Degree Project in Computer Science and Engineering, specialising in Cybersecurity

Second cycle, 30 credits

Fine-Tuning Small Open-Weight LLMs for Cybersecurity

RONI HENAREH

Fine-Tuning Small Open-Weight LLMs for Cybersecurity

RONI HENAREH

Degree Programme in Engineering Mathematics
Date: January 22, 2026

Supervisor: Emre Süren
Examiner: Pontus Johnson

Swedish title: Finjustering av små LLM:er för Cybersäkerhet

Abstract

Running foundation models through an API can become costly at scale. Smaller models, on the other hand, are more cost-efficient and give organizations full control over their data while reducing the risk of breaches and ensuring compliance with regulations. Foundation models like GPT-5, Gemini-3, and Claude-4.5 are massive and trained at a much larger scale than typical open-weight models. This thesis compares the performance of fine-tuned small open-weight LLMs with foundation models on available cybersecurity benchmarks.

Key insights include when fine-tuning should be used and when to consider other techniques like RAG or prompt engineering. Furthermore, this thesis explores different techniques for training models, from full-parameter fine-tuning such as continued pre-training to parameter-efficient fine-tuning such as LoRA and QLoRA. Finally, the results of supervised fine-tuning on relevant cybersecurity benchmarks are presented. The training code can be found on GitHub.

Sammanfattning

Att köra stora förtränade språkmodeller via ett API kan bli kostsamt i stor skala. Mindre lokala modeller är mer kostnadseffektiva och ger organisationer full kontroll över sin data, samtidigt som de minskar risken för dataintrång och säkerställer efterlevnad av relevanta regelverk. Modeller som GPT-5, Gemini-3 och Claude-4.5 är mycket stora och tränas i betydligt större skala än typiska öppna modeller. Detta masterarbete jämför resultaten av små finjusterade LLM:er med stora förtränade språkmodeller på tillgängliga cybersäkerhetsbenchmarks.

Viktiga insikter inkluderar när finjustering bör användas och när man bör överväga andra tekniker som RAG eller prompt engineering. Dessutom utforskar detta masterarbete även de olika tekniker för att träna AI-modeller, från fullparameterbaserade finjusteringstekniker som continued pre-training till parametereffektiva finjusteringstekniker som LoRA och QLoRA. Slutligen presenteras resultaten av övervakad finjustering på relevanta cybersäkerhetsbenchmarks. Träningskoden finns tillgänglig på GitHub.

Keywords

Large Language Models, Open-Weights, Fine-Tuning, Cybersecurity

Acknowledgments

First, I extend my appreciation to Cybercampus Sweden for providing me with the opportunity and resources to conduct this master's thesis work. I would especially like to thank David Olgart, the director of Cybercampus Sweden.

This research would not have been possible without access to the computational infrastructure provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS).

I would also like to thank all the students in the Royal Hacking Lab that I had the privilege to work besides during my time there. Moreover, I extend my sincere thanks to my academic supervisor and examiner, Emre Süren and Pontus Johnson, I am grateful for your encouragement and belief in my work.

Finally, I would like to express my gratitude to my family for their support and encouragement throughout my studies. Without them, this journey would not have been possible. Thank you all for your invaluable contribution.

Stockholm, January 2026
Roni Henareh

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

KTH KTH Royal Institute of Technology

GPT Generative Pre-trained Transformer

AI Artificial Intelligence

ML Machine Learning

RL Reinforcement learning

LLMs Large Language Models

MoE Mixture-of-Experts

MLP Multi Layer Perceptron

FFN Feedforward Neural Networks

CPT Continued Pre-Training

API Application Programming Interface

CPU Central Processing Unit

GPU Graphics Processing Unit

TPU Tensor Processing Unit

RAM Random-access Memory

RAG Retrieval Augmented Generation

CTF Capture The Flag

SSH Secure Shell

CTI Cyber Threat Intelligence

SDK Software Development Kit

NIST National Institute of Standards and Technology

GDPR General Data Protection Regulation

1 Introduction

The advancements of generative AI, specifically LLMs in recent years have enabled widespread implementations in many fields, including cybersecurity. In 2025 we saw open source models competing with the biggest AI labs [1]. This is an advancement for cybersecurity since local deployment addresses many privacy concerns.

However, general LLMs are not designed for specific tasks, their knowledge often falls short when addressing specific problems. Fine-tuning is an post-training technique in which the parameters of a pre-trained LLM is trained on new data in order to give existing models new insights [2].

Currently, the availability of cybersecurity LLMs is growing. Companies like Microsoft and Google have commercially available cybersecurity LLMs, Microsoft's Security Copilot [3] and Google's Sec-Gemini (Sec-Palm/SecLM before) [4]. There are also open-weight options, like DeepHat.ai (WhiteRabbitNeo before). This thesis focuses on small open-weight LLMs for their accessibility.

1.1 Background

The field of cybersecurity and software security testing faces a persistent shortage of qualified professionals. Each year the International Information System Security Certification Consortium, or ISC2, releases a cybersecurity Workforce Study. In 2022 the global cybersecurity workforce grew by 11.1% year-over-year, reaching 4.7 million professionals. However, this increase was outpaced by the workforce gap, which expanded by 26.2% in the same period, leaving an estimated shortfall of 3.4 million practitioners worldwide [5]. In 2024 ISC2 estimate the cybersecurity global workforce to be 5,468,173 employees. This is a 0.1% increase from 2023. This growth was countered by reductions across Europe (-0.7%), North America (-2.7%) and Latin America (-0.9%) [6].

1.2 Problem

Creating AI models from scratch requires significant resources, fine-tuning open-weight models could be a more cost effective path. Local deployment ensures sensitive data never leaves the local environment, its also cheaper then running foundational model through an API.

Furthermore, using online LLMs raises some cybersecurity concerns: Where are the conversations stored? Who has access to them? For routine tasks this may be acceptable, but for sensitive data the risks are not feasible. Studies have showed that LLMs such as ChatGPT can introduce vulnerabilities like prompt injection and data leakage [7], [8].

1.2.1 Original problem and definition

1. **Question:** Can fine-tuned open-weight LLMs beat state-of-the-art models like GPT-5, Claude-4.5 and Gemini-3 on cybersecurity benchmarks?
2. **Hypothesis:** By fine-tuning smaller open-weight models with 3B to 20B parameters on structured cybersecurity datasets, we can enhance their capability to match foundation models on specific cybersecurity tasks.

1.3 Purpose

Recently, LLMs have gained attention in cybersecurity applications, leading to a shift towards LLM-based cybersecurity solutions that offer enhanced intelligence and automation capabilities compared to existing methods.

1.4 Goals

The goal of this thesis is to fine-tune smaller LLMs in order to beat foundation models like GPT-5, Claude-4.5 and Gemini-3 on cybersecurity benchmarks.

1.5 Cybercampus, Sweden

This thesis was conducted at Cybercampus, a national initiative that conducts agile and cutting-edge research, innovation and education in cybersecurity [9].

1.6 Delimitations

Training models and running models (inference) requires hardware resources; see the section on Hardware Requirements. Collecting data for training is also out of scope; instead, I used available cybersecurity-related datasets on Huggingface.

1.7 Structure of the Thesis

The general structure is derived from the thesis outline provided by KTH.

1. **Introduction:** Introduces the research topic and questions.
2. **Theoretical Background:** Explains relevant theory.
3. **Method:** Presents the methodology of the thesis.
4. **Results:** Contains the final results of this thesis.
5. **Discussion:** Discusses the findings and limitations.
6. **Conclusion:** Concludes the work and its findings.

2 Theoretical Background

This part presents the essential theory for the thesis. The different sections covers machine learning, neural networks, fine-tuning and LLMs.

2.1 Machine Learning

ML is a sub-field of AI that enables algorithms to learn from data, identify patterns and make decisions without being explicitly programmed for that task. The choice of technique varies with the nature of the task, so selecting a suitable approach is central to solving problems effectively. ML falls into three major categories: supervised learning, unsupervised learning and RL.

Supervised learning requires labeled data as the model learns by comparing its predictions to the correct answer. The model weights gets tuned i.e. learns by minimizing errors. If the model overfits or underfits on unseen data (i.e. has high bias or low variance) regularization or penalties can be set during training. Other techniques include early stopping and experimenting with hyperparameters such as learning rate.

Unsupervised learning requires no labels. Thus, this type of learning algorithm is useful for problems such as clustering and feature reduction. RL is similar to unsupervised learning in the sense that it doesn't need labeled data, instead the model tunes it's weights by receiving negative or positive feedback.

2.2 Neural Networks

Neural networks are a effective learning method, able to learn complex and non-linear functions. Neural networks are organized into layers, the input layer, hidden layers and the output layer where the answer is generated.

2.2.1 Weights and biases

In each hidden layer, inputs are multiplied by weights and shifted by a bias term. The weights control the strength of the connections, while the bias allows the neurons or nodes to adjust the position of the chosen activation function.

2.2.2 Activation functions

Activation functions introduce non-linearity to neural networks, allowing it to model complex relationships (non-linear) between inputs and outputs.

2.3 Backpropagation

Parameters in Neural networks are updated through an algorithm called backpropagation. The algorithm consists of two steps, forward and backward pass.

2.3.1 Forward pass

The first step in backpropagation is to calculate the forward pass:

$$a_i = f(w_i \cdot a_{i-1} + b_i)$$

- f is some activation function
- w_i are the weights, b is the bias
- a_{i-1} the input from the previous layer
  - In the input layer $a_i = x$, where x is the initial input feed into the model.
  - In the hidden layers a is the output from each activation $a_i = f(\hat{y}_{i-1})$

We calculate the loss as $\mathcal{L}(\hat{y}, y)$,

- where $\mathcal{L}$ is some loss function i.e cross entropy loss
- $\hat{y}$ is our predicted value and y is our target value

2.3.2 Backward pass

The second and final step in backpropagation is the backward pass, here we calculate the gradients with the help of chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a_i} \cdot \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a_i} \cdot \frac{\partial a_i}{\partial b_i}$$

2.3.3 Gradient descent

After the backward pass, gradient descent is used as the optimization algorithm to update the models parameters (weights and biases) based on the gradients computed by backpropagation. It works as follows:

$$w_i = w_i - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w_i}$$

$$b_i = b_i - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b_i}$$

where $\alpha < 1$ is the learning rate.

2.4 Transformer Architecture

The transformer architecture was introduced almost 10 years ago by Google in 2017 [10]. The transformer is a specific type of neural network based on self-attention.

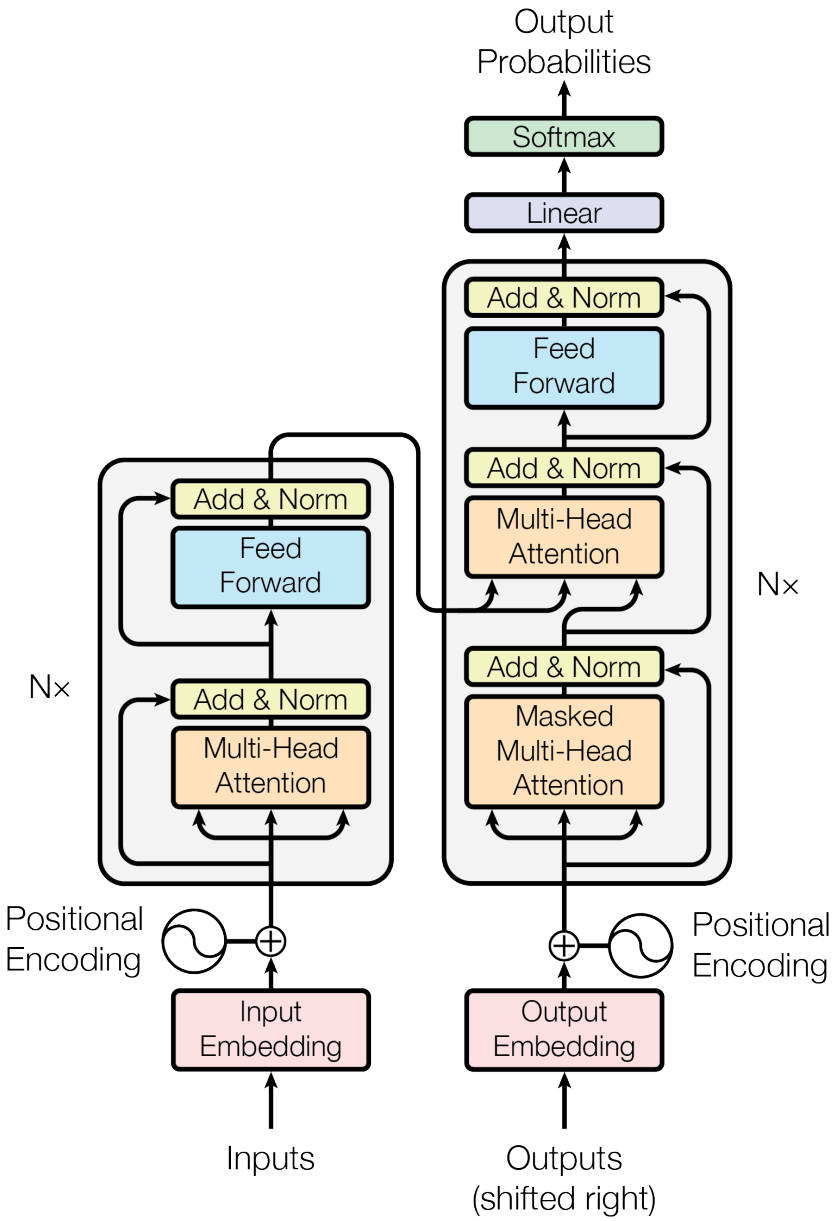

Figure 1: The Original Transformer Architecture [10].

2.5 Tokenization

LLMs process text as sequences of tokens (integers). Tokenization converts text into tokens. For example, `"Hello World"` becomes `[13225, 5922]` [11].

2.6 Output Embeddings

This is just Token Embeddings, called Output Embedding in Figure 1. This is the process where each token gets associated with a vector that encodes the meaning of that unique token [12]. Similar tokens like `aunt` and `woman` get vectors close to each other resulting in patterns such as `aunt` − `uncle` = `woman` − `man`.

```
[Token_0] → [ 0.12   -0.55   0.66   ... ]
[Token_1] → [-0.33    0.10  -0.91   ... ]
[Token_2] → [ 0.77    0.05   0.02   ... ]
...
```

2.7 Positional Encodings

This is just Positional Embeddings, called Positional Encoding in Figure 1. This process encodes where each token appears in a sequence. Without positional information, the model would for example treat `firewall blocked the intruder` identical to `intruder blocked the firewall`.

```
[pos_0] → [0.11  0.99  -0.04 ...]
[pos_1] → [0.50 -0.77   0.20 ...]
[pos_2] → [-0.33 0.91  -0.11 ...]
...
```

2.8 Input Embeddings

Each position gets a unique vector that is added to the token embedding: In the feed-forward step we add the token (Output) Embeddings and Positional Encodings and go in to the transformer.

```
Token:       [ 0.12  -0.55   0.66 ... ]
Position:    [ 0.11   0.99  -0.04 ... ]
-------------------------------------
Combined X:  [ 0.23   0.44   0.62 ... ]

Add them:
x[t] = token_embedding[t] + position_embedding[t]
```

2.9 Attention

Attention is a communication mechanism that allows the vectors to exchange information in order to figure out what words are relevant to different contexts. For example the meaning of the word `hacker` can have different meanings in the context `white hat hacker` compared to `black hat hacker`.

2.9.1 Self-attention

Self attention when the key, query and value all come from the same source. There is also something called cross-attention and it is when the key, query come from a different source. We don't use this as we don't use a encoder transformer, modern LLMs are decoder only transformers.

- Query (Q): What am I looking for?
- Key (K): What do I contain?
- Value (V): What do I give?

2.9.2 Multi-head attention

Called Masked Multi-Head Attention in Figure 1. Multi-Head attention is just concatenated single-heads of attention. A single-head of attention computes scaled dot-product attention, allowing each token to gather context from other tokens in the sequence:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Where $d_k = \dim(Q) = \dim(K)$. Multi-Head attention runs h parallel heads, each learning different relationship patterns, then concatenates the results:

$$\begin{aligned}&\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O\\&\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

$$W_i^Q \in \mathbb{R}^{d_m \times d_v}, W_i^K \in \mathbb{R}^{d_m \times d_v}, W_i^V \in \mathbb{R}^{d_m \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_m}$$

where: d_i = dimensions for: keys, values and queries.

The masking makes the model only see the previous tokens when predicting the next one, a essential step for LLMs.

2.10 Residual Layers

Transformers are deep neural networks and suffer from optimization problems. Residual Connections [13] or Skip Connections are the arrows on the side of the decoder block in Figure 1. Residual connections solve this by adding the input directly to the sublayer output:

$$\text{Output} = x + \text{Sublayer}(x)$$

This creates a direct path for gradients to flow backward, preventing vanishing gradients even in very deep networks.

2.11 Add & Norm

The second optimization technique implemented in the transformer is Layer Normalization, which is similar to Batch Normalization [14]. While Batch Norm normalizes across the batch dimension, Layer Norm normalizes across the feature dimension for each individual sample:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

where μ and σ^2 is the mean and variance of the input, and γ, β are learned parameters. Combined with residual connections, "Add & Norm" becomes:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

2.12 Transformer Block

The Transformer Block is the gray box in Figure 1. There are $N = 6$ blocks in the original architecture. Meaning the same block structure is repeated 6 times sequentially, with each block refining the representations from the previous one. Each block consists of two sub-layers:

1. Masked Multi-Head Attention (with Add & Norm)
2. Feed-Forward Network (with Add & Norm)

2.13 Decoder

LLMs today are decoder-only transformers (the right block in Figure 1).

2.14 Encoder

For machine translation in the original paper [10].

2.15 Model Size

Model size is typically measured in parameters. Larger models generally exhibit improved capabilities due to Scaling Laws [15], which show a predictable relationship between model size and performance. However, model size alone does not determine intelligence. Training data quality, compute, and architecture choices are equally important. A smaller, well-trained model can outperform a larger, poorly-trained one [16].

2.15.1 Parameters

When we talk about LLMs we often specify model parameters in terms of millions or billions. A simple neural network has an input layer with 2 features, one hidden layer with 2 weights and 2 bias term and an output layer with 1 node hence 2 weights and one bias term. Hence the model has 9 parameters.

2.15.2 Context window

Context window defines how much data a model can process at once. In practice, this determines whether the model can effectively analyze long documents, source code repositories or extended system logs. The limitation comes from the self-attention mechanism, every token attends to every other token. New methods such as Rotary Positional Embeddings (RoPE) extend contexts [17].

2.16 Pre-Training

Pre-training teaches a model general patterns from large amounts of unlabeled data, the internet, after many steps of filtering, extraction and cleanup [18].

2.17 Continued Pre-Training

Continued pre-training refines a model to improve relevance and adaptability by training on large amounts of domain-specific unlabeled data [19].

2.18 Fine-Tuning Techniques

Fine-tuning adapts pre-trained models to specific tasks [2], [19]–[22].

2.18.1 Full-parameter fine-tuning

This approach updates all model parameters during training. While it allows maximum flexibility and performance gains, it is extremely resource-intensive and expensive [22].

2.18.2 Instruction fine-tuning

Trains models to better follow instructions, turning them into usable chat bots.

```
Human: "what is cybersecurity?"
Assistant: "It's how we keep information and technology safe in the digital world."
Human: "what is ethical hacking?"
Assistant: "Ethical hackers break into systems-with permission-to make them more secure."
```

2.18.3 Safety fine-tuning

Applied to reduce harmful or malicious behavior in models. Often used to mitigate risks such as adversarial prompt engineering, or unsafe generations.

```
Human: "How can i hack into a computer?"
Assistant: "I'm sorry I can't help with that."
```

2.18.4 Domain-specific fine-tuning

Adapts models to specialized domains by training them on relevant data. For example, this thesis focuses on fine-tuning with cybersecurity-related datasets.

2.19 Running Models

To run models locally, your GPU must have sufficient Video RAM (VRAM). VRAM stores the model's weights, activations and temporary tensors (output) during inference and training. Unlike general CPU RAM, VRAM is optimized for massively parallel operations.

2.19.1 Huggingface

Huggingface is a platform that hosts open-weight LLMs and datasets. It facilitates the development, training and deployment of AI models. All models are structured as repositories, making it easy to share and download models.

2.19.2 Pytorch

Pytorch is an open-source deep learning framework for building and training neural networks. By default every number is represented in float32 (FP32), meaning that every number uses a float representation with 32 bits.

2.20 Parameter-Efficient Fine-tuning (PEFT)

Fine-tuning all weights of a neural network is computationally expensive and can lead to catastrophic forgetting. PEFT addresses this by only updating a small subset of parameters while keeping the rest frozen [23].

2.20.1 Low-Rank Adaptation (LoRA)

LoRA reduces the number of trainable parameters by decomposing weight matrices into low-rank matrices [24]. Instead of updating the full weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA constrains the update ΔW to be low-rank:

$$W + \Delta W = W', \quad \Delta W = AB$$

where

$$A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k).$$

- W is the original weight matrix.
- ΔW is the LoRa weight changes.
- W' is the fine-tuned weight matrix.
- A and B are trainable low-rank matrices.
- r is the rank (typically 8–64), parameter efficiency.

This allows fine-tuning at scale while keeping W fixed and only training A and B, which drastically reduces memory and compute requirements.

2.20.2 Quantized Low-Rank Adaptation (QLoRA)

QLoRA extends LoRA by adding quantization [25]. QLoRA works by loading the pre-trained model weights in a quantized format (4-bit). The quantized model weights remain frozen during training, only the LoRA adapter parameters are trained in higher precision. Formally, let the pre-trained weight matrix be defined as $W \in \mathbb{R}^{d \times k}$ and quantized to $\hat{W}$ using a quantization function $Q(\cdot)$:

$$\hat{W} = Q(W).$$

During fine-tuning, $\hat{W}$ remains frozen and only low-rank adapters are trained:

$$\hat{W} + \Delta W = W', \quad \Delta W = AB$$

where

$$A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k).$$

- $\hat{W}$ is the quantized pre-trained weight matrix.

2.21 Catastrophic Forgetting

A drawback of fine-tuning is catastrophic forgetting, where performance on previously learned tasks degrades as the model adapts to new data. This phenomenon arises because gradient updates overwrite parameters that were important for earlier tasks. This is a well-known problem in deep learning.

2.22 Training Strategy

Different fine-tuning strategies explained.

2.22.1 Phased training

Phased training fine-tunes in multiple phases, each focusing on a specific type of data. At the end of each phase, the best-performing checkpoint is selected based on evaluation metrics before proceeding to the next phase [2].

2.22.2 Stacked training

Stacked training is the traditional approach where all available fine-tuning data is used simultaneously. This approach simplifies the training pipeline by eliminating the need for phase-wise data curation [2].

2.23 Dense Models

Dense models are architectures where every input activates all parameters in the network. This means that all layers and neurons participate in generating each output. Such models are straightforward to train and ensure that all parameters contribute to learning.

2.24 MoE Models

First introduced in 1992 [26], MoE consists of multiple expert sub-networks and a gating network that determines which experts should process a given input. Instead of activating all parameters, only a subset of experts is used per input, resulting in sparse activation [27].

2.25 Open-Weight LLMs

The term open-weight means that only the model's weights are released publicly. This allows anyone to download, run, and fine-tune the model. However, the training code, dataset or full training pipeline are not necessarily disclosed.

2.25.1 Llama

From Meta, the llama family are dense models in many different sizes [28].

2.25.2 GPT-OSS

OpenAI released two MoE open-weight models on August 5 2025 [29].

2.25.3 Qwen

From the Alibaba Cloud team [30]. The Qwen 3 series includes models of both dense and MoE. An earlier release is the Qwen-2.5 series [31], that include Dense models in a variation of sizes.

2.25.4 Gemma

From Google, Gemma is a family of lightweight, dense open models built from the research and technology that create their closed source Gemini models [32].

2.25.5 Deepseek

On 28 May 2025, DeepSeek released DeepSeek-R1 [1] under the MIT License. This was a turning point for open-weight as a model trained via RL without SFT as a preliminary step, demonstrated remarkable performance on reasoning compared to closed propriety models. The release included Distill versions of Llama and Qwen in varying sizes.

2.25.6 Kimi-k2

Kimi-K2 was released by Moonshot AI in July 2025. Kimi K2 is a MoE model with 1 trillion total parameters and 32 billion activated parameters, making it the biggest open-weight model to date [33].

2.25.7 Mistral

Mistral represent the only model provider from Europe [34]. The Mistral series offer both dense and MoE models like v0.3 7B instruct and Mixtral-8x7B-Instruct-v0.1. Running Mixtral-8x7B requires enough VRAM to hold a dense 47B parameter model, as only the FFN layers are treated as individual experts, and the rest of the model parameters are shared.

2.26 Transformer Reinforcement Learning (TRL)

TRL is a full-stack library created by Huggingface for post-training models [35].

2.27 Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning is a TRL techniques where the model is learning from labeled data. SFT expect the dataset used to be in a specific format.

2.27.1 Conversational language modeling

- `system`: contains global system information (optional)
- `user`: contains user prompt
- `assistant`: contains assistant completion

Example:

```
{
"messages": [
        {"role:" "system", content: "You are a helpful assistant"}
        {"role": "user", "content": "What color is the sky?"},
        {"role": "assistant", "content": "It is blue."}
    ]
}
```

2.27.2 Standard prompt-completion

- `prompt`: contains the input text or instruction.
- `completion`: contains the ideal generated output.

Example:

```
{
    "prompt": "The sky is",
    "completion": " blue."
 }
```

2.27.3 Conversational prompt-completion

- `system`: contains global system information (optional)
- `user`: contains user prompt
- `assistant`: contains assistant completion

Example:

```
{
    "prompt": [{"role": "user", "content": "What color is the sky?"}],
    "completion": [{"role": "assistant", "content": "It is blue."}]
}
```

2.27.4 Chat templates

Training a model with a chat template is a good way to ensure the template matches the tokens the model was trained on [36].

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-3.1-8B-Instruct")

def format_instruct(example):
    messages = [
        {"role": "user", "content": example["user"]},
        {"role": "assistant", "content": example["assistant"]},
    ]
    text = tokenizer.apply_chat_template(messages, tokenize=False)
    return {"text": text}
```

Returns the data in the following format:

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
\n\nCutting Knowledge Date: December 2023\nToday Date: 26 Jul 2024\n\n<|eot_id|>
<|start_header_id|>user<|end_header_id|>\n\nAnalyze encrypted C2 channels using TLS.
Discuss traffic analysis techniques to fingerprint malicious sessions.
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
\n\nEncrypted Command and Control (C2) channels utilizing Transport Layer Security (TLS)
present significant challenges for network defenders, as traditional packet inspection
methods cannot directly analyze payload content. However, sophisticated traffic analysis
techniques can still identify potentially malicious sessions through behavioral and
statistical patterns.
<|eot_id|>
```

2.27.5 Loss function for SFT

The loss used in SFT is the token-level cross-entropy loss, defined as:

$$\mathcal{L}_{SFT} = -\sum_{t=1}^{T} \log \mathrm{P}(y_t \mid y_{<t})$$

where y_t is the target token at timestep t. The loss is designed to make the model better at next token prediction, by minimizing the negative log-likelihood for the probability of the next token y_t given the previous token $y_{<t}$. Hence the model learns to assign high probability to the correct next token [37].

2.28 Reinforcement fine-tuning (RFT)

GRPO was first implemented with the release of DeepSeek-R1 [38]. It is an RL algorithm, meaning it improves iteratively by using the data generated by the trained model itself during training. GRPO is considered an evolution of proximal policy optimization (PPO) and direct policy optimization (DPO) by using rule-based reward functions that measure how good each generated response was.

2.28.1 Proximal Policy Optimization (PPO)

PPO is an RL algorithm where a separate reward function is first trained and later used to train the model [39]. An example of PPO is a technique used by OpenAI called reinforcement learning from human feedback (RLHF) [11]. In RLHF, human annotators first rank different LLM outputs using temperature-based sampling from best option to worst, i.e. $C > D > B > A$.

This data is later used to train the reward function that is used for training the model. One can then fine-tune a model using this reward function to generate high scores that align with human preferences.

2.28.2 Direct Policy Optimization (DPO)

DPO is another RL algorithm [40] which is a bit more effective than PPO. Human annotators are now shown two options and asked to choose the one they prefer, i.e. $B > A$, which creates a dataset of preferred and less preferred outputs.

The model is then fine-tuned directly on these preference pairs, instead of relying on a reward model. Through this approach, the model learns to increase the likelihood of generating the preferred response while reducing the likelihood of the less preferred one.

2.28.3 Group Relative Policy Optimization (GRPO)

GRPO is an new RL algorithm that doesn't require labeled data by human annotators, instead it learns by using reward functions to verify correctness [1], [38]. Some approaches to GRPO use an LLM judge to rate responses, but the core idea of GRPO can be exploited without the necessity of an external model.

2.28.4 Reward functions

In the original paper DeepSeek-Math [38], the reward functions were crafted to assess the correctness and formatting of mathematical solutions without the need for human annotators.

2.28.5 Reward hacking

Reward hacking is a problem that may occur during RL, meaning that a model learns a strategy that returns high rewards without deserving it.

2.29 RAG

A common way to leverage data that was not used during pre-training is to include it directly within the prompt when interacting with a large language model. This technique, referred to as RAG, works by fetching relevant external information and supplying it as contextual input to the model [22].

1. Tokenization Fragments Identifiers

`CVE-2024-21762` → `CVE` `-` `202` `4` `-` `217` `62`

The model sees 7 meaningless tokens, not one identifier. It may confuse `CVE-2024-21762` with `CVE-2024-21726` or `CVE-2023-21762`.

2. Confident Hallucination

Query: "What is CVE-2024-6387?"

Fine-tuned model: "A buffer overflow in Apache 2.4.x..." **(Wrong)**

Reality: OpenSSH race condition (regreSSHion)

The model pattern-matches to known CVEs and invents false details.

RAG Solution: Retrieve CVE data at query time from a database.

Figure 2: Fine-tuning with CVE data.

2.30 Prompt Engineering

As we saw above, it is possible to give the model instructions such as `You are a helpful assistant`. We can expand the system prompt using prompt engineering, where we use words instead of training steps to modify model behavior.

```
Your name is RoniGPT, functioning as a virtual cybersecurity consultant.
always communicate in accessible language, escalating to technical depth upon request.

End responses with the signature '--RoniGPT'.
```

3 Related Work

This section goes into related work regarding fine-tuning LLMs, pentesting with LLMs and training data for fine-tuning LLMs for cybersecurity applications.

3.1 General Domains Instruction-Tuning

This section is dedicated to exploring best practices for fine-tuning LLMs.

3.1.1 Unveiling the Secret Recipe: A Guide For Supervised Fine-Tuning Small LLMs

Best Practices from Red Hat AI Innovation, MIT-IBM Watson AI Lab, IBM Research, Massachusetts Institute of Technology (MIT) and the University of Massachusetts Amherst [2]. The paper is a Comprehensive study on supervised fine-tuning of small-sized LLMs using instruction-tuning datasets spanning diverse knowledge domains and skills. Key insights from the research include:

1. Larger batch sizes paired with lower learning rates lead to improved results. For stacked training, larger batch sizes uniformly resulted in improved performance while phased training, a batch size of 3,840 samples outperformed the smaller batch size of 128 samples.
2. No significant difference in performance between phased (sequentially training on data divided into phases) and stacked (training on the entire dataset at once) strategies, yet stacked training is simpler and more efficient.
3. Different instruction-following datasets was implemented with 308,343 samples, a foundational knowledge dataset with 231,178 samples, a complex skills dataset with 285,966 samples. **Only 2-4% improvements.**

3.1.2 The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities

Best Practices Ireland's AI Center include [22]:

1. Use low learning rates, between 1e-4 to 2e-4, ensures stable convergence.
2. Mixed precision training involves using both 16-bit and 32-bit floating-point types to reduce memory usage and increase computational efficiency.
3. Regularly saving checkpoints ensures model weights at various intervals across 5-8 epochs to capture optimal performance without overfitting while continuously evaluating the model performance using a validation set.
4. Datasets, with at least 1000 samples recommended for effective fine-tuning. However, large datasets pose challenges in terms of storage, computational requirements, and processing time

3.2 Domain specific Fine-Tuning

This section includes, to our knowledge, all instruction-following models have been developed for general cybersecurity tasks.

3.2.1 Llama-3.1-FoundationAI-SecurityLLM-8B-Instruct Technical Report

In [21] the authors fine-tune Llama-3.1-8B-Instruct on cybersecurity data, resulting in Foundation-Sec-8B-Instruct a model they say outperforms Llama 3.1-8B-Instruct on a range of cybersecurity tasks.

It is not clear where the training data comes from, how it was formatted or what the hyperparameters where. The authors say that SFT and RL was utilized to train the model with instructions. The results show that the fine-tuned model performed worse then the base model for CyberMetric and SecEval.

Model	CTIBench-RCM	CTIBench-MCQA	CTIBench-VSP	CyberMetric-500	SecBench	SecEval
Llama 3.1-8B-Instruct	0.558±0.007	0.617±0.004	0.815±0.002	0.847±0.005	0.723±0.010	0.855±0.003
Foundation-Sec-8B-Instruct	**0.692±0.005** (↑24.03%)	0.644±0.003 (↑4.40%)	0.802±0.004 (↓-1.67%)	0.830±0.005 (↓-2.01%)	0.685±0.006 (↓-5.21%)	0.833±0.003 (↓-2.50%)

Table 1: Foundation-Sec-8B-Instruct results.

3.2.2 Primus: A Pioneering Collection of open-weight Datasets for cybersecurity LLM Training

In [19] the authors present datasets covering all major training stages, continued pre-training, instruction and reasoning fine-tuning. The authors mean that Llama-3.1-8B-Instruct improved 15.9% (aggregate score) on cybersecurity benchmarks after continued pre-training. Comparing Llama-3.1-8B-Instruct before and after CPT (Primus-Seed + FineWeb) the results show 0.01, 0.006 and 0.004 improvement on CyberMetric and 0.0041, 0.0009 and 0.0023 on SecEval.

Model	CISSP	CTI-MCQ	CTI-RCM	CTI-VSP	CTI-ATE	CyberMetric-500	SecEval	Agg.
Llama-3.1-8B-Instruct	0.7073	0.6420	0.5910	1.2712	0.2721	0.8560	0.4966	2.29
+ Primus-Seed	0.7132	0.6608	0.6100	1.2848	0.2829	0.8600	0.4998	2.34 ↑2.1%
+ Primus-FineWeb	0.7191	0.6600	0.6680	1.1499	0.3006	0.8620	0.4984	2.56 ↑11.5%
+ Primus-Seed+FineWeb	**0.7230**	**0.6676**	**0.6780**	**1.0912**	**0.3140**	**0.8660**	**0.5007**	**2.66** ↑15.9%

Table 2: Primus CPT results.

Comparing Llama-3.1-8B-Instruct before and after Instruct-tuning results show 0.008 increase on CyberMetric, and a 0.0023 decrease on SecEval.

Model	CISSP	CTI-MCQ	CTI-RCM	CTI-VSP	CTI-ATE	CyberMetric	SecEval	MT-Bench	Agg.
Llama-3.1-8B-Instruct	0.7073	0.6420	0.5910	1.2712	0.2721	0.8560	0.4966	**8.3491**	4.11
Llama-Primus-Instruct	0.7132	**0.6660**	**0.6660**	**1.1161**	0.3348	0.8640	0.4943	7.9063	4.21 ↑2.4%

Table 3: Primus Instruct-tuning results.

3.3 Prompt Engineering and RAG

This section includes papers for building cybersecurity agents on top of foundation models by using OpenAI Agents SDK, Prompt Engineering or RAG.

3.3.1 CAI: An Open, Bug Bounty-Ready cybersecurity AI

The authors of [41] argue that by 2028 most cybersecurity actions will be autonomous, with humans in the loop. The authors introduce cybersecurity AI (CAI), a well-maintained open-source framework that can be found on Github. CAI uses OpenAI Agents SDK to give AI agents access to all traditional hacking tools: reconnaissance, network, web, lateral movement, exploitation and data exfiltration. The entire list of tools can be found on Github.

3.3.2 PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing

In [42] the authors built PentestGPT. The repository is fully open-source and available on Github.

The authors conclude that PentestGPT outperforms other LLMs like GPT-3.5, GPT-4 and Bard (now Gemini) on their own crafted benchmarks, but also that PentestGPT proves effective in tackling real-world penetration testing targets and CTF challenges. At the time (August 2024) there were some drawbacks, such as a short context size, which made the model unable to understand the whole picture. This result was achieved by prompt engineering available OpenAI models; no fine-tuning or RAG was implemented.

3.3.3 PentestAgent: Incorporating LLM Agents to Automated Penetration Testing

The authors of [43] bring up the importance of penetration testing, but also the fact that it is time-consuming and expensive. They bring up the opportunities in automated penetration testing, but also that such systems often fall short in real-world applications due to limitations in flexibility and adaptability. The paper presents a solution to these gaps, specifically PentestAgent, a terminal-based LLM for automated penetration testing that leverages RAG to enhance penetration testing knowledge and automate various tasks. The project is fully open-source and can be found on Github.

The authors compare their model with PentestGPT [42], they conclude that these recent attempts to utilize LLMs for automating penetration testing have shown some promising initial results. However, two crucial gaps still need to be addressed for practical use: limited pentesting knowledge (data) and insufficient automation from the models.

3.4 Creating Domain Specific Datasets

This section includes academic papers for creating cybersecurity datasets for training AI models.

3.4.1 CyberPal.AI

CyberPal [44] is a collaboration between Ben-Gurion University and IBM Research. The paper presents SecKnowledge, a domain-knowledge-driven cybersecurity instruction dataset with approximately 400,000 samples. The authors also created SecKnowledge-Eval, a cybersecurity evaluation benchmark.

CyberPal.AI refers to the family of models fine-tuned using SecKnowledge. The authors claim that the results show a significant average improvement of up to 24% over base models. The results show 1.6% improvement on CyberMetric but 22.43 % improvement on SecEval for the fine-tuned Llama model.

Model	CISSP Assessment	SecMMLU	cybersecurity Skill Assessment	CyberMetric	CTI-MCQ	CTI-RCM	SecEval	Avg.
Meta-Llama-3-8B-Instruct	71.71	74.00	82.24	83.20	63.28	41.45	32.61	64.07
CyberPal.AI-Llama	**90.40**	**77.00**	**86.98**	**84.80**	**66.41**	**60.65**	**55.04**	**74.47**

Table 4: CyberPal.AI Instruct-tuning results.

3.4.2 CyberLLMInstruct

CyberLLMInstruct [45] is a dataset of 54,928 instruction-response pairs, spanning cybersecurity tasks including malware analysis, phishing simulations, and zero-day vulnerabilities, the training code can be found on Github.

The paper reveals a critical trade-off: while fine-tuning improved the model's cybersecurity task performance (achieving up to 92.50% accuracy on CyberMetric), it also severely compromised safety resilience across all tested models and attack vectors (i.e. Llama 3.1-8B security score against prompt injection dropped from 0.95 to 0.15).

LLM Model	80 Q	500 Q	2k Q	10k Q
Llama 3.1-8B	81.25 → **92.50**	76.20 → 87.80	73.05 → **91.25**	71.25 → **88.50**

Table 5: CyberLLMInstruct Instruct-tuning results on CyberMetric.

3.5 Summary

Research on best practices for fine-tuning small open-weight LLMs shows that better results can be achieved by using larger batch sizes paired with lower learning rates, mixed precision during training, frequent checkpointing, and stacked training instead of phased training.

Recent research highlights the rapid development of LLM-driven cybersecurity agents, spanning vulnerability detection and penetration testing. Frameworks such as CAI demonstrate the feasibility of fully autonomous pentesting at scale. PentestGPT, PentestAgent and similar efforts focus on leveraging existing LLMs through prompt engineering and RAG, whereas Primus explore fine-tuning with specialized datasets to improve performance and domain knowledge.

Complementary work like CyberLLMInstruct and CyberPal.AI addresses the lack of high-quality cybersecurity datasets, which remains a bottleneck for advancing model accuracy and reliability. Unfortunately, neither CyberLLMInstruct nor CyberPal.AI shared their datasets publicly.

4 Methods

This part covers the methodology used for this thesis. The different sections cover the training data, training strategy, benchmarking and hyperparameters.

4.1 Data Collection

For this thesis, no data was collected for training; instead, available cybersecurity datasets found on Huggingface were used for instruct-tuning models.

4.1.1 HackMentor-Instruct

The 14k dataset [20] can be found on Huggingface. The dataset includes various concepts in cybersecurity, such as DDoS attacks and encryption algorithms.

4.1.2 Trendyol-Instruct

The 53k dataset [46] can be found on Huggingface. The dataset includes domains such as cloud security, threat intelligence, incident response, forensics, AI/ML security and computer networking.

4.1.3 Fenrir-Instruct

The 84k dataset [47] can be found on Huggingface. The dataset includes domains such as cloud security, cybersecurity hygiene, detection and response.

4.2 Data Formatting

4.2.1 No system prompt

It is possible to use no system prompt.

```
messages = [
    {"role": "user", "content": example["user"]},
    {"role": "assistant", "content": example["assistant"]},
]
```

4.2.2 System prompt

It might be relevant to use a fixed prompt.

```
messages = [
    {"role": "system", "content": "You are an expert cybersecurity assistant.
    You provide accurate, detailed, and safe information about cybersecurity
    threats, vulnerabilities, and mitigations."},
    {"role": "user", "content": example["user"]},
    {"role": "assistant", "content": example["assistant"]},
]
```

4.3 Training

Model training was performed using the Alvis cluster, provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), dedicated for Artificial Intelligence and Machine Learning research [48]. For this thesis, training was run on 80GB Nvidia A100 GPUs.

4.3.1 Stacked Training

After evaluating training methods in 2.22, Stacked training was selected.

4.4 Inference

Inference refers to running the trained model to generate outputs, such as text predictions. Performance is often measured in tokens per second, indicating how many tokens the model can process or generate each second. For efficient inference, vLLM was used to optimize memory management and throughput [49]. For this thesis, inference was run on 48GB Nvidia A40 GPUs.

4.5 Model Selection

After evaluating several small open-weight LLMs from 2.25, it was clear that instruct models where the best choice due to their ability to give direct answers. Additionally, the majority of papers reviewed in 3 employed Llama-3.1-8B-Instruct, hence this model was selected as our base model for comparisons.

4.6 Evaluation Benchmarks

This section evaluates the performance of fine-tuned models on benchmarks.

4.6.1 CyberMetric

CyberMetric is a benchmark that provides a standardized evaluation framework with 10,000 multiple-choice questions spanning network security, cryptography, web security, malware analysis, digital forensics and more. The questions where specifically designed to assess the cybersecurity knowledge of LLMs [50]. The dataset is split into subsets of 80, 500, and 2000 questions.


```
"question": "In cryptography, what is the purpose of
using a key-derivation function (KDF)?",
"answers": {
    "A": "Generate public keys",
    "B": "Authenticate digital signatures",
    "C": "Encrypt data using a password",
    "D": "KDF are algorithms used to transform a
    secret into crucial parameters like keys and Initialization Vectors (IVs)"
},
"solution": "D"
```

4.6.2 SecEval

SecEval is a benchmark that provides over 2,189 multiple-choice questions covering nine cybersecurity domains including system security, application security, pentesting, memory safety, network security, web security, vulnerability detection, software security and cryptography [51].

```
"question": "Which of the following statements is NOT
a vulnerability when developing an iOS application?",
"choices": [
    "A: Allowing the application to execute unsigned code.",
    "B: Implementing strict mode to limit the permissions granted to the application.",
    "C: Enabling debugging tools in the production version of the application.",
    "D: Disabling ATS (App Transport Security) to allow the app to communicate over HTTP."
],
"answer": "B"
```

4.6.3 AthenaBench

AthenaBench [52] is an updated version of CTIBench [53], the update include improved tasks, data, and evaluation. The benchmark can be found on Github. Due to its recent release and the absence of independent validation in peer-reviewed literature, this benchmark was excluded from this thesis work.

4.7 Data Splitting

The dataset is divided into training and testing subsets, with 80% used for training and 20% of the data reserved for validation. This ensures that the model is evaluated on unseen data during training in order to detect overfitting.

4.8 Hyperparameters

This section explains the hyperparameters used during training. To see exact hyperparameters used during training see Appendix **??**.

4.8.1 Sequence length

The sequence length is the maximum length of tokens the tokenizer will keep in each text chunk. If the sequence length is too small, a significant portion of tokens will be discarded and therefore won't contribute to the training.

4.8.2 Learning rate

The learning rate dictates the speed at which the model adapts during training. Smaller learning rates require more training iterations since the weights are updated only slightly each time, while larger learning rates cause faster changes. However, if the learning rate is too high, the model may fail to converge.

4.8.3 Learning rate scheduler

A learning rate scheduler adjusts the learning rate over the course of training rather than keeping it fixed. Its purpose is to improve convergence—often starting with a warmup phase that gradually increases the rate, followed by decay to allow finer adjustments as the model approaches an optimum.

4.8.4 Batch size

The batch size defines the number of training steps performed before the model weights are updated (backward/update pass). Smaller batch sizes result in more frequent updates, which can introduce noise into the training process but may help the model generalize better. Larger batch sizes provide more stable and accurate gradient estimates, leading to smoother training, but they require more memory. Gradient accumulation is a technique to simulate larger batches.

4.8.5 Number of epochs

The number of epochs specifies how many times the entire training dataset is passed through the model during training. Increasing the number of epochs allows the model to learn more thoroughly from the data, as the weights are updated repeatedly based on all available samples. However, training for too many epochs can lead to overfitting, where the model begins to memorize the training data and performs worse on unseen data.

4.8.6 Optimizer

The optimizer determines how the model parameters are updated based on the computed gradients during training. It defines the strategy used to minimize the loss function by adjusting the weights.

4.8.7 Max gradient norm

The max gradient norm is a threshold used for gradient clipping, a technique that limits the magnitude of the gradients during training. If the norm of the gradients exceeds this maximum value, the gradients are scaled down to prevent excessively large updates to the model parameters.

4.8.8 Warmup steps

Warmup steps mean the learning rate is gradually increased from a small initial value to the target learning rate during the early phase of training. This helps stabilize training at the beginning, when model parameters are often randomly initialized and large updates could otherwise cause instability or divergence.

4.9 Inference Hyperparameters

This section specifies the hyperparameters used during inference.

4.9.1 Temperature

Temperature controls randomness in LLM outputs during inference. Lower values make responses more deterministic, while higher values increases variability.

4.10 Monitoring

W&B is an AI developer platform to train and fine-tune models, manage models from experimentation to production and track and monitor jobs. It was used to monitor the training progress of the model.

5 Results

This section presents the results of the experiments.

5.1 Training

During supervised fine-tuning, the reported loss is the token-level cross-entropy loss (Section 2.27.5), i.e. how well the model predicts the next token of the reference answer given the prompt and previous target tokens. Lower loss means the model is assigning higher probability to the ground-truth tokens in the training set.

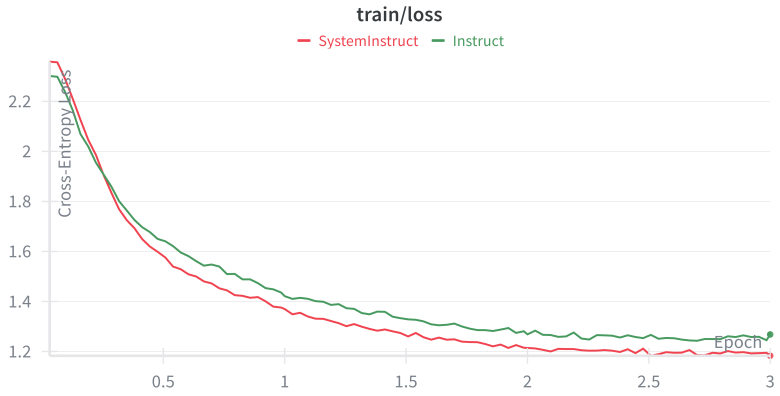

Figure 3: Training loss during supervised fine-tuning (W&B).

5.2 Quantitative Summary

Table 6 summarizes the best-performing fine-tuned configuration per dataset and compares it against the base model and the tested foundation models.

Model / Setting	CyberMetric-80 (%)	CyberMetric-500 (%)	CyberMetric-2000 (%)	SecEval (%)
BaseModel (Llama-3.1-8B-Instruct)	91.25	86.00	86.80	48.74
Best (HackMentor-Instruct, 14k)	91.25	86.90	86.80	48.29
Best (Trendyol-Instruct, 53k)	**92.50**	87.40	**87.20**	48.79
Best (Fenrir-Instruct, 84k)	91.25	**87.60**	86.95	**48.88**
Best (Stacked Training, 150k)	87.50	86.6	85.55	48.61
Primus-Seed+FineWeb	-	86.60	-	50.07
Llama-Primus-Instruct	-	86.40	-	49.43
CyberPal.AI-Llama	-	84.80	-	55.04
CyberLLMInstruct-Llama	92.50	87.80	91.25	-
GPT-5	**97.50**	96.20	94.15	51.21
Claude-4.5	**97.50**	**96.40**	**95.05**	53.81
Gemini-3	**97.50**	**96.40**	94.00	**53.95**

Table 6: Summary of benchmark results.

5.3 Evaluation

This subsection reports the benchmark results for our base model Llama-3.1-8B-Instruct referred to as `BaseModel` below. Furthermore, `Instruct` is our fine-tuned model using no system prompt and `SystemInstruct` is our fine-tuned model using a system prompt. The formatting can be found in section 4.2.

5.3.1 Results HackMentor-Instruct

Benchmark results from the HackMentor dataset 4.1.1.

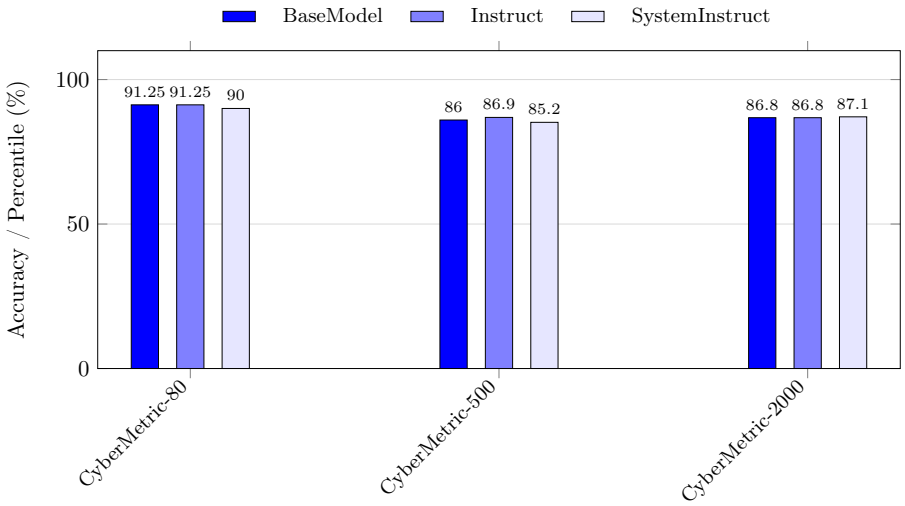

Figure 4: Benchmark performance with 14k samples (80/20 split: 11,200 training samples and 2,800 validation samples.)

SecEval	BaseModel	Instruct	SystemInstruct
Accuracy / Percentile (%)	48.74	47.97	48.29

Table 7: SecEval Benchmark performance with 14k samples.

5.3.2 Results Trendyol-Instruct

Benchmark results from the Trendyol dataset 4.1.2.

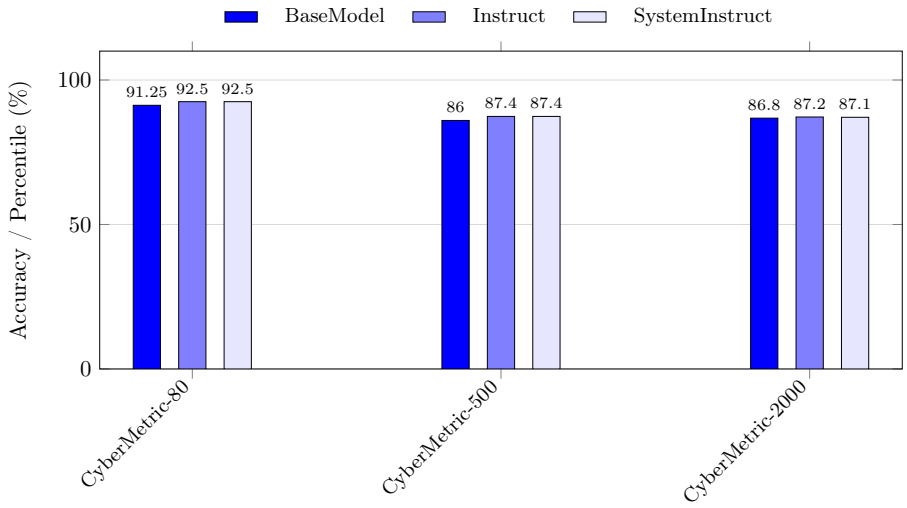

Figure 5: CyberMetric benchmark performance with 53k samples (80/20 split: 42,400 training samples and 10,600 validation samples).

SecEval	BaseModel	Instruct	SystemInstruct
Accuracy / Percentile (%)	48.74	48.65	48.79

Table 8: SecEval Benchmark performance with 53k samples.

5.3.3 Results Fenrir-Instruct

Benchmark results from the Fenrir dataset 4.1.3, where `Instruct` is our fine-tuned model using no system prompt and `SystemInstruct` is our fine-tuned model using a system prompt. Formatting can be found in section 4.2.

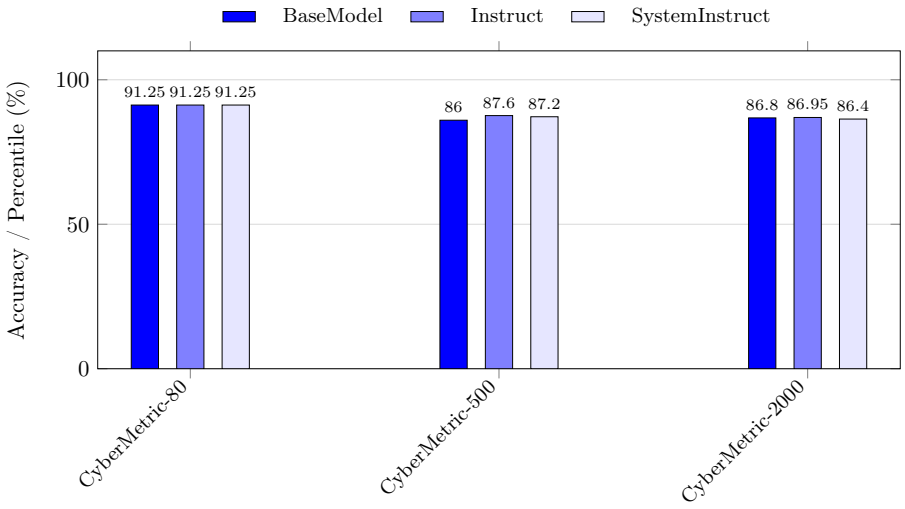

Figure 6: Benchmark performance with 84k samples (80/20 split: 67,200 training samples and 16,800 validation samples).

SecEval	BaseModel	Instruct	SystemInstruct
Accuracy / Percentile (%)	48.74	48.61	48.88

Table 9: SecEval Benchmark performance with 84k samples.

5.3.4 Results Stacked Training

Benchmark results from stacked training of all datasets.

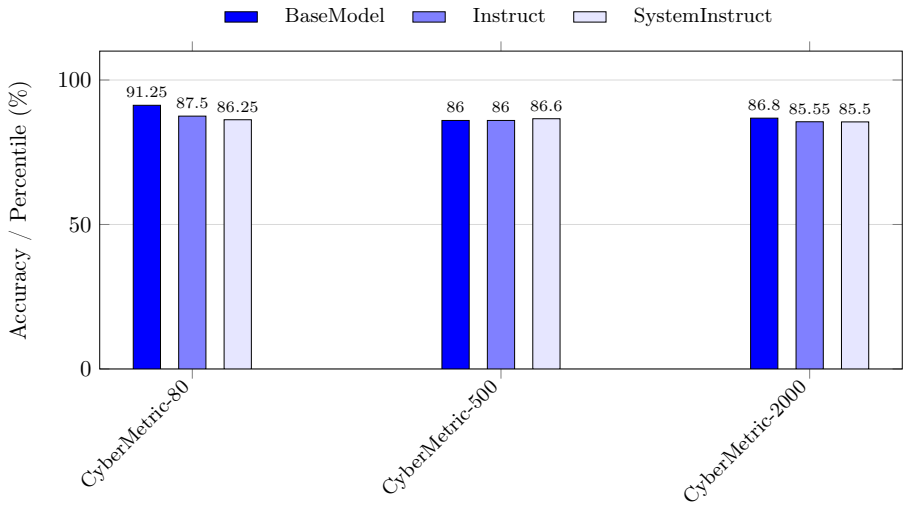

Figure 7: Benchmark performance with 150k samples (80/20 split: 120,000 training samples and 30,000 validation samples).

SecEval	BaseModel	Instruct	SystemInstruct
Accuracy / Percentile (%)	48.74	48.61	48.24

Table 10: SecEval Benchmark performance with 150k samples.

5.3.5 Results GPT-5, Claude-4.5 and Gemini-3

Benchmark results from the foundation models GPT-5, Claude-4.5 and Gemini-3. In order to benchmark foundation models effectively, LiteLLM was used [54].

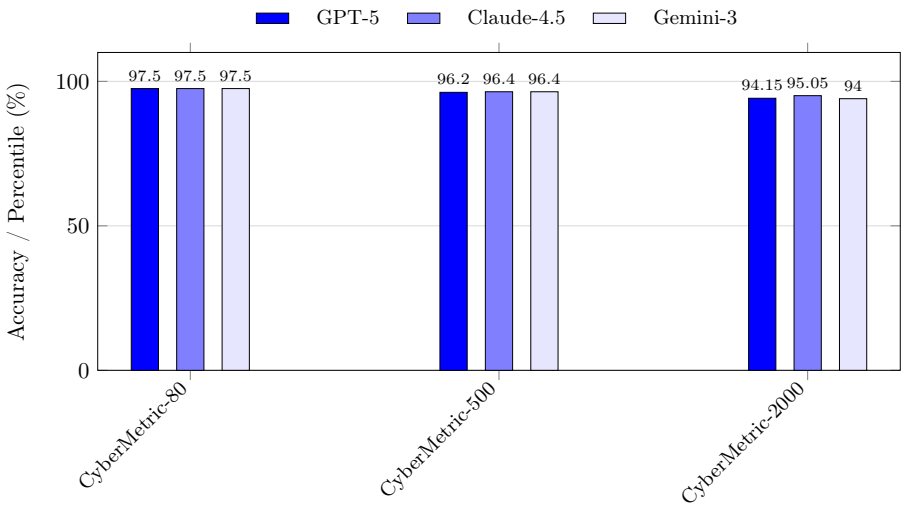

Figure 8: Benchmark performance for GPT-5, Claude-4.5 and Gemini-3.

SecEval	GPT-5	Claude-4.5	Gemini-3
Accuracy / Percentile (%)	51.21	53.81	53.95

Table 11: SecEval Benchmark performance for GPT-5, Claude-4.5, Gemini-3.

6 Discussion

This section discusses the results and experiments.

6.1 Hyperparameters Insights

The hyperparameters selected for this thesis followed best practices identified in related work 3. Specifically, the combination of a large effective batch size (3,840) with a conservative learning rate (2e-5) aligns with recommendations from section 3, which found this pairing leads to improved results for stacked training approaches.

The use of QLoRA with rank 64 and alpha 128 provided a reasonable balance between parameter efficiency and model capacity. Training on an 80GB A100 GPU allowed for sequence lengths of 2,048 tokens, which proved sufficient for most cybersecurity instruction-response pairs. The cosine learning rate scheduler with a 3% warmup ratio helped stabilize early training phases.

6.2 Data Format Insights

Experiments compared two formatting approaches: instruction-tuning without a system prompt (`Instruct`) and instruction-tuning with a fixed cybersecurity-focused system prompt (`SystemInstruct`). The results indicate minimal difference between these approaches across all tested cybersecurity benchmarks.

For the HackMentor-Instruct dataset (14k samples), the system prompt variant performed marginally worse on CyberMetric-80 (90.0% vs 91.25%) and CyberMetric-500 (85.20% vs 86.90%), but slightly better on CyberMetric-2000 (87.10% vs 86.80%). On SecEval, the difference was negligible (48.29% vs 47.97%).

For the Trendyol-Instruct dataset (53k samples), both approaches achieved nearly identical results across all CyberMetric subsets (92.50%, 87.40%, and 87.10% vs 87.20%). SecEval performance was also virtually identical (48.65% vs 48.79%).

For the Fenrir-Instruct dataset (84k samples), the `Instruct` approach slightly outperformed the system prompt variant on CyberMetric-500 (87.60% vs 87.20%) and CyberMetric-2000 (86.95% vs 86.40%), while both achieved 91.25% on CyberMetric-80.

For the stacked dataset (150k samples), performance actually degraded. This counterintuitive result suggests that combining heterogeneous datasets may introduce conflicting patterns or noise that harms generalization, supporting the observation that dataset quality and coherence matter more than sample count.

6.3 Dataset Insights

The results reveal an interesting relationship between dataset size and benchmark performance, though the pattern is not strictly linear.

6.3.1 HackMentor-Instruct (14k samples)

This smallest dataset produced essentially no improvement over the base model. On CyberMetric-500, the `Instruct` variant achieved 86.90% compared to the baseline 86.00%, a marginal 0.9 percentage point gain. The `SystemInstruct` variant actually performed worse (85.20%). SecEval scores decreased slightly from 48.74% to 47.97-48.29%.

6.3.2 Trendyol-Instruct (53k samples)

This dataset yielded the most consistent improvements. CyberMetric-80 improved from 91.25% to 92.50%, CyberMetric-500 improved from 86.00% to 87.40% and CyberMetric-2000 improved from 86.80% to 87.20%. SecEval remained essentially unchanged at 48.65-48.79%.

6.3.3 Fenrir-Instruct (84k samples)

Despite being the largest dataset, improvements were comparable to or slightly less than Trendyol. CyberMetric-80 showed no improvement (91.25%), CyberMetric-500 improved to 87.60% for `Instruct`, and CyberMetric-2000 showed minimal change (86.95%). This suggests diminishing returns beyond a certain dataset size, or that dataset quality and composition matter more than sample count.

6.4 Comparison to Related Work

Comparing these results to related work reveals important context:

- The Primus paper [19] reported only 0.8% improvement on CyberMetric after instruction-tuning.
- The CyberLLMInstruct paper [45] reported improvements up to 92.50% on CyberMetric-80, which matches our best result with the Trendyol dataset.
- The Foundation-Sec-8B-Instruct model [21] actually showed decreased performance on CyberMetric compared to the base Llama-3.1-8B-Instruct.
- CyberPal.AI [44] reported 1.6% improvement on CyberMetric but a substantial 22.43% improvement on SecEval. Our inability to replicate SecEval improvements suggests their SecKnowledge dataset may have been specifically curated to address SecEval content. Unfortunately, this dataset is not publicly available for comparison.

6.5 Foundation Model Comparison

The benchmark results for GPT-5, Claude-4.5, and Gemini-3 demonstrate a substantial performance gap compared to fine-tuned open-weight models. On CyberMetric-80, all three foundation models achieved 97.5% accuracy compared to our best result of 92.50%, a gap of **5%**. On CyberMetric-500, GPT-5 achieved 96.0% while Claude-4.5 and Gemini-3 achieved 96.4%, compared to our best fine-tuned result of 87.60%, a gap of **8.8%**. On the larger CyberMetric-2000 subset, foundation models scored between 94.0% and 95.05%, while our best fine-tuned model achieved only 87.20%, a gap of **7.85%**.

On SecEval, the gap was smaller but still notable. Foundation models achieved between 51.21% (GPT-5) and 53.95% (Gemini-3), compared to our best result of 48.88%, a difference of approximately **5%**. Interestingly, Claude-4.5 (53.81%) and Gemini-3 (53.95%) outperformed GPT-5 on SecEval, while all three performed similarly on CyberMetric. This performance gap likely reflects several factors:

1. **Pre-training scale:** Foundation models benefit from vastly larger pre-training corpora (estimated near 1 trillion parameters) that include substantial cybersecurity content from diverse sources.

2. **Alignment procedures:** These models have undergone extensive RLHF and other alignment procedures that improve their instruction-following capabilities and factual accuracy.

3. **Model capacity:** The sheer parameter count provides greater representational capacity than an 8 billion parameter model, allowing for more nuanced understanding of complex security concepts.

6.6 Key Observations

Several patterns emerge from the experimental results:

1. **Modest but consistent gains on CyberMetric:** Fine-tuning with quality cybersecurity instruction data yields 1-2 percentage point improvements on CyberMetric benchmarks, consistent with related work.

2. **No improvement on SecEval:** None of the tested datasets or formatting approaches improved SecEval performance, suggesting a fundamental mismatch between available instruction datasets and SecEval's domains.

3. **Dataset quality over quantity:** The 53k Trendyol dataset outperformed the 84k Fenrir dataset on several metrics, indicating that dataset composition and quality matter more than raw size.

4. **System prompts are optional:** For instruction-tuned base models, adding domain-specific system prompts during fine-tuning provides no measurable benefit.

5. **Foundation models remain superior: 8.8%** gap on CyberMetric-500 demonstrates that current fine-tuning approaches cannot close the performance gap with state-of-the-art foundation models like GPT-5.

6.7 Limitations

Several limitations affect the interpretation of these results:

1. **Limited benchmark coverage:** Only two benchmarks (CyberMetric and SecEval) were evaluated. The recently released AthenaBench [52] was excluded due to lack of independent validation.

2. **Multiple-choice format:** Both benchmarks rely on multiple-choice questions, which may not capture practical cybersecurity capabilities such as pentesting, vulnerability analysis, incident response, or threat detection.

3. **No safety evaluation:** Given prior findings that domain fine-tuning can compromise model safety [45], this will be important area to research.

4. **Single base model:** All experiments used Llama-3.1-8B-Instruct. Results may differ with other architectures or model sizes from section 2.25.

5. **Same number of Epochs**: bigger datasets might need more epochs to learn the data, for this thesis the number of epochs was set to 3.

7 Conclusions

This section includes the conclusions, limitations, and future work.

7.1 Performance

The experimental results provide a partial answer to the research question. Fine-tuned open-weight LLMs with 8 billion parameters cannot currently match foundation models like GPT-5, Claude-4.5 and Gemini-3 on cybersecurity benchmarks. The best fine-tuned model achieved 87.60% on CyberMetric-500, compared to approximately 96% for foundation models, a gap that instruction-tuning alone does not close.

However, the hypothesis that fine-tuning can enhance cybersecurity capabilities of smaller models is supported, with modest effect sizes. The Trendyol-Instruct dataset produced consistent improvements of **1-1.5%** across CyberMetric subsets. These gains, while small, are statistically meaningful and align with findings from related work.

7.2 Takeaways

Several practical insights emerge from this research.

7.2.1 Dataset size matters, but with diminishing returns

The 53k-sample dataset outperformed the 14k dataset, but the 84k dataset did not yield proportionally better results. Simply adding more instruction data does not guarantee proportional performance gains.

7.2.2 System prompts provide minimal benefit during fine-tuning

For instruction-tuned base models, adding a domain-specific system prompt during SFT does not meaningfully improve benchmark performance.

7.2.3 Local deployment remains a valid trade-off

Despite lower benchmark scores, fine-tuned open-weight models offer complete data control, elimination of API costs, and compliance with regulations like GDPR, advantages that may outweigh the accuracy gap for specific use cases.

7.2.4 The gap to foundation models remains substantial

GPT-5 and Claude-4.5 achieved 97.5% on CyberMetric-80 and approximately 96% on CyberMetric-500, compared to our best results of 92.50% and 87.60% respectively. For applications requiring maximum accuracy, fine-tuned 8B models are not yet competitive with state-of-the-art foundation models.

7.3 Contributions

This thesis makes the following contributions:

1. **Systematic comparison:** A comprehensive evaluation of three publicly available cybersecurity instruction datasets on two standardized benchmarks, providing reproducible baselines for future research.
2. **Foundation model benchmarks:** Updated benchmark results for GPT-5, Claude-4.5, and Gemini-3 on CyberMetric and SecEval, establishing current state-of-the-art performance levels.
3. **Practical insights:** Recommendations regarding system prompts, dataset selection and hyperparameters for training cybersecurity LLMs.
4. **Open-source code:** Training code, benchmarking code, data formating code and inference code is available on Github for reproducibility.

7.4 Future Work

Several directions could extend this research.

7.4.1 Explore CPT

Following the Primus [19] approach, incorporating CPT on cybersecurity data before instruction-tuning may provide a better way of adding new data (even though it did not for Primus).

7.4.2 Exploring GRPO

Develop cybersecurity specific reward functions in order to improve reasoning capabilities, as demonstrated by DeepSeek-R1 [38].

7.4.3 Safety evaluation

Comprehensive safety testing before and after fine-tuning is essential, given prior findings that domain fine-tuning can compromise model safety in CyberLLMInstruct [45].

7.4.4 Larger base models

Evaluate whether the performance gap narrows with larger open-weight models i.e. Llama-3.1-70B, Qwen-72B, or MoE architectures like Mixtral.

References

[1] DeepSeek-AI, D. Guo, D. Yang, *et al.*, *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*, 2025. arXiv: 2501.12948 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2501.12948.

[2] A. Pareja, N. S. Nayak, H. Wang, *et al.*, *Unveiling the secret recipe: A guide for supervised fine-tuning small llms*, 2024. arXiv: 2412.13337 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2412.13337.

[3] Microsoft, "Microsoft security copilot," Tech. Rep., 2024. [Online]. Available: https://www.microsoft.com/sv-se/security/business/ai-machine-learning/microsoft-security-copilot.

[4] Google, "Google announces sec-gemini v1, a new experimental cybersecurity model," Tech. Rep., 2025. [Online]. Available: https://security.googleblog.com/2025/04/google-launches-sec-gemini-v1-new.html.

[5] ISC2, "Isc2 research reveals the cybersecurity profession must grow by 3.4 mil to close workforce gap," 2022, Accessed: 2025-08-24. [Online]. Available: https://www.isc2.org/Insights/2022/10/ISC2-Research-Reveals-the-Cybersecurity-Profession-Must-Grow-by-3-4-Mil-to-Close-Workforce-Gap.

[6] ISC2, "2024 isc2 cybersecurity workforce study," 2024, Accessed: 2025-08-24. [Online]. Available: https://www.isc2.org/Insights/2022/10/ISC2-Research-Reveals-the-Cybersecurity-Profession-Must-Grow-by-3-4-Mil-to-Close-Workforce-Gap.

[7] V. Benjamin, E. Braca, I. Carter, *et al.*, *Systematically analyzing prompt injection vulnerabilities in diverse llm architectures*, 2024. arXiv: 2410.23308 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2410.23308.

[8] J. Yu, Y. Wu, D. Shu, M. Jin, S. Yang, and X. Xing, *Assessing prompt injection risks in 200+ custom gpts*, 2024. arXiv: 2311.11538 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2311.11538.

[9] Cybercampus Sverige. "About us." (2025), [Online]. Available: https://www.cybercampus.se/en/about (visited on 01/24/2026).

[10] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[11] L. Ouyang, J. Wu, X. Jiang, *et al.*, *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2203.02155.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1301.3781.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1607.06450.

[15] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, *Scaling laws for neural language models*, 2020. arXiv: 2001.08361 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2001.08361.

[16] L. Ouyang, J. Wu, X. Jiang, *et al.*, *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2203.02155.

[17] B. Yang, B. Venkitesh, D. Talupuru, *et al.*, *Rope to nope and back again: A new hybrid attention strategy*, 2025. arXiv: 2501.18795 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2501.18795.

[18] G. Penedo, H. Kydlíček, L. B. allal, *et al.*, *The fineweb datasets: Decanting the web for the finest text data at scale*, 2024. arXiv: 2406.17557 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2406.17557.

[19] Y.-C. Yu, T.-H. Chiang, C.-W. Tsai, C.-M. Huang, and W.-K. Tsao, *Primus: A pioneering collection of open-source datasets for cybersecurity llm training*, 2025. arXiv: 2502.11191 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2502.11191.

[20] J. Zhang, H. Wen, L. Deng, *et al.*, "Hackmentor: Fine-tuning large language models for cybersecurity," in *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2023, pp. 452–461. DOI: 10.1109/TrustCom60117.2023.00076.

[21] S. Weerawardhena, P. Kassianik, B. Nelson, *et al.*, *Llama-3.1-foundationai-securityllm-8b-instruct technical report*, 2025. arXiv: 2508.01059 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2508.01059.

[22] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid, *The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities*, 2024. arXiv: 2408.13296 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.13296.

[23] R. A. Bafghi, C. Bagwell, A. Ravichandran, A. Shrivastava, and M. Raissi, *Fine tuning without catastrophic forgetting via selective low rank adaptation*, 2025. arXiv: 2501.15377 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2501.15377.

[24] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021. arXiv: 2106.09685 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2106.09685.

[25] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *Qlora: Efficient finetuning of quantized llms*, 2023. arXiv: 2305.14314 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.14314.

[26] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," eng, *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991, ISSN: 0899-7667.

[27] O. Sanseviero, L. Tunstall, P. Schmid, S. Mangrulkar, Y. Belkada, and P. Cuenca, *Mixture of experts explained*, 2023. [Online]. Available: https://huggingface.co/blog/moe.

[28] A. Grattafiori, A. Dubey, A. Jauhri, *et al.*, *The llama 3 herd of models*, 2024. arXiv: 2407.21783 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2407.21783.

[29] OpenAI, : S. Agarwal, *et al.*, *Gpt-oss-120b gpt-oss-20b model card*, 2025. arXiv: 2508.10925 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2508.10925.

[30] A. Yang, A. Li, B. Yang, *et al.*, *Qwen3 technical report*, 2025. arXiv: 2505.09388 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2505.09388.

[31] Qwen, : A. Yang, *et al.*, *Qwen2.5 technical report*, 2025. arXiv: 2412.15115 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2412.15115.

[32] G. Team, T. Mesnard, C. Hardin, *et al.*, *Gemma: Open models based on gemini research and technology*, 2024. arXiv: 2403.08295 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2403.08295.

[33] K. Team, Y. Bai, Y. Bao, *et al.*, *Kimi k2: Open agentic intelligence*, 2025. arXiv: 2507.20534 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2507.20534.

[34] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.06825.

[35] HugginfFace. "Trl - transformer reinforcement learning." Accessed: 2025-08-24. (2025), [Online]. Available: https://huggingface.co/docs/trl/index.

[36] H. Face, *Chat templates — transformers documentation*, Hugging Face, 2025. [Online]. Available: https://huggingface.co/docs/transformers/chat_templating.

[37] Hugging Face, *Sft trainer — computing the loss*, https://huggingface.co/docs/trl/en/sft_trainer#computing-the-loss, Accessed: 2026-01-26, 2026. [Online]. Available: https://huggingface.co/docs/trl/en/sft_trainer#computing-the-loss.

[38] Z. Shao, P. Wang, Q. Zhu, *et al.*, *Deepseekmath: Pushing the limits of mathematical reasoning in open language models*, 2024. arXiv: 2402.03300 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.03300.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1707.06347.

[40] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, *Direct preference optimization: Your language model is secretly a reward model*, 2024. arXiv: 2305.18290 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.18290.

[41] V. Mayoral-Vilches, L. J. Navarrete-Lozano, M. Sanz-Gómez, *et al.*, *Cai: An open, bug bounty-ready cybersecurity ai*, 2025. arXiv: 2504.06017 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2504.06017.

[42] G. Deng, Y. Liu, V. Mayoral-Vilches, *et al.*, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 847–864, ISBN: 978-1-939133-44-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/deng.

[43] X. Shen, L. Wang, Z. Li, *et al.*, *Pentestagent: Incorporating llm agents to automated penetration testing*, 2024. arXiv: 2411.05185 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2411.05185.

[44] M. Levi, Y. Allouche, D. Ohayon, and A. Puzanov, "Cyberpal. ai: Empowering llms with expert-driven cybersecurity instructions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, 2025, pp. 24 402–24 412. DOI: 10.1609/aaai.v39i23.34618. [Online]. Available: https://doi.org/10.1609/aaai.v39i23.34618.

[45] A. ElZemity, B. Arief, and S. Li, *Cyberllminstruct: A new dataset for analysing safety of fine-tuned llms using cyber security data*, 2025. arXiv: 2503.09334 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2503.09334.

[46] Trendyol Security Team, *Trendyol cybersecurity defense instruction-tuning dataset v2.0*, version 2.0.0, Hugging Face, Jul. 2025.

[47] A. Kiraz, *Fenrir v2.0 — cybersecurity defense instruction-tuning dataset*, Hugging Face, 2025. [Online]. Available: https://huggingface.co/datasets/AlicanKiraz0/Cybersecurity-Dataset-Heimdall-v2.0.

[48] Alvis, "Alvis," Tech. Rep., 2025. [Online]. Available: https://www.c3se.chalmers.se/about/Alvis/.

[49] VLLM, "Vllm," Tech. Rep., 2025. [Online]. Available: https://docs.vllm.ai/en/latest/.

[50] N. Tihanyi, M. A. Ferrag, R. Jain, T. Bisztray, and M. Debbah, "Cybermetric: A benchmark dataset based on retrieval-augmented generation for evaluating llms in cybersecurity knowledge," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024, pp. 296–302. DOI: 10.1109/CSR61664.2024.10679494.

[51] G. Li, Y. Li, W. Guannan, H. Yang, and Y. Yu, *Seceval: A comprehensive benchmark for evaluating cybersecurity knowledge of foundation models*, https://github.com/XuanwuAI/SecEval, 2023.

[52] M. T. Alam, D. Bhusal, S. Ahmad, N. Rastogi, and P. Worth, *Athenabench: A dynamic benchmark for evaluating llms in cyber threat intelligence*, 2025. arXiv: 2511.01144 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2511.01144.

[53] M. T. Alam, D. Bhusal, L. Nguyen, and N. Rastogi, *Ctibench: A benchmark for evaluating llms in cyber threat intelligence*, 2024. arXiv: 2406.07599 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2406.07599.

[54] G. Jocher, *Litellm*, version v1.77.1.dev.1, 2025. [Online]. Available: https://github.com/BerriAI/litellm.

TRITA – XXX-EX 2025:0000
Stockholm, Sweden 2025

www.kth.se