Efficient Fine-Tuning and Quantized Deployment of 7B Cybersecurity Language Model & Performance Prediction by EdgeProfile(Extention work)

Final Report

Abdul Kader

November 15, 2025

Abstract

Large language models (LLMs) have shown strong potential in automating cybersecurity tasks such as vulnerability analysis and exploit reasoning; however, their practical deployment is often limited by high computational and memory requirements. This work explores whether a security-focused LLM can be fine-tuned, compressed, and deployed locally while remaining effective and efficient. Using the DeepHat-V1-7B model as a base, we apply parameter-efficient fine-tuning with Low-Rank Adaptation (LoRA) and the Unsloth training framework on a CVE-to-Metasploit dataset. To further reduce deployment overhead, the model is quantized to multiple precision formats, including 8-bit and 4-bit variants. Model efficiency is evaluated using **EdgeProfiler**, which analytically predicts latency, memory footprint, and energy consumption without executing the model. Experimental results show that quantization reduces model size by up to 69% compared to the FP16 baseline while maintaining usability for cybersecurity reasoning tasks. Finally, the system is deployed locally using Ollama and OpenWebUI, demonstrating that advanced cybersecurity-oriented LLMs can operate efficiently on consumer-grade hardware without reliance on cloud infrastructure.

1 Introduction

The rapid growth of software complexity has significantly expanded the attack surface of modern computing systems, making vulnerability discovery, analysis, and exploitation increasingly challenging. Public vulnerability disclosures, such as Common Vulnerabilities and Exposures (CVEs), provide critical information, but translating these descriptions into actionable security insights or exploit modules often requires expert knowledge and substantial manual effort.

Recent advances in large language models (LLMs) offer promising opportunities to automate and accelerate cybersecurity workflows. However, deploying large-scale models in security-sensitive environments presents challenges related to computational cost, memory footprint, and inference latency. These challenges are particularly pronounced when attempting to run models locally or at the edge, where cloud-based solutions may be undesirable due to privacy, latency, or cost constraints.

This project investigates whether a large language model can be fine-tuned for cybersecurity tasks while remaining lightweight, efficient, and deployable on consumer-grade hardware. Using the DeepHat-V1-7B base model, the system is fine-tuned on a CVE-to-Metasploit dataset using parameter-efficient techniques such as Low-Rank Adaptation (LoRA) and the Unsloth training framework. To further reduce deployment overhead, multiple quantization strategies are evaluated. Finally, EdgeProfiler is used to predict performance characteristics such as latency and energy consumption without executing the model.

The key contributions of this work are:

- Fine-tuning a 7B-parameter LLM for cybersecurity exploitation tasks using resource-efficient methods.
- Demonstrating significant memory reduction through quantization while preserving usability.
- Evaluating model efficiency using analytical performance prediction via EdgeProfiler.
- Deploying the system locally using Ollama and OpenWebUI without reliance on cloud services.

2 Related Work

Large language models have increasingly been applied to security-related tasks such as vulnerability classification, exploit generation, and code analysis. Models such as GPT-based systems and domain-specific LLMs have demonstrated strong reasoning capabilities but typically require substantial computational resources.

DeepHat-V1-7B is a Qwen2-based architecture designed for security-focused language modeling. Its 7.62 billion parameters provide sufficient expressive capacity while remaining more tractable than larger foundation models. Hugging Face hosts multiple variants of DeepHat, enabling experimentation with different precision formats.

Parameter-efficient fine-tuning methods such as Low-Rank Adaptation (LoRA) allow models to be adapted to specialized domains without updating the full parameter set. LoRA introduces trainable low-rank matrices into attention layers, significantly reducing memory and compute requirements.

Unsloth is an open-source framework designed to accelerate LLM fine-tuning through optimized kernels and a custom backpropagation engine. Prior work has shown that Unsloth can achieve faster training times and reduced memory usage while maintaining compatibility with popular transformer architectures.

Model quantization has also been widely studied as a method to reduce inference cost. Techniques such as INT8 and 4-bit quantization have been shown to reduce model size substantially with acceptable performance degradation. Tools such as Ollama integrate quantization directly into local model deployment workflows.

EdgeProfiler represents recent work in analytical performance modeling for LLMs. Rather than executing a model, EdgeProfiler estimates FLOPs, memory usage, latency, and energy consumption based on hardware characteristics and model parameters, enabling rapid evaluation across devices.

3 Methodology

3.1 Dataset Description and Preprocessing

The supervised fine-tuning dataset used in this project is the *CVE-to-Metasploit Module* dataset, publicly available on Hugging Face (3). This dataset is specifically designed for cybersecurity research and contains mappings between publicly disclosed vulnerabilities and their corresponding Metasploit exploitation modules.

Each data sample consists of a textual representation of a vulnerability description derived from Common Vulnerabilities and Exposures (CVEs), paired with exploit-related information such as Metasploit module metadata, exploitation logic, and contextual technical details. The dataset is structured in a single `text` field, enabling direct use for supervised fine-tuning of large language models using instruction-style or completion-based training.

This dataset is particularly well-suited for the objectives of this project for three key reasons. First, it captures real-world security vulnerabilities rather than synthetic or simulated examples, grounding the model in practical cybersecurity scenarios. Second, the inclusion of exploitation context allows the model to learn relationships between vulnerability descriptions and corresponding attack mechanisms. Third, the dataset aligns naturally with language modeling objectives, as both inputs and targets are represented in natural language form.

Prior to training, samples exceeding the maximum sequence length of 2048 tokens were truncated to ensure compatibility with the model architecture. Dataset preprocessing was parallelized using multiple CPU processes to improve throughput. No manual label engineering was required, as the dataset is already formatted for supervised fine-tuning. This streamlined preprocessing pipeline enabled efficient experimentation while preserving the semantic richness of the original data.

3.2 Fine-Tuning Strategy

To minimize computational overhead while preserving model expressiveness, supervised fine-tuning is conducted using the `SFTTrainer` from the TRL library. This trainer provides a streamlined interface for instruction-style fine-tuning of large language models and integrates seamlessly with parameter-efficient adaptation methods.

3.2.1 Training Configuration

The maximum input sequence length is set to 2048 tokens to balance contextual coverage and memory usage. Dataset preprocessing is parallelized using two CPU worker processes, improving data loading throughput and reducing training idle time. These settings enable efficient utilization of available hardware resources during fine-tuning.

Table 1: Key Fine-Tuning Configuration Parameters

Parameter	**Value / Description**
Training Method	Supervised Fine-Tuning (SFT)
Trainer	TRL `SFTTrainer`
Maximum Sequence Length	2048 tokens
Dataset Text Field	`text`
Preprocessing Workers	2 CPU processes
Optimizer	AdamW (8-bit)
Learning Rate Scheduler	Linear

3.2.2 Parameter-Efficient Adaptation with LoRA

Low-Rank Adaptation (LoRA) is applied to the transformer attention layers to enable domain-specific adaptation without updating the full set of model parameters. Instead of modifying all weights, LoRA injects trainable low-rank matrices into key projection layers of the attention mechanism.

This design dramatically reduces the number of trainable parameters, resulting in lower GPU memory consumption and faster convergence. By freezing the base model weights and optimizing only the LoRA parameters, the fine-tuning process remains computationally tractable even for a 7B-parameter model.

Table 2: LoRA Adaptation Characteristics

Aspect	**Description**
Adaptation Scope	Attention projection layers
Trainable Parameters	Low-rank adapter matrices only
Base Model Weights	Frozen
Memory Impact	Significantly reduced GPU usage
Training Speed	Faster than full fine-tuning

Key Insight: By combining supervised fine-tuning with LoRA-based parameter-efficient adaptation, the training process avoids the prohibitive cost of full-model optimization while still achieving meaningful domain specialization.

3.3 Optimization and Precision

Training uses the AdamW optimizer in 8-bit mode to further reduce memory usage. A linear learning rate scheduler is employed for stability, and weight decay is applied to mitigate overfitting. Mixed-precision training is enabled using FP16 or BF16 depending on hardware support.

3.4 Quantization

Post-training quantization is applied using multiple precision formats, including Q8_0 (8-bit) and Q4_K_M (4-bit). These formats are evaluated based on memory footprint reduction and suitability for local inference.

3.5 Model Fine-Tuning Implementation

3.5.1 Dataset Loading

The fine-tuning dataset is loaded directly from Hugging Face using the `datasets` library. The dataset contains paired CVE descriptions and corresponding Metasploit module information formatted for supervised fine-tuning.

Listing 1: Loading the CVE-to-Metasploit dataset

```python
from datasets import load_dataset

dataset = load_dataset("icantiemyshoe/cve-to-metasploit-module")
print(dataset["train"][0])
```

3.5.2 Training Environment Setup

The training environment installs Unsloth and its required dependencies, including TRL for supervised fine-tuning, PEFT for parameter-efficient adapters, and BitsAndBytes for low-precision optimization.

Listing 2: Installing required libraries

```
!pip install unsloth trl peft accelerate bitsandbytes
```

3.5.3 Model Initialization

The DeepHat-V1-7B model is loaded using Unsloth with 4-bit quantization enabled to minimize GPU memory usage. The maximum sequence length is set to 2048 tokens.

Listing 3: Loading the base model with Unsloth

```python
from unsloth import FastLanguageModel
import torch

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="DeepHat/DeepHat-V1-7B",
    max_seq_length=2048,
    load_in_4bit=True,
    dtype=None
)
```

3.5.4 LoRA Adapter Configuration

Low-Rank Adaptation (LoRA) is applied to attention layers to enable parameter-efficient fine-tuning without updating the full model.

Listing 4: Applying LoRA adapters

```
model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    bias="none",
    use_gradient_checkpointing=True
)
```

3.5.5 Supervised Fine-Tuning

Training is performed using the SFTTrainer from the TRL library. A linear learning-rate scheduler and 8-bit AdamW optimizer are used to improve stability and reduce memory consumption.

Listing 5: SFTTrainer configuration

```
from trl import SFTTrainer
from transformers import TrainingArguments

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset["train"],
    # .......(Please check github repository for full code)
        optim="adamw_8bit",
        weight_decay=0.01,
        lr_scheduler_type="linear",
        output_dir="./results"
    )
)
```

3.5.6 Training Execution

The fine-tuning process is initiated using the trainer interface.

Listing 6: Starting model training

```
trainer.train()
```

3.6 Model Conversion and Deployment with Ollama

Following fine-tuning, the adapted DeepHat-V1-7B model is prepared for efficient local deployment using Ollama. To enable compatibility with Ollama's inference engine, the model is converted to the GGUF (GPT-Generated Unified Format), a format designed for efficient loading and execution of large language models on both CPU and GPU environments.

Multiple quantized variants of the fine-tuned model are generated to evaluate trade-offs between memory footprint and numerical precision. Specifically, two quantization formats are produced: `q4_K_M`, a 4-bit quantization scheme optimized for minimal memory usage, and `q8_0`, an 8-bit quantization scheme that balances reduced model size with higher numerical fidelity. These formats are commonly used in practical deployments due to their favorable performance characteristics.

Once converted, the GGUF models are pushed to the Ollama model registry, enabling streamlined local inference and version management. Ollama handles optimized model loading, quantized execution, and hardware-aware scheduling, allowing the fine-tuned cybersecurity model to run efficiently on consumer-grade hardware without reliance on external cloud infrastructure.

This deployment pipeline demonstrates that large language models fine-tuned for specialized domains can be packaged, distributed, and executed locally while maintaining strong performance and significantly reduced resource requirements. The availability of multiple quantized variants further enables flexible deployment across heterogeneous systems with varying memory and compute constraints.

3.7 Local Inference Interface Using OpenWebUI and Docker

To enable interactive testing, evaluation, and prompt management, the fine-tuned and quantized models are deployed locally using OpenWebUI as the front-end interface and Ollama as the backend model runner. Both components are executed within a local environment, eliminating any dependency on external cloud services and ensuring data privacy during inference.

Ollama is responsible for loading the GGUF-formatted models, applying quantization (including INT8 and 4-bit variants), and executing inference efficiently on available CPU or GPU resources. Its runtime handles memory-efficient model loading, optimized kernel execution, and hardware-aware scheduling, allowing large language models to operate on consumer-grade systems.

OpenWebUI is deployed using Docker and provides a web-based interface for interacting with the locally hosted models. Through this interface, users can perform conversational testing, manage prompts, switch between quantized model variants, and monitor inference behavior. The Docker-based deployment ensures environment consistency and simplifies setup across different systems.

The combined OpenWebUI and Ollama deployment enables a fully local, self-contained inference pipeline. This design is particularly well-suited for cybersecurity applications, where sensitive vulnerability data and exploit-related prompts should not be transmitted to external servers. By running the entire system locally, the deployment preserves confidentiality while maintaining low-latency interaction and flexible model management.

4 System Implementation

The system architecture consists of four primary components: the fine-tuned LLM, the quantized inference engine, the performance profiling module, and the user-facing interface.

Fine-tuning is performed using Unsloth with LoRA adapters applied to the DeepHat-V1-7B base model. After training, the model is exported and served locally using Ollama, which handles optimized loading, quantization, and CPU/GPU execution.

OpenWebUI is deployed via Docker and serves as the front-end interface for interacting with the model. It provides chat-based testing, prompt management, and model control, enabling rapid evaluation without external dependencies.

EdgeProfiler is used independently to predict performance metrics. Hardware characteristics are collected using system tools such as `lsblk`, `dd`, `lspci`, `nvidia-smi`, and `ethtool`. These measurements are then used as inputs to EdgeProfiler's analytical cost model.

5 Empirical Results and Analysis

5.1 Impact of Quantization on Model Size

Table 3: Effect of Quantization on Model Storage Size

Model Variant	**Precision**	**Size (GB)**	**Reduction**
Base Model	FP16	15.23	–
Quantized Model	Q8_0	8.10	46.8%
Quantized Model	Q4_K_M	4.70	69.1%

Quantization yields substantial reductions in model size. The original FP16 model occupies approximately 15.23 GB. Using Q8_0 quantization reduces the size to 8.1 GB, corresponding to a 46.8% reduction. Further compression with Q4_K_M reduces the model to approximately 4.7 GB, achieving a total size reduction of 69.1%.

5.2 Performance Prediction via EdgeProfiler$^{\textbf{(Extension work)}}$

EdgeProfiler is an analytical profiling framework designed to estimate the performance characteristics of large language models deployed on edge and resource-constrained systems without executing the model (1). Unlike traditional benchmarking approaches, EdgeProfiler relies on a cost model that combines static model properties with hardware capability parameters to predict metrics such as FLOPs per token, memory footprint, inference latency, and energy consumption.

The framework separates profiling inputs into two components: a `ModelConfig`, which describes architectural properties such as layer count, hidden size, attention heads, sequence length, and numerical precision; and a `HardwareConfig`, which captures system capabilities including peak FLOPs, memory bandwidth, storage throughput, PCIe bandwidth, and utilization factors. This separation enables rapid evaluation of different model variants and quantization strategies across heterogeneous hardware platforms.

Table 4: System Metrics Collected for EdgeProfiler Analysis

Subsystem	**Tools Used**	**Purpose**
Storage	`lsblk`, `dd`	Measure sequential read bandwidth and storage throughput
Accelerators	`lspci`, `nvidia-smi`	Collect PCIe topology, GPU type, and compute capability
Network	`ip route`, `ethtool`	Determine network interface speed and routing characteristics

Formula

The peak memory bandwidth can be estimated as:
with System Specs: 13th Gen Intel i5-1334U, Cores: 12

$$\text{Memory Bandwidth (Bytes/sec)} = \text{Memory Speed} \times \text{Bus Width} \times \text{Number of Channels}$$

$$\begin{aligned}\text{Memory Speed (MT/s)} &= 5600 \times 10^6 \text{ transfers/sec}\\ \text{Bus Width per Channel} &= 64 \text{ bits} = 8 \text{ bytes}\\ \text{Number of Channels} &= 2\end{aligned}$$

$$\begin{aligned}\text{Bandwidth} &= (5.6 \times 10^9) \times 8 \times 2\\ &= 89.6 \times 10^9 \text{ Bytes/sec}\\ &\approx 89.6 \text{ GB/s}\end{aligned}$$

The comparison between EdgeProfiler predictions and empirical CPU measurements highlights the strengths and limitations of analytical profiling when applied to larger domain-adapted models. While EdgeProfiler effectively captures relative trends in compute and memory behavior across quantization levels, absolute latency and energy estimates diverge for the 7B-parameter model evaluated in this work.

This discrepancy is consistent with the scope described by Pinnock et al. (1), as the framework was primarily validated on lightweight LLMs and does not explicitly model runtime optimizations such as kernel fusion, thread scheduling, or CPU-specific inference engines. Nevertheless, EdgeProfiler remains valuable for early-stage deployment planning and comparative analysis, particularly when evaluating quantization strategies prior to full-scale execution.

Table 5: EdgeProfiler Predictions vs. Measured CPU Performance for `finetunecyberexpert-INT8`

Metric	EdgeProfiler (Predicted)	Actual CPU (Measured)
Parameters (M)	6738.4	**7000**
FLOPs per Token (G)	149.1	**∼137**
Memory Usage (MB)	6946.3	**15300**
Computation Time (ms)	5.04	**2.35**
Memory Time (ms)	135.49	**N/A**
I/O Time (ms)	0.00	**∼0**
Host-to-Device (ms)	0.00	**0**
Network Time (ms)	104.86	79.01
End-to-End Time (ms)	1.34	**2.35**
Arithmetic Intensity	1.53	**∼5.0**
Energy (J)	14.590	**168.5 (∼0.082 J/token)**

Due to a lot of system performance depends on the underlaying operating system the result can shows significant differneces.

5.3 Qualitative Case Study: CVE Identification and Exploit-Aware Reasoning

To evaluate real-world cybersecurity usefulness beyond aggregate metrics, we performed a qualitative test using an HTML snippet representative of vulnerable web content. The model was prompted to identify a relevant CVE based on the provided snippet and then asked to produce an exploit-aware response.

5.3.1 Prompt and Output (Redacted)

Listing 7: Evaluation prompt using an HTML snippet (post fine-tuning)

```
html = """<header class=page-header><h1 class=entry-title>Hello world!
<div class=post-tags></div></div></footer>"""

messages = [
  {"role": "user",
   "content": f"Can you find the CVE code for any vulnerability found 
      here: {html}"}
]
```

Outcome Summary: The fine-tuned model was able to (i) output a CVE identifier given a snippet-style prompt and (ii) generate a structured exploit-aware response. Exploit code details are redacted for responsible reporting.

5.3.2 Discussion

This case study suggests that fine-tuning on CVE-to-exploit artifacts improves the model's ability to map vulnerability descriptions to standardized identifiers and produce structured security reasoning. Importantly, such capability should be deployed with safeguards (access controls, logging, and restricted prompt templates) to reduce the risk of misuse.

5.4 Safety Considerations

Because exploit development content can be dual-use, this project treats generated exploit code as sensitive. Experimental outputs that contained actionable exploitation instructions were not retained in the final report. The evaluation focuses on capability demonstration through high-level reasoning and structured metadata rather than providing step-by-step exploit code that could enable misuse.

6 Implications and Insights

The results demonstrate that large language models can be adapted for cybersecurity tasks without requiring large-scale cloud infrastructure. Parameter-efficient fine-tuning combined with aggressive quantization enables practical deployment on consumer hardware.

Analytical performance modeling via EdgeProfiler provides valuable insights early in the development process, reducing the need for costly empirical benchmarking. This approach is particularly useful when evaluating deployment feasibility across heterogeneous hardware environments.

7 Conclusion

This project demonstrates the feasibility of building a lightweight yet capable cybersecurity-focused language model through a combination of efficient fine-tuning, quantization, and performance modeling. By leveraging Unsloth, LoRA, and EdgeProfiler, the system achieves strong practicality while remaining deployable entirely on local hardware.

Future work includes expanding the dataset to additional vulnerability classes, conducting user studies with security professionals, and validating EdgeProfiler predictions against real-world energy measurements.

8 Code and Reproducibility

All code, configuration files, and instructions to reproduce the experiments are publicly available at:

`https://github.com/getgh/llm-fine-tune-quantized7B`

The repository includes a .ipynb file which contains:

- Source code

- Environment setup instructions
- Dataset preprocessing scripts
- Experiment configuration files
- Quantization methods

8.1 Model Availability

In addition to the source code repository, the fine-tuned and quantized models produced in this project have been publicly published through the Ollama model registry. This enables straightforward local deployment and reproducibility without requiring manual model conversion or cloud-based services.

The models are available at:

https://ollama.com/getgh

Ollama provides optimized execution, built-in support for INT8 and low-bit quantization, and efficient CPU/GPU inference.

References

[1] A. Pinnock, S. Jayakody, K. A. Roxy, and M. R. Ahmed, *EdgeProfiler: A Fast Profiling Framework for Lightweight LLMs on Edge Using Analytical Model*, arXiv preprint arXiv:2506.09061v3, 2025. Available: https://arxiv.org/abs/2506.09061v3

[2] DeepHat Team, *DeepHat-V1-7B*, Hugging Face, 2024. Available: https://huggingface.co/DeepHat/DeepHat-V1-7B

[3] icantiemyshoe, *CVE to Metasploit Module Dataset*, Hugging Face Datasets, 2023. Available: https://huggingface.co/datasets/icantiemyshoe/cve-to-metasploit-module

[4] Unsloth.ai, *Unsloth Documentation*, 2024. Available: https://docs.unsloth.ai/

[5] E. Hu et al., *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv preprint arXiv:2106.09685, 2021.

[6] L. von Werra et al., *TRL: Transformer Reinforcement Learning*, GitHub Repository, 2022. Available: https://github.com/huggingface/trl

[7] Ollama, *Newly published Models by Author*, 2025. Available: https://ollama.com/getgh

[8] OpenWebUI, *Quick Start – Starting with Ollama*, 2024. Available: https://docs.openwebui.com/getting-started/quick-start/starting-with-ollama/

[9] Docker Inc., *Open-WebUI Docker Desktop Model Runner*, 2024. Available: https://www.docker.com/blog/open-webui-docker-desktop-model-runner/

[10] L. H. Newman, *Apple, Google, and Microsoft Just Fixed Multiple Zero-Day Flaws*, WIRED, Sep. 2023. Available: https://www.wired.com/story/apple-google-microsoft-zero-days-september-2023/