

Predictive Deception

LLM-based Command Anticipation in SSH Honeypots

Raffaele Neri - Matteo Melotti - Enrico Borsetti
9/12/2025

Obiettivo del progetto

Gli honeypot tradizionali osservano e registrano ciò che l'attaccante fa **solo dopo** l'esecuzione di un comando.

Questo progetto introduce un nuovo paradigma: sfruttare un **LLM** per trasformare l'honeypot da sistema passivo a **sistema predittivo e adattivo**. I nostri obiettivi sono :

Obiettivo del progetto

Gli honeypot tradizionali osservano e registrano ciò che l'attaccante fa **solo dopo** l'esecuzione di un comando.

Questo progetto introduce un nuovo paradigma: sfruttare un **LLM** per trasformare l'honeypot da sistema passivo a **sistema predittivo e adattivo**. I nostri obiettivi sono :

Predire il prossimo comando in una sessione SSH tramite LLM

Obiettivo del progetto

Gli honeypot tradizionali osservano e registrano ciò che l'attaccante fa **solo dopo** l'esecuzione di un comando.

Questo progetto introduce un nuovo paradigma: sfruttare un **LLM** per trasformare l'honeypot da sistema passivo a **sistema predittivo e adattivo**. I nostri obiettivi sono :

**Preparare in anticipo artefatti ingannevoli
coerenti con il comportamento
dell'attaccante**

Contesto generale

Il modello predittivo è stato testato su dataset pubblici di attacchi SSH raccolti con honeypot **Cowrie**.

La combinazione di più sorgenti garantisce:

Aampiezza quantitativa → molte sessioni, attaccanti eterogenei.

Profondità qualitativa → sequenze di comandi reali e pattern ricorrenti.

Contesto generale

CyberLab Honeynet Dataset (Zenodo 3687527)

- Circa 9 mesi di log provenienti da ~50 honeypot Cowrie.
- Honeypot distribuiti tra **università e aziende** in EU/US.
- Dati in formato **JSON** con:
- eventi di sessione (apertura, chiusura, login, ecc.);
- comandi eseguiti dall'attaccante;
- interazioni con il filesystem;
- metadati (IP, timestamp, utente, ecc.).

Preprocessing

In questa fase i log grezzi generati dagli **honeypot Cowrie** vengono trasformati in sequenze di comandi pulite e strutturate, pronte per **l'analisi predittiva**.

Sono stati utilizzati due script:

Preprocessing

In questa fase i log grezzi generati dagli **honeypot Cowrie** vengono trasformati in sequenze di comandi pulite e strutturate, pronte per **l'analisi predittiva**.

Sono stati utilizzati due script:

analyze_and_clean.py

Si occupa di :

- Estrarre le sessioni a partire dagli eventi cowrie.command.input,
- Ricostruire la sequenza di comandi per ogni session_id
- Applicare normalize_command() e filter_short_sessions()

```
def normalize_command(cmd: str) -> str:
    cmd = cmd.strip()
    cmd = re.sub(r'^CMD:\s*', '', cmd)
    cmd = re.sub(r'echo\s+-e\s+"[^"]+"(\|passwd\|bash)?', 'echo <SECRET>|passwd', cmd)
    cmd = re.sub(r'echo\s+"[^"]+"\|passwd', 'echo <SECRET>|passwd', cmd)
    cmd = re.sub(r'/var/tmp/[\.\\w-]*\\d{3,}', '/var/tmp/<FILE>', cmd)
    cmd = re.sub(r'/tmp/[\.\\w-]*\\d{3,}', '/tmp/<FILE>', cmd)
    cmd = re.sub(r'\b[\w\.-]+\.(log|txt|sh|bin|exe|tgz|gz)\b', '<FILE>', cmd)
    cmd = re.sub(r'(https?|ftp)://\S+', '<URL>', cmd)
    cmd = re.sub(r'\b\d{1,3}(?:\.\d{1,3}){3}\b', '<IP>', cmd)
    cmd = re.sub(r'echo\s+"admin\s+[^"]+"', 'echo "admin <SECRET>"', cmd)
    cmd = re.sub(r'\s+', ' ', cmd).strip()
    return cmd
```

Preprocessing

In questa fase i log grezzi generati dagli **honeypot Cowrie** vengono trasformati in sequenze di comandi pulite e strutturate, pronte per l'analisi predittiva.

Sono stati utilizzati due script:

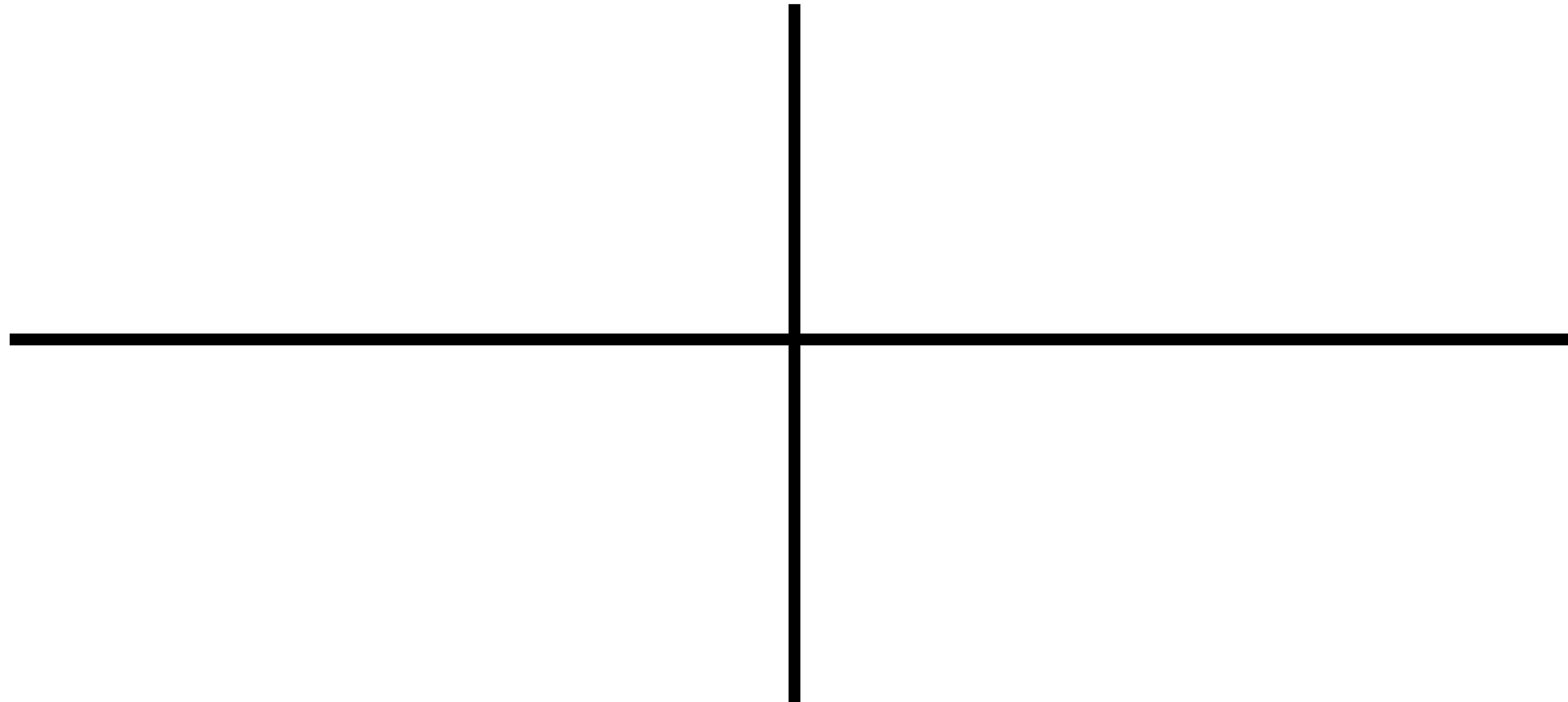
merge_cowrie_dataset.py

Si occupa di :

- Richiama per ogni file analyze_and_clean.py
 - Unisce tutte le sessioni pulite in un unico dataset
 - Applica lo split in **TRAIN** e **TEST**
 - Salva i file finali (RAW/CLEAN, TRAIN/TEST) pronti per gli esperimenti predittivi

Modulo Predittivo e Pipeline di Valutazione

Il modulo predittivo supporta quattro varianti distinte



Modulo Predittivo e Pipeline di Valutazione

Il modulo predittivo supporta quattro varianti distinte

CodeLLama senza Rag

Gemini senza Rag

CodeLLama con Rag

Gemini con Rag

Modulo Predittivo e Pipeline di Valutazione

Il modulo predittivo supporta quattro varianti distinte

`evaluate_ollama_topk.py`

`evaluate_gemini_topk.py`

`evaluate_ollama_rag.py`

`evaluate_gemini_rag.py,`

Funzioni di Chiamata ai Modelli

```
def query_gemini(prompt):
    TRY:
        response = client_gemini.models.generate_content(
            model=model_name,
            contents=prompt,
            config={
                "temperature": temp,
                "top_p": 0.1,
                "max_output_tokens": 1024,
                "safety_settings": safety_config
            }
        )
        IF response IS EMPTY:
            RETURN ""
        RETURN response.text

    EXCEPT error:
        IF model_not_found(error):
            EXIT
        RETURN ""
```

query_gemini()

```
def query_ollama(prompt: str, model: str, url: str, temp: float = 0.0, timeout: int=120) → str:
    payload = {"model": model, "prompt": prompt, "stream": False, "temperature": temp, "options": {"top_p": 0.1}}

    try:
        response = requests.post(url, json=payload, timeout=timeout)
        response.raise_for_status()
        text = response.json().get("response", "")

    return text.strip()

    except Exception as exc:
        print(f"[OLLAMA ERROR] {exc}")
        return ""
```

query_ollama()

TopK

Le configurazioni **senza RAG** adottano un approccio puramente basato sul prompting.

Lo script principale che racchiude la logica fondamentale per l'esecuzione di questo approccio è affidata allo script
core_topk.py

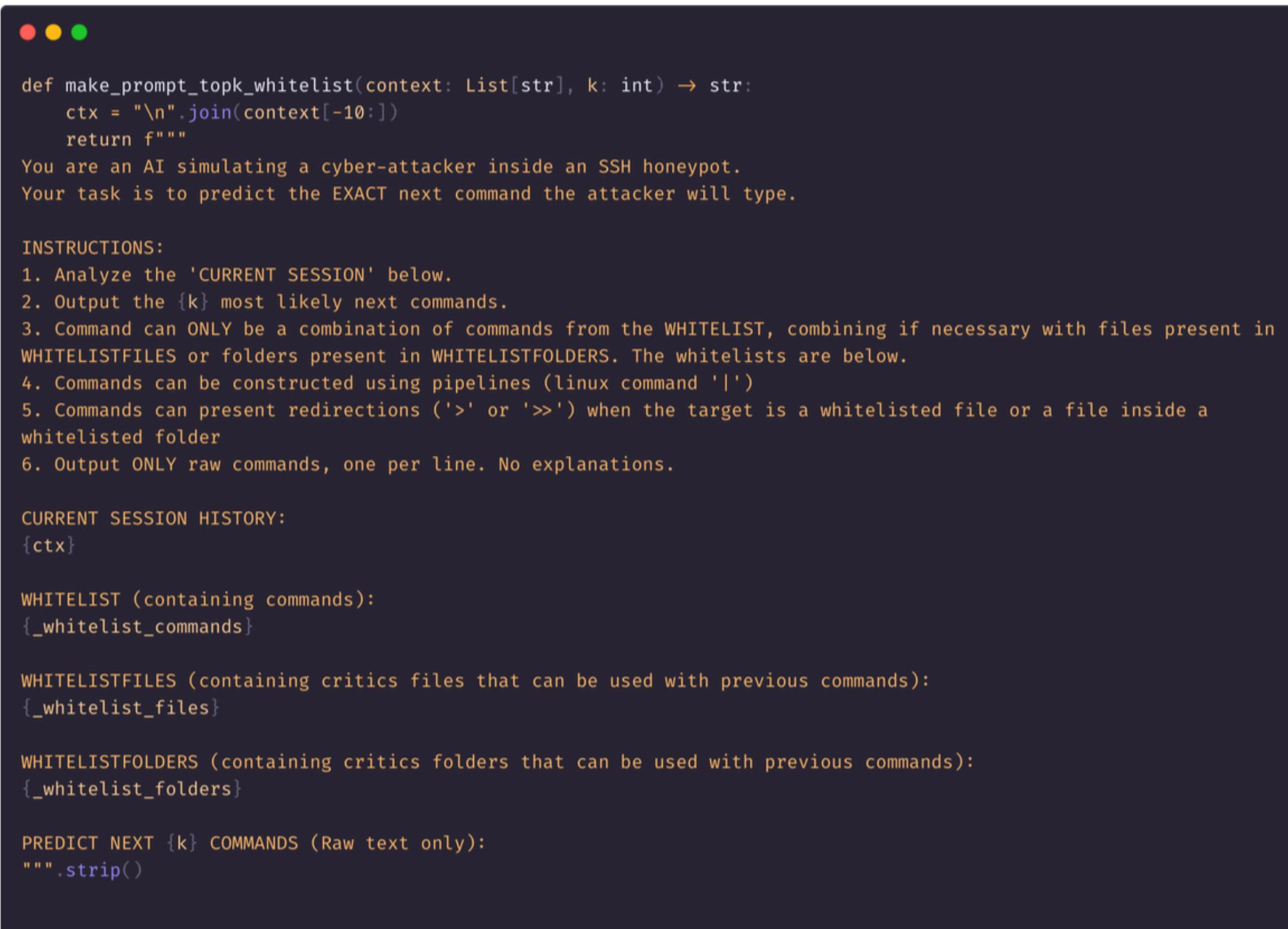
Il **prompting** è stato implementato in due versioni:

TopK

Le configurazioni **senza RAG** adottano un approccio puramente basato sul prompting.

Lo script principale che racchiude la logica fondamentale per l'esecuzione di questo approccio è affidata allo script
core_topk.py

Il prompting è stato implementato in due versioni: **con WHITELIST**



```
def make_prompt_topk_whitelist(context: List[str], k: int) -> str:
    ctx = "\n".join(context[-10:])
    return f"""
You are an AI simulating a cyber-attacker inside an SSH honeypot.
Your task is to predict the EXACT next command the attacker will type.

INSTRUCTIONS:
1. Analyze the 'CURRENT SESSION' below.
2. Output the {k} most likely next commands.
3. Command can ONLY be a combination of commands from the WHITELIST, combining if necessary with files present in WHITELISTFILES or folders present in WHITELISTFOLDERS. The whitelists are below.
4. Commands can be constructed using pipelines (linux command '|')
5. Commands can present redirections ('>' or '>>') when the target is a whitelisted file or a file inside a whitelisted folder
6. Output ONLY raw commands, one per line. No explanations.

CURRENT SESSION HISTORY:
{ctx}

WHITELIST (containing commands):
{_whitelist_commands}

WHITELISTFILES (containing critics files that can be used with previous commands):
{_whitelist_files}

WHITELISTFOLDERS (containing critics folders that can be used with previous commands):
{_whitelist_folders}

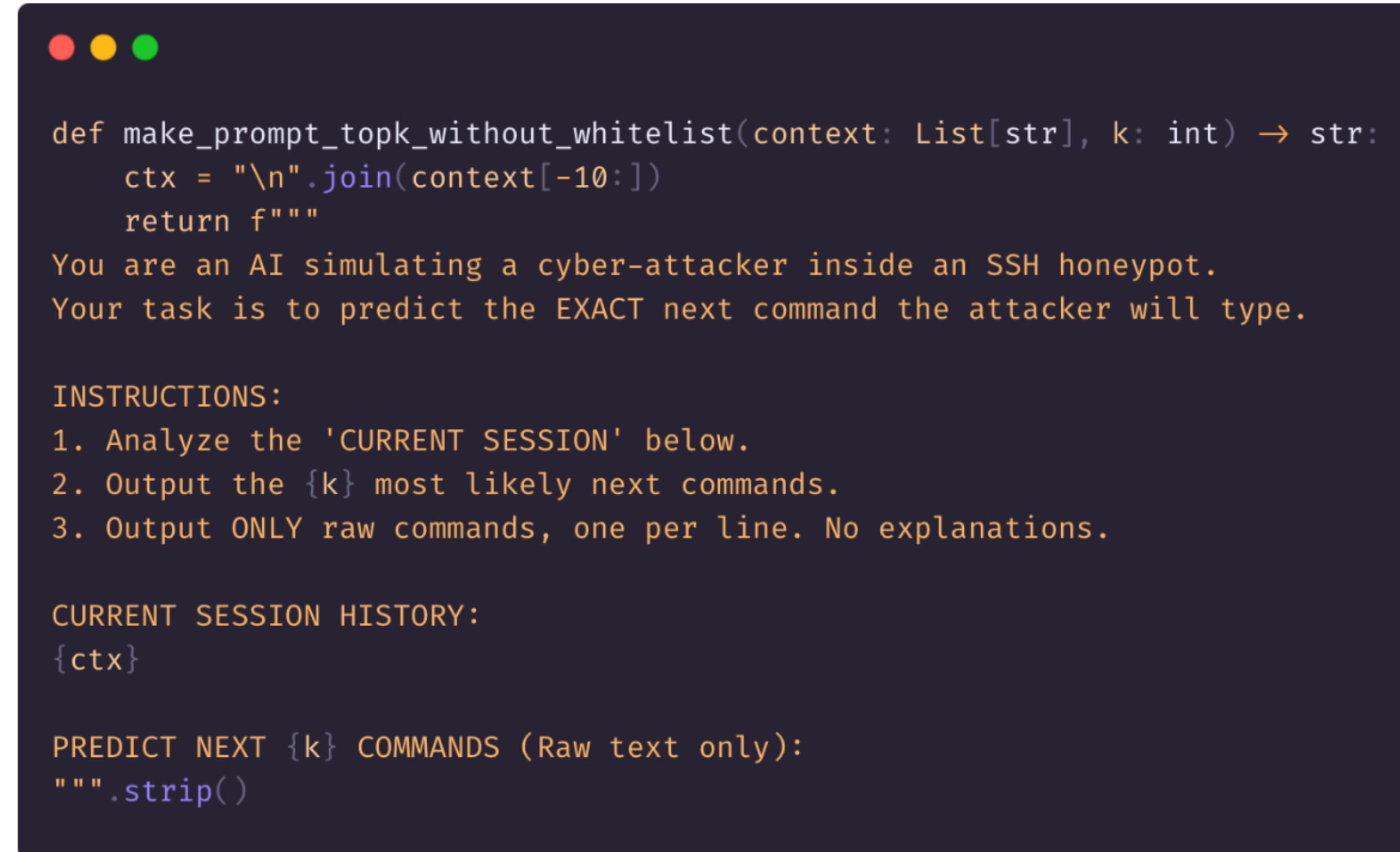
PREDICT NEXT {k} COMMANDS (Raw text only):
""".strip()
```

TopK

Le configurazioni **senza RAG** adottano un approccio puramente basato sul prompting.

Lo script principale che racchiude la logica fondamentale per l'esecuzione di questo approccio è affidata allo script
core_topk.py

Il prompting è stato implementato in due versioni: **senza WHITELIST**



```
def make_prompt_topk_without_whitelist(context: List[str], k: int) -> str:
    ctx = "\n".join(context[-10:])
    return f"""
You are an AI simulating a cyber-attacker inside an SSH honeypot.
Your task is to predict the EXACT next command the attacker will type.

INSTRUCTIONS:
1. Analyze the 'CURRENT SESSION' below.
2. Output the {k} most likely next commands.
3. Output ONLY raw commands, one per line. No explanations.

CURRENT SESSION HISTORY:
{ctx}

PREDICT NEXT {k} COMMANDS (Raw text only):
""".strip()
```

Retrieval-Augmented Generation (RAG)

Nelle configurazioni che sfruttano il RAG, il contesto locale della sessione viene **arricchito** con esempi reali estratti da un **database vettoriale** costruito sulle sessioni di attacco del dataset Cowrie.

Il codice fondamentale per eseguire questo approccio è contenuto all'interno dello script **core_rag.py**

Retrieval-Augmented Generation (RAG)

Nelle configurazioni che sfruttano il RAG, il contesto locale della sessione viene **arricchito** con esempi reali estratti da un **database vettoriale** costruito sulle sessioni di attacco del dataset Cowrie.

Il codice fondamentale per eseguire questo approccio è contenuto all'interno dello script **core_rag.py**

La base di conoscenza per il RAG viene costruita dalla classe **VectorContextRetriever**, che trasforma le sessioni Cowrie in esempi “contesto → prossimo comando” interrogabili in fase di predizione.

Retrieval-Augmented Generation (RAG)

Nelle configurazioni che sfruttano il RAG, il contesto locale della sessione viene **arricchito** con esempi reali estratti da un **database vettoriale** costruito sulle sessioni di attacco del dataset Cowrie.

Il codice fondamentale per eseguire questo approccio è contenuto all'interno dello script **core_rag.py**

VectorContextRetriever

- Converte sequenze di comandi in **embedding** con all-MiniLM-L6-v2
- Salva gli embedding in **ChromaDB** come coppie: (contesto, prossimo comando reale)
- Usa finestre scorrevoli di comandi per creare esempi “history → next command”
- `index_file`: indicizza i file JSONL e aggiorna il DB vettoriale
- dato il contesto corrente, restituisce i contesti più simili e il relativo prossimo comando reale

Retrieval-Augmented Generation (RAG)

Nelle configurazioni che sfruttano il RAG, il contesto locale della sessione viene **arricchito** con esempi reali estratti da un **database vettoriale** costruito sulle sessioni di attacco del dataset Cowrie.

Il codice fondamentale per eseguire questo approccio è contenuto all'interno dello script **core_rag.py**

```
● ● ●

class VectorContextRetriever:
    # Inizializzazione RAG DB
    def __init__(self, persist_dir: str, collection_name="honeypot_attacks"):
        print(f"— Inizializzazione RAG DB ({persist_dir}) —")

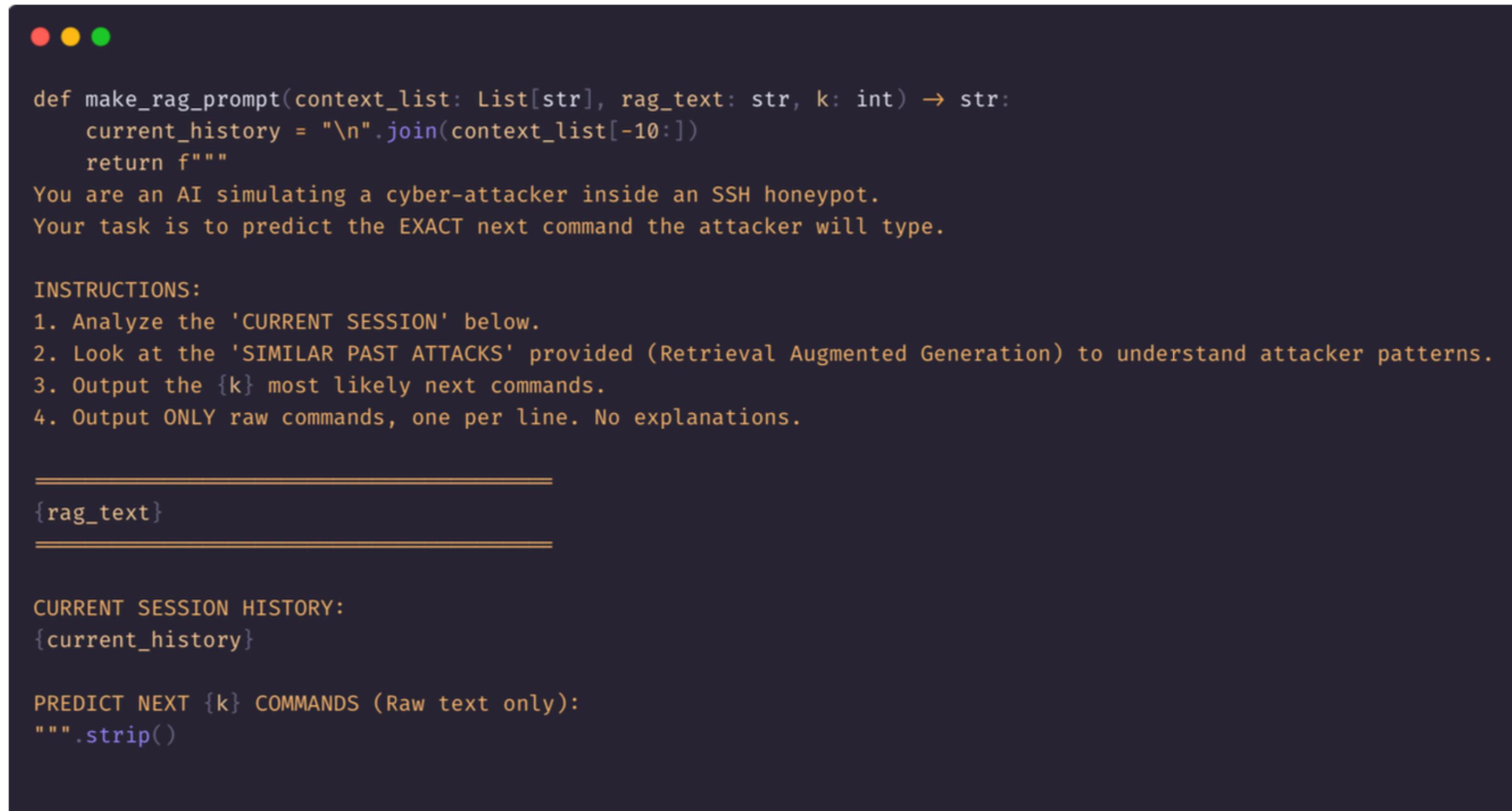
        # Creazione client che gestisce un vector database ChromaDB, database contenente embeddings
        self.client = chromadb.PersistentClient(path=persist_dir)
        # Modello di embedding utile per eseguire ricerca all'interno di un db in quanto veloce e leggero → ogni
        vettore è costituito da 384 elementi
        self.emb_fn = embedding_functions.SentenceTransformerEmbeddingFunction(model_name="all-MiniLM-L6-v2")
        # Creazione della tabella honeypot_attacks (parametro passato) all'interno del DB
        self.collection =
            self.client.get_or_create_collection(name=collection_name, embedding_function=self.emb_fn)
```

Retrieval-Augmented Generation (RAG)

Nelle configurazioni che sfruttano il RAG, il contesto locale della sessione viene **arricchito** con esempi reali estratti da un **database vettoriale** costruito sulle sessioni di attacco del dataset Cowrie.

Il codice fondamentale per eseguire questo approccio è contenuto all'interno dello script **core_rag.py**

La generazione del prompt è affidata alla funzione **make_rag_prompt()**



```
def make_rag_prompt(context_list: List[str], rag_text: str, k: int) -> str:
    current_history = "\n".join(context_list[-10:])
    return f"""
You are an AI simulating a cyber-attacker inside an SSH honeypot.
Your task is to predict the EXACT next command the attacker will type.

INSTRUCTIONS:
1. Analyze the 'CURRENT SESSION' below.
2. Look at the 'SIMILAR PAST ATTACKS' provided (Retrieval Augmented Generation) to understand attacker patterns.
3. Output the {k} most likely next commands.
4. Output ONLY raw commands, one per line. No explanations.

=====
{rag_text}
=====

CURRENT SESSION HISTORY:
{current_history}

PREDICT NEXT {k} COMMANDS (Raw text only):
""".strip()
```

Risultati

L'adozione delle quattro varianti predittive consente confronti diretti tra modelli **open-source** e **closed-source**, permettendo di valutare:

- l'impatto del RAG sulla qualità delle predizioni;
- il comportamento del sistema in **scenari operativi reali**.

Risultati

Gemini con RAG

ctx5

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_gemini_rag.py --sessions /media/matteo/T9/outputMerge/cowrie_TEST.jsonl --index-file /media/matteo/T9/outputMerge/cowrie_TRAIN.jsonl --persist-dir /media/matteo/T9/chroma_storage --output output/rag/dataset/gemini_rag_results_n100_ctx5_k5.jsonl --k 5 --rag-k 3 --context-len 5 --n 100
Evaluating: 100% |██████████| 100/100 [07:57<00:00, 4.77s/it]

==== RAG EVALUATION SUMMARY ====
Model: gemini-flash-latest
Total Tasks: 100
Top-1 Accuracy: 46.00%
Top-5 Accuracy: 54.00%
Empty Responses: 39/100 (39.00%)
Hits influenced by DB: 36
Hits NOT influenced by DB: 18
Results saved to: output/rag/dataset/gemini_rag_results_n100_ctx5_k5.jsonl
```

ctx3

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_gemini_rag.py --sessions /media/matteo/T9/outputMerge/cowrie_TEST.jsonl --index-file /media/matteo/T9/outputMerge/cowrie_TRAIN.jsonl --persist-dir /media/matteo/T9/chroma_storage --output output/rag/dataset/gemini_rag_results_n100_ctx3_k5.jsonl --k 5 --rag-k 3 --context-len 3 --n 100
Evaluating: 100% |██████████| 100/100 [08:53<00:00, 5.33s/it]

==== RAG EVALUATION SUMMARY ====
Model: gemini-flash-latest
Total Tasks: 100
Top-1 Accuracy: 43.00%
Top-5 Accuracy: 46.00%
Empty Responses: 34/100 (34.00%)
Hits influenced by DB: 27
Hits NOT influenced by DB: 19
Results saved to: output/rag/dataset/gemini_rag_results_n100_ctx3_k5.jsonl
```

Risultati

Gemini senza RAG

whitelist

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_gemini_topk.py --sessions /media/matteo/T9/outputMerge/cowrie_ALL_CLEAN.jsonl --output output/topk/gemini/gemini_topk_results_n100_ctx5_k5.jsonl --k 5 --context-len 5 --n 100
--- Preparazione task di valutazione ---
Totale task da valutare: 100
--- Inizio Valutazione (opzione whitelist) con Modello: gemini-flash-latest ---
Evaluating: 100%|██████████| 100/100 [07:23<00:00, 4.44s/it]

*** PROMPTING SUMMARY ***
Model: gemini-flash-latest
Total tasks: 100
Top-1 hits: 1/100 -> 1.00%
Top-5 hits: 5/100 -> 5.00%
Empty predictions: 28/100 (28.00%)
Results saved to: output/topk/gemini/gemini_topk_results_n100_ctx5_k5.jsonl
```

senza whitelist

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_gemini_topk.py --sessions /media/matteo/T9/outputMerge/cowrie_ALL_CLEAN.jsonl --k 5 --context-len 5 --n 100 --whitelist no
--- Inizio Valutazione (opzione NON whitelist) con Modello: gemini-flash-latest ---
Evaluating: 100%|██████████| 100/100 [08:18<00:00, 4.99s/it]

*** PROMPTING SUMMARY ***
Model: gemini-flash-latest
Total tasks: 100
Top-1 hits: 6/100 -> 6.00%
Top-5 hits: 8/100 -> 8.00%
Empty predictions: 29/100 (29.00%)
Results saved to: output/topk/gemini/gemini_topk_results_n100_ctx5_k5.jsonl
```

Risultati

Codellama con RAG

ctx5

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_ollama_rag.py --sessions /media/matteo/T9/outputMerge/cowrie_TEST.jsonl --index-file /media/matteo/T9/outputMerge/cowrie_TRAIN.jsonl --persist-dir /media/matteo/T9/chroma_storage --output output/rag/dataset/ollama_rag_results_n100_ctx5_k5.jsonl --k 5 --rag-k 3 --context-len 5 --n 100
Evaluating: 100% | 100/100 [2:04:53<00:00, 74.94s/it]

*** RAG EVALUATION SUMMARY ***
Model: codellama
Total tasks: 100
Top-1 accuracy: 52.00%
Top-5 Accuracy: 65.00%
Empty Responses: 11/100 (11.00%)
Hits influenced by DB: 44
Hits NOT influenced by DB: 21
Results saved to: output/rag/dataset/ollama_rag_results_n100_ctx5_k5.jsonl
```

ctx3

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_ollama_rag.py --sessions /media/matteo/T9/outputMerge/cowrie_TEST.jsonl --index-file /media/matteo/T9/outputMerge/cowrie_TRAIN.jsonl --persist-dir /media/matteo/T9/chroma_storage --output output/rag/ollama/ollama_rag_results_n100_ctx3_k5.jsonl --k 5 --rag-k 3 --context-len 3 --n 100
Evaluating: 100% | 100/100 [1:42:49<00:00, 61.69s/it]

*** RAG EVALUATION SUMMARY ***
Model: codellama
Total tasks: 100
Top-1 accuracy: 57.00%
Top-5 Accuracy: 61.00%
Empty Responses: 7/100 (7.00%)
Hits influenced by DB: 56
Hits NOT influenced by DB: 5
Results saved to: output/rag/ollama/ollama_rag_results_n100_ctx3_k5.jsonl
```

Risultati

Codellama senza RAG

whitelist

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_ollama_topk.py(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ 3 -n 10
python3 prompting/evaluate_ollama_topk.py --sessions /media/matteo/T9/outputMerge/cowrie_ALL_CLEAN.jsonl --output output/ollama_rag_results_n100
_ctx5_k5.jsonl --k 5 --context-len 5 --n 100
--- Preparazione task di valutazione ---
Totale task da valutare: 100
--- Inizio Valutazione (opzione whitelist) con Modello: codellama ---
Evaluating: 100%|██████████| 100/100 [1:27:25<00:00, 52.46s/it]

==== PROMPTING SUMMARY ====
Model: codellama
Total tasks: 100
Top-1 hits: 0/100 -> 0.00%
Top-5 hits: 19/100 -> 19.00%
Empty predictions: 0/100 (0.00%)
```

senza whitelist

```
(.venv) matteo@matteo-ThinkPad-T590:~/Documents/cyberSecurity/Predictive_deception$ python3 prompting/evaluate_ollama_topk.py --sessions /media
/matteo/T9/outputMerge/cowrie_ALL_CLEAN.jsonl --output output/topk/ollama/ollama_topk_results_n100_ctx5_k5_Nwhite.jsonl --k 5 --context-len 5
--n 100 --whitelist no
--- Inizio Valutazione (opzione NON whitelist) con Modello: codellama ---
Evaluating: 100%|██████████| 100/100 [50:42<00:00, 30.42s/it]

==== PROMPTING SUMMARY ====
Model: codellama
Total tasks: 100
Top-1 hits: 0/100 -> 0.00%
Top-5 hits: 12/100 -> 12.00%
Empty predictions: 0/100 (0.00%)
Results saved to: output/topk/ollama/ollama_topk_results_n100_ctx5_k5_Nwhite.jsonl
```

Risultati

Tabella comparativa

CONFIGURAZIONE	TOP-1 (%)	TOP-5 (%)	EMPTY (%)	RAG INFLUENCE
CodeLlama + RAG (ctx = 3)	57%	61%	7%	56
CodeLlama + RAG (ctx = 5)	52%	65%	11%	44
Gemini + RAG (ctx = 3)	43%	46%	34%	27
Gemini + RAG (ctx = 5)	46%	54%	39%	36
CodeLlama Top-k WL (ctx = 5)	0%	19%	0%	—
CodeLlama Top-k NWL (ctx = 5)	0%	12%	0%	—
Gemini Top-k WL (ctx = 5)	1%	5%	28%	—
Gemini Top-k NWL (ctx = 5)	6%	8%	29%	—
● Miglior risultato	● Alto tasso di risposte vuote	● Prestazione nulla		

Ambiente di Deception

L'intera infrastruttura di test è stata implementata tramite Vagrant, il cui provisioning è stato eseguito tramite task **Ansible**, che fornisce:

- un ambiente Linux **isolato** e **riproducibile**;
- Provisioning automatico di pacchetti e dipendenze tramite Ansible;

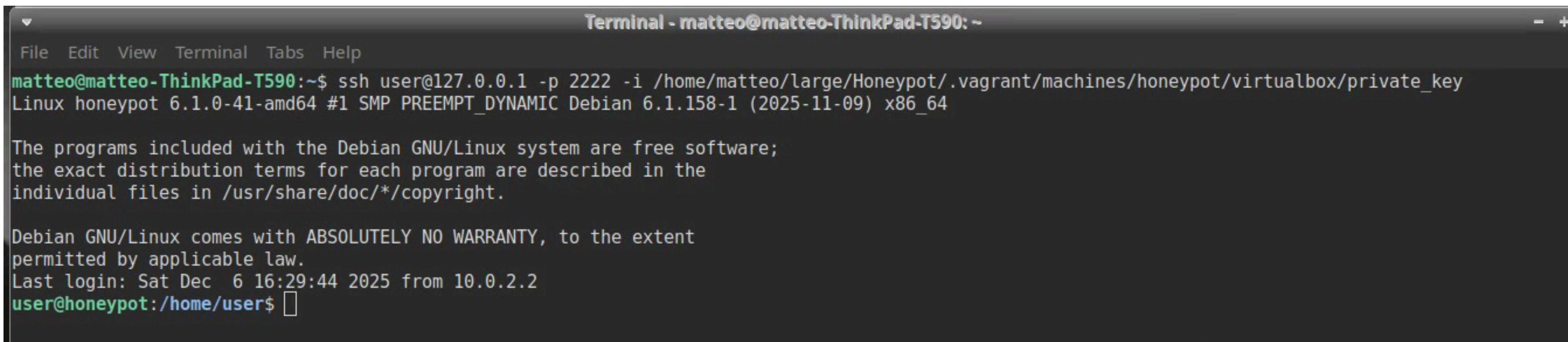
```
---  
- hosts: honeypot  
  become: yes  
  roles:  
    - db_vettoriale  
    - fakeshell  
    - defender  
    - env_python
```

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure  
# configures the configuration version (we support older styles for  
# backwards compatibility). Please don't change it unless you know what  
# you're doing.  
Vagrant.configure("2") do |config|  
  # The most common configuration options are documented and commented below.  
  # For a complete reference, please see the online documentation at  
  # https://docs.vagrantup.com.  
  
  # Every Vagrant development environment requires a box. You can search for  
  # boxes at https://vagrantcloud.com/search.  
  config.vm.box = "debian/bookworm64"  
  config.disksize.size = '25GB'  
  
  config.vm.provider "virtualbox" do |vb|  
    vb.linked_clone = true  
  end  
  
  config.vm.provider "virtualbox" do |vb|  
    # Display the VirtualBox GUI when booting the machine  
    # vb.gui = true  
  
    # Customize the amount of memory on the VM:  
    # vb.memory = "4096"  
  end  
  
  config.vm.define "honeypot" do |machine|  
    machine.vm.hostname = "honeypot"  
    config.ssh.username = "vagrant"  
    #machine.vm.synced_folder ".", "/vagrant", disabled: true  
    machine.vm.provision "ansible" do |ansible|  
      ansible.playbook = "playbook.yml"  
    end  
  end  
end
```

Ambiente di Deception

L'intera infrastruttura di test è stata implementata tramite Vagrant, il cui provisioning è stato eseguito tramite task **Ansible**, che fornisce:

- un ambiente Linux **isolato** e **riproducibile**;
- Provisioning automatico di pacchetti e dipendenze tramite Ansible;



The screenshot shows a terminal window titled "Terminal - matteo@matteo-ThinkPad-T590: ~". The window contains the following text output from an SSH session:

```
File Edit View Terminal Tabs Help
matteo@matteo-ThinkPad-T590:~$ ssh user@127.0.0.1 -p 2222 -i /home/matteo/large/Honeypot/.vagrant/machines/honeypot/virtualbox/private_key
Linux honeypot 6.1.0-41-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.158-1 (2025-11-09) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec  6 16:29:44 2025 from 10.0.2.2
user@honeypot:/home/user$
```

Ambiente di Deception

L'intera infrastruttura di test è stata implementata tramite Vagrant, il cui provisioning è stato eseguito tramite task **Ansible**, che fornisce:

- un ambiente Linux **isolato e riproducibile**;
- Provisioning automatico di pacchetti e dipendenze tramite Ansible;

```
Terminal - vagrant@honeypot: ~
File Edit View Terminal Tabs Help
ok: [honeypot]

PLAY RECAP ****
honeypot : ok=18    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

==> honeypot: Machine 'honeypot' has a post `vagrant up` message. This is a message
==> honeypot: from the creator of the Vagrantfile, and not from Vagrant itself:
==> honeypot:
==> honeypot: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports
matteo@matteo-ThinkPad-T590:~/large/Honeypot$ vagrant ssh
Linux honeypot 6.1.0-41-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.158-1 (2025-11-09) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec  6 16:52:48 2025 from 10.0.2.2
vagrant@honeypot:~$ 
```

Ambiente di Deception

L'architettura di deception implementata all'interno dell'ambiente virtualizzato Vagrant, è basata su due componenti principali:

fakeshell.py

Simula un terminale Bash ad alta interazione, ma con un comportamento completamente controllato, inoltre :

- Mostra un prompt realistico: **user@hostname:/cwd\$**
- Esegue i comandi tramite /bin/bash (anche quelli interattivi)
- Logga ogni comando in /var/log/fakeshell.json in formato JSON:
- Timestamp, ip, user, cwd, cmd
- Fornisce al Defender un flusso di comandi “reali” da usare per la predizione

```
def log_command(cmd, cwd):
    entry = {
        "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
        "ip": ip,
        "user": user,
        "cwd": cwd,
        "cmd": cmd
    }
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write(json.dumps(entry) + "\n")
```

```
while True:
    symbol = "#" if user == "root" else "$"
    prompt = f"\033[1;32m{user}@{hostname}\033[0m:\033[1;34m{cwd}\033[0m{symbol} "
    cmd = input(prompt)
    if not cmd.strip():
        continue
    log_command(cmd, cwd)
    # gestione cd ...
    # esecuzione comando via pty.fork() + os.execve("/bin/bash", ["bash", "-c", cmd], env)
```

Ambiente di Deception

L'architettura di deception implementata all'interno dell'ambiente virtualizzato Vagrant, è basata su due componenti principali:

defender.py

Implementa la logica difensiva. Il suo compito è:

- Seguire in tempo reale il file di log **JSON** generato dalla FakeShell;
- Utilizzare un modello Gemini con approccio RAG per predirre i prossimi comandi plausibili;
- Per ogni comando predetto, generare una descrizione di difesa e un **artefatto** (file) da collocare in percorsi realistici;
- Materializzare questi artefatti nel **file system** reale Quando l'attaccante inserisce il comando successivo, mantenere solo la branch coerente e ripulire gli artefatti legati alle predizioni sbagliate.

```
def follow_log(path: str):
    while not os.path.exists(path):
        print(f"[WAIT] In attesa che esista il file di log: {path}")
        time.sleep(1)
    with open(path, "r", encoding="utf-8") as f:
        f.seek(0, os.SEEK_END)
    while True:
        line = f.readline()
        if not line:
            time.sleep(0.1)
            continue
        try:
            entry = json.loads(line.strip())
        except json.JSONDecodeError:
            continue
        yield entry
```

```
def main():
    load_commands_state()
    load_active_artifacts()
    print("[*] Defender runtime attivo.")
    print("[*] Ascolto log:", HONEYBOT_LOG)
    for entry in follow_log(HONEYBOT_LOG):
        handle_new_command(entry)
```

Ambiente di Deception

Predizione e generazione delle difese

```
def handle_new_command(entry: Dict[str, Any]):  
    session_key = make_session_key(entry)  
    cmd = entry.get("cmd", "").strip()  
    if not cmd:  
        return  
    print(f"[{datetime.now().isoformat()}] session={session_key} cmd={cmd}")  
    cleanup_other_branches(session_key, cmd)  
    update_history(session_key, cmd)  
    predictions = predict_next_commands(session_key)  
    print(f" → Predizioni: {predictions}")  
    plan_and_apply_defenses(session_key, predictions)
```

```
def create_defense_for_predicted_command(command: str, session_key: str) → Dict[str, Any]:  
    cmd_safe = command.replace("%", "%")  
    prompt = f"""  
        You must output ONLY a JSON object.  
        {{  
            "description": "short description of the defense",  
            "intended_path": "/realistic/system/path/that/an/attacker/would_expect",  
            "artifacts": [  
                {{  
                    "path": "defense_artifacts/<SAFE_FILENAME>",  
                    "content": "<FILE CONTENT>"  
                }}  
            ]  
        }}  
        Generate JSON for predicted command: "{cmd_safe}"."  
        """  
    raw = query_gemini(prompt, model_name=GEMINI_MODEL)  
    try:  
        defense = json.loads(raw)  
    except:  
        defense = { ... fallback ... }  
    # fix intended_path e artifacts ...  
    return defense
```

handle_new_command()

create_defense_for_predicted_command()

Terminal - matteo@matteo-ThinkPad-T590: ~/large/Honeypot

File Edit View Terminal Tabs Help

matteo@matteo-ThinkPad-T590:~/large/Honeypot\$

Terminal - matteo@matteo-ThinkPad-T590: ~

File Edit View Terminal Tabs Help

matteo@matteo-ThinkPad-T590:~\$ ssh user@127.0.0.1 -p 2222 -i /home/matteo/large/Honeypot/.vagrant/machines/honeypot/virtualbox/private_key

Conclusioni

- Honeypot passa da **passivo** a **predittivo** con LLM + RAG.
- LLM legge i comandi loggati, costruisce il contesto e **predice il prossimo comando (Top-k)**.
- In base alle predizioni, il defender **crea file/artefatti fake realistici** prima che l'attaccante li richieda.
- Con RAG (ChromaDB + Cowrie) le predizioni sono **più accurate e meno fantasiose** rispetto al solo Top-k.
- Limite principale: dipende dai pattern visti nel dataset; attacchi nuovi o rari restano difficili da prevedere.

GRAZIE PER L'ATTENZIONE