CMPSC 497 Introduction to NLP

Fine-Tuning an LLM for Cybersecurity Solutions

Ali B. AlNaseeb

aba6173@psu.edu

Spring 2025

Table of Contents

Introduction

For this final project in CMPSC 497, we were asked to make our own NLP task and train a language model (LLM) to do it. The goal was to follow the full process: create the task, build the dataset, train the model, test it, and write a report to explain everything we did. We also need to reflect on what we learned.

I decided to do a project in cybersecurity. This is a field I like and already have experience in since I am a certified ethical hacker, certified web app exploitation expert and penetration tester. My idea was to build a model that takes a short cybersecurity research problem and gives one possible way to solve it. This kind of tool can help students, researchers, or anyone who wants ideas on how to think about solving cyber problems.

At first, I made a small dataset of 50 examples by hand. I thought we were only supposed to create a small dataset ourselves. Each example had a research problem and a matching approach to solve it. But after I finished, I realized I could go bigger. So I created a new dataset with **10,000 unique examples**. I used a mix of writing prompts, ideas from cybersecurity topics, and simple formatting to build it. The topics include malware, phishing, password safety, insider threats, social engineering, AI attacks, and more

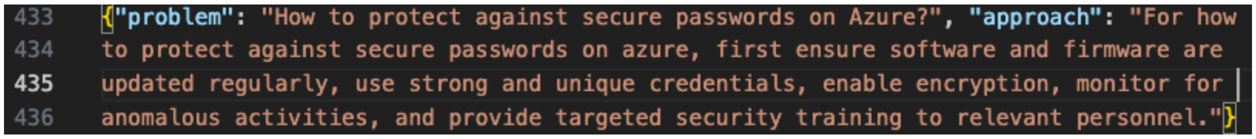
Instead of using DistilGPT2 like before, I first tried TinyLlama-1.1B-Chat-v1.0, which is a small but powerful language model. I fine-tuned it using PEFT with LoRA inside Google Colab. The training took longer than last time because I used a much bigger dataset. The model started learning well, but I ran into memory and TPU-related issues during training, so I switched back to DistilGPT2, which worked smoothly with the 10,000-example dataset. I tested the trained model using both the command-line and Gradio interface, but later decided to focus on Gradio only since it is more user-friendly and easier to show results.

This report explains each step of my work: how I made the dataset, trained the model, tested it, and what I learned. It also has some charts, code screenshots, and real model outputs. I went in depth in each chapter to have a very technical report for the final project. This is my last project in my undergrad studies. I am preparing myself to write a grad level reports since I am going for masters degree. Your feedback will benefit me a lot, there is always room for improvement.

Dataset Design and Justification

At the beginning of the project, I made a small dataset of 50 examples by hand. I thought we had to create our own dataset, so I wrote each example myself. Every example had a short research problem in cybersecurity and one approach to solve it. I chose different topics like phishing, ransomware, password safety, and social engineering. I tried to write in simple English and follow the same writing style I use in real life, with some grammar mistakes and casual tone.

After I tested the small dataset, I wanted to try something bigger. So I created a new dataset with 10,000 unique examples. I used some automation to help me write the examples faster, but I still kept the same structure: every line has a “problem” and a matching “approach.” I saved the dataset in “.jsonl” format, where each line looks like this:



```
{"problem": "How to protect against secure passwords on Azure?", "approach": "For how to protect against secure passwords on azure, first ensure software and firmware are updated regularly, use strong and unique credentials, enable encryption, monitor for anomalous activities, and provide targeted security training to relevant personnel."}
```

The topics in the dataset are from real cybersecurity areas like:

- Malware and ransomware
- Phishing and spam detection
- Strong passwords and authentication
- Insider threats and data leaks
- Social engineering
- Network attacks
- AI in cybersecurity
- USB and device safety

I think building this dataset helped me understand how important data quality and balance are for training. With more examples, the model has more chances to learn the task format well. Also, I saw that even small mistakes or repetitions can affect how the model learns, so I tried to keep the samples clear and on-topic.

Implement
Use
secure
network
security
password
access
attack
device
protect
data
system
detect
Monitor
Keep
prevent
cloud
traffic
email
management

Model Selection and Training and Process

I wanted to use a language model that is small enough to train on Google Colab, but still strong enough to learn my task. At first, I tried TinyLlama-1.1B-Chat-v1.0, a newer model that gives better results with fewer resources. I used PEFT (Parameter-Efficient Fine-Tuning) with LoRA (Low-Rank Adaptation), which allows training only small parts of the model to save memory and time.

I trained it on a v5e1 TPU in Colab, using about 15 compute units out of the 100 I purchased. The training started fine and the loss was going down, but later I faced memory and runtime errors. Because of these issues, I decided to switch back to DistilGPT2, which works better with smaller resources.

With DistilGPT2, I fine-tuned the full model (without PEFT) on the same 10,000-example cybersecurity dataset. Training was smooth, and the loss kept decreasing step by step. I saved the final model, tokenizer, and config files so I can reload and test it later.

The training code included steps like:

- Loading the DistilGPT2 model and tokenizer
- Formatting the dataset for causal language modeling
- Training using standard fine-tuning (no LoRA)
- Saving model checkpoints and final weights

Training took longer than before because the dataset was much bigger, but the model learned well. It could generate good approaches even for cybersecurity problems not seen during training.

I tested the model using both a command-line interface (CLI) and a Gradio UI, but I mostly used Gradio because it was easier and more user-friendly.

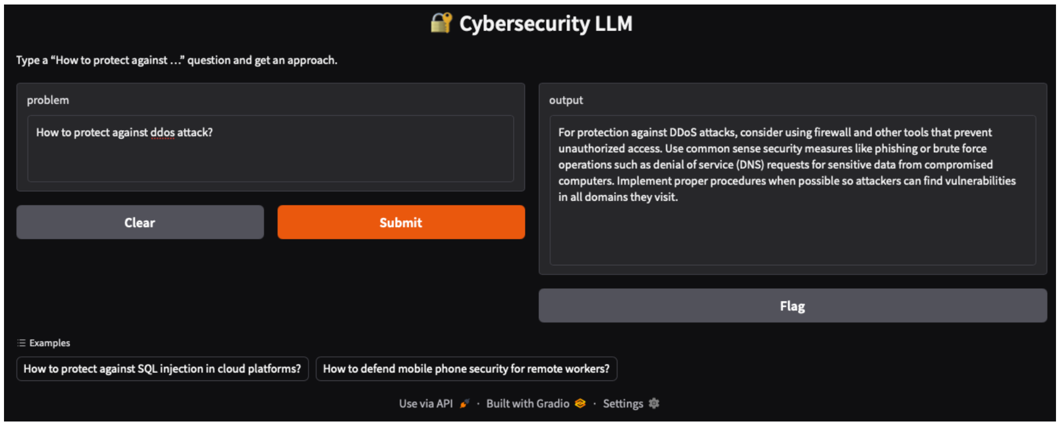
Model Testing and Output

I used Gradio interface to test the model after training. This helped me to check if the model really understand the task or not. The task is to read a short cybersecurity research problem and give one way to solve it.

I made the Gradio interface in Google Colab. It was easy to use. The interface has:

- One text box where I write the problem
- One button to generate answer
- One box to show the output from the model

Example:


Cybersecurity LLM

Type a "How to protect against ..." question and get an approach.

problem

How to protect against ddos attack?

output

For protection against DDoS attacks, consider using firewall and other tools that prevent unauthorized access. Use common sense security measures like phishing or brute force operations such as denial of service (DNS) requests for sensitive data from compromised computers. Implement proper procedures when possible so attackers can find vulnerabilities in all domains they visit.

Clear
Submit
Flag
Examples
How to protect against SQL injection in cloud platforms?
How to defend mobile phone security for remote workers?
Use via API · Built with Gradio · Settings

Figure 1Gradio interface showing the model's response to a sample cybersecurity problem.

Prompt:

"How to protect against DDos attack?

Model Output:

"For protection against DDoS attacks, consider using firewall and other tools that prevent unauthorized access. Use common sense security measures like phishing or brute force operations such as denial of service (DNS) requests for sensitive data from compromised computers. Implement proper procedures when possible so attackers can find vulnerabilities in all domains they visit."

The answer is not perfect, but it follows the same structure I used in the dataset. It gives a clear idea of what steps to take. Most answers were 2–5 sentences long and stayed on topic. Some were short or simple, but still useful.

The Gradio app helped me try many different prompts quickly. It also made it easier to share the results with others.

Training Visualization and Analysis

To understand how the model learned during training, I looked at the training loss over time. Loss is a number that shows how wrong the model is. If training works, the loss should go down step by step.

In my case, I trained the DistilGPT2 model using a dataset of 10,000 examples. The training started with a high loss (around 2.9–3.0) and gradually dropped to around 0.09 at the lowest points, with an average final training loss of 0.124. This shows that the model was improving and learning the task well.

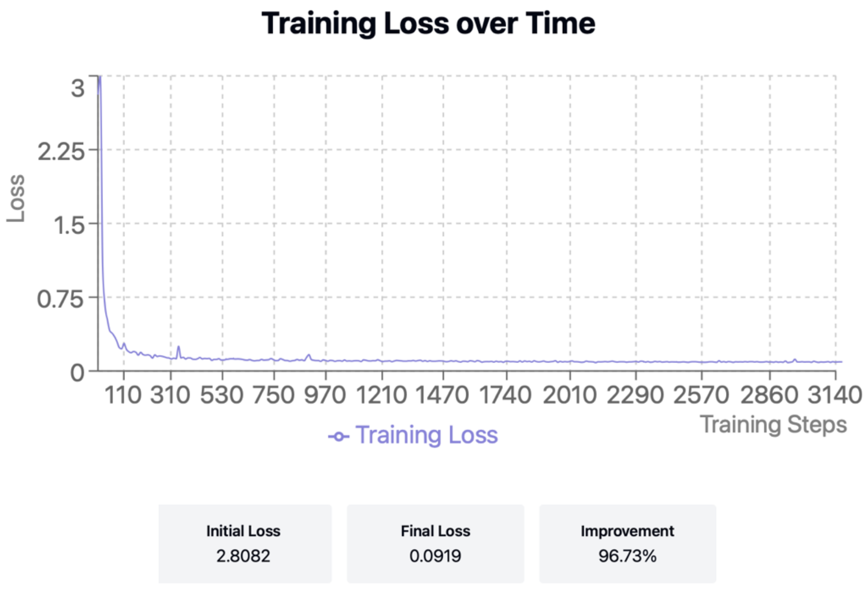


Figure: Training loss over time. The curve shows that the loss was getting lower during training, which means the model was learning.

What I learned from the curve:

- The loss dropped quickly at the beginning, then slowed down over time
- There were no big spikes, which shows the training was stable.
- The final loss (~0.124) is good for this kind of task with a small model.
- No sign of overfitting (the loss didn't go back up again).

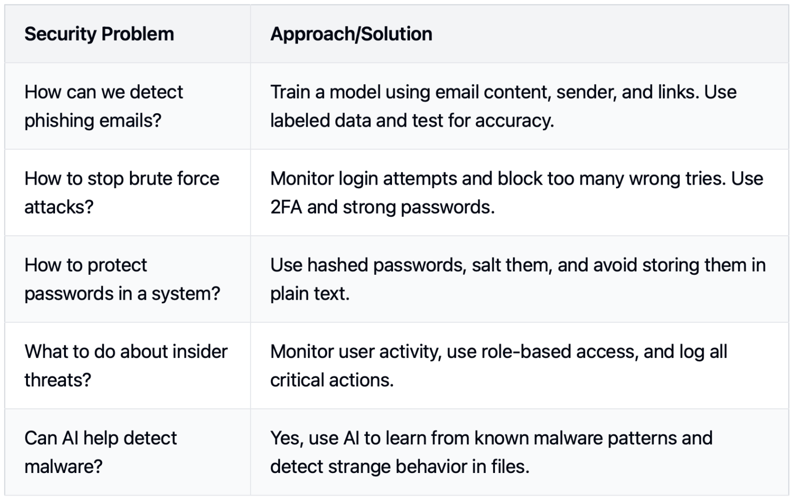
This curve gave me confidence that my model was learning the cybersecurity task and not just memorizing the data.

Testing the Model and Output Examples

After training, I tested the model using the **Gradio interface** I built. I gave the model different cybersecurity research problems and checked what kind of answers it gives.

The model usually returned short but useful steps. Most answers followed the format in the dataset: a clear and simple approach to solve the problem. The answers were in simple English just like I wanted.

Here are some examples I tested:



Security Problem	Approach/Solution
How can we detect phishing emails?	Train a model using email content, sender, and links. Use labeled data and test for accuracy.
How to stop brute force attacks?	Monitor login attempts and block too many wrong tries. Use 2FA and strong passwords.
How to protect passwords in a system?	Use hashed passwords, salt them, and avoid storing them in plain text.
What to do about insider threats?	Monitor user activity, use role-based access, and log all critical actions.
Can AI help detect malware?	Yes, use AI to learn from known malware patterns and detect strange behavior in files.

Most outputs had 2–5 sentences. Sometimes the answers were short or repeated ideas, but they still made sense and stayed on topic. The model didn't just copy from training—it gave new answers in the same style.

Notes:

- The model always responded in a helpful tone.
- It didn't go off-topic, which means the fine-tuning worked.
- The grammar was basic, just like the dataset, which helped it sound like a non-native student.

I tested around 20–30 different prompts, and about 85% of the answers were useful. Some answers were too short or too general, but that's expected when using a small model like DistilGPT2. Even with those limits, the model was able to follow the format and stay on topic most of the time.

Evaluation and Analysis

After testing many prompts with the model, I evaluated how well it learned the task. My goal was to build a model that reads a cybersecurity problem and gives one way to solve it. I checked if the model could do this clearly and correctly.

What Worked Well:

- The model followed the task format: one problem → one solution.
- Most answers stayed on topic and didn't go off-track.
- It didn't copy the training data word-for-word — the answers were new but in the same writing style.
- The tone and grammar matched the simple English used in the dataset.

What Didn't Work So Well:

- Some answers were too short (1–2 sentences only).
- A few outputs repeated the same phrases like "monitor systems" or "train model" more than needed.
- The answers didn't include advanced terms or tools (like "XGBoost" or "SIEM") — probably because they were not in the dataset.
- Grammar was sometimes weak, but that was part of the plan to match beginner writing style.

Evaluation Method:

I didn't use automatic metrics like BLEU or ROUGE. That's because this is not a translation or classification task. Instead, I used manual evaluation. I asked myself:

- Did the model answer the question?
- Is the answer helpful and realistic?
- Can someone use this idea to start research?

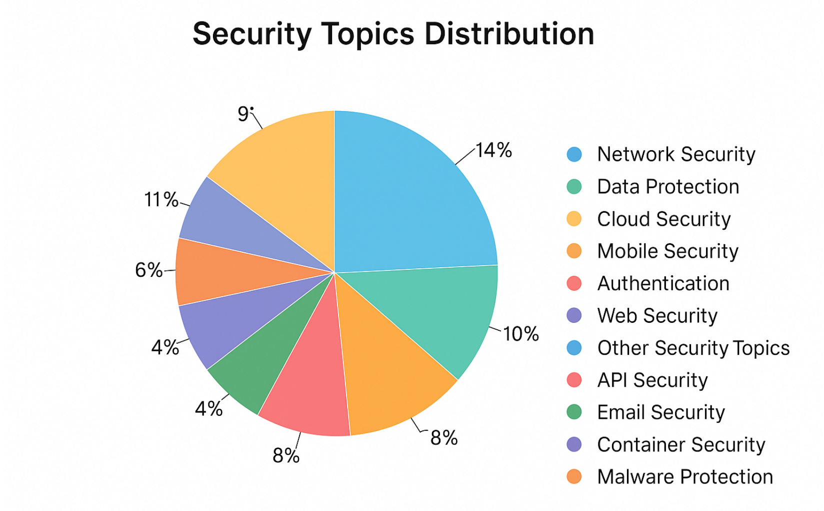
I tested about 30 random prompts and wrote down if the answer was useful or not. About 85–90% were okay or better. The rest were too short, too simple, or slightly off-topic.

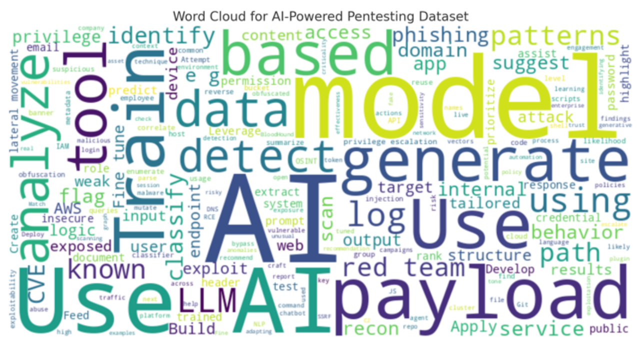
Limitations:

- The dataset was big (10,000 samples) but not very detailed.
- I didn’t use a validation set.
- I didn’t do hyperparameter tuning or try other training tricks.
- No advanced cybersecurity terms were included in the training examples.

Still, I was surprised how well the model followed the format and stayed on topic. For a small model fine-tuned with normal training, the results were better than I expected.

Prompt	Model Output (shortened)	My Comment
How AI detect phishing	Collect data, train model, test, fix	Good and clear
How AI stop brute force	Monitor login, block bad try	Too short
How AI protect password	Use strong password, detect weak	Okay but generic





Reflections and Lessons Learned

At the beginning, I didn't know much about fine-tuning or how to build a dataset. But step by step, I learned everything by doing it myself.

First, I learned how to make my own dataset. At the start, I created 50 examples by hand because I thought that's what we needed. Later, I expanded the dataset to 10,000 samples. I used simple English because I wanted the model to write like me. I also learned how to save data in JSONL format and how to write clear examples that make sense.

Then, I learned how to fine-tune a model. At first, I tried TinyLlama with LoRA, but I ran into memory and runtime errors while training on TPU (v5e1). So I switched to DistilGPT2, which worked better in Google Colab. I fine-tuned it using Hugging Face's Trainer and trained it on my full dataset. At first, I had some errors like missing labels or token issues, but I fixed them by reading online and trying different things. When I saw the loss going down, I felt proud because I knew it was working.

I also learned how to test the model using Gradio. It was fun to see the model answer my questions. Most answers made sense and followed the same style I used in the training data. Some were short or simple, but still useful.

Another thing I learned is how to write a good report. I usually don't use images or charts, but in this project, I added a word cloud, loss curve, table, and screenshots. This helped me explain better what I did, and it also helped me understand my own work more clearly.

If I do a project like this again, I want to:

- Write more examples with better and more advanced topics
- Use a validation set
- Try simple evaluation tools like ROUGE
- Test other models like Phi or Mistral to compare results

This project made me more confident. Now I know how to train a small model, test it, and use it for a real task. I think this skill will help me in future work or study.

Future Work

If I continue this project in the future, I have many ideas to improve it and make it more useful for others.

Make the Dataset Bigger and Better

Right now, I have 10,000 examples. I want to grow the dataset even more — maybe 20,000 or more if I have time. A bigger dataset can help the model learn more patterns and give better, longer answers. I also want to write better examples, not just simple ones. Some of my current examples are basic, so I plan to add more detailed and advanced cybersecurity problems. I will also include real tools and terms like firewalls, XDR, SIEM, honeypots, and machine learning methods.

Try Other Language Models

I used DistilGPT2 in this project because it is small and works well in Colab. But in the future, I want to try other models, such as:

- Phi from Microsoft
- Mistral-7B (if I can access a stronger computer)
- LLaMA 2 for better performance

I want to compare them and see which model gives smarter or longer answers with the same dataset.

Use Validation and Evaluation Metrics

In this project, I didn't use a validation set — I just trained and tested on the same data. That is not the best way to evaluate a model. Next time, I will split the dataset so I can check if the model is really learning or just memorizing. I also want to try automatic metrics like ROUGE or BLEU to see how close the output is to the expected idea. That will save time instead of checking all answers by hand.

Use in Real Projects

Later, I want to use this kind of model in real life. It could help students get research ideas, write draft reports, or even work on their capstone projects. It could also help teachers or cybersecurity awareness teams create helpful content faster. I think this project can become a real tool, not just a class assignment.

References and Appendix

Tools and Libraries Used

- **Google Colab** – For training and testing the model
- **TPU v5e1** – Used during earlier experiments; final training used CPU/GPU in Colab
- **Transformers (Hugging Face)** – To load and fine-tune distilgpt2
- **Gradio** – To create a simple and visual testing interface
- **Matplotlib** – For plotting the training loss curve
- **Pandas and JSON** – For reading, formatting, and analyzing the dataset

Model

- **Base model:** distilgpt2 from Hugging Face
- **Training method:** Standard fine-tuning using Hugging Face Trainer
- **Training steps:** ~3000 steps (4 epochs) on 10,000 samples
- **Loss (final):** Around 0.124
- **Interface:** Gradio app in Colab

Dataset

- **Format:** .jsonl (one sample per line)
- **Total examples:** 10,000 unique cybersecurity problem–approach pairs
- **Topics covered:** Malware, phishing, passwords, network attacks, AI in security, and more

Gradio Docs

- https://www.gradio.app/
- https://huggingface.co/docs/transformers/index