

RELATÓRIO TÉCNICO DE PESQUISA: Avaliação de SLMs Locais para Automação de SOC

Pesquisador: Nicol [Seu Sobrenome] **Projeto:** PIBIC - Engenharia de Software **Data:** Janeiro de 2026

1. Resumo Executivo

Este relatório documenta os experimentos realizados para validar a viabilidade de **Small Language Models (SLMs)** rodando localmente (On-Premise) para tarefas de Cibersegurança. O foco foi a análise automática de logs de segurança e a geração de Playbooks de Resposta a Incidentes, comparando modelos generalistas contra modelos especializados (*Fine-Tuned*).

2. Especificações de Hardware e Ambiente

Para medir o impacto do hardware no tempo de inferência (uma das métricas críticas solicitadas), os testes foram segregados em dois cenários distintos.

Cenário A: Processamento via CPU (Notebook)

Utilizado para estabelecer a linha de base (*baseline*) de performance em hardware restrito.

- **Dispositivo:** Notebook Pessoal
- **Processador (CPU):** Ryzen 5 3500U
- **Memória RAM:** 12GB DDR4
- **Aceleração Gráfica:** Nenhuma (Processamento exclusivo via CPU)
- **Sistema Operacional:** Windows 11

Cenário B: Processamento via GPU Dedicada (Desktop)

Utilizado para validar o ganho de performance com *Hardware Offloading*.

- **Dispositivo:** Desktop Workstation
- **Processador (CPU):** Ryzen 5 5500X
- **Memória RAM:** 16GB DDR4
- **Placa de Vídeo (GPU):** RX 6600 8GB
- **VRAM Utilizada:** ~5GB para alocação do modelo Q4_K_M.

Stack de Software

- **Engine de Inferência:** Ollama v0.5.4
- **Linguagem:** Python 3.12
- **Formato de Modelo:** GGUF (Quantização Q4_K_M)
- **Bibliotecas:** `ollama` (Python), `json`, `litellm`

3. Metodologia de Engenharia

3.1. Arquitetura Modular

O projeto evoluiu de scripts monolíticos para uma arquitetura de software desacoplada, visando facilitar a troca de prompts e modelos sem refatoração de código.

- **config.py**: Centralizador de configurações. Armazena o *System Prompt* (instruções da IA) e define o modelo alvo.
- **testar_cyber.py**: Script de validação unitária. Carrega o log, consulta a IA e realiza a verificação automática de IoCs (Hash).
- **dados/log2.json**: Dataset de teste enriquecido manualmente.

3.2. Engenharia de Dados (Data Engineering)

Durante a fase inicial, detectou-se que os modelos falhavam em encontrar evidências ("alucinação"). A investigação revelou que os logs originais não possuíam campos explícitos de hash.

- **Ação:** Enriquecimento do dataset (**log2.json**).
- **Implementação:** Inserção manual do campo "**TargetHash": "2d1f6f8a...**" para simular um log real de EDR (Endpoint Detection and Response).

3.3. Engenharia de Prompt (Prompt Engineering)

A qualidade da resposta foi drasticamente otimizada através da evolução do *System Prompt*, saindo de instruções em linguagem natural simples para uma estrutura técnica avançada baseada em tags XML.

Técnicas Aplicadas:

1. **LOG-ONLY Restriction:** Instrução explícita proibindo o uso de conhecimento externo não contido no log.
2. **Explicit Field Extraction:** Comando imperativo para buscar chaves específicas (**TargetHash**, **MD5**).
3. **Chain of Thought (CoT):** Estruturação do raciocínio da IA antes da geração da resposta final.

4. Modelos Avaliados

Foram comparados modelos na faixa de **7 a 8 Bilhões de Parâmetros (7B/8B)**, ideais para execução local (Edge AI).

Modelo	Tipo	Tamanho	Quantização	Função no Teste
Llama 3.1 (8B)	Base (Generalista)	8B	Q4_K_M	Baseline de comparação.
Mistral (7B)	Base (Generalista)	7B	Q4_K_M	Comparativo de arquitetura.
Qwen 2.5 (7B)	Base (Generalista)	7B	Q4_K_M	Comparativo de performance.
Llama-3-Cybersecurity	Fine-Tuned (Especialista)	8B	Q4_K_M	Modelo proposto para validação.

5. Resultados Obtidos

5.1. Performance de Inferência (Tempo)

O uso de GPU demonstrou ser mandatório para viabilidade operacional em um SOC, reduzindo o tempo de resposta drasticamente.

Cenário	Modelo	Tempo Médio de Resposta	Observação
A (CPU)	Llama 3.1	~375 segundos	Inviável para tempo real.
B (GPU)	Llama 3.1	~102 segundos	Ganho de 3.6x.
B (GPU)	Llama-3-Cyber	~75 segundos	Melhor tempo registrado.

Obs: O modelo Fine-Tuned (Cyber) apresentou menor latência, provavelmente devido a uma "convergência" mais rápida na resposta, gerando textos mais concisos e diretos.

5.2. Qualidade e Precisão (Teste do Hash)

O objetivo crítico era identificar um Hash Malicioso ([2d1f6f8a...](#)) oculto no JSON.

- **Modelos Generalistas (Llama Base/Mistral):**
 - Falharam consistentemente na extração exata do Hash.
 - Tendência a ignorar campos aninhados profundamente no JSON.
- **Modelo Especialista (Llama-3-Cyber):**
 - **Resultado:** SUCESSO (100% de Precisão).
 - **Análise:** Com o Prompt otimizado, o modelo não apenas extraiu o hash correto, como classificou a severidade do incidente e sugeriu ações de contenção (bloqueio de perfil) coerentes com a natureza da ameaça.

6. Conclusão Parcial e Próximos Passos

Os experimentos validaram que **Small Language Models (SLMs) Fine-Tunados** superam modelos generalistas de mesmo tamanho em tarefas específicas de cibersegurança, desde que apoiados por:

1. **Hardware com Aceleração (GPU).**
2. **Engenharia de Prompt Robusta (Contexto XML).**
3. **Dados Estruturados.**

Próximos Passos (Solicitação da Orientação):

- Implementar métricas de qualidade quantitativas ("LLM-as-a-Judge").
- Explorar o uso de LLMs maiores para gerar dados sintéticos e treinar o SLM (Knowledge Distillation).