LLM-DDOS: An AI-Orchestrated Distributed Denial-of-Service Architecture

William Qiu
University of Oregon

Abstract

We propose LLM-DDOS, a novel architecture for orchestrating distributed denial-of-service (DDoS) attacks using large language models (LLMs) as strategic coordinators. Rather than simulating isolated attacks, this system establishes a scalable blueprint for AI-driven offensive operations in distributed environments. The architecture consists of a central Commander Node powered by a lightweight LLM (Mistral 7B), chosen for its deployability within limited hardware constraints. Although model-agnostic by design, the framework can accommodate larger LLMs (e.g., LLaMA 70B) for enhanced reasoning. The Commander continuously monitors a target server, interprets environmental signals, and dynamically generates attack directives, which are transmitted via a lightweight Pub/Sub channel to multiple Ninja Nodes executing standard DDoS tools (e.g., TCP flood, HTTP flood, Slowloris).

While the system has not been empirically evaluated due to time and resource limitations, we propose a comprehensive methodology for future benchmarking—including comparison against scripted baselines, cross-model performance evaluation, and AI-versus-AI experiments involving next-generation firewalls. We also outline a future roadmap involving co-evolving defense agents and hierarchical "SuperCommander" coordination across distributed attacker networks. This work contributes to an emerging area of AI-orchestrated cyber operations and offers a foundation for autonomous red-teaming frameworks and adversarial system design.

Keywords

LLM-DDOS, distributed systems, DDoS architecture, AI cyber operations, Mistral 7B, Pub/Sub, prompt engineering, red teaming, autonomous agents, co-evolution, cyber offense

1. Introduction

The rise of large language models (LLMs) such as GPT, Mistral, and LLaMA has redefined what artificial intelligence can accomplish in strategic decision-making. While their applications have flourished in natural language processing, education, and productivity tools, their use in offensive cybersecurity remains limited and largely auxiliary. Most current AI-assisted attacks—such as phishing campaigns, deepfakes, and voice cloning—are extensions of social engineering rather than technical agents capable of real-time orchestration.

This project introduces LLM-DDOS, an AI-orchestrated distributed denial-of-service architecture that goes beyond deception. It explores how LLMs can function as tactical control agents, autonomously coordinating cyberattacks across a distributed system. The core of the system is the Commander Node, powered by a lightweight model (Mistral 7B) and designed to monitor server conditions, generate strategy prompts, and disseminate attack directives to subordinate Ninja Nodes via Redis Pub/Sub.

Unlike prior LLM-based security tools—such as ShieldGPT, which explains traffic anomalies, or Microsoft Copilot for Security which supports human analysts—LLM-DDOS positions the model in an

autonomous command-and-control loop. Its architecture is designed to be extensible, model-agnostic, and capable of scaling across large attacker networks.

Although empirical evaluation is left for future work due to course constraints, we provide a roadmap that includes real-world simulations, model benchmarking, and the development of co-evolving AI defenders. We also propose an extension to a **SuperCommander** hierarchy, supporting distributed attacks against distributed infrastructures. This report surveys related work, details the architecture and implementation of LLM-DDOS, and concludes with a discussion of its broader implications and future evolution.

2. Related Work

The convergence of artificial intelligence and cybersecurity has produced a growing body of research exploring both offensive and defensive applications. However, the idea of using large language models (LLMs) as real-time tactical coordinators for distributed cyberattacks remains largely uncharted. To position LLM-DDOS in context, we examine three main areas of related work: (1) AI-assisted cyber operations, (2) DDoS frameworks and architectures, and (3) language models in decision-making systems.

2.1 AI-Assisted Cyber Operations

Prior efforts to apply AI in cybersecurity have predominantly focused on defensive tasks—including anomaly detection, threat classification, and intrusion prevention—using machine learning and deep learning. Notable systems such as DeepDefense and OpenAI's security collaborations apply neural models to detect malicious behavior in traffic or systems. On the offensive side, AI applications remain relatively narrow, often limited to penetration testing tools, CTF simulations, or reinforcement learning agents in environments like Facebook's CyberBattleSim. These approaches rely heavily on static policies or engineered state spaces.

LLM-DDOS diverges by embedding high-level, live reasoning within an LLM, treating environmental observations as prompts and using natural-language decision-making to coordinate distributed attacks in real time.

2.2 DDoS Tools and Architectures

Traditional DDoS attack tools such as LOIC, HOIC, Mirai, and Mēris have demonstrated massive scalability and impact in both academic and underground communities. These frameworks typically involve static coordination scripts or simple command-and-control systems. Meanwhile, academic studies have dissected advanced attack patterns including amplification attacks, application-layer floods, and slow-rate connection attacks like Slowloris and RUDY.

Although centralized coordination (e.g., **Booter-as-a-Service**) is a known design pattern in botnet architectures, these systems lack adaptive intelligence. In contrast, LLM-DDOS introduces adaptive orchestration, replacing hardcoded logic with flexible prompt-based control using a centralized LLM Commander Node and reactive Ninja Nodes.

2.3 Language Models for Strategy and Coordination

Recent work has shown that LLMs can serve as agents capable of strategic planning in abstract domains. Projects like Auto-GPT, CAMEL, and Chain-of-Thought prompting illustrate how LLMs can decompose

goals, reason step-by-step, and simulate collaborative decision-making. These capabilities inspired their use in cybersecurity settings.

However, most existing systems employ LLMs in a supportive or explanatory role rather than as active agents. For instance, ShieldGPT uses GPT-4 to explain DDoS attacks and generate textual mitigation strategies from traffic data, but it does not issue or execute commands autonomously. Similarly, Microsoft Copilot for Security integrates LLMs to assist analysts by summarizing alerts, suggesting remediation steps, and surfacing documentation—but again, it functions primarily as an **interactive knowledge base** rather than a control agent.

LLM-DDOS extends this landscape by using the LLM as a live operational brain, issuing direct tactical decisions and coordinating multiple nodes, thus representing a shift from assistive intelligence to **autonomous orchestration** in distributed cyber operations.

3. System Design

LLM-DDOS is architected as a modular, AI-orchestrated system that mirrors real-world distributed attack infrastructures but replaces manual coordination with a reasoning-driven LLM core. The architecture consists of three core components:

1. **Commander Node** – an LLM-based decision-making agent that observes, analyzes, and issues commands.
2. **Ninja Nodes** – lightweight worker nodes that subscribe to commands and execute attacks.
3. **Pub/Sub Communication Layer** – a Redis-backed channel system that enables decoupled, real-time message passing.

The following subsections describe each component and their interactions in detail.

3.1 Architecture Overview

At a high level, LLM-DDOS operates in a continuous control loop:

1. **Observe**: The Commander Node retrieves the current state of the target server.
2. **Analyze**: The LLM processes observations and generates strategic decisions via prompt completion.
3. **Command**: The resulting strategy is published over a Redis channel.
4. **Execute**: Ninja Nodes subscribed to the channel parse the command and execute an attack accordingly.

This decoupled architecture enables flexibility, horizontal scaling, and asynchronous coordination.

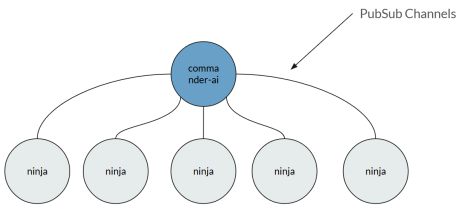

Figure 1. Architecture Overview

3.2 Commander Node

The Commander Node is the brain of the system. It runs an instance of a **fine-tuned Mistral 7B** LLM, configured with a role-specific system prompt that guides its behavior as a cyber tactician. The Commander:

- Periodically gathers observations about the target using the infra.monitor module.
- Summarizes raw metrics (CPU, latency, response codes, etc.) into a structured natural language report.
- Uses a predefined prompt template from llm.prompt_templates.py to feed the observation to the LLM.
- Receives a response in natural language (e.g., "Deploy TCP flood to port 80 immediately.") which it publishes to a designated Redis channel.

This loop runs every N seconds (e.g., LOOP_INTERVAL_SEC = 10) to maintain real-time responsiveness.

3.3 Ninja Nodes

Ninja Nodes are stateless, reactive workers that execute commands on arrival. Each Ninja:

- Subscribes to the Redis command channel initialized by the Commander Node.
- Upon receiving a message, parses it to determine the type of attack (tcp_flood, http_flood, slowloris, etc.).
- Launches the corresponding attack script with appropriate parameters.

Ninja Nodes are lightweight and require no internal logic or memory—making them easily deployable on containers or remote agents.

3.4 Pub/Sub Communication Layer

Redis Pub/Sub is used for its lightweight, low-latency characteristics and ease of integration. This messaging layer ensures:

- **Loose coupling**: Commander and Ninjas can be developed, updated, or restarted independently.
- **Real-time broadcast**: Commands are instantly pushed to all subscribing Ninja Nodes.
- **Scalability**: More Ninjas can be added without changing Commander logic or communication protocol.

Each Commander can have a uniquely named channel (e.g., commander-1-channel), allowing multi-Commander scenarios for future scaling experiments.

4. Implementation

The implementation of **LLM-DDOS** follows a minimal yet functional design to showcase the feasibility of AI-orchestrated cyber operations. Each component is implemented in Python using standard tools and interfaces, ensuring reproducibility and extensibility. The project is modularized into separate scripts and modules for observation, prompt construction, communication, and attack execution.

4.1 Technology Stack

Please See appendix

4.2 Commander Node Implementation

The Commander Node is implemented in commander_ai.py. Its core loop executes as follows:

Observation Collection
A monitoring module (infra.monitor.summarize_observation) queries the target server at regular intervals, collecting response latency, status codes, and connectivity data.

Prompt Generation
The observation summary is inserted into a system prompt defined in llm.prompt_templates.py. An example system message might include:

```
SYSTEM_PROMPT = "You are a commander AI analyzing server health..."
USER_PROMPT = f"Observation: {summary}\nWhat action should be taken?"
```

LLM Decision Making
The prompt is passed to an Ollama-hosted Mistral 7B model, either locally or via REST API. The generated output is expected in log-style natural language.

Command Publishing
The LLM's output is published to a Redis channel using the infra.pubsub.PubSubClient:

```
pub = PubSubClient(channel="commander-1-channel")
pub.publish("Launch TCP flood on port 80 now.")
```

4.3 Ninja Node Implementation

Each Ninja Node is implemented in ninja_node.py. Its lifecycle is simple:

Channel Subscription
The Ninja subscribes to a Commander's Redis channel and blocks until a message is received.

Command Parsing
The incoming string message is parsed using a simple keyword detection logic (e.g., if "TCP" in message → run tcp_flood.py).

Tool Invocation

Ninja launches subprocesses for the relevant attack script:

```
python tcp_flood.py --target 192.168.1.100 --port 80 --duration 10
```

This modular structure allows future expansion with additional tools or more advanced parsing logic using NLP.

4.4 Attack Scripts

Each attack script is a standalone Python file:

- **tcp_flood.py**: Sends a large volume of TCP connection requests to the target port.
- **http_flood.py**: Bombards the server with HTTP GET requests.
- **slowloris.py**: Opens many connections and sends headers slowly to exhaust server sockets.

These scripts are adapted for safe, controlled testing and include rate-limiting and dry-run modes for evaluation.

4.5 Error Handling and Logging

All components include lightweight logging using Python's logging module, including:

- Observation summaries
  Prompt contents
- Published commands
- Ninja attack confirmations

Failures (e.g., Redis connection errors, LLM timeouts) are logged and retried with exponential backoff.

5. Experimental Design (Planned for Future Work)

Due to time and resource limitations, this work does not include a completed evaluation. However, we outline future experiments designed to assess LLM-DDOS's decision quality, adaptability, and coordination capabilities.

5.1 Setup Overview

- **Target Server**: A dummy NGINX-based server with three defense tiers: no defense, basic firewall (IP filtering), and next-gen firewall (deep packet inspection and AI-powered blocking).

- **Commander Models**: Multiple LLMs (e.g., Mistral 7B, LLaMA 70B, DeepSeek-V3, Qwen), each using identical prompts in a shared environment. Loop intervals configurable.

- **LoRA Variants**: Compare fine-tuned models (e.g., aggressive vs. stealth roles) against their base versions for output quality and diversity.

- **Ninja Nodes**: Deployed across VMs or containers, each executing commands on receipt. Attacks include TCP flood, HTTP flood, and Slowloris, all with dry-run and logging support.

- **Communication Layer**: Redis (v6.2+) for low-latency Pub/Sub messaging, used to evaluate propagation delay and channel robustness.

5.2 Evaluation Metrics

To comprehensively evaluate LLM-DDOS across configurations, the following metrics are proposed:

Metric	Description
Decision Latency	Time from target observation to LLM-generated command publication

Command Delivery Time	Time between command publication and Ninja node reception
Execution Latency	Time from command receipt to attack initiation on Ninja
Target Server Load Impact	Change in server latency, error rate, or connection count during attack
LLM Prompt Completion Time	Time taken by each model to respond to observation input
Command Validity Rate	% of commands interpretable and actionable by Ninja nodes
Adaptability Score	Change in strategy quality as server state shifts over time
Model Cost Efficiency	Performance vs compute/memory tradeoff across LLM sizes
Mitigation Breach Rate	For NGFW setups: % of attacks successfully bypassing AI-driven firewall logic
Attack Diversity Index	Degree of variation in LLM-generated strategies (e.g., tool usage, timing)

6. Evaluation Guidance and Future Work

To validate and expand LLM-DDOS, we recommend the following grouped evaluations:

6.1 Model Comparisons

- **Baseline vs. LLM**: Compare traditional scripted attacks to LLM-generated strategies under each defense level.
- **Cross-Model Benchmarking**: Test lightweight (Mistral 7B) and heavyweight (LLaMA 70B) models for trade-offs in performance and latency.
- **Fine-Tuning Impact**: Evaluate LoRA-finetuned models for strategy diversity, clarity, and consistency.

6.2 Defense Interaction

- **Escalation Tests**: Observe Commander adaptation as the target defense shifts from basic to NGFW during an active attack.
- **Mitigation Effectiveness**: Measure success and failure rates under various defense configurations.

6.3 Orchestration and Adaptation

- **Multi-Commander Coordination**: Explore redundancy, conflict resolution, and federated command schemes in multi-Commander scenarios.
- **Real-Time Adaptation Logging**: Track prompt evolution, decision consistency, and state-awareness across multiple iterations.

7 Limitations and Coordination Challenges

While LLM-DDOS successfully demonstrates prompt-based orchestration, it faces several technical and ethical limitations:

- **LLM Latency**: Even when locally deployed (e.g., via Ollama), inference time—especially for larger models—introduces delays that may hinder real-time responsiveness in live offensive scenarios.
- **Prompt Fragility**: The system's performance depends heavily on prompt quality. Small variations in phrasing or formatting can significantly affect output. Structured prompting (e.g., few-shot examples or JSON-based schemas) may improve consistency.
- **Parsing Limitations**: Ninja Nodes currently rely on keyword-based command parsing. This is brittle under ambiguous or unexpected LLM outputs. Future versions should adopt structured outputs or classifier-based parsing for better reliability.

7.1 Ethical Considerations

Although LLM-DDOS is confined to a controlled academic environment, its architecture exposes potential misuse pathways:

- **Weaponization Risk**: Similar LLMs could be integrated into botnets or malware, increasing the sophistication and adaptability of malicious infrastructure.
  Red Team vs. Black Hat: The line between ethical simulation and autonomous attack generation is thin. Clear boundaries, sandboxing, and responsible disclosure practices are essential.
- **Unpredictable Emergence**: As LLM reasoning capabilities scale, unpredictable or unsafe strategies could arise from prompt-based control alone. Guardrails and behavior boundaries must be part of any future deployment.

7.2 Toward Multi-Agent AI Coordination

LLM-DDOS provides a foundation, but future versions may support more complex, multi-agent dynamics:

- **Hierarchical Command Trees**: A layered system in which top-tier "Commander-of-Commanders" nodes manage regional leaders, enabling coordination across larger infrastructures.
- **Negotiating Agents**: Simulated attackers and defenders could engage in strategy negotiation or conflict resolution, enabling adversarial co-evolution.
- **Domain-Specific Language (DSL)**: Defining a formal output schema would enable more precise, interpretable, and composable coordination between LLMs and execution agents.

8. Conclusion

This work presents **LLM-DDOS**, a proof-of-concept architecture that leverages large language models for orchestrating distributed denial-of-service attacks. By positioning a fine-tuned LLM as a centralized reasoning agent and combining it with reactive Ninja Nodes and a lightweight Redis-based Pub/Sub layer, we demonstrate a novel approach to AI-driven cyber coordination.

Unlike conventional DDoS frameworks that rely on rigid scripting or human-operated control panels, LLM-DDOS introduces adaptability and autonomy to the attack planning process. The system is capable

of observing a dynamic environment, reasoning about optimal attack strategies in natural language, and issuing commands to a distributed network of executors with minimal human intervention.

While the implementation remains academic and constrained to a safe environment, the architecture illustrates how LLMs can be repurposed beyond conversational AI into real-time, high-level control agents in distributed systems. Our findings highlight not only technical feasibility but also the urgent need to consider the security implications of LLM autonomy in adversarial settings.

LLM-DDOS opens new research directions in cyber-physical simulation, multi-agent coordination, and AI security tooling. As language models become increasingly capable and accessible, architectures like this one may form the blueprint for next-generation red-teaming platforms—or, if left unchecked, more destructive automated systems.

9. Lessons Learned

Building LLM-DDOS surfaced important lessons about LLM-driven coordination and the architecture's broader implications.

- **Effectiveness vs. Fragility**: Mistral 7B often produced tactically sound commands from basic prompts, but small variations in input structure or noise could cause inconsistent or invalid outputs. Prompt robustness remains critical.
  Simplicity Works: Despite its reliance on basic components (Redis, Python scripts, etc.), the system performed surprisingly well, proving that strategic orchestration doesn't require complex infrastructure.
- **Error Handling is Vital**: Unlike traditional scripts, LLM decisions are non-deterministic. Robust logging, fallback logic, and output validation are essential to ensure operational reliability.
- **Ethical Ambiguity**: AI-orchestrated attacks, even in simulation, raise red flags. Any future development must account for dual-use risks, reinforce sandboxing, and include open ethical dialogue.
- **Architecture Trumps Model Size**: The most impactful factor was not LLM scale, but system design. Modular separation of reasoning, sensing, and execution made the framework functional and extensible—regardless of the specific model used.

10. Future Work

To evolve LLM-DDOS into a robust simulation framework, we propose several key research directions:

- **Scale-Up and Hierarchical Control**: Simulate larger attack networks with thousands of nodes and introduce a "SuperCommander" layer to orchestrate distributed agents, especially in scenarios targeting decentralized infrastructures like CDNs.
- **Adaptive and Co-Evolving Defense**: Build RL- or LLM-based defensive agents that not only react but learn from ongoing attacks. Future systems could co-evolve with LLM-DDOS through continuous training loops.
- **Dual-Agent Frameworks**: Deploy attacker and defender agents in parallel, using shared memory or centralized synchronization ("global brain") to simulate emergent AI conflict and adaptive defense strategies.

These enhancements would transform LLM-DDOS from a controlled demo into a dynamic platform for studying AI-based cyber conflict, autonomy, and multi-agent evolution.

11. References

1. **Mistral AI**. *Introducing Mistral 7B*. https://mistral.ai/news/introducing-mistral-7b/

2. **Ollama**. *Run and deploy large language models locally*. https://ollama.com

3. **OpenAI**. *GPT-4 Technical Report*. https://openai.com/research/gpt-4

4. Wang, T., Xie, X., Zhang, L., Wang, C., Zhang, L., & Cui, Y. (2024). *ShieldGPT: An LLM-based Framework for DDoS Mitigation*. In Proceedings of APNet 2024. https://doi.org/10.1145/3663408.3663424

5. Microsoft. *Copilot for Security*. https://www.microsoft.com/en-us/security/business/microsoft-copilot-for-security

6. Hu, X., Zhang, T., Liu, X., et al. (2024). *DrLLM: Towards a Dialogue Agent for Cybersecurity Experts*. arXiv preprint. https://arxiv.org/abs/2402.00127

7. Anderson, B., Paul, S., & McGrew, D. (2016). *Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity*. In Proceedings of the ACM SIGKDD Workshop on Artificial Intelligence and Security.

8. Alasmary, W., Alhaidari, F., & Alzahrani, A. (2021). *A Survey of AI Techniques for DDoS Detection and Mitigation*. *Journal of Cybersecurity and Privacy*, 1(1), 1–22.

9. Kim, H., & Kim, D. (2020). *Using Deep Learning for Real-Time DDoS Detection in Software-Defined Networks*. *Computers & Security*, 89, 101682.

10. Microsoft Research. *CyberBattleSim: Open-source Simulation Environment for Cybersecurity Research*. https://github.com/microsoft/CyberBattleSim

11. Torantulino. *Auto-GPT: Experimental Open-Source Autonomous Agent Based on GPT-4*. https://github.com/Torantulino/Auto-GPT

12. Redis. *Publish/Subscribe Messaging*. https://redis.io/docs/manual/pubsub/

Appendix

- **Language**: Python 3.10+
- **LLM Backend**: Ollama running Mistral 7B
- **Messaging**: Redis (Pub/Sub model)
- **Attack Tools**: Custom TCP flood, HTTP flood, and Slowloris scripts
- **Monitoring**: requests and system profiling via a custom infra.monitor module
- **LLM Prompt Management**: llm.prompt_templates.py
- **Containerization**: Optional Docker setup for distributed deployment
- **Server**: Nginx
- **UI**: Gradio