspecification in the dataset. While there exist numerous approaches to using LLMs for generating customized training data, we were unable to find an approach where an LLM analyzes a report like the one produced by Trustee in order to explain a dataset's bias in conversational language. Thus, our novel approach aims to make the trust report produced by Trustee accessible to non-domain experts by analyzing the decision tree and producing a report in common English that a networking student or trainee could understand.

3 Design/Approach/Methodology

Our goal in this experiment was to determine if Gemini would be able to pick up on the exact underspecification in a dataset that we purposely introduced. Our general approach involved creating a biased dataset, generating a Trustee trust report for the dataset and the Fugu machine learning model, passing the trust report decision tree (in Graphviz DOT representation) to an interface for Google's Gemini AI that we developed, and analyzing Gemini's output for accuracy.

The main strategy for skewing our datasets was to remove important groups of datapoints from the distribution–we anticipated that these reduced datasets should each have their own biases compared to the original datasets since these simpler datasets do not contain the same diversity of training data. We documented the groups of datapoints that we removed (the skew in each dataset) for comparison purposes in the evaluation stage.

Next, we developed our Gemini interface with a trial-and-error process: we passed our baseline trust report (from the original NetReplica dataset) to Gemini and analyzed its understanding of what the decision tree represented. We then adjusted our prompt until we were content with Gemini's evaluation of the trust report.

Before obtaining any actual LLM output, our plan for analysis was to first ensure that the LLM produced very similar output with each trial on the same decision tree, then to compare the bias identified by the LLM to the bias that we had purposely introduced.

4 Implementation

Our key preliminary step was skewing the datasets we had acquired in various ways in order to artificially introduce bias. Tests 1-5 exclusively use the dataset generated with NetReplica (netrep_100ms.csv), and test 6 incorporates the dataset from Stanford's Puffer experiment (oct_100ms.csv).

1. ascend_descend: we noticed that the 'delivery_rate', 'cwnd', and 'size' parameters tend to increase in netrep_100ms.csv from chunk 0 to chunk 8. We thought this could be caused by a strong network connection on an uncongested network, allowing for throughput to improve. However, in some situations these parameters decreased over time, potentially indicating un-ideal network conditions. We predicted that excluding these cases from training data might cause the model to only learn the happy "ascending" cases without getting insight from the arguably more important "descending" cases where network conditions aren't ideal. Datapoints where two or more of the three variables are increasing are called "ascending" and datapoints where two or more of the three variables are decreasing are called "descending."
2. cwnd_len: we noticed a long-tailed distribution in cwnd (number of in-flight packets allowed without a response). We wanted to see what would happen if we separated these out (about 4% of the total).
3. delivery_rate_len: we noticed a spike and then a big drop in datapoint frequency at around 310 Mbps, indicating a paradigm shift in network conditions, so we split the data here. (23% of datapoints had delivery_rate ¿ 340)
4. rtt_len: we noticed a long-tailed distribution in rtt, indicating some tests ($\tilde{3}$%) experienced high latency. We wanted to isolate these to see if removing them impacted data quality.
5. trans_time_len: we noticed a long-tailed distributed in trans_time, indicating some tests ($\tilde{3}$%) experienced high latency. We also wanted to isolate these to see if removing them impacted data quality.
6. compare_netrep_oct: we wanted to train the model on one dataset and evaluate it on the other to see how well the model transfers to new data. We also wanted to train and evaluate the model on the union of both datasets to see if more diverse input data improves black-box model and Trustee decision tree quality.

To split these datasets, we wrote a script (split.py) that separated the data into all categories being tested, scanned for ascending and descending sequences of datapoints, and generated subsets of the original datasets based on our criteria.

Once we had prepared a collection of purposely-skewed datasets, our next step was developing an interface to interact with Google's Gemini AI. We decided on using Gemini because Google provides free API keys for developers to use.

The first challenge we faced was Gemini going out of context due to the length of the decision tree text representation. To solve this, we used Gemini's context-cache feature1 , which stores a large amount of text, audio, or video for the model to use when answering questions.

Another challenge we encountered was deciding how much context to give Gemini to prepare it for an extensive and complicated decision tree without compelling it to be too specific or overly critical. We began with providing basic information about the datasets and what they contained ("networking data with features such as IP addresses, ports, protocols, traffic volume, packet timing, etc."), then provided five rows of our NetReplica dataset as an example. We then specified the dataset issues we wanted Gemini to diagnose by evaluating the decision tree–we asked that it "analyze the tree structure, decision splits, feature usage, and any patterns that might indicate bias in the underlying networking dataset."

The last step of preparing our Gemini interface was deciding our desired output format. We determined that we wanted a boolean answer for whether or not the dataset was biased, a brief explanation of the bias or lack of bias, a list of potential causes if bias was found, and a list of features that were seemingly causing bias.

With our skewed datasets and LLM interface prepared, we then generated a Trustee trust report for each of the six biased datasets and fed each trust report to the LLM a minimum of three times for comparison between each other, with the intuition that the output should be just about the same each time for one trust report.

One last difficulty we had when testing the LLM interface was preventing "overfitting," where the LLM was noticing small issues in single nodes rather than finding patterns of bias in the overall tree. We decided to take just the first 20 lines of the decision tree text representation, which defines the top 20 nodes in the tree, and we found that our results were more generalized without being too vague.

5 Evaluation

Since our project produces a natural language assessment of Trustee's reports, it is necessarily inexact and subjective to evaluate the quality of Gemini's output. However, we use the following 3 guiding principles to test if the LLM seems to have understood the specific biases of the test cases:

1. Identification of Biased Features - whether the LLMs output highlighted the features that were skewed on purpose in the test cases
2. Explanation of Impact - whether it properly identified how the features were skewed and offered viable solutions
3. Consistency of Output - whether the LLM regularly arrived to the correct conclusions

During the final evaluation of our experiment, we prompted Gemini five times for each of the five test cases that appear on our github repository. We did this to test principle 3: Consistency of Output. Furthermore, in order to make the LLM output more feature-specific while being human-readable, we asked Gemini to format its answers in json format, where it not only straightforwardly determined whether or not the dataset was biased, but also listed each of the features it determined was causing the bias and the model's reasoning about the causes and strategies of remediation.

The following example is from a Gemini response to test case 1: ascend-descend, where we specifically removed from the overall dataset any network data that had "descending" values in the delivery_rate, cwnd, and size parameters (as defined in section 4), so that the model could only train on data with "ascending" params:

Example output from Gemini for test case 1

```
{
  "biased_feature": "delivery_rate8",
  "causes": "The model's early
  reliance on `delivery_rate8`
  indicates an over-emphasis on the
  network conditions during the 8th
  transmission window, potentially
  skewed by specific congestion
  events or routing policies active
  during that time.",
  "strategy": "Resample the training
  data to ensure a more balanced
  distribution of `delivery_rate`
  values across different time slices.
  Introduce synthetic data points with
  diverse `delivery_rate` values to
  cover a wider range of network
  conditions."
},
{
  "biased_feature": "cwnd6",
  "causes": "Frequent use of `cwnd6`
  suggests a bias towards connection
  initiation phases or short-lived
  connections, failing to represent
  longer-running or diverse connection
  behaviors accurately.",
  "strategy": "Feature engineer `cwnd`
  delta (change) over time, moving
  averages, or standard deviations to
  capture congestion trends rather
  than instantaneous values. Also,
  introduce more data with varying
  congestion levels."
}
```

It is notable that in all of the five iterations testing on ascend-descend, the LLM identified the delivery_rate and cwnd features, but the explanations of their skewed-ness varied across answers. This remained true for the other four test cases: when Gemini was able to identify the key features, it would be able to do so with consistency. However, its reasoning about the features is not always clear or helpful even when it manages to correctly choose the biased ones.

For example, in test case 2, the model was able to identify the cwnd6, cwnd7, and cwnd8 features even though it consistently gave almost the same reasoning for all three features, which may not be helpful for understanding the specific problem of a long-tailed distribution. In cases 3, 4, and 5, Gemini was also able to determine the features that we had manipulated on purpose, but the "causes" that it gave were of mixed quality and especially for non-expert users may not provide enough information.

Furthermore, no matter which test case we ran the model on, it would always identify first that the trans_time7 feature was too heavily weighted. Although this was not intentional on our part, it may be a bias in the original dataset. This is an area where further research would be helpful to determine whether Gemini had some legitimate reason for concern, or if it was incorrect and there are inherent limits in how LLMs can reason about decision trees.

6 Conclusion

Making tools like Trustee more accessible to non-domain experts is a step towards the normalization of using machine learning models to detect complex patterns in network traffic and address network security problems in the real world. The complex ways that individual packets and other network traffic data are interconnected means that we cannot simply and easily modify its features without possibly introducing more biases, so having a tool like ours which can be run iteratively and check the validity of the dataset while providing clear, interpretable feedback is crucial. This iterative validation process helps ensure that any adjustments made to the data or model do not inadvertently amplify existing biases or create new ones. This will ultimately lead to more robust, trustworthy, and fair network security solutions that can be adopted confidently by users regardless of their machine learning expertise.

References

[1] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. 2022. AI/ML for Network Security: The Emperor has no Clothes. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560609

[2] NetReplica Github

[3] Roman Beltiukov, Wenbo Guo, Arpit Gupta, and Walter Willinger. 2023. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623075

[4] Create a Context Cache (Vertex AI)