RELATÓRIO TÉCNICO: Avaliação de LLMs Locais para Automação de SOC e Threat Hunting

Autor: Nicollas Avila **Data:** Dezembro de 2025 **Contexto:** Projeto de Pesquisa (PIBIC) - Engenharia de Software

---

1. Objetivo

Avaliar a viabilidade técnica e a precisão de **Grandes Modelos de Linguagem (LLMs)** rodando localmente (on-premise) para atuar como assistentes de Nível 1 em um Centro de Operações de Segurança (SOC). O foco principal é a análise automática de logs JSON brutos e a geração de Playbooks de Resposta a Incidentes.

2. Metodologia e Ambiente

2.1. Infraestrutura de Hardware

Os testes foram conduzidos em dois cenários para validar a escalabilidade:

- **Cenário A (Notebook):** Processamento via CPU.
- **Cenário B (Desktop):** Processamento com aceleração de GPU Dedicada.
    - *Resultado:* O uso de GPU reduziu o tempo de inferência em aproximadamente **73%** (de ~375s para ~102s no modelo Llama 3.1), viabilizando o uso em tempo real.

2.2. Stack Tecnológico

- **Engine de IA:** Ollama (versão 0.5.4).
- **Linguagem:** Python 3.12.
- **Bibliotecas Principais:**
    - `ollama`: Para interação direta e gerenciamento de modelos customizados.
    - `litellm`: Para padronização de chamadas entre diferentes arquiteturas de modelos.
- **IDE:** VS Code com extensões de produtividade.

---

3. Engenharia de Software e Arquitetura

Para garantir a reprodutibilidade e a manutenção do código, o projeto evoluiu de scripts isolados para uma arquitetura modular.

3.1. Centralização de Configurações (`config.py`)

Foi criado um módulo dedicado para separar a lógica de programação (Python) das regras de negócio (Prompts). Isso permitiu iterar rapidamente nas instruções sem alterar o código-fonte dos testes.

3.2. Data Engineering (Curadoria de Dados)

Durante a fase de validação, identificou-se que os logs originais careciam de campos explícitos de IoCs.

- **Ação:** Enriquecimento manual do dataset (`log2.json`).
- **Alteração:** Inclusão explícita do campo `TargetHash` contendo o hash malicioso, simulando um log real de EDR.

---

4. Engenharia de Prompt (Prompt Engineering)

O prompt evoluiu de uma instrução simples para uma estrutura complexa baseada nas melhores práticas (Anthropic/XML).

4.1. Técnicas Utilizadas

1. **Role Prompting:** Definição clara da persona ("Analista de SOC Sênior").
2. **XML Tagging:** Uso de tags (`<log_data>`, `<playbook>`) para delimitar escopo.
3. **Chain of Thought:** Instrução explícita (`<thinking>`) forçando o raciocínio antes da resposta.
4. **Explicit Field Extraction:** Ordem direta para buscar chaves específicas no JSON.

---

5. Modelos Avaliados

Foram selecionados modelos na faixa de 7 a 8 Bilhões de parâmetros (7B/8B):

1. **Llama 3.1 (8B) - Meta:** Baseline (Generalista).
2. **Mistral (7B) - Mistral AI:** Comparativo de arquitetura.
3. **Qwen 2.5 (7B) - Alibaba:** Comparativo de performance.
4. **Llama-3-Cybersecurity (8B) - Fine-Tuned:** Modelo Especialista convertido para GGUF.

---

6. Resultados Obtidos

6.1. Performance (GPU)

- **Llama 3.1:** ~102 segundos.
- **Mistral:** ~141 segundos.
- **Llama-3-Cyber:** ~75 segundos (Otimizado).

6.2. Precisão (Identificação de Hash)

O teste crítico consistiu na identificação de um hash oculto no JSON.

- **Generalistas:** Falharam ou ignoraram o campo.
- **Especialista (Llama-3-Cyber):** Identificou corretamente o hash `2d1f6f8a...` e sugeriu contenção apropriada após a correção do Prompt e dos Dados.

---

7. Conclusão

O experimento validou a hipótese de que modelos **Fine-Tuned (Especialistas)** superam modelos generalistas em tarefas de nicho como Cibersegurança, desde que apoiados por uma boa Engenharia de Dados e Prompts estruturados.