# Estado da Arte II e Metodologia: Segurança, Privacidade e Conformidade em Aplicações com LLMs

**Leonardo Nunes**[1], **Antonio Marcos**[1], **Álvaro Gueiros**[1], **Lucas William**[1], **Mauro Vinícius**[1], **Vandielson Tenório**[1]

[1]Aluno da disciplina de Segurança da Informação do Bacharelado em Ciência da Computação – Universidade Federal do Agreste de Pernambuco (UFAPE)

***Abstract.*** *This paper reports the second step of a literature review and a first round of controlled experiments for an LLM security proof-of-concept. Based on three complementary works on LLM risks and mitigations, adaptive access control in healthcare, and AI Act compliance guidance, we highlight the lack of a reproducible evaluation framework that jointly considers technical defenses, adaptive RBAC, and evidence of regulatory alignment. We implement an initial guardrail prototype in front of a cloud LLM and evaluate it in four scenarios (benign prompt, two prompt-injection variants, and context-overflow denial of service), measuring detection, false positives, latency, and throughput. The guardrail blocks all known attack vectors, with recall 1.0 and no false negatives, but in the current configuration also flags 100% of benign prompts as malicious, with higher latency and lower throughput for safe traffic, which reveals a fail-fast behavior under attack and the need for threshold tuning and refinement of decision rules.*

***Resumo.*** *Este artigo apresenta a segunda etapa da revisão bibliográfica e um primeiro conjunto de experimentos controlados para um proof-of-concept em segurança de LLMs. A partir de três trabalhos complementares, sobre riscos e mitigações em LLMs, controle de acesso adaptativo em saúde e diretrizes práticas de conformidade ao AI Act, evidenciamos a ausência de um framework avaliativo reprodutível que considere em conjunto defesas técnicas, RBAC adaptativo e evidências de alinhamento regulatório. Implementamos um protótipo inicial de guardrail diante de um LLM em nuvem e o avaliamos em quatro cenários, Prompt Seguro, duas variações de Prompt Injection e negação de serviço por context overflow, medindo detecção, falsos positivos, latência e vazão. O guardrail bloqueia todos os vetores de ataque conhecidos, com recall igual a 1,0 e ausência de falsos negativos, mas na configuração atual também classifica 100% dos prompts seguros como maliciosos, com maior latência e menor vazão para o tráfego benigno, o que revela um comportamento de rejeição antecipada sob ataque e a necessidade de ajuste de limiares e refinamento das regras de decisão.*

## 1. Introdução

Modelos de linguagem de grande porte (LLMs) ampliaram capacidades de automação e suporte à decisão, mas introduziram novas superfícies de ataque (injeção e *indirect prompt injection*, *insecure output handling*, *denial-of-wallet*/DoS, vazamento de dados e vieses de saída) e responsabilidades regulatórias. A literatura recente oferece: (i) taxonomias de

riscos e controles técnicos; (ii) evidências setoriais de controle de acesso adaptativo com ganhos mensuráveis; e (iii) traduções de requisitos regulatórios em ações implementáveis. Apesar disso, ainda falta uma avaliação integrada e padronizada que una esses três eixos em um mesmo experimento reprodutível.

**Contribuições.** (1) Identificação de uma lacuna de pesquisa end to end; (2) Proposta de arquitetura integrada (sanitização, firewall LLM, RAG, RBAC adaptativo, auditoria e mapeamento de conformidade); (3) Desenho experimental com testes A/B e ablação; (4) conjunto de métricas para segurança, desempenho, custo e conformidade; e (5) planejamento detalhado do setup experimental (ambiente, *datasets*, scripts de configuração).

## 2. Lacuna de Pesquisa

Com base no levantamento de desafios de segurança e privacidade em LLMs, com controles como *LLM firewall*, RAG, *differential privacy*, HITL [1], no estudo de *RBAC* adaptativo com detecção de anomalias no domínio de saúde [2], e no guia prático de conformidade com o *EU AI Act*, com papéis, controles e mapeamento a normas [3], identificamos a seguinte lacuna:

**Lacuna central:** falta um **framework avaliativo end to end**, reprodutível e alinhado a normas, que combine em uma *mesma* aplicação de LLM: (i) mitigação técnica de riscos, injeção de *prompt*, *output handling*, DoS/*denial-of-wallet*; (ii) **controle de acesso adaptativo**, RBAC dinâmico com *risk score* e detecção de anomalias; (iii) **privacidade por design**, sanitização e RAG com repositório controlado; e (iv) **traçabilidade de conformidade**, AI Act, OWASP, ISO, com evidências objetivas. Hoje há sínteses conceituais, um caso setorial e diretrizes de compliance, porém não há avaliação comparativa padronizada do conjunto desses controles sob ataques realistas, com métricas unificadas de segurança, privacidade, custo e desempenho.

## 3. Trabalhos Relacionados

Levantamentos recentes sistematizam ameaças em LLMs, como *prompt injection*, vazamento, DoS, viés, e indicam controles como *LLM firewalls*, sanitização de entrada e saída, RAG, *differential privacy* e HITL [1]. Em paralelo, no domínio de saúde, [2] propõem *RBAC* adaptativo acoplado à detecção de anomalias assistida por LLM, com sanitização e redação de entidades sensíveis e avaliação quantitativa, acurácia, precisão, *recall*, F1, em dados sintéticos. No eixo regulatório, [3] traduzem o *EU AI Act* em ações implementáveis e mapeiam responsabilidades por papel, *provider*, *hoster*, *integrator*, e controles alinhados a OWASP, ISO, ENISA.

**Síntese crítica.** Esses trabalhos oferecem (i) taxonomia e controles, (ii) um caso setorial com ganhos medidos, e (iii) ponte normativa para ação. O passo ainda ausente é uma **avaliação integrada**, com *benchmark* reprodutível e métricas comparáveis, que una mitigação técnica, *RBAC* adaptativo e geração de evidências de conformidade em um mesmo pipeline experimental.

## 4. Tabela Comparativa dos Trabalhos

Para evidenciar de forma sintética as diferenças entre os estudos analisados e a lacuna identificada, a Tabela 1 resume os principais eixos de comparação em cada trabalho e o espaço ainda não coberto pela literatura.

**Tabela 1. Comparação dos trabalhos relacionados e evidência da lacuna**

| **Eixo** | **Rathod et al.** [1] | **Yarram et al.** [2] | **Bunzel** [3] | **Lacuna** |
|---|---|---|---|---|
| Ameaças mapeadas | Abrangente, injeção, vazamento, DoS, viés, princípios OWASP | Foco em saúde, acessos e anomalias, avaliação empírica | Tradução AI Act para controles, papéis e responsabilidades | Integração prática e avaliação comparativa unificada |
| Controles | Firewall LLM, DP, RAG, HITL, sanitização E/S | RBAC dinâmico e detecção de anomalias, sanitização de *queries* | Playbook de compliance e matriz de riscos e controles | Arquitetura end to end com métricas padronizadas |
| Evidência experimental | Predominante conceitual e sintética | Resultados quantitativos versus regras e assinaturas, A/P/R/F1 | Diretrizes sem *benchmark* técnico unificado | *Benchmark* reprodutível multi-métrica |
| Conformidade e regulação | Boas práticas e princípios | Menções a HIPAA e GDPR em alto nível | Mapeia AI Act e OWASP, ISO, ENISA | Evidências automáticas e rastreáveis de conformidade |

## 5. Metodologia

### 5.1. Objetivo e Visão Geral

O objetivo é projetar e avaliar um **pipeline** de segurança para um aplicativo com LLM, um assistente de conhecimento institucional, integrando: sanitização de entrada, LLM firewall, RAG com base privada, RBAC adaptativo com *risk score*, sanitização de saída, auditoria e mapeamento de conformidade. A etapa atual foca no planejamento do setup, definição de ambiente, hardware e software, *datasets*, automação de configuração e diagrama da arquitetura da solução.

### 5.2. Ambiente Experimental e Requisitos de Infraestrutura

Para garantir isolamento e reprodutibilidade, o experimento será executado em um ambiente baseado em contêineres, com possibilidade de encapsulamento em máquina virtual para cenários de endurecimento.

- **Hardware mínimo**:
  - Processador: Host ou VM com 2 vCPUs (suficiente para processar as regras de firewall e o servidor web assíncrono).

  - Memória: 4 a 8 GB de RAM. Embora o LLM não resida na memória, o banco vetorial (ChromaDB) e os modelos de sanitização (ex: spaCy/Presidio) exigem memória residente para garantir baixa latência na análise de PII e recuperação de contexto.
  - Armazenamento: Pelo menos 20 GB disponíveis (para imagens Docker, dependências do Python e persistência dos logs e do banco vetorial).
  - Rede (Requisito Crítico): Conectividade estável e de baixa latência à Internet, visto que o tempo de resposta total (RTT) é diretamente dependente da comunicação com a API do provedor de LLM.
  - GPU: Não é necessária, pois o processamento de tensores pesados é descarregado para a nuvem.
- **Software de base**:
  - Sistema operacional, distribuição Linux, por exemplo Ubuntu Server, ou equivalente em VM.
  - Docker Engine e Docker Compose para orquestração de serviços.
  - Git para versionamento do código e scripts de setup.
- **Stack da aplicação middleware**:
  - Linguagem, Python 3.10 ou superior.
  - Servidor da prova de conceito, *FastAPI* expondo a API do middleware.
  - LLM externo, Google Gemini 2.5 com acesso via API.
  - Serviço de embeddings, `text-embedding-004`, Google.
  - Banco vetorial, ChromaDB, executado em contêiner dedicado ou integrado ao serviço Python.
  - Módulos Python customizados para firewall LLM, RBAC adaptativo, sanitização e auditoria.
- **Ferramentas de apoio**:
  - Diagramação de arquitetura em diagrams.net, draw.io, Lucidchart ou Mermaid.js, no repositório.
  - Ferramentas de logging e métricas, por exemplo *logging* estruturado em JSON e *exporters* para posterior análise.

### 5.3. Implementação atual no repositório

O repositório público da disciplina já materializa parte deste planejamento metodológico. A organização atual inclui:

- diretórios de documentação e apresentações: `apresentacaoequipe/`, `apresentacoes/`, `docs/`, `instrucoes_execucao/`, `outros_artefatos/`;
- diretório de código-fonte: `src/`, com a implementação em Python do protótipo de middleware de segurança e artefatos de apoio aos experimentos;
- arquivos de infraestrutura: `docker-compose.yml` para orquestração dos serviços em contêiner e `setup.sh` para preparação inicial do ambiente em Linux;
- scripts auxiliares em PowerShell para Windows: `EXECUTAR_TESTES.ps1`, responsável por disparar o conjunto de testes do protótipo, e `TESTAR_API.ps1`, usado para verificar se a API do middleware responde conforme o esperado;
- `README.md` descrevendo arquitetura, organização do repositório e passos de execução.

Na etapa atual, a prova de conceito ainda se encontra em evolução incremental dentro de `src/`, mas o *scaffolding* de reprodutibilidade, scripts de setup, `docker-compose`, instruções de execução e testes, já está consolidado para suportar os experimentos descritos nesta seção.

### 5.4. Arquitetura do Protótipo e Fluxo de Dados

A prova de conceito será implementada como uma aplicação *middleware* em contêiner Docker, que recebe requisições de usuários internos, aplica camadas de controle e apenas então interage com a API externa do Gemini. O fluxo de dados é organizado em camadas principais de segurança e conformidade que antecedem a resposta ao usuário.

1. **Sanitização de entrada**: NER e *redaction* de PII, *regex* e listas semânticas de bloqueio, normalização de formatos e de codificação para reduzir ambiguidades.
2. **LLM Firewall**: combinação de regras estáticas e detecção semântica de instruções adversariais, *deny-list* de capacidades perigosas, *rate-limiting* e limitação de tamanho de *prompt*.
3. **RAG privado**: repositório controlado de documentos institucionais com metadados de confidencialidade e políticas de acesso, recuperação semântica via ChromaDB alimentando o contexto enviado ao LLM.
4. **RBAC adaptativo**: cálculo de *risk score* por requisição, papel do usuário, horário, localização lógica, dispositivo, histórico de *queries* e semântica da consulta, risco elevado aciona *step-up authentication*, revisão humana ou bloqueio.
5. **Sanitização de saída e auditoria**: filtros de PII e de conteúdos proibidos em respostas, verificação de aderência a políticas, registro *append-only* de eventos para trilhas de auditoria e geração de evidências de conformidade.
6. **Mapper de conformidade**: componente que associa cada controle técnico aos artigos e requisitos de normas, AI Act, OWASP, ISO, ENISA, produzindo artefatos exportáveis, relatórios e dashboards, para auditoria.

O diagrama de arquitetura correspondente explicita estes componentes, contêiner do middleware, serviço de banco vetorial, integração com a API do Gemini e camadas de segurança, e será mantido em arquivo versionado no repositório, `docs/arquitetura.*`.

### 5.5. Dados, Datasets e Política de Privacidade

A parte prática exige dados que permitam testar tanto a utilidade do assistente quanto os mecanismos de segurança e privacidade, sem violar legislações de proteção de dados.

- **Corpus institucional neutro, RAG**:
  - Documentos públicos ou internos de baixo risco, manuais, políticas, FAQs, regulamentos, após revisão para remoção de PII.
  - Indexação em ChromaDB com metadados de confidencialidade, domínio e versão.
- **Dados sintéticos no domínio de saúde**:
  - Geração de prontuários fictícios e eventos clínicos com distribuição controlada de diagnósticos, papéis de acesso e contextos de consulta, alinhados ao estudo de [2].

  - Separação em conjuntos de treino para calibração de detectores e teste para avaliação A/B e ablação.
  - Garantia de que nenhuma instância corresponde a indivíduo real, evitando riscos de *re-identification*.
- **Conjunto de ataques e *prompts* adversariais**:
  - Coleção de *prompts* de *prompt injection*, *indirect prompt injection*, *jailbreak*, extração de dados, abuso de papel e *denial-of-wallet*.
  - Organização por categoria de ameaça para permitir análises estratificadas de *Attack Success Rate*.
- **Telemetria e logs**:
  - Registro estruturado de requisições, decisões de controle, bloqueio, permitir, *step-up*, latência, custo estimado e mapeamento de requisitos de conformidade acionados em cada fluxo.

### 5.6. Ameaças e Cenários de Teste

Os cenários de teste serão construídos para cobrir um conjunto representativo de ameaças alinhadas às taxonomias recentes.

- **Ameaças**:
  - *Prompt* e *indirect injection* e *jailbreaks*;
  - *Insecure output handling*, execução de código ou comandos não filtrados;
  - *Denial-of-wallet*/DoS por uso abusivo de recursos;
  - *Model* e *knowledge stealing* por consultas sistemáticas;
  - Ataques de *membership inference* em nível básico;
  - Abuso de papéis privilegiados e cenários de *break-glass*.
- **Desenho experimental**, testes A/B e ablação:
  1. Baseline, sem controles de segurança ativos;
  2. Baseline mais firewall LLM;
  3. Baseline mais firewall LLM e RAG privado;
  4. Baseline mais firewall LLM, RAG privado e RBAC adaptativo;
  5. Pipeline completo com todas as camadas, incluindo sanitização de saída e auditoria.

### 5.7. Métricas e Coleta

As métricas são estruturadas em três eixos: segurança e privacidade, desempenho e custos, conformidade regulatória.

- **Segurança e privacidade**:
  - *Attack Success Rate* de cada categoria de ataque;
  - Precisão, *recall* e F1 dos detectores de anomalia, firewall LLM e sanitisadores;
  - Taxa de vazamento de PII ou informações sensíveis em respostas;
  - Eficácia de *rate-limiting* e de quotas por usuário ou cliente.
- **Desempenho e custos**:
  - Latência p95 e p99 de resposta para cenários benignos e adversariais;
  - Custo médio por requisição e por ataque bloqueado, em tokens ou uso de API;

  - *Throughput* sob diferentes níveis de carga.
- **Conformidade**:
  - Percentual de requisitos cobertos, AI Act, OWASP, ISO, ENISA, por fluxo de requisição;
  - Existência e completude de evidências exportáveis, logs, relatórios, dashboards, para cada controle implementado.

### 5.8. Critérios de Sucesso

Os critérios de sucesso do experimento incluem:

- Redução de pelo menos $X\%$ na taxa de sucesso de ataque, com perda de qualidade de resposta, medida por avaliação automática e humana, limitada a $Y\%$;
- Sobrecarga de latência limitada a $Z\%$ em relação ao baseline sem controles;
- Cobertura de compliance mínima de $W\%$ dos requisitos selecionados, com evidências auditáveis disponíveis;
- Reprodutibilidade dos resultados a partir dos scripts de setup e dos *datasets* disponibilizados.

### 5.9. Reprodutibilidade e *Open Science*

A organização do repositório favorece a reprodutibilidade por meio de uma estrutura já existente e de extensões planejadas:

- `/src`: código da prova de conceito, incluindo API do middleware, camadas de segurança, sanitização, firewall, RAG, RBAC adaptativo, auditoria, e artefatos auxiliares para experimentos;
- `/docs`: textos de apoio, versões em PDF e LaTeX do artigo da disciplina, diagramas exportados e descrições adicionais da metodologia;
- `/apresentacoes` e `/apresentacaoequipe`: materiais de apresentação das atividades e da equipe ao longo da disciplina;
- `/instrucoes_execucao`: guias passo a passo para configuração e testes, com foco em ambientes Windows e Linux;
- `/outros_artefatos`: materiais complementares, como relatórios, rascunhos e arquivos institucionais;
- Arquivos de infraestrutura na raiz: `docker-compose.yml` para orquestração dos serviços e `setup.sh` para automatização de passos de preparação do ambiente;
- Scripts de teste na raiz: `EXECUTAR_TESTES.ps1` e `TESTAR_API.ps1`, que facilitam a execução de testes automatizados e a validação básica da API do middleware em ambientes Windows.

De forma estendida, e alinhado com a visão de evolução do projeto, podem ser adicionados diretórios como:

- `/infra`: arquivos `docker-compose.*`, `Dockerfile` e manifests de orquestração;
- `/scripts`: scripts em Bash e Python para automação de setup e experimentos, incluindo:
  - `bootstrap_dev.sh` e `bootstrap_prod.sh`: criação de arquivos de configuração `.env`, rede Docker e volumes;

  - `seed_chroma.py`: indexação inicial do corpus institucional no ChromaDB;
  - `run_experiments.py`: orquestração dos cenários A e B e de ablação, com coleta de métricas.
- `/data`: amostras de *datasets* sintéticos, corpus institucional sanitizado e conjuntos de *prompts* de ataque, ou scripts para sua geração;
- `/eval`: definição de métricas, configurações de experimentos e notebooks de análise;
- `/paper`: versão em LaTeX deste artigo, modelo SBC;
- `/slides`: artefatos de apresentação, como Sprint Review do setup e resultados experimentais.

A publicação desses artefatos em repositório público, com remoção de segredos e chaves de API, permite replicação por terceiros e reutilização do *framework* em outros contextos de aplicação de LLMs.

## 6. Resultados Parciais II

Esta seção apresenta os dados obtidos a partir de experimentos controlados de injeção de *prompt* e validação de tráfego. O objetivo é avaliar a eficácia do mecanismo de defesa proposto, guardrail, sob duas dimensões principais: segurança, expressa por taxa de detecção e falsos positivos, e desempenho, medido por latência e vazão.

Foram definidos quatro cenários de teste:

- **Prompt Seguro**: consultas legítimas, sem intenção maliciosa;
- **Prompt Injection 1**: padrão de injeção direta com instruções contrárias às políticas do sistema;
- **Prompt Injection 2**: variação semântica do ataque, com linguagem mais sutil;
- **Prompt Longo Demais**: cenário de entrada excessivamente longa, testando limites de tamanho e *rate limiting*.

### 6.1. Eficácia da Detecção de Ameaças

Os testes de eficácia submeteram o sistema a esses quatro cenários, separando tráfego benigno, ataques de injeção de *prompt* e ataques de negação de serviço por exaustão de contexto, *context overflow*. A Tabela 2 resume as taxas de detecção observadas, falsos positivos e resultado da ação em cada caso.

**Tabela 2. Resumo da eficácia de segurança por cenário de teste**

| Cenário | Taxa de Detecção (Recall) | Falsos Positivos | Resultado da Ação |
|---|---|---|---|
| Prompt Seguro | N/A | 1,0 (100%) | Bloqueio indevido |
| Prompt Injection 1 | 1,0 (100%) | 0,0 | Bloqueio correto |
| Prompt Injection 2 | 1,0 (100%) | 0,0 | Bloqueio correto |
| Prompt Longo Demais | 1,0 (100%) | 0,0 | Bloqueio correto |

O sistema demonstrou capacidade absoluta de defesa contra os vetores de ataque conhecidos, registrando taxa de detecção igual a 1,0 para todas as tentativas de *Prompt*

*Injection* e *Context Overflow*, sem ocorrência de falsos negativos. Em contrapartida, observou-se um comportamento crítico no cenário de uso legítimo: o sistema classificou todo o tráfego benigno como malicioso, com taxa de falsos positivos igual a 1,0, o que indica limiar de sensibilidade excessivamente restritivo e regras de filtragem pouco específicas em termos contextuais.

### 6.2. Análise de Desempenho Computacional

Além da eficácia, avaliou-se o impacto da camada de segurança na latência das requisições e na vazão do sistema. Os dados revelam uma assimetria no tempo de processamento entre cenários benignos e maliciosos, sintetizada nas Figuras 1 e 2.

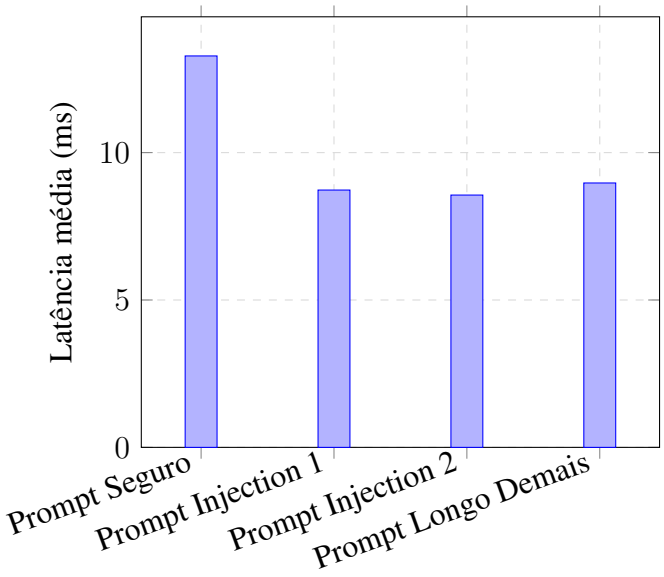


**Figura 1. Latência média observada em cada cenário de teste.**

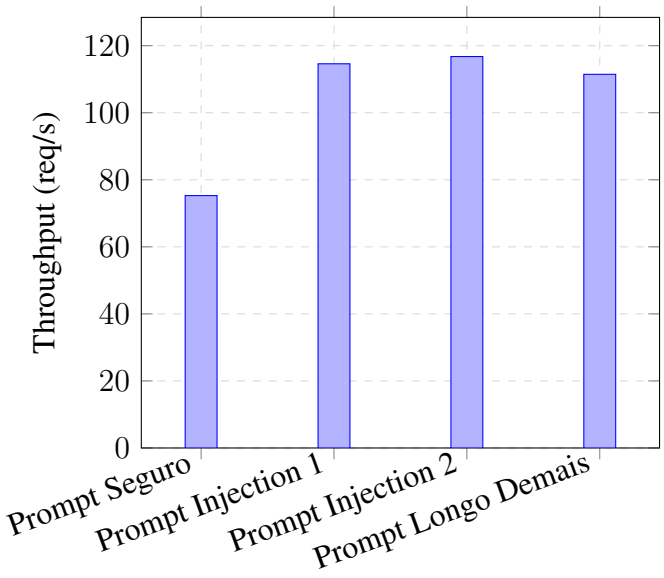


**Figura 2. Throughput observado em cada cenário de teste.**

Nos cenários de ataque, a latência média variou entre aproximadamente 8,56 ms e 8,97 ms, com picos de vazão em torno de 111,46 a 116,76 requisições por segundo. No cenário benigno, a latência média foi de 13,28 ms, com vazão aproximada de 75,28 requisições por segundo.

### 6.3. Discussão e Interpretação dos Dados

A análise combinada de segurança e desempenho sugere que o algoritmo de defesa opera com um mecanismo de rejeição antecipada, *fail-fast*:

1. Otimização sob ataque: a baixa latência nos cenários de injeção de *prompt* e de *Prompt Longo Demais* indica que o sistema identifica indícios de ataque nas camadas iniciais, possivelmente por checagem de tamanho de entrada ou *blacklist* de tokens, interrompendo o processamento o quanto antes e respondendo rapidamente com bloqueio.
2. Custo da validação profunda em tráfego legítimo: no caso de *Prompts Seguros*, o *input* atravessa todas as camadas de análise sem acionar bloqueios imediatos, o que gera latência maior, cerca de 55% acima dos cenários de ataque. Ainda assim, devido a limiares excessivamente rígidos ou regras muito genéricas, o sistema rejeita o tráfego benigno na etapa final, sacrificando disponibilidade em favor de uma integridade exageradamente conservadora.

Na prática, o sistema respondeu mais rápido sob ataque do que em tráfego normal, pois, uma vez detectadas assinaturas de ameaça nas camadas superficiais da verificação, o fluxo de processamento é interrompido de forma antecipada, economizando ciclos de CPU. Já as requisições legítimas percorrem todas as camadas de validação, consumindo mais tempo de processamento para, ao final, ainda serem rejeitadas devido à má calibração dos limiares de decisão.

Do ponto de vista de segurança, o comportamento observado é desejável sob ataque, pois não foram encontrados falsos negativos. No entanto, a incapacidade de distinguir nuances semânticas em interações legítimas torna a solução inviável para uso em produção no estado atual, pois o serviço permanece indisponível para todos os usuários legítimos.

### 6.4. Conclusão da Análise

Os resultados validam a arquitetura como altamente resiliente a vetores de ataque conhecidos, mas mostram que a configuração atual do guardrail é inadequada para cenários reais, devido à taxa de 100% de falsos positivos em tráfego benigno. Trabalhos futuros devem priorizar:

- ajuste fino dos limiares e pesos das regras de decisão, reduzindo a taxa de rejeição de usuários legítimos;
- adoção de sinais contextuais mais ricos e de classificadores semânticos complementares, mitigando a dependência exclusiva de *keywords* ou heurísticas simples;
- integração das métricas de conformidade regulatória e de privacidade aos experimentos automatizados, permitindo avaliar o equilíbrio entre segurança robusta, usabilidade e aderência a normas.

## 7. Conclusão e Próximos Passos

Apresentamos a lacuna de pesquisa e um plano metodológico para avaliá-la de forma integrada, com métricas comparáveis e geração de evidências de conformidade. Nesta etapa, avançamos do nível puramente conceitual para o planejamento detalhado do setup experimental, especificando hardware, software, *datasets*, arquitetura de middleware, contêineres e scripts de automação necessários para reproduzir o ambiente, bem como a organização concreta do repositório do projeto. Além disso, reportamos resultados experimentais iniciais que demonstram alta resiliência contra vetores de ataque conhecidos, ao custo de uma taxa de falsos positivos ainda inaceitável em cenários de uso legítimo, como evidenciado pelas Figuras 1 e 2.

Como próximos passos: materializar o *docker-compose* e os scripts de configuração adicionais, validando o setup em ambiente controlado; completar a implementação dos módulos de segurança, firewall LLM, RBAC adaptativo, sanitisadores e auditoria, dentro da estrutura de `src/`; definir em detalhe os cenários de ataque e parâmetros de testes A e B e de ablação, incluindo ataques mais sofisticados; executar os experimentos em escala ampliada, analisando *trade-offs* entre segurança, custo e latência, com especial atenção à redução de falsos positivos em tráfego benigno; e disponibilizar publicamente os artefatos e relatórios, consolidando o *framework* como referência reprodutível para avaliação de segurança, privacidade e conformidade em aplicações com LLMs.

## Referências


[1] Rathod, et al. (2024). Privacy and Security Challenges in Large Language Models.

[2] Yarram, et al. (2024). Privacy-Preserving Healthcare Data Security Using LLMs and Adaptive Access Control.

[3] Bunzel (2024). Compliance Made Practical: Translating the EU AI Act into Implementable Security Actions.