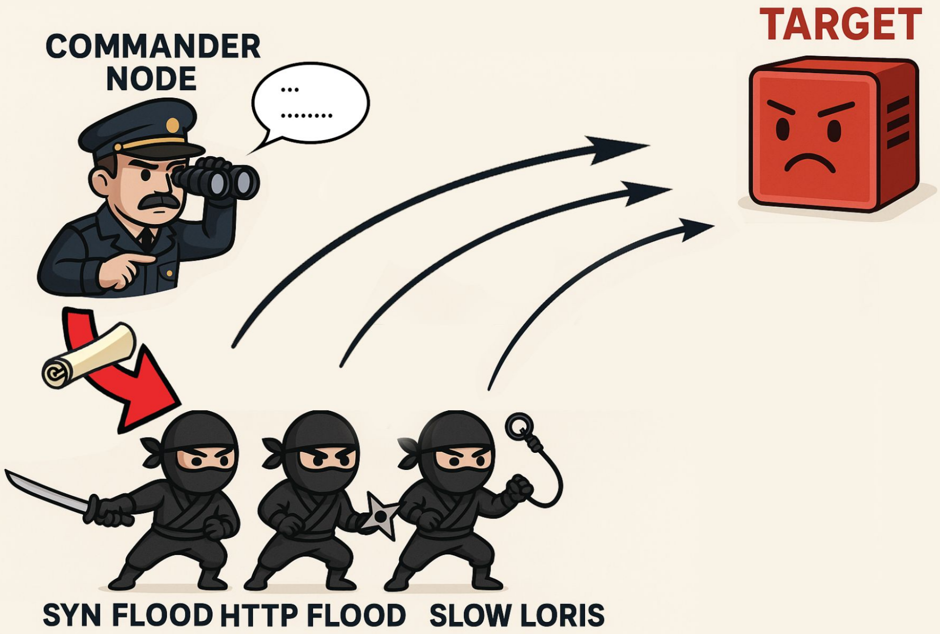
LLM-Driven DDoS Simulation: An AI-Enhanced Framework for Distributed Attack Orchestration

630 Project – Research Prototype Presentation

William Qiu

(definitely not a botnet)



Scope & Terminology

- Large Language Model(LLM)
- Fine Tune(Fine Tuned LLM)
- Distributed Denial of Service (DDOS)
- Hacker: In this project/presentation, we use the term hacker to the broadest meaning(whitehat, blackhat, etc).

- Scope:
    - Limited Resource

Motivation & Problem Statement

Why simulate DDoS in a new way?

- **Static scripts dominate existing DDoS simulations**, relying on pre-defined behaviors with minimal ability to react or adapt.
- **Real-world attackers don't follow scripts**—they observe, learn, and adapt in real-time.
- **Existing tools are non-adaptive and unrealistic**, failing to simulate intelligent adversaries.
- **High-performance, open-source LLMs open a new possibility**: using them to emulate the intelligence, unpredictability, and strategic thinking of real attackers.
- This project explores the feasibility and implications of **AI-driven, behaviorally rich, and low-cost** simulations that mimic adversarial decision-making, potentially offering a **more accurate and challenging testbed** for cyber defense systems.

Research Gap

🔍 Existing Research Landscape

ShieldGPT: This framework utilizes LLMs for DDoS mitigation by combining traffic representation, domain knowledge, and role representation to generate mitigation strategies.

DrLLM: This approach employs LLMs in a zero-shot learning context for DDoS resistance, focusing on prompt engineering and progressive role reasoning

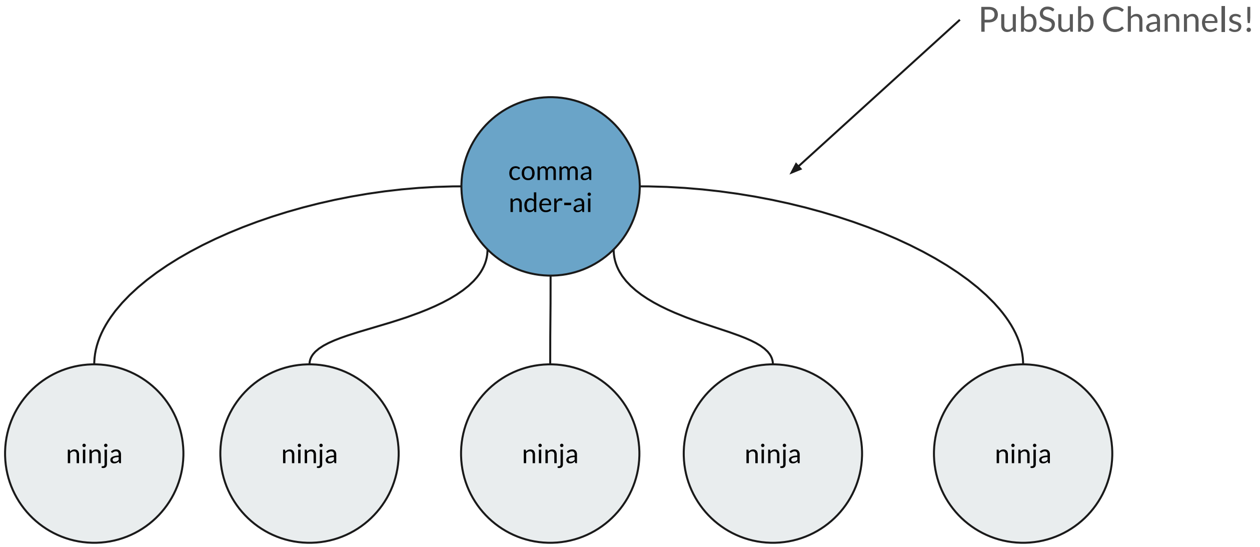
Similarly, **Copilot for Secuity**.

AI phishing attackes: voice clone, deepfake, etc.

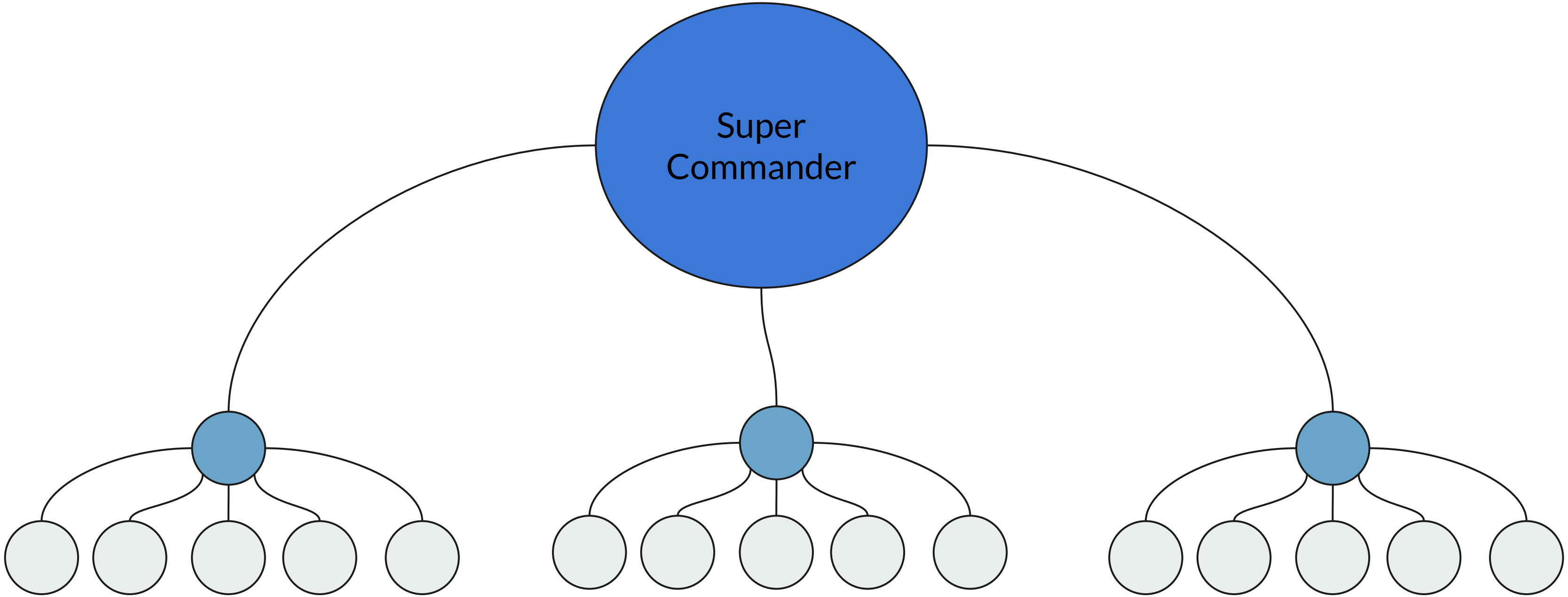
Project Objective

- Build a research-grade LLM-based system to simulate DDoS attack patterns
- Showcase intelligent behavior coordination using a Commander/Ninja model
- Use mainstream LLMs (e.g., Mistral 7B in this case) in a resource-constrained but functional setting

System Architecture Overview



System Architecture Overview



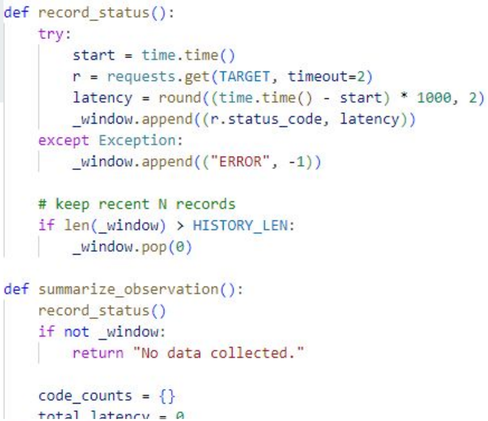
Role of LLMs

How LLMs are embedded:

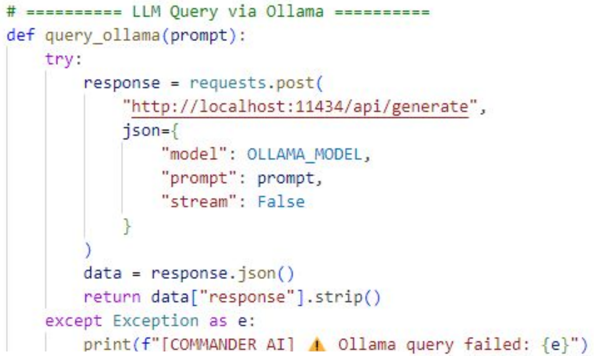
- Commander: interprets logs/status, generates attack plans
- Ninjas: execute commands, use tools (e.g., curl, slowloris), give feedback

Commander Node

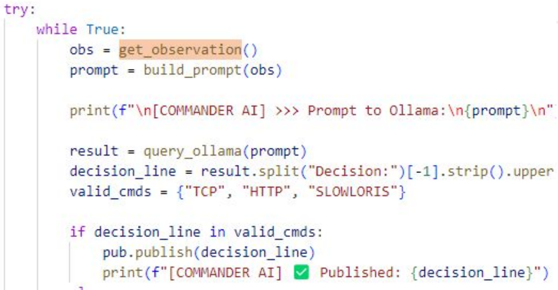
- Monitoring
- LLM-empowered



```python
def record_status():
    try:
        start = time.time()
        r = requests.get(TARGET, timeout=2)
        latency = round((time.time() - start) * 1000, 2)
        _window.append((r.status_code, latency))
    except Exception:
        _window.append(("ERROR", -1))

    # keep recent N records
    if len(_window) > HISTORY_LEN:
        _window.pop(0)

def summarize_observation():
    record_status()
    if not _window:
        return "No data collected."

    code_counts = {}
    total_latency = 0
```



```python
# ========== LLM Query via Ollama ==========
def query_ollama(prompt):
    try:
        response = requests.post(
            "http://localhost:11434/api/generate",
            json={
                "model": OLLAMA_MODEL,
                "prompt": prompt,
                "stream": False
            }
        )
        data = response.json()
        return data["response"].strip()
    except Exception as e:
        print(f"[COMMANDER AI] ⚠️ Ollama query failed: {e}")
```



```python
try:
    while True:
        obs = get_observation()
        prompt = build_prompt(obs)

        print(f"\n[COMMANDER AI] >>> Prompt to Ollama:\n{prompt}\n"

        result = query_ollama(prompt)
        decision_line = result.split("Decision:")[-1].strip().upper
        valid_cmds = {"TCP", "HTTP", "SLOWLORIS"}

        if decision_line in valid_cmds:
            pub.publish(decision_line)
            print(f"[COMMANDER AI] ✅ Published: {decision_line}")
```

Commander Node-continued

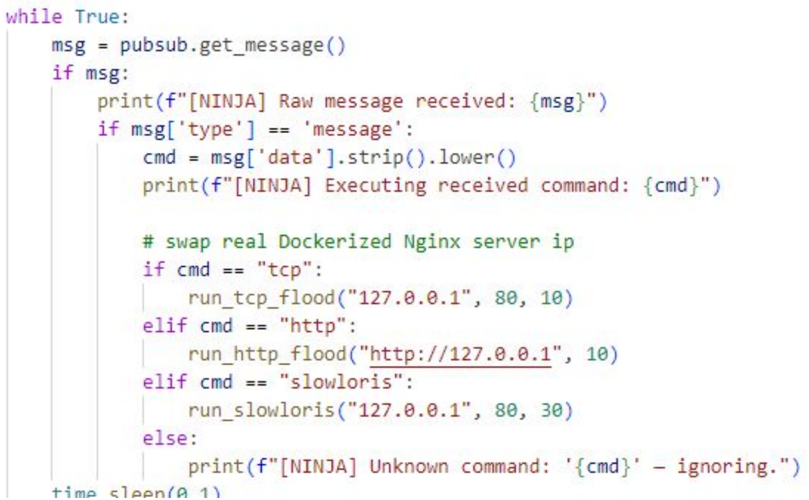
- LLM Lora Fine-tuned

```
SYSTEM_PROMPT = """You are an AI commander responsible for simulating cyberattacks based on observed
server behavior.

Your goal is to:
1. Analyze simplified logs or status reports from a target server.
2. Decide the most appropriate type of attack from this limited set:
   - TCP
   - HTTP
   - SLOWLORIS
3. Respond only with a decision line in this format:

Decision: <ONE OF TCP | HTTP | SLOWLORIS>

Do not include any commentary, reasoning, or additional output. Keep it short and in log-style.
```

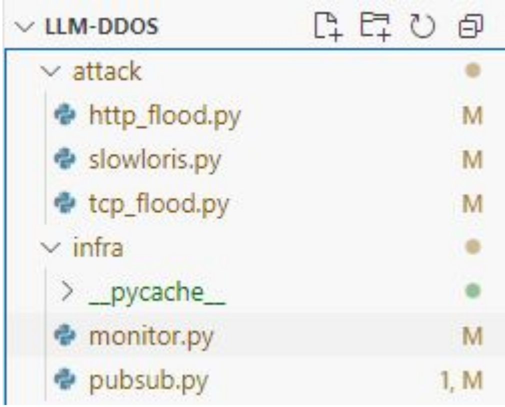
Ninja Node

Sub to Channel

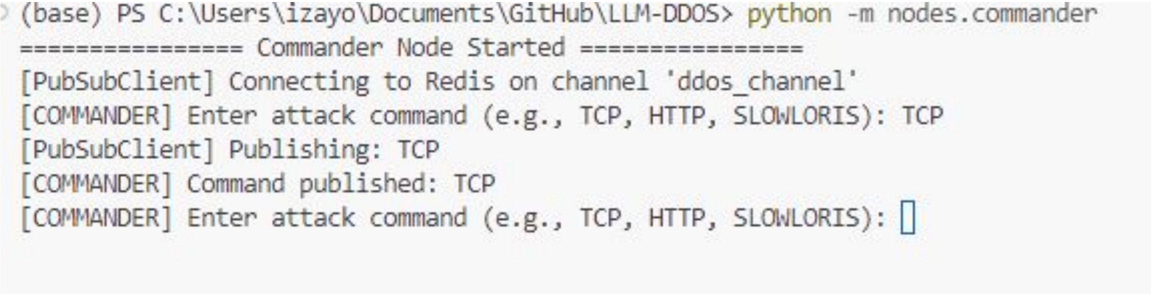
Use DDOS-scripts



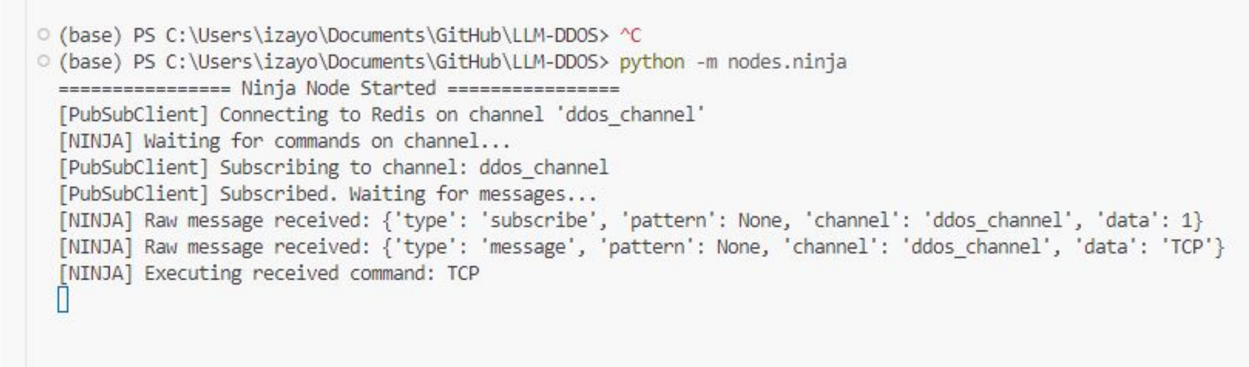
```
while True:
    msg = pubsub.get_message()
    if msg:
        print(f"[NINJA] Raw message received: {msg}")
        if msg['type'] == 'message':
            cmd = msg['data'].strip().lower()
            print(f"[NINJA] Executing received command: {cmd}")

            # swap real Dockerized Nginx server ip
            if cmd == "tcp":
                run_tcp_flood("127.0.0.1", 80, 10)
            elif cmd == "http":
                run_http_flood("http://127.0.0.1", 10)
            elif cmd == "slowloris":
                run_slowloris("127.0.0.1", 80, 30)
            else:
                print(f"[NINJA] Unknown command: '{cmd}' — ignoring.")
    time.sleep(0.1)
```



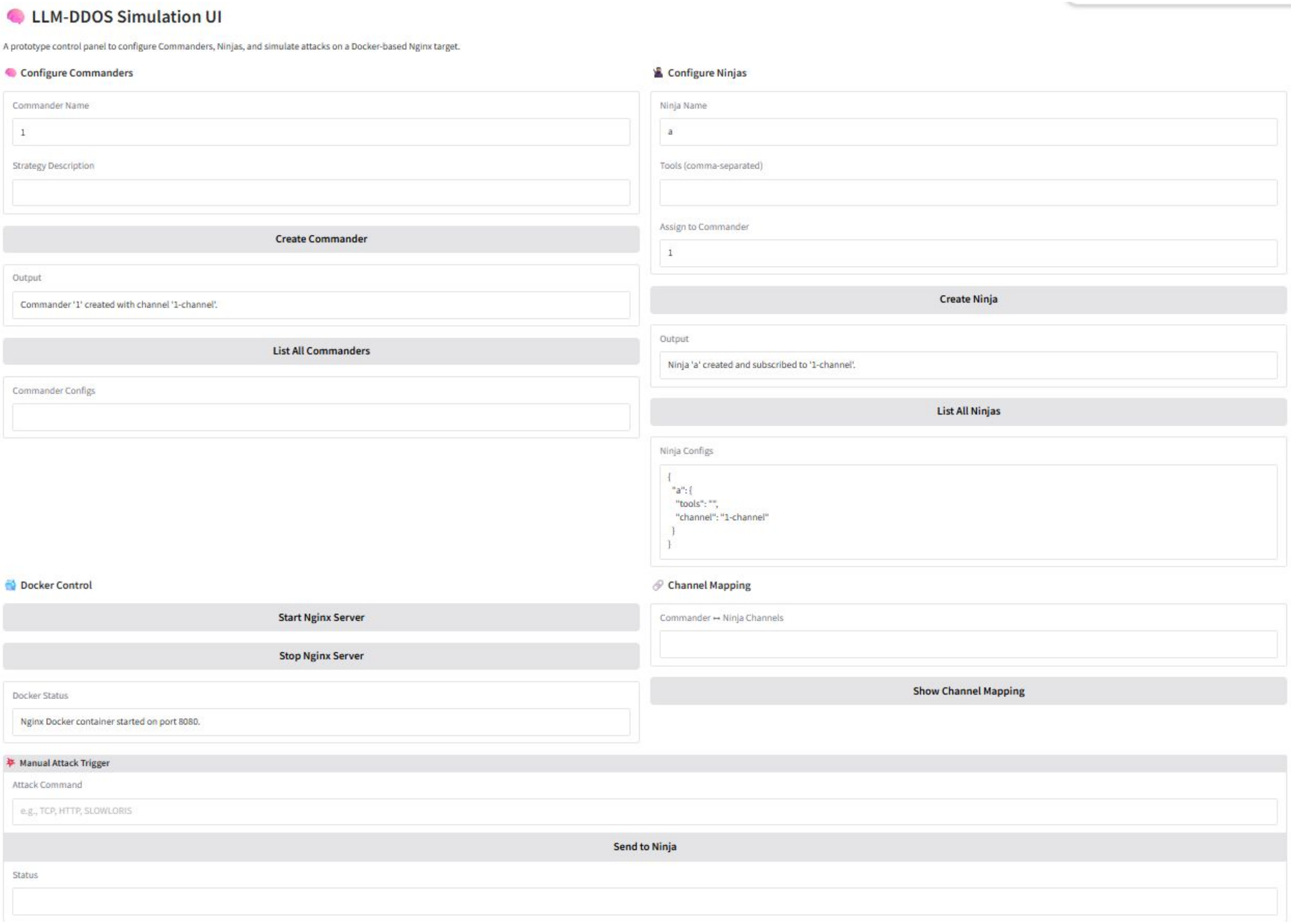
Demo



```
(base) PS C:\Users\izayo\Documents\GitHub\LLM-DDOS> python -m nodes.commander
================ Commander Node Started ================
[PubSubClient] Connecting to Redis on channel 'ddos_channel'
[COMMANDER] Enter attack command (e.g., TCP, HTTP, SLOWLORIS): TCP
[PubSubClient] Publishing: TCP
[COMMANDER] Command published: TCP
[COMMANDER] Enter attack command (e.g., TCP, HTTP, SLOWLORIS):
```



```
○ (base) PS C:\Users\izayo\Documents\GitHub\LLM-DDOS> ^C
○ (base) PS C:\Users\izayo\Documents\GitHub\LLM-DDOS> python -m nodes.ninja
  ================ Ninja Node Started ================
  [PubSubClient] Connecting to Redis on channel 'ddos_channel'
  [NINJA] Waiting for commands on channel...
  [PubSubClient] Subscribing to channel: ddos_channel
  [PubSubClient] Subscribed. Waiting for messages...
  [NINJA] Raw message received: {'type': 'subscribe', 'pattern': None, 'channel': 'ddos_channel', 'data': 1}
  [NINJA] Raw message received: {'type': 'message', 'pattern': None, 'channel': 'ddos_channel', 'data': 'TCP'}
  [NINJA] Executing received command: TCP
```



LLM-DDOS Simulation UI

A prototype control panel to configure Commanders, Ninjas, and simulate attacks on a Docker-based Nginx target.

Configure Commanders

Commander Name: 1

Strategy Description:

Output: Commander '1' created with channel '1-channel'.

Commander Configs:

Configure Ninjas

Ninja Name: a

Tools (comma-separated):

Assign to Commander: 1

Output: Ninja 'a' created and subscribed to '1-channel'.

Ninja Configs:

```
{
  "a": {
    "tools": "",
    "channel": "1-channel"
  }
}
```

Docker Control

Docker Status: Nginx Docker container started on port 8080.

Channel Mapping

Commander ↔ Ninja Channels:

Manual Attack Trigger

Attack Command: e.g., TCP, HTTP, SLOWLORIS

Status:

Technical Stack

Python backend

Redis (Pub/Sub system)

Open-source LLMs (Mistral 7B via Ollama)

Docker/Nginx for target simulation

Future Direction

Scale Up:

- Simulate at real-world attack scale
- Deploy SuperCommander architecture for large-scale coordination

Defense Side:

- Develop adaptive defense agents based on learned attack patterns
- Explore LLM- or RL-powered agents that co-evolve with attackers

Co-Evolution Framework:

- Agents learn to attack and defend in parallel
- Periodically sync with a central brain to refine shared defense strategies