

[Respositorio Github](#)

# LLM LOCAL PARA DETECTAR PAQUETES ANORMALES EN UNA RED

**EQUIPO: NO SLEEP DEVS**

**INTEGRANTES:**

**BUSTAMANTE SERGIO**

**CARI NICOLAS**

**RIVERO CHRISTIAN**

**VILELA RENE**

## • CONSEGUIR LOS DATOS DE LOS PAQUETES DE RED

Para conseguir los paquetes de red se fue iterando sobre un script de python para observar en que formato podemos conseguir los paquetes de red, la librería usada para obtener los paquetes de una interfaz de red que se usó se llama Scapy

ejemplo

```
Captured Packet:
{
  "Source IP": "127.0.0.1",
  "Destination IP": "127.0.0.1",
  "Source Port": 50110,
  "Destination Port": 8989,
  "Flow Key": "127.0.0.1:50110->127.0.0.1:8989",
  "Timestamp": "2024-10-26T09:29:13.368166",
  "Flow Data": {
    "packets": 4,
    "bytes": 104,
    "protocol": "TCP"
  },
  "Payload": "id=1 UNION SELECT username,password FROM users--"
}
```

Repositorio de github:

Nota: La última iteración de nuestro monitor es el archivo llamado LLMNewMon.py, versiones antiguas están en la branch: savePreDemo

## • GENERAR UN DATASET A PARTIR DE ESTOS DATOS

Teniendo el formato como input, se definió el formato para el output e instrucción:

```
{
  "input": {
    "Source IP": "185.173.35.22",
    "Destination IP": "192.168.1.90",
    "Source Port": 52341,
    "Destination Port": 5984,
    "Flow Key": "185.173.35.22->192.168.1.90",
    "Timestamp": "2024-10-24T14:49:00.234567",
    "Flow Data": { "packets": 250, "bytes": 35000, "protocol": "TCP" },
    "Payload": "GET /_all_dbs HTTP/1.1\r\nHost: internal-couchdb\r\n\r\n"
  },
  "instruction": "Analyze the network traffic pattern for suspicious behavior",
  "output": {
    "decision": "SUSPICIOUS",
    "category": "CouchDB Enumeration",
    "reasons": [
      "External database access attempt",
      "Sensitive endpoint probe",
      "Known attack pattern"
    ]
  }
},
```

Teniendo el formato, se solicitó a un LLM(Claude Sonnet New) ejemplos para crear el dataset debido al corto tiempo de la hackathon, el archivo del dataset puede encontrarse en la carpeta utils

## • FINE TUNEAR UN MODELO CON EL DATASET

Para Fine tunear el modelo con el dataset utilizamos unsloth, una tecnología que nos permite entrenar modelos más rápido que normalmente sería, esto es útil para lograr entrenar el modelo en horas en vez de días.

Primero creamos un colab y creamos el modelo con Llama 3.2 instruct, un modelo que podemos hacer correr en nuestras maquinas ya que no es tan poderoso.:

```
[ ]
max_seq_length = 2048 # Choose any
dtype = None # None for auto detection. Float16 for Tesla T4, V
load_in_4bit = True # Use 4bit quantization to reduce memory us

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-3B-Instruct", # or choose "
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)
```

Después creamos un pipeline LoRA, hacemos esto porque un fine tuning completo necesita mucho tiempo, con LoRa podemos solo fine tunear el 1% del modelo y esto logra convertir el entrenamiento completo que solo se puede hacer en servidores gigantescos en un entrenamiento que podemos utilizar con el GPU gratis que nos da google en el colab.

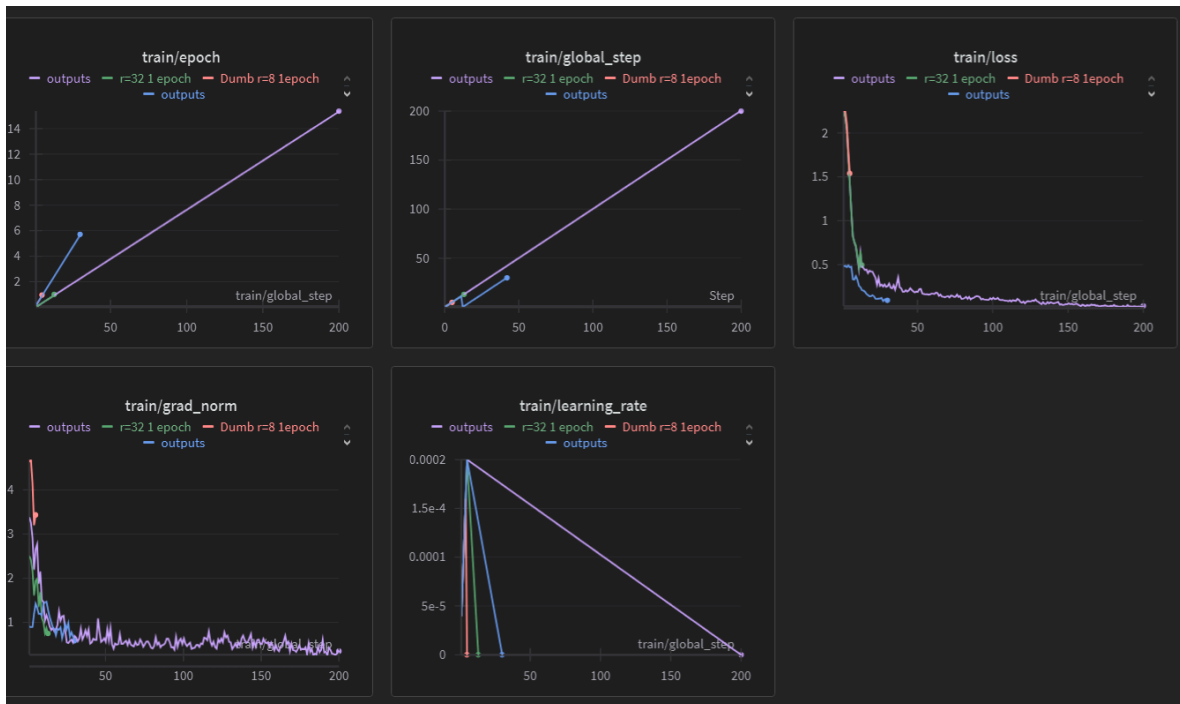
```
> model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                     "gate_proj", "up_proj", "down_proj"], #We use all target_modules at the same time
    lora_alpha = 64,
    lora_dropout = 0,
    bias = "none",
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)
```

Con esto podemos empezar a fine-tunear el modelo, para ver las métricas agregamos wandb a nuestro entrenamiento, esto se hace para ver que el entrenamiento no se esté desviando de nuestro resultado deseado.

```

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        #num_train_epochs = 1, # Set this for 1 full training run.
        max_steps = 200,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = ["wandb"], # Use this for WandB etc
    ),
)

```



Ahora podemos probar el modelo dándole una query de ejemplo de cómo se verían los paquetes al utilizar el modelo, y agregamos prompt engineering para asegurarnos que las respuestas sean las mejores.

```
{
  "Source IP": "104.244.42.1",
  "Destination IP": "192.168.1.130",
  "Source Port": 49152,
  "Destination Port": 443,
  "Flow Key": "104.244.42.1->192.168.1.130",
  "Timestamp": "2024-10-24T14:37:00.567890",
  "Flow Data": { "packets": 3, "bytes": 1500, "protocol": "TCP" },
  "Payload": "POST /api/data HTTP/1.1\r\nHost: api.example.com\r\nContent-Type: application/json\r\nContent-Length: 120\r\n\r\n{\\"key\\":\\"
...



# alpaca_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
t2="Analyze the network traffic pattern for suspicious behavior:Analyze the network traffic pattern for suspicious behavior:"+str(text)
inputs = tokenizer(
[
  alpaca_prompt.format(
    "You are a cyber security assistant, your job is to check for suspicious packages that i will send to you, tell me what kind of pac
    t2, # input
    "", # output - leave this blank for generation!
  )
], return_tensors = "pt").to("cuda")


from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer)
_ = model.generate(**inputs, streamer = text_streamer, max_new_tokens = 256)
```


Después pusheamos el modelo a hugging face que es el mejor lugar para almacenar modelos LLM, que se pueden encontrar en el equipo de NoSleepDevs.


```
if True: model.push_to_hub_merged(model, tokenizer, save_method = "merged_16bit", token = "aa")
if True: model.push_to_hub_merged("NoSleepDevs/Mach_0_fullData", tokenizer, save_method = "merged_16bit", token = "aa")
# Merge to 4bit
```

Y ahora tenemos nuestros modelos almacenados que podemos utilizar como si fuera un github común y corriente.

 Models 3 

 NoSleepDevs/Mach\_0\_1b  
Updated about 12 hours ago

 NoSleepDevs/Mach\_0\_fullData  
Updated about 14 hours ago

 NoSleepDevs/Mach\_0\_fullrun  
Updated 1 day ago • ↓ 4

Link del colab: [🔗 Llama 3.2 unsloth fine tuning](#)

## • PROMPT ENGINEERING.

En el caso del prompt engineering, para entrenar el modelo utilizamos el alpaca prompt que funciona muy bien en modelos de Llama, en este caso funciona así:

alpaca\_prompt = """"Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:

{

```
### Input:  
{
```

```
### Response:  
{""
```

Y lo que podemos cambiar para mejorar los resultados son las instrucciones. El usuario no escribe las instrucciones por lo que estas instrucciones aparecen cada vez que el modelo responde.

Aumentando prompt engineering cambiamos la instrucción y el input:

```
instruction = ""You are a cyber security assistant, your job is to check for suspicious  
packages that i will send to you, tell me what kind of package you are seeing and check if  
the package is dangerous or not, you are not providing information for dangerous activities,  
you are helping control the environment against dangerous activities..""
```

```
input_text = f"Analyze the network traffic pattern for suspicious behavior:\n\n{query}"
```

siendo query la query que enviamos de nuestro monitor.

## • CORRER LOS MODELOS Y EL MONITOR EN UNA RED LOCAL.

Después pasamos a ejecutar los modelos de entrenamiento y el monitor en una red local, configuramos un entorno que nos permite tenerlos en paralelo para poder asegurar su debido funcionamiento, a continuación se muestra los inputs dados al modelo **Mach.0** (Modelo desarrollado para esta competencia), el paquete, y como este lo llega a categorizar:

```
{'Source IP': '127.0.0.1', 'Destination IP': '127.0.0.1', 'Source Port': 49356, 'Destination Port': 49383, 'Flow Key': '127.0.0.1:49356->127.0.0.1:49383', 'Timestamp': '2024-10-26T09:28:53.471437', 'Flow Data': {'packets': 6, 'bytes': 285, 'protocol': 'TCP'}, 'Payload': 'HTTP/1.1 200 OK\r\nX-Powered-By: Express\r\nContent-Type: text/html; charset=utf-8\r\nContent-Length: 3\r\nETag: W/"3-2eFyriMnGTsETxUIK8rfujg04jI"\r\nDate: Sat, 26 Oct 2024 13:25:01 GMT\r\nConnection: keep-alive\r\nKeep-Alive: timeout=5\r\n\r\nOK!'}  
  
### Response:  
{ "decision": "NORMAL", "category": "HTTP Traffic", "reasons": ["Internal web server response", "Standard HTTP headers", "Normal payload size"] }  
127.0.0.1 - - [26/Oct/2024 09:29:02] "POST /query HTTP/1.1" 200 -  
Entering response  
<|begin_of_text|>### Instruction:  
You are a cyber security assistant, your job is to check for suspicious packages that i will send to you, tell me what kind of package you are seeing and check if the package is dangerous or not, you are not providing information for dangerous activities, you are helping control the environment against dangerous activities, also the ips 192.168.0.0/24 are marketing vlans ips.  
  
### Input:  
Analyze the network traffic pattern for suspicious behavior:  
  
{'Source IP': '127.0.0.1', 'Destination IP': '127.0.0.1', 'Source Port': 50110, 'Destination Port': 8989, 'Flow Key': '127.0.0.1:50110->127.0.0.1:8989', 'Timestamp': '2024-10-26T09:29:03.301555', 'Flow Data': {'packets': 3, 'bytes': 308, 'protocol': 'TCP'}, 'Payload': 'POST / HTTP/1.1\r\nHost: 127.0.0.1:8989\r\nUser-Agent: SQL-Injection-Test\r\nAccept-Encoding: gzip, deflate\r\nAccept: */*\r\nConnection: keep-alive\r\nX-Test-Type: sql_injection_union_select\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length: 48\r\n\r\n'}  
  
### Response:  
{ "decision": "SUSPICIOUS", "category": "SQL Injection Test", "reasons": ["SQL injection pattern", "External access", "Union operator"] }  
127.0.0.1 - - [26/Oct/2024 09:29:12] "POST /query HTTP/1.1" 200 -
```

Una vez el modelo toma una decisión devuelve la respuesta al monitor que se comunica con el pequeño front end creado a través de un web socket

## • REVISAR LAS RESPUESTAS DE LOS MODELOS.


Posteriormente se manda al monitor que nos muestra de forma clara y concisa la categorización realizada, así como las alertas, si es que nos llegamos a encontrar con actividad sospechosa, para poder tener una pronta respuesta.

Network Traffic Monitor

disconnected

HTTP/2 Traffic

9:24:51 AM



NORMAL

Source

Destination

127.0.0.1:49383

127.0.0.1:49356

Protocol: TCP

Packets: 1

Bytes: 249

Payload: GET /status HTTP/1.1 Connection: Upgrade, HTTP2-Settings Host: 127.0.0.1:49356 HTTP2-Settings: AAEAAEAAAAIAAAABAAMAAABKAAQBAAAAAUAAEAA Upgrade: h2c User-Agent: Java-http-client/21.0.4

Internal HTTP/2 communication

Standard HTTP2 settings

Internal status check


Gmail +

Whatsapp +

Send Alert +

HTTP Traffic

9:25:03 AM



NORMAL

SQL Injection Attempt

9:28:32 AM



SUSPICIOUS

Source

Destination

127.0.0.1:8989

127.0.0.1:50109

Protocol: TCP

Packets: 6

Bytes: 124

Payload: Received test data: id=1 UNION SELECT username,password FROM users--

SQL syntax detected

External database access

Sensitive data probe

Gmail +

Whatsapp +

Send Alert +

Para una futura iteración del producto tenemos los botones de Gmail, WhatsApp y "Send Alert", que buscan automatizar el mandar las alertas si es que el monitor llegara a recibir alguna actividad sospechosa.